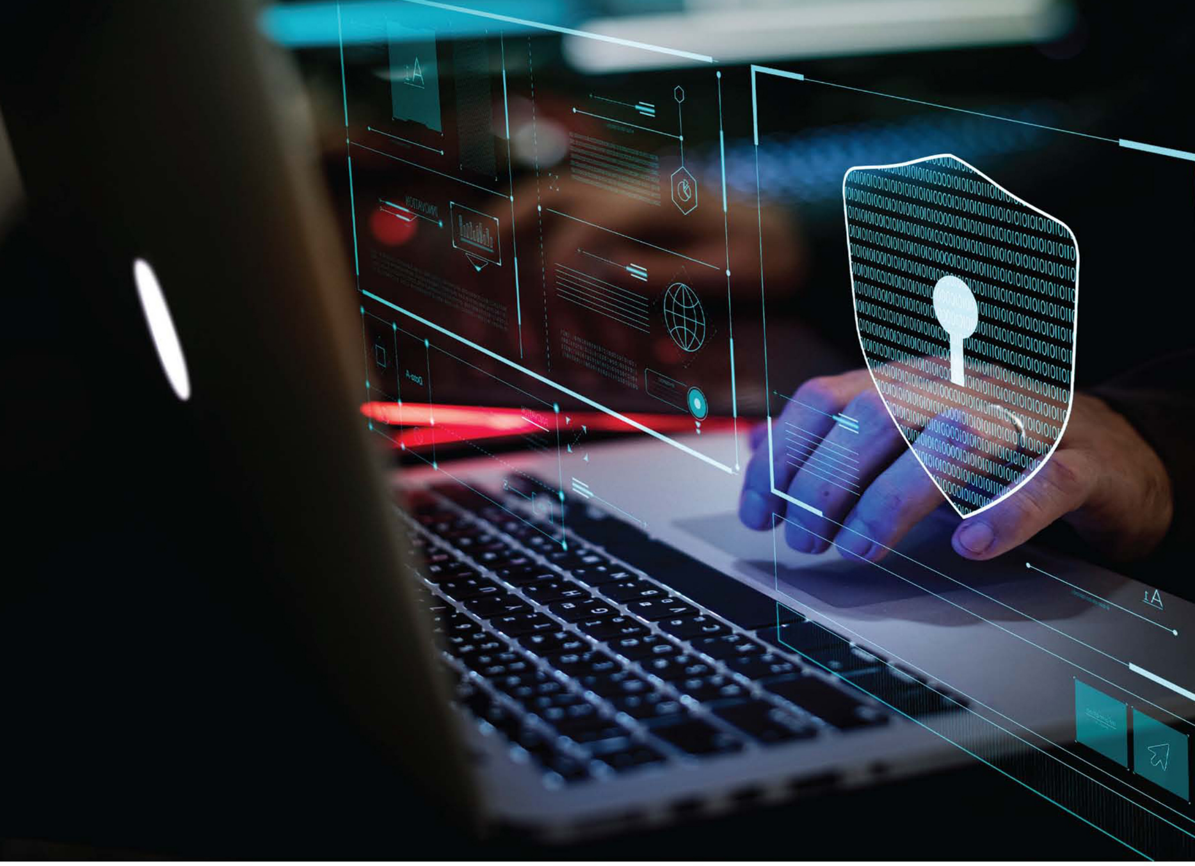


Secure Computing

Edited by: Cate Thomas

BIBLIOTEX
Digital Library

Secure Computing

Secure Computing

Editor:

Cate Thomas

BIBLIOTEX
Digital Library
www.bibliotex.com

Secure Computing

Editor: Cate Thomas

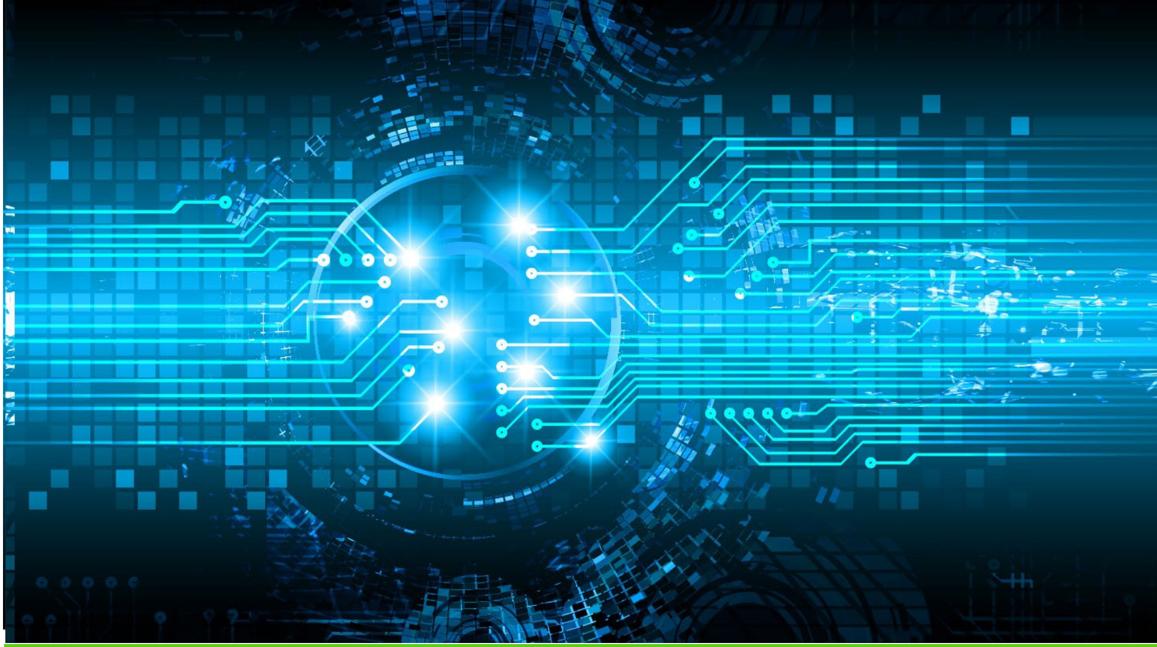
www.bibliotex.com

email: info@bibliotex.com

e-book Edition 2022

ISBN: 978-1-98467-777-8 (e-book)

TABLE OF CONTENTS



PREFACE

Data security has consistently been a major issue in information technology. In the cloud computing environment, it becomes particularly serious because the data is located in different places even in all the globe. The security of computer networks plays a strategic role in modern computer systems. In order to enforce high protection levels against malicious attack, a number of software tools have been currently developed. Intrusion Detection System has recently become a heated research topic due to its capability of detecting and preventing the attacks from malicious network users. Data security and privacy protection are the two main factors of user's concerns about the cloud technology. Though many techniques on the topics in cloud computing have been investigated in both academics and industries, data security and privacy protection are becoming more important for the future development of cloud computing technology in government, industry, and business. Data security and privacy protection issues are relevant to both hardware and software in the cloud architecture.

This book is aimed to cover different security techniques and challenges from both software and hardware aspects for protecting data in the cloud and aims at enhancing the data security and privacy protection for the trustworthy cloud environment. Network and computer security is critical to the financial health of every organization. Over the past few years, Internet-enabled business, or e-business, has drastically improved efficiency and revenue growth. E-business applications such as e-commerce, supply-chain management, and remote access allow companies to streamline processes, lower operating costs, and increase customer satisfaction. Such applications require mission-critical networks that accommodate voice, video, and data traffic, and these networks must be scalable to support increasing numbers of users and the need for greater capacity and performance. However, as networks enable more and more applications and are available to more and more users, they become ever more vulnerable to a wider range of security threats. To combat those threats and ensure that e-business transactions are not compromised, security technology must play a major role in today's networks. As time goes on, more and more new technology will be developed to further improve the efficiency of business and communications. At the same time, breakthroughs in technology will provide even greater network security, therefore, greater piece of mind to operate in cutting edge business environments. Provided that enterprises stay on top of this emerging technology, as well as the latest security threats and dangers, the benefits of networks will most certainly outweigh the risks.

CHAPTER 1

INFORMATION SECURITY

INTRODUCTION

Information security is a set of practices designed to keep personal data secure from unauthorized access and alteration during storing or transmitting from one place to another. Information security is designed and implemented to protect the print, electronic and other private, sensitive and personal data from unauthorized persons. It is used to protect data from being misused, disclosure, destruction, modification, and disruption. Information security, sometimes abbreviated to *infosec,* is a set of practices intended to keep data secure from unauthorized access or alterations, both when it›s being stored and when it›s being transmitted from one machine or physical location to another. You might sometimes see it referred to as *data security.* As knowledge has become one of the 21st century's most important assets, efforts to keep information secure have correspondingly become increasingly important.

Information security or infosec is concerned with protecting information from unauthorized access. It's part of information risk management and involves preventing or reducing the probability of unauthorized access, use, disclosure, disruption, deletion, corruption, modification, inspect, or recording. If a security incident does occur, information security professionals are involved with reducing the negative impact of the incident. Note information can be electronic or physical, tangible or intangible.

While the primary focus of any information security program is protecting the confidentiality, integrity and availability (the CIA triad) of information, maintaining organizational productivity is often an important consideration. This has led the information security industry to offer guidance, information security policies, and industry standards on passwords, antivirus software, firewalls, encryption software, legal liability and security awareness, to share best practices.

1.1 INFORMATION SECURITY (INFOSEC)

Information security (infosec) is a set of strategies for managing the processes, tools and policies necessary to prevent, detect, document and counter threats to digital and non-digital information. Infosec responsibilities include establishing a set of business processes that will protect information assets regardless of how the information is formatted or whether it is in transit, is being processed or is at rest in storage.

Many large enterprises employ a dedicated security group to implement and maintain the organization's infosec program. Typically, this group is led by a chief information security officer. The security group is generally responsible for conducting risk management, a process through which vulnerabilities and threats to information assets are continuously assessed, and the appropriate protective controls are decided on and applied. The value of an organization lies within its information its security is critical for business operations, as well as retaining credibility and earning the trust of clients.

1.1.1 Importance of Information Security

Information security, sometimes shortened to infosec, is the practice of protecting information by mitigating information risks. It is part of information risk management. It typically involves preventing or at least reducing the probability of unauthorized/inappropriate access to data, or the unlawful use, disclosure, disruption, deletion, corruption, modification, inspection, recording or devaluation of information. It also involves actions intended to reduce the adverse impacts of such incidents. Protected information may take any form, e.g. electronic or physical, tangible (e.g. paperwork) or intangible (e.g. knowledge). Information security's primary focus is the balanced protection of the confidentiality, integrity and availability of data (also known as the CIA triad) while maintaining a focus on efficient policy implementation, all without hampering organization productivity. This is largely achieved through a structured risk management process that involves:

- Identifying information and related assets, plus potential threats, vulnerabilities and impacts;
- Evaluating the risks;
- Deciding how to address or treat the risks i.e. to avoid, mitigate, share or accept them;
- Where risk mitigation is required, selecting or designing appropriate security controls and implementing them;
- Monitoring the activities, making adjustments as necessary to address any issues, changes and improvement opportunities.

To standardize this discipline, academics and professionals collaborate to offer guidance, policies, and industry standards on password, antivirus software, firewall, encryption software, legal liability, security awareness and training, and so forth. This standardization may be further driven by a wide variety of laws and regulations that affect how data is accessed, processed, stored, transferred and destroyed. However, the implementation of any standards and guidance within an entity may have limited effect if a culture of continual improvement is not adopted.

1.1.2 Principles of Information Security

The basic components of information security are most often summed up by the so-called CIA triad: confidentiality, integrity, and availability.

- ***Confidentiality*** is perhaps the element of the triad that most immediately comes to mind when you think of information security. Data is confidential when only those people who are authorized to access it can do so; to ensure confidentiality, you need to be able to identify who is trying to access data and block attempts by those without authorization. Passwords, encryption, authentication, and defense against penetration attacks are all techniques designed to ensure confidentiality.
- ***Integrity*** means maintaining data in its correct state and preventing it from being improperly modified, either by accident or maliciously. Many of the techniques that ensure confidentiality will also protect data integrity—after all, a hacker cannot change data they can't access—but there are other tools that help provide a defense of integrity in depth: checksums can help you verify data integrity, for instance, and version control software and frequent backups can help you restore data to a correct state if need be. Integrity also covers the concept of non-repudiation: you must be able to *prove* that you've maintained the integrity of your data, especially in legal contexts.
- ***Availability*** is the mirror image of confidentiality: while you need to make sure that your data can›t be accessed by unauthorized users, you also need to ensure that it *can* be accessed by those who have the proper permissions. Ensuring data availability means matching network and computing resources to the volume of data access you expect and implementing a good backup policy for disaster recovery purposes.

Apart from this there is one more principle that governs information security programs. This is Non repudiation.

- ***Non repudiation*** **–** means one party cannot deny receiving a message or a transaction nor can the other party deny sending a message or a transaction. For example in cryptography it is sufficient to show that message matches the digital signature signed with sender's private key and that sender could have a sent a message and nobody else could have altered it in transit. Data Integrity and Authenticity are pre-requisites for Non repudiation.
- ***Authenticity*** **–** means verifying that users are who they say they are and that each input arriving at destination is from a trusted source. This principle if followed guarantees the valid and genuine message received from a trusted source through a valid transmission. For example if take above example sender sends the message along with digital signature which was generated using the hash value of message and private key. Now at the receiver side this digital signature is decrypted using the public key generating a hash value and message is again hashed to generate the hash value. If the 2 value matches then it is known as valid transmission with the authentic or we say genuine message received at the recepient side
- ***Accountability*** **–** means that it should be possible to trace actions of an entity uniquely to that entity. For example as we discussed in Integrity section Not every employee should be allowed to do changes in other employees data. For this there is a separate department in an organization that is responsible for making such changes and when they receive request for a change then that letter must be signed by higher authority for example Director of college and person that is allotted that change will be able to do change after verifying his bio metrics, thus timestamp with the user(doing changes) details get recorded. Thus we can say if a change goes like this then

it will be possible to trace the actions uniquely to an entity.

At the core of Information Security is Information Assurance, which means the act of maintaining CIA of information, ensuring that information is not compromised in any way when critical issues arise. These issues are not limited to natural disasters, computer/server malfunctions etc.

Thus, the field of information security has grown and evolved significantly in recent years. It offers many areas for specialization, including securing networks and allied infrastructure, securing applications and databases, security testing, information systems auditing, business continuity planning etc.

In an ideal world, your data should always be kept confidential, in its correct state, and available; in practice, of course, you often need to make choices about which information security principles to emphasize, and that requires assessing your data. If you are storing sensitive medical information, for instance, you will focus on confidentiality, whereas a financial institution might emphasize data integrity to ensure that nobody's bank account is credited or debited incorrectly.

Infosec programs are built around the core objectives of the CIA triad: maintaining the confidentiality, integrity and availability of IT systems and business data. These objectives ensure that sensitive information is only disclosed to authorized parties (confidentiality), prevent unauthorized modification of data (integrity) and guarantee the data can be accessed by authorized parties when requested (availability).

The first security consideration, confidentiality, usually requires the use of encryption and encryption keys. The second consideration, integrity, implies that when data is read back, it will be exactly the same as when it was written. (In some cases, it may be necessary to send the same data to two different locations in order to protect against data corruption at one place.) The third part of the CIA is availability. This part of the triad seeks to ensure that new data can be used in a timely manner and backup data can be restored in an acceptable recovery time.

1.1.3 Threats and Threat Responses

Threats to sensitive and private information come in many different forms, such as malware and phishing attacks, identity theft and ransomware. To deter attackers and mitigate vulnerabilities at various points, multiple security controls are implemented and coordinated as part of a layered defense in depth strategy. This should minimize the impact of an attack. To be prepared for a security breach, security groups should have an incident response plan (IRP) in place. This should allow them to contain and limit the damage, remove the cause and apply updated defense controls.

Information security processes and policies typically involve physical and digital security measures to protect data from unauthorized access, use, replication or destruction. These measures can include mantraps, encryption key management, network intrusion detection systems, password policies and regulatory compliance. A security audit may be conducted to evaluate the organization's ability to maintain secure systems against a set of established criteria.

1.1.4 Information security vs. Network Security

In modern enterprise computing infrastructure, data is as likely to be in motion as it is to be at rest. This is where network security comes in. While technically a subset of cybersecurity, network security is primarily concerned with the networking infrastructure of the enterprise. It deals with issues such as securing the edge of the network; the data transport mechanisms, such as switches and routers; and those pieces of technology that provide protection for data as it moves between computing nodes. Where cybersecurity and network security differ is mostly in the application of security planning. A cybersecurity plan without a plan for network security is incomplete; however, a network security plan can typically stand alone.

Jobs in InfoSec

Jobs within the information security field vary in their titles, but some common designations include IT chief security officer (CSO), chief information security officer (CISO), security engineer, information security analyst, security systems administrator and IT security consultant.

InfoSec certifications

- *Certified Ethical Hacker (CEH)*: This is a vendor-neutral certification from the EC-Council, one of the leading certification bodies. This security certification, which validates how much an individual knows about network security, is best suited for a penetration tester role. This certification covers more than 270 attacks technologies. Prerequisites for this certification include attending official training offered by the EC-Council or its affiliates and having at least two years of information security-related experience.
- *Certified Information Systems Auditor (CISA):* This certification is offered by ISACA, a nonprofit, independent association that advocates for professionals involved in information security, assurance, risk management and governance. The exam certifies the knowledge and skills of security professionals. To qualify for this certification, candidates must have five years of professional work experience related to information systems auditing, control or security.
- *Certified information security manager (CISM):* CISM is an advanced certification offered by ISACA that provides validation for individuals who have demonstrated the in-depth knowledge and experience required to develop and manage an enterprise information security program.

The certification is aimed at information security managers, aspiring managers or IT consultants who support information security program management.

- *GIAC Security Essentials (GSEC):* This certification created and administered by the Global Information Assurance Certification organization is geared toward security professionals who want to demonstrate they are qualified for IT systems hands-on roles with respect to security tasks. Candidates are required to demonstrate they understand information security beyond simple terminology and concepts.

1.2 CONCEPT OF COMPUTER SECURITY

Computer security, the protection of computer systems and information from harm, theft, and unauthorized use. Computer hardware is typically protected by the same means used to protect other valuable or sensitive equipment, namely, serial numbers, doors and locks, and alarms. The protection of information and system access, on the other hand, is achieved through other tactics, some of them quite complex.

The security precautions related to computer information and access address four major threats: (1) theft of data, such as that of military secrets from government computers; (2) vandalism, including the destruction of data by a computer virus; (3) fraud, such as employees at a bank channeling funds into their own accounts; and (4) invasion of privacy, such as the illegal accessing of protected personal financial or medical data from a large database. The most basic means of protecting a computer system against theft, vandalism, invasion of privacy, and other irresponsible behaviors is to electronically track and record the access to, and activities of, the various users of a computer system. This is commonly done by assigning an individual password to each person who has access to a system. The computer system itself can then automatically track the use of these passwords, recording such data as which files were accessed under particular passwords and so on. Another security measure is to store a system's data on a separate device, or medium, such as magnetic tape or disks, that is normally inaccessible through the computer system. Finally, data is often encrypted so that it can be deciphered only by holders of a singular encryption key.

Computer security has become increasingly important since the late 1960s, when modems (devices that allow computers to communicate over telephone lines) were introduced. The proliferation of personal computers in the 1980s compounded the problem because they enabled hackers to illegally access major computer systems from the privacy of their homes. The development of advanced security techniques continues to diminish such threats, though concurrent refinements in the methods of computer crime pose ongoing hazards.

Computer security deals with the protection of computer systems and information from harm, theft, and unauthorized use. The main reason users get attacked frequently is that they lack adequate defenses to keep out intruders, and cybercriminals are quick to exploit such weaknesses. Computer security ensures the confidentiality, integrity, and availability of your computers and their stored data.

1.2.1 The Challenges of Computer Security

Computer security is both fascinating and complex. Some of the reasons follow:

- Computer security is not as simple as it might first appear to the novice. The requirements seem to be straightforward; indeed, most of the major requirements for security services can be given self-explanatory one-word labels: confidentiality, authentication, nonrepudiation, integrity. But the mechanisms used to meet those requirements can be quite complex, and understanding them may involve rather subtle reasoning.
- In developing a particular security mechanism or algorithm, one must always consider potential attacks on those security features. In many cases, successful attacks are designed by looking at the problem in a completely different way, therefore exploiting an unexpected weakness in the mechanism.

- Because of point 2, the procedures used to provide particular services are often counterintuitive. Typically, a security mechanism is complex, and it is not obvious from the statement of a particular requirement that such elaborate measures are needed. It is only when the various aspects of the threat are considered that elaborate security mechanisms make sense.

- Having designed various security mechanisms, it is necessary to decide where to use them. This is true both in terms of physical placement (e.g., at what points in a network are certain security mechanisms needed) and in a logical sense [e.g., at what layer or layers of an architecture such as TCP/IP (Transmission Control Protocol/Internet Protocol) should mechanisms be placed].
- Security mechanisms typically involve more than a particular algorithm or protocol. They also require that participants be in possession of some secret information (e.g., an encryption key), which raises questions about the creation, distribution, and protection of that secret information. There may also be a reliance on communications protocols whose behavior may complicate the task of developing the security mechanism. For example, if the proper functioning of the security mechanism requires setting time limits on the transit time of a message from sender to receiver, then any protocol or network that introduces variable, unpredictable delays may render such time limits meaningless.
- Computer security is essentially a battle of wits between a perpetrator who tries to find holes and the designer or administrator who tries to close them. The great advantage that the attacker has is that he or she need only find a single weakness while the designer must find and eliminate all weaknesses to achieve perfect security.
- There is a natural tendency on the part of users and system managers to perceive little benefit from security investment until a security failure occurs.
- Security requires regular, even constant, monitoring, and this is difficult in today's short-term, overloaded environment.

- Security is still too often an afterthought to be incorporated into a system after the design is complete rather than being an integral part of the design process.
- Many users and even security administrators view strong security as an impediment to efficient and user-friendly operation of an information system or use of information.

1.2.2 A Model for Computer Security

Table 1 defines terms and Figure 1, based on [CCPS12a], shows the relationship among some of these terms. We start with the concept of a system resource, or asset, that users and owners wish to protect. The assets of a computer system can be categorized as follows:

- Hardware: Including computer systems and other data processing, data storage, and data communications devices
- Software: Including the operating system, system utilities, and applications.
- Data: Including files and databases, as well as security-related data, such as password files.

- Communication facilities and networks: Local and wide area network communication links, bridges, routers, and so on.

Table 1. Computer Security Terminology

Adversary (threat agent)
An entity that attacks, or is a threat to, a system.

Attack
An assault on system security that derives from an intelligent threat; that is, an intelligent act that is a deliberate attempt (especially in the sense of a method or technique) to evade security services and violate the security policy of a system.

Countermeasure
An action, device, procedure, or technique that reduces a threat, a vulnerability, or an attack by eliminating or preventing it, by minimizing the harm it can cause, or by discovering and reporting it so that corrective action can be taken.

Risk
An expectation of loss expressed as the probability that a particular threat will exploit a particular vulnerability with a particular harmful result.

Security Policy
A set of rules and practices that specify or regulate how a system or organization provides security services to protect sensitive and critical system resources.

System Resource (Asset)
Data contained in an information system; or a service provided by a system; or a system capability, such as processing power or communication bandwidth; or an item of system equipment (i.e., a system component— hardware, firmware, software, or documentation); or a facility that houses system operations and equipment.

Threat
A potential for violation of security, which exists when there is a circumstance, capability, action, or event, that could breach security and cause harm. That is, a threat is a possible danger that might exploit a vulnerability.

Vulnerability
A flaw or weakness in a system's design, implementation, or operation and management that could be exploited to violate the system's security policy.

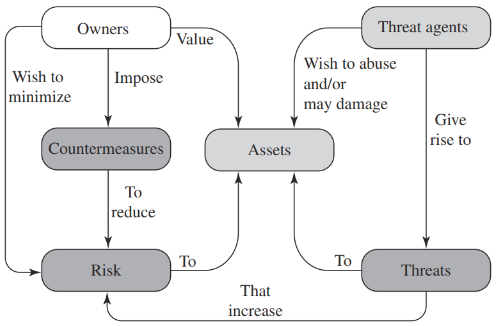


Figure 1. Security Concepts and Relationships.

In the context of security, our concern is with the vulnerabilities of system resources. Lists the following general categories of vulnerabilities of a computer system or network asset:

- It can be corrupted, so that it does the wrong thing or gives wrong answers. For example, stored data values may differ from what they should be because they have been improperly modified.
- It can become leaky. For example, someone who should not have access to some or all of the information available through the network obtains such access.
- It can become unavailable or very slow. That is, using the system or network becomes impossible or impractical.

Corresponding to the various types of vulnerabilities to a system resource are threats that are capable of exploiting those vulnerabilities. A threat represents a potential security harm to an asset. An attack is a threat that is carried out (threat action) and, if successful, leads to an undesirable violation of security, or threat consequence. The agent carrying out the attack is referred to as an attacker, or threat agent. We can distinguish two types of attacks:

- Active attack: An attempt to alter system resources or affect their operation.
- Passive attack: An attempt to learn or make use of information from the system that does not affect system

resources. We can also classify attacks based on the origin of the attack:

- Inside attack: Initiated by an entity inside the security perimeter (an "insider"). The insider is authorized to access system resources but uses them in a way not approved by those who granted the authorization.
- Outside attack: Initiated from outside the perimeter, by an unauthorized or illegitimate user of the system (an "outsider"). On the Internet, potential outside attackers range from amateur pranksters to organized criminals, international terrorists, and hostile governments.

Finally, a countermeasure is any means taken to deal with a security attack. Ideally, a countermeasure can be devised to prevent a particular type of attack from succeeding. When prevention is not possible, or fails in some instance, the goal is to detect the attack and then recover from the effects of the attack. A countermeasure may itself introduce new vulnerabilities. In any case, residual vulnerabilities may remain after the imposition of countermeasures. Such vulnerabilities may be exploited by threat agents representing a residual level of risk to the assets. Owners will seek to minimize that risk given other constraints.

1.2.3 Threats, Attacks, and Assets

We look at the types of security threats that must be dealt with, and then give some examples of the types of threats that apply to different categories of assets.

Threats and Attacks

Table 2, based on RFC 4949, describes four kinds of threat consequences and lists the kinds of attacks that result in each consequence.

Table 2. Threat Consequences, and the Types of Threat Actions that Cause Each Consequence

Threat Consequence	Threat Action (Attack)
Unauthorized Disclosure A circumstance or event whereby an entity gains access to data for which the entity is not authorized.	**Exposure:** Sensitive data are directly released to an unauthorized entity. **Interception:** An unauthorized entity directly accesses sensitive data traveling between authorized sources and destinations. **Inference:** A threat action whereby an unauthorized entity indirectly accesses sensitive data (but not necessarily the data contained in the communication) by reasoning from characteristics or by-products of communications. **Intrusion:** An unauthorized entity gains access to sensitive data by circumventing a system's security protections.
Deception A circumstance or event that may result in an authorized entity receiving false data and believing it to be true.	**Masquerade:** An unauthorized entity gains access to a system or performs a malicious act by posing as an authorized entity. **Falsification:** False data deceive an authorized entity. **Repudiation:** An entity deceives another by falsely denying responsibility for an act.
Disruption A circumstance or event that interrupts or prevents the correct operation of system services and functions.	**Incapacitation:** Prevents or interrupts system operation by disabling a system component. **Corruption:** Undesirably alters system operation by adversely modifying system functions or data. **Obstruction:** A threat action that interrupts delivery of system services by hindering system operation.
Usurpation A circumstance or event that results in control of system services or functions by an unauthorized entity.	**Misappropriation:** An entity assumes unauthorized logical or physical control of a system resource. **Misuse**: Causes a system component to perform a function or service that is detrimental to system security.

Unauthorized disclosure is a threat to confidentiality. The following types of attacks can result in this threat consequence:

- Exposure: This can be deliberate, as when an insider intentionally releases sensitive information, such as credit card numbers, to an outsider. It can also be the result of a human, hardware, or software error, which results in an entity gaining unauthorized knowledge of sensitive data. There have been numerous instances of this, such as universities accidentally posting student confidential information on the Web.
- Interception: Interception is a common attack in the context of communications. On a shared local area network (LAN), such as a wireless LAN or a broadcast Ethernet, any device attached to the LAN can receive a copy of packets intended for another device. On the Internet, a determined hacker can gain access to e-mail traffic and other data transfers. All of these situations create the potential for unauthorized access to data.
- Inference: An example of inference is known as traffic analysis, in which an adversary is able to gain information from observing the pattern of traffic on a network, such as the amount of traffic between particular pairs of hosts on the network. Another example is the inference of detailed information from a database by a user who has only limited access; this is accomplished by repeated queries whose combined results enable inference.
- Intrusion: An example of intrusion is an adversary gaining unauthorized access to sensitive data by overcoming the system's access control protections.

Deception is a threat to either system integrity or data integrity. The following types of attacks can result in this threat consequence:

- Masquerade: One example of masquerade is an attempt by an unauthorized user to gain access to a system by posing as an authorized user; this could happen if the unauthorized user has learned another user's logon ID and password. Another example is malicious logic, such as a Trojan horse, that appears to perform a useful or desirable function but actually gains unauthorized access to system resources or tricks a user into executing other malicious logic.

- Falsification: This refers to the altering or replacing of valid data or the introduction of false data into a file or database. For example, a student may alter his or her grades on a school database.
- Repudiation: In this case, a user either denies sending data or a user denies receiving or possessing the data.

Disruption is a threat to availability or system integrity. The following types of attacks can result in this threat consequence:

- Incapacitation: This is an attack on system availability. This could occur as a result of physical destruction of or damage to system hardware. More typically, malicious software, such as Trojan horses, viruses, or worms, could operate in such a way as to disable a system or some of its services.
- Corruption: This is an attack on system integrity. Malicious software in this context could operate in such a way that system resources or services function in an unintended manner. Or a user could gain unauthorized access to a system and modify some of its functions. An example of the latter is a user placing backdoor logic in the system to provide subsequent access to a system and its resources by other than the usual procedure.
- Obstruction: One way to obstruct system operation is to interfere with communications by disabling communication links or altering communication control information. Another way is to overload the system by placing excess burden on communication traffic or processing resources.

Usurpation is a threat to system integrity. The following types of attacks can result in this threat consequence:

- Misappropriation: This can include theft of service. An example is a distributed denial of service attack, when malicious software is installed on a number of hosts to be used as platforms to launch traffic at a target host. In this case, the malicious software makes unauthorized use of processor and operating system resources.
- Misuse: Misuse can occur by means of either malicious logic or a hacker that has gained unauthorized access to a system. In either case, security functions can be disabled or thwarted.

Threats and Assets

The assets of a computer system can be categorized as hardware, software, data, and communication lines and networks. We briefly describe these four categories and relate these to the concepts of integrity, confidentiality, and availability introduced (see Figure 2 and Table 3).

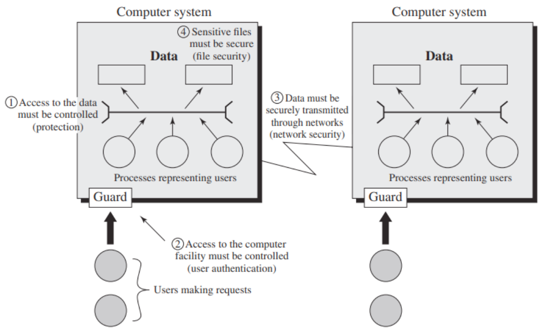

Computer system
④ Sensitive files must be secure (file security)
Data
① Access to the data must be controlled (protection)
Processes representing users
Guard
③ Data must be securely transmitted through networks (network security)
Computer system
Data
Processes representing users
Guard
② Access to the computer facility must be controlled (user authentication)
Users making requests

Figure 2. Scope of Computer Security.

Table 3. Computer and Network Assets, with Examples of Threats

	Availability	Confidentiality	Integrity
Hardware	Equipment is stolen or disabled, thus denying service.	An unencrypted CD-ROM or DVD is stolen.	
Software	Programs are deleted, denying access to users.	An unauthorized copy of software is made.	A working program is modified, either to cause it to fail during execution or to cause it to do some unintended task.
Data	Files are deleted, denying access to users.	An unauthorized read of data is performed. An analysis of statistical data reveals underlying data.	Existing files are modified or new files are fabricated.
Communication Lines and Networks	Messages are destroyed or deleted. Communication lines or networks are rendered unavailable.	Messages are read. The traffic pattern of messages is observed.	Messages are modified, delayed, reordered, or duplicated. False messages are fabricated.

Hardware

A major threat to computer system hardware is the threat to availability. Hardware is the most vulnerable to attack and the least susceptible to automated controls. Threats include accidental and deliberate damage to equipment as well as theft. The proliferation of personal computers and workstations and the widespread use of LANs increase the potential for losses in this area. Theft of CD-ROMs and DVDs can lead to loss of confidentiality. Physical and administrative security measures are needed to deal with these threats.

Software

Software includes the operating system, utilities, and application programs. A key threat to software is an attack on availability. Software, especially application software, is often easy to delete. Software can also be altered or damaged to render it useless. Careful software configuration management, which includes making backups of the most recent version of software, can maintain high availability. A more difficult problem to deal with is software modification that results in a program that still functions but that behaves differently than before, which is a threat to integrity/authenticity. Computer viruses and related attacks fall into this category. A final problem is protection against software piracy. Although certain countermeasures are available, by and large the problem of unauthorized copying of software has not been solved.

Data

Hardware and software security are typically concerns of computing center professionals or individual concerns of personal computer users. A much more widespread problem is data security, which involves files and other forms of data controlled by individuals, groups, and business organizations. Security concerns with respect to data are broad, encompassing availability, secrecy, and integrity. In the case of availability, the concern is with the destruction of data files, which can occur either accidentally or maliciously.

The obvious concern with secrecy is the unauthorized reading of data files or databases, and this area has been the subject of perhaps more research and effort than any other area of computer security. A less obvious threat to secrecy involves the analysis of data and manifests itself in the use of so-called statistical databases, which provide summary or aggregate information. Presumably, the existence of aggregate information does not threaten the privacy of the individuals involved. However, as the use of statistical databases grows, there is an increasing potential for disclosure of personal information. In essence, characteristics of constituent individuals may be identified through careful analysis. For example, if one table records the aggregate of the incomes of respondents A, B, C, and D and another records the aggregate of the incomes of A, B, C, D, and E, the difference between the two aggregates would be the income of E. This problem is exacerbated by the increasing desire to combine data sets. In many cases, matching several sets of data for consistency at different levels of aggregation requires access to individual units. Thus, the individual units, which are the subject of privacy concerns, are available at various stages in the processing of data sets.

Finally, data integrity is a major concern in most installations. Modifications to data files can have consequences ranging from minor to disastrous.

1.2.4 Communication Lines and Networks

Network security attacks can be classified as passive attacks and active attacks. A passive attack attempts to learn or make use of information from the system but does not affect system resources. An active attack attempts to alter system resources or affect their operation. Passive attacks are in the nature of eavesdropping on, or monitoring of, transmissions. The goal of the attacker is to obtain information that is being transmitted. Two types of passive attacks are the release of message contents and traffic analysis. The release of message contents is easily understood. A telephone conversation, an electronic mail message, and a transferred file may contain sensitive or confidential information. We would like to prevent an opponent from learning the contents of these transmissions.

A second type of passive attack, traffic analysis, is subtler. Suppose that we had a way of masking the contents of messages or other information traffic so that opponents, even if they captured the message, could not extract the information from the message. The common technique for masking contents is encryption. If we had encryption protection in place, an opponent might still be able to observe the pattern of these messages. The opponent could determine the location and identity of communicating hosts and could observe the frequency and length of messages being exchanged. This information might be useful in guessing the nature of the communication that was taking place.

Passive attacks are very difficult to detect because they do not involve any alteration of the data. Typically, the message traffic is sent and received in an apparently normal fashion and neither the sender nor receiver is aware that a third party has read the messages or observed the traffic pattern. However, it is feasible to prevent the success of these attacks, usually by means of encryption. Thus, the emphasis in dealing with passive attacks is on prevention rather than detection.

Active attacks involve some modification of the data stream or the creation of a false stream and can be subdivided into four categories: replay, masquerade, modification of messages, and denial of service. Replay involves the passive capture of a data unit and its subsequent retransmission to produce an unauthorized effect. A masquerade takes place when one entity pretends to be a different entity. A masquerade attack usually includes one of the other forms of active attack. For example, authentication sequences can be captured and replayed after a valid authentication sequence has taken place, thus enabling an authorized entity with few privileges to obtain extra privileges by impersonating an entity that has those privileges.

Modification of messages simply means that some portion of a legitimate message is altered, or that messages are delayed or reordered, to produce an unauthorized effect. For example, a message stating, "Allow John Smith to read confidential file accounts" is modified to say, "Allow Fred Brown to read confidential file accounts."

The denial of service prevents or inhibits the normal use or management of communication facilities. This attack may have a specific target; for example, an entity may suppress all messages directed to a particular destination (e.g., the security audit service). Another form of service denial is the disruption of an entire network, either by disabling the network or by overloading it with messages so as to degrade performance.

Active attacks present the opposite characteristics of passive attacks. Whereas passive attacks are difficult to detect, measures are available to prevent their success. On the other hand, it is quite difficult to prevent active attacks absolutely, because to do so would require physical protection of all communication facilities and paths at all times. Instead, the goal is to detect them and to recover from any disruption or delays caused by them. Because the detection has a deterrent effect, it may also contribute to prevention.

1.2.5 Security Functional Requirements

There are a number of ways of classifying and characterizing the countermeasures that may be used to reduce vulnerabilities and deal with threats to system assets. We view countermeasures in terms of functional requirements, and we follow the classification defined in FIPS 200 (Minimum Security Requirements for Federal Information and Information Systems). This standard enumerates 17 security-related areas with regard to protecting the confidentiality, integrity, and availability of information systems and the information processed, stored, and transmitted by those systems. The areas are defined in Table 4.

The requirements listed in FIPS 200 encompass a wide range of countermeasures to security vulnerabilities and threats. Roughly, we can divide these countermeasures into two categories: those that require computer security technical measures, either hardware or software, or both; and those that are fundamentally management issues.

Each of the functional areas may involve both computer security technical measures and management measures. Functional areas that primarily require computer security technical measures include access control, identification and authentication, system and communication protection, and system and information integrity. Functional areas that primarily involve management controls and procedures include awareness and training; audit and accountability; certification, accreditation, and security assessments; contingency planning; maintenance; physical and environmental protection; planning; personnel security; risk assessment; and systems and services acquisition. Functional areas that overlap computer security technical measures and management controls include configuration management, incident response, and media protection.

Note that the majority of the functional requirements areas in FIPS 200 are either primarily issues of management or at least have a significant management component, as opposed to purely software or hardware solutions. But as one computer security expert observed, “If you think technology can solve your security problems, then you don’t understand the problems and you don’t technology”.

Table 4. Security Requirements

Access Control: Limit information system access to authorized users, processes acting on behalf of authorized users, or devices (including other information systems) and to the types of transactions and functions that authorized users are permitted to exercise.
Awareness and Training: (i) Ensure that managers and users of organizational information systems are made aware of the security risks associated with their activities and of the applicable laws, regulation, and policies related to the security of organizational information systems; and (ii) ensure that personnel are adequately trained to carry out their assigned information security-related duties and responsibilities.
Audit and Accountability: (i) Create, protect, and retain information system audit records to the extent needed to enable the monitoring, analysis, investigation, and reporting of unlawful, unauthorized, or inappropriate information system activity; and (ii) ensure that the actions of individual information system users can be uniquely traced to those users so they can be held accountable for their actions.
Certification, Accreditation, and Security Assessments: (i) Periodically assess the security controls in organizational information systems to determine if the controls are effective in their application; (ii) develop and implement plans of action designed to correct deficiencies and reduce or eliminate vulnerabilities in organizational information systems; (iii) authorize the operation of organizational information systems and any associated information system connections; and (iv) monitor information system security controls on an ongoing basis to ensure the continued effectiveness of the controls.
Configuration Management: (i) Establish and maintain baseline configurations and inventories of organizational information systems (including hardware, software, firmware, and documentation) throughout the respective system development life cycles; and (ii) establish and enforce security configuration settings for information technology products employed in organizational information systems.
Contingency Planning: Establish, maintain, and implement plans for emergency response, backup operations, and postdisaster recovery for organizational information systems to ensure the availability of critical information resources and continuity of operations in emergency situations.
Identification and Authentication: Identify information system users, processes acting on behalf of users, or devices, and authenticate (or verify) the identities of those users, processes, or devices, as a prerequisite to allowing access to organizational information systems.
Incident Response: (i) Establish an operational incident-handling capability for organizational information systems that includes adequate preparation, detection, analysis, containment, recovery, and user-response activities; and (ii) track, document, and report incidents to appropriate organizational officials and/or authorities.
Maintenance: (i) Perform periodic and timely maintenance on organizational information systems; and (ii) provide effective controls on the tools, techniques, mechanisms, and personnel used to conduct information system maintenance.
Media Protection: (i) Protect information system media, both paper and digital; (ii) limit access to information on information system media to authorized users; and (iii) sanitize or destroy information system media before disposal or release for reuse.

Physical and Environmental Protection: (i) Limit physical access to information systems, equipment, and the respective operating environments to authorized individuals; (ii) protect the physical plant and support infrastructure for information systems; (iii) provide supporting utilities for information systems; (iv) protect information systems against environmental hazards; and (v) provide appropriate environmental controls in facilities containing information systems.

Planning: Develop, document, periodically update, and implement security plans for organizational information systems that describe the security controls in place or planned for the information systems and the rules of behavior for individuals accessing the information systems.

Personnel Security: (i) Ensure that individuals occupying positions of responsibility within organizations (including third-party service providers) are trustworthy and meet established security criteria for those positions; (ii) ensure that organizational information and information systems are protected during and after personnel actions such as terminations and transfers; and (iii) employ formal sanctions for personnel failing to comply with organizational security policies and procedures.

Risk Assessment: Periodically assess the risk to organizational operations (including mission, functions, image, or reputation), organizational assets, and individuals, resulting from the operation of organizational information systems and the associated processing, storage, or transmission of organizational information.

Systems and Services Acquisition: (i) Allocate sufficient resources to adequately protect organizational information systems; (ii) employ system development life cycle processes that incorporate information security considerations; (iii) employ software usage and installation restrictions; and (iv) ensure that thirdparty providers employ adequate security measures to protect information, applications, and/or services outsourced from the organization.

System and Communications Protection: (i) Monitor, control, and protect organizational communications (i.e., information transmitted or received by organizational information systems) at the external boundaries and key internal boundaries of the information systems; and (ii) employ architectural designs, software development techniques, and systems engineering principles that promote effective information security within organizational information systems.

System and Information Integrity: (i) Identify, report, and correct information and information system flaws in a timely manner; (ii) provide protection from malicious code at appropriate locations within organizational information systems; and (iii) monitor information system security alerts and advisories and take appropriate actions in response.

1.3 FUNDAMENTAL SECURITY DESIGN PRINCIPLES

Despite years of research and development, it has not been possible to develop security design and implementation techniques that systematically exclude security flaws and prevent all unauthorized actions. In the absence of such foolproof techniques, it is useful to have a set of widely agreed design principles that can guide the development of protection mechanisms. The National Centers of Academic Excellence in Information Assurance/Cyber Defense, which is jointly sponsored by the U.S. National Security Agency and the U. S. Department of Homeland Security, list the following as fundamental security design principles:

- Economy of mechanism
- Fail-safe defaults
- Complete mediation
- Open design
- Separation of privilege
- Least privilege
- Least common mechanism
- Psychological acceptability
- Isolation
- Encapsulation
- Modularity
- Layering
- Least astonishment

Economy of mechanism means that the design of security measures embodied in both hardware and software should be as simple and small as possible. The motivation for this principle is that relatively simple, small design is easier to test and verify thoroughly. With a complex design, there are many more opportunities for an adversary to discover subtle weaknesses to exploit that may be difficult to spot ahead of time. The more complex the mechanism, the more likely it is to possess exploitable flaws. Simple mechanisms tend to have fewer exploitable flaws and require less maintenance. Furthermore, because configuration management issues are simplified, updating or replacing a simple mechanism becomes a less intensive process. In practice, this is perhaps the most difficult principle to honor. There is a constant demand for new features in both hardware and software, complicating the security design task. The best that can be done is to keep this principle in mind during system design to try to eliminate unnecessary complexity.

Fail-safe default means that access decisions should be based on permission rather than exclusion. That is, the default situation is lack of access, and the protection scheme identifies conditions under which access is permitted. This approach exhibits a better failure mode than the alternative approach, where the default is to permit access. A design or implementation mistake in a mechanism that gives explicit permission tends to fail by refusing permission, a safe situation that can be quickly detected. On the other hand, a design or implementation mistake in a mechanism that explicitly excludes access tends to fail by allowing access, a failure that may long go unnoticed in normal use. For example, most file access systems work on this principle and virtually all protected services on client/server systems work this way.

Complete mediation means that every access must be checked against the access control mechanism. Systems should not rely on access decisions retrieved from a cache. In a system designed to operate continuously, this principle requires that, if access decisions are remembered for future use, careful consideration be given to how changes in authority are propagated into such local memories. File access systems appear to provide an example of a system that complies with this principle. However, typically, once a user has opened a file, no check is made to see of permissions change. To fully implement complete mediation, every time a user reads a field or record in a file, or a data item in a database, the system must exercise access control. This resource-intensive approach is rarely used.

Open design means that the design of a security mechanism should be open rather than secret. For example, although encryption keys must be secret, encryption algorithms should be open to public scrutiny. The algorithms can then be reviewed by many experts, and users can therefore have high confidence in them. This is the philosophy behind the National Institute of Standards and Technology (NIST) program of standardizing encryption and hash algorithms, and has led to the widespread adoption of NIST-approved algorithms.

Separation of privilege is defined in [SALT75] as a practice in which multiple privilege attributes are required to achieve access to a restricted resource. A good example of this is multifactor user authentication, which requires the use of multiple techniques, such as a password and a smart card, to authorize a user. The term is also now applied to any technique in which a program is divided into parts that are limited to the specific privileges they require in order to perform a specific task. This is used to mitigate the potential damage of a computer security attack. One example of this latter interpretation of the principle is removing high privilege operations to another process and running that process with the higher privileges required to perform its tasks. Day-to-day interfaces are executed in a lower privileged process.

Least privilege means that every process and every user of the system should operate using the least set of privileges necessary to perform the task. A good example of the use of this principle is role-based access control. The system security policy can identify and define the various roles of users or processes. Each role is assigned only those permissions needed to perform its functions. Each permission specifies a permitted access to a particular resource (such as read and write access to a specified file or directory, and connect access to a given host and port). Unless permission is granted explicitly, the user or process should not be able to access the protected resource. More generally, any access control system should allow each user only the privileges that are authorized for that user. There is also a temporal aspect to the least privilege principle. For example, system programs or administrators who have special privileges should have those privileges only when necessary; when they are doing ordinary activities the privileges should be withdrawn. Leaving them in place just opens the door to accidents.

Least common mechanism means that the design should minimize the functions shared by different users, providing mutual security. This principle helps reduce the number of unintended communication paths and reduces the amount of hardware and software on which all users depend, thus making it easier to verify if there are any undesirable security implications.

Psychological acceptability implies that the security mechanisms should not interfere unduly with the work of users, while at the same time meeting the needs of those who authorize access. If security mechanisms hinder the usability or accessibility of resources, users may opt to turn off those mechanisms. Where possible, security mechanisms should be transparent to the users of the system or at most introduce minimal obstruction. In addition to not being intrusive or burdensome, security procedures must reflect the user's mental model of protection. If the protection procedures do not make sense to the user or if the user must translate his image of protection into a substantially different protocol, the user is likely to make errors.

Isolation is a principle that applies in three contexts. First, public access systems should be isolated from critical resources (data, processes, etc.) to prevent disclosure or tampering. In cases where the sensitivity or criticality of the information is high, organizations may want to limit the number of systems on which that data are stored and isolate them, either physically or logically. Physical isolation may include ensuring that no physical connection exists between an organization's public access information resources and an organization's critical information. When implementing logical isolation solutions, layers of security services and mechanisms should be established between public systems and secure systems responsible for protecting critical resources. Second, the processes and files of individual users should be isolated from one another except where it is explicitly desired. All modern operating systems provide facilities for such isolation, so that individual users have separate, isolated process space, memory space, and file space, with protections for preventing unauthorized access. And finally, security mechanisms should be isolated in the sense of preventing access to those mechanisms. For example, logical access control may provide a means of isolating cryptographic software from other parts of the host system and for protecting cryptographic software from tampering and the keys from replacement or disclosure.

Encapsulation can be viewed as a specific form of isolation based on object oriented functionality. Protection is provided by encapsulating a collection of procedures and data objects in a domain of its own so that the internal structure of a data object is accessible only to the procedures of the protected subsystem and the procedures may be called only at designated domain entry points.

Modularity in the context of security refers both to the development of security functions as separate, protected modules and to the use of a modular architecture for mechanism design and implementation. With respect to the use of separate security modules, the design goal here is to provide common security functions and services, such as cryptographic functions, as common modules. For example, numerous protocols and applications make use of cryptographic functions. Rather than implementing such functions in each protocol or application, a more secure design is provided by developing a common cryptographic module that can be invoked by numerous protocols and applications. The design and implementation effort can then focus on the secure design and implementation of a single cryptographic module, including mechanisms to protect the module from tampering. With respect to the use of a modular architecture, each security mechanism should be able to support migration to new technology or upgrade of new features without requiring an entire system redesign. The security design should be modular so that individual parts of the security design can be upgraded without the requirement to modify the entire system.

Layering refers to the use of multiple, overlapping protection approaches addressing the people, technology, and operational aspects of information systems. By using multiple, overlapping protection approaches, the failure or circumvention of any individual protection approach will not leave the system unprotected.

Least astonishment means that a program or user interface should always respond in the way that is least likely to astonish the user. For example, the mechanism for authorization should be transparent enough to a user that the user has a good intuitive understanding of how the security goals map to the provided security mechanism.

1.3.1 Attack Surfaces and Attack Trees

We provided an overview of the spectrum of security threats and attacks facing computer and network systems. We elaborate on two concepts that are useful in evaluating and classifying threats: attack surfaces and attack trees.

Attack Surfaces

An attack surface consists of the reachable and exploitable vulnerabilities in a system. Examples of attack surfaces are the following:

- Open ports on outward facing Web and other servers, and code listening on those ports
- Services available on the inside of a firewall
- Code that processes incoming data, email, XML, office documents, and industry specific custom data exchange formats
- Interfaces, SQL, and Web forms
- An employee with access to sensitive information vulnerable to a social engineering attack

Attack surfaces can be categorized in the following way:

- Network attack surface: This category refers to vulnerabilities over an enterprise network, wide-area network, or the Internet. Included in this category are network protocol vulnerabilities, such as those used for a denial-of-service attack, disruption of communications links, and various forms of intruder attacks.

- Software attack surface: This refers to vulnerabilities in application, utility, or operating system code. A particular focus in this category is Web server software.
- Human attack surface: This category refers to vulnerabilities created by personnel or outsiders, such as social engineering, human error, and trusted insiders.

An attack surface analysis is a useful technique for assessing the scale and severity of threats to a system. A systematic analysis of points of vulnerability makes developers and security analysts aware of where security mechanisms are required. Once an attack surface is defined, designers may be able to find ways to make the surface smaller, thus making the task of the adversary more difficult. The attack surface also provides guidance on setting priorities for testing, strengthening security measures, or modifying the service or application.

As illustrated in Figure 3, the use of layering, or defense in depth, and attack surface reduction complement each other in mitigating security risk.

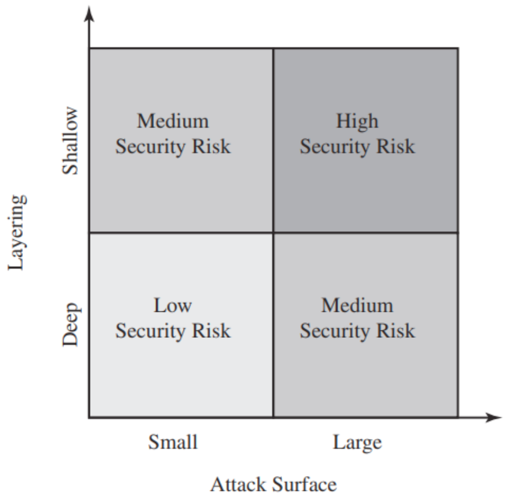


Figure 3. Defense in Depth and Attack Surface.

Attack Trees

An attack tree is a branching, hierarchical data structure that represents a set of potential techniques for exploiting security vulnerabilities. The security incident that is the goal of the attack is represented as the root node of the tree, and the ways that an attacker could reach that goal are iteratively and incrementally represented as branches and subnodes of the tree. Each subnode defines a subgoal, and each subgoal may have its own set of further subgoals, etc. The final nodes on the paths outward from the root, i.e., the leaf nodes, represent different ways to initiate an attack. Each node other than a leaf is either an AND-node or an OR-node. To achieve the goal represented by an AND-node, the subgoals represented by all of that node's subnodes must be achieved; and for an OR-node, at least one of the subgoals must be achieved. Branches can be labeled with values representing difficulty, cost, or other attack attributes, so that alternative attacks can be compared.

The motivation for the use of attack trees is to effectively exploit the information available on attack patterns. Organizations such as CERT publish security advisories that have enabled the development of a body of knowledge about both general attack strategies and specific attack patterns. Security analysts can use the attack tree to document security attacks in a structured form that reveals key vulnerabilities. The attack tree can guide both the design of systems and applications, and the choice and strength of countermeasures.

Figure 4, is an example of an attack tree analysis for an Internet banking authentication application. The root of the tree is the objective of the attacker, which is to compromise a user's account. The shaded boxes on the tree are the leaf nodes, which represent events that comprise the attacks. The white boxes are categories which consist of one or more specific attack events (leaf nodes). Note that in this tree, all the nodes other than leaf nodes are OR-nodes. The analysis used to generate this tree considered the three components involved in authentication:

- User terminal and user (UT/U): These attacks target the user equipment, including the tokens that may be involved, such as smartcards or other password generators, as well as the actions of the user.
- Communications channel (CC): This type of attack focuses on communication links.
- Internet banking server (IBS): These types of attacks are offline attack against the servers that host the Internet banking application.

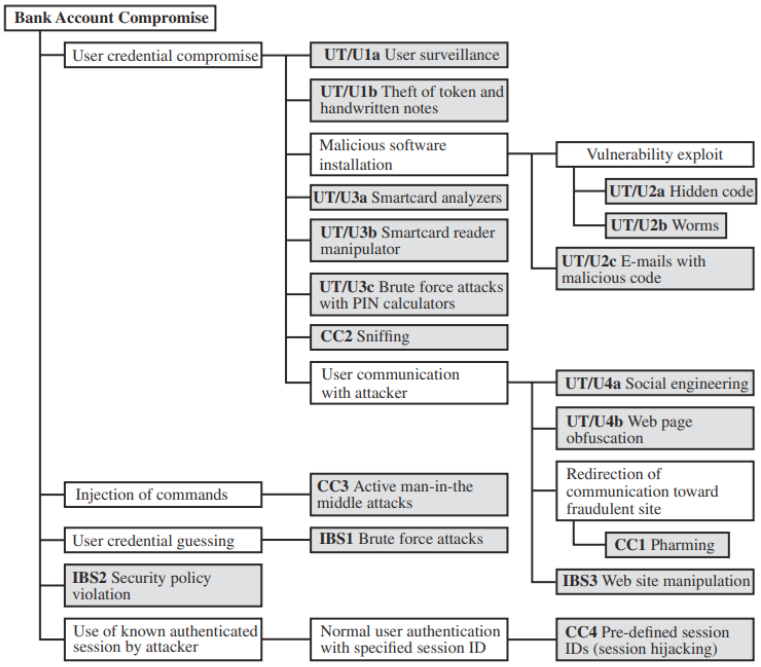

Bank Account Compromise
User credential compromise
UT/U1a User surveillance
UT/U1b Theft of token and handwritten notes
Malicious software installation
Vulnerability exploit
UT/U2a Hidden code
UT/U2b Worms
UT/U2c E-mails with malicious code
UT/U3a Smartcard analyzers
UT/U3b Smartcard reader manipulator
UT/U3c Brute force attacks with PIN calculators
CC2 Sniffing
User communication with attacker
UT/U4a Social engineering
UT/U4b Web page obfuscation
Redirection of communication toward fraudulent site
CC1 Pharming
IBS3 Web site manipulation
Injection of commands
CC3 Active man-in-the middle attacks
User credential guessing
IBS1 Brute force attacks
IBS2 Security policy violation
Use of known authenticated session by attacker
Normal user authentication with specified session ID
CC4 Pre-defined session IDs (session hijacking)

Figure 4. An Attack Tree for Internet Banking Authentication.

Five overall attack strategies can be identified, each of which exploits one or more of the three components. The five strategies are as follows:

- User credential compromise: This strategy can be used against many elements of the attack surface. There are procedural attacks, such as monitoring a user's action to observe a PIN or other credential, or theft of the user's token or handwritten notes. An adversary may also compromise token information using a variety of token attack tools, such as hacking the smartcard or using a brute force approach to guess the PIN. Another possible strategy is to embed malicious software to compromise the user's login and password. An adversary may also attempt to obtain credential information via the communication channel (sniffing). Finally, an adversary may use various means to engage in communication with the target user, as shown in Figure 4.

- Injection of commands: In this type of attack, the attacker is able to intercept communication between the UT and the IBS. Various schemes can be used to be able to impersonate the valid user and so gain access to the banking system.
- User credential guessing: It is reported in [HILT06] that brute force attacks against some banking authentication schemes are feasible by sending random usernames and passwords. The attack mechanism is based on distributed zombie personal computers, hosting automated programs for username- or password-based calculation.
- Security policy violation: For example, violating the bank's security policy in combination with weak access control and logging mechanisms, an employee may cause an internal security incident and expose a customer's account.
- Use of known authenticated session: This type of attack persuades or forces the user to connect to the IBS with a preset session ID. Once the user authenticates to the server, the attacker may utilize the known session ID to send packets to the IBS, spoofing the user's identity.

Figure 4 provides a thorough view of the different types of attacks on an Internet banking authentication application. Using this tree as a starting point, security analysts can assess the risk of each attack and, using the design principles outlined in the preceding section, design a comprehensive security facility.

1.4 COMPUTER SECURITY STRATEGY

We conclude this chapter with a brief look at the overall strategy for providing computer security. Suggests that a comprehensive security strategy involves three aspects:

- Specification/policy: What is the security scheme supposed to do?
- Implementation/mechanisms: How does it do it?
- Correctness/assurance: Does it really work?

1.4.1 Security Policy

The first step in devising security services and mechanisms is to develop a security policy. Those involved with computer security use the term security policy in various ways. At the least, a security policy is an informal description of desired system behavior. Such informal policies may reference requirements for security, integrity, and availability. More usefully, a security policy is a formal statement of rules and practices that specify or regulate how a system or organization provides security services to protect sensitive and critical system resources. Such a formal security policy lends itself to being enforced by the system's technical controls as well as its management and operational controls.

In developing a security policy, a security manager needs to consider the following factors:

- The value of the assets being protected
- The vulnerabilities of the system
- Potential threats and the likelihood of attacks

Further, the manager must consider the following trade-offs:

- Ease of use versus security: Virtually all security measures involve some penalty in the area of ease of use. The following are some examples. Access control mechanisms require users to remember passwords and perhaps perform other access control actions. Firewalls and other network security measures may reduce available transmission capacity or slow response time. Virus-checking software reduces available processing power and introduces the possibility of system crashes or malfunctions due to improper interaction between the security software and the operating system.

- Cost of security versus cost of failure and recovery: In addition to ease of use and performance costs, there are direct monetary costs in implementing and maintaining security measures. All of these costs must be balanced against the cost of security failure and recovery if certain security measures are lacking. The cost of security failure and recovery must take into account not only the value of the assets being protected and the damages resulting from a security violation, but also the risk, which is the probability that a particular threat will exploit a particular vulnerability with a particular harmful result.

1.4.2 Security Implementation

Security implementation involves four complementary courses of action:

- Prevention: An ideal security scheme is one in which no attack is successful. Although this is not practical in all cases, there is a wide range of threats in which prevention is a reasonable goal. For example, consider the transmission of encrypted data. If a secure encryption algorithm is used, and if measures are in place to prevent unauthorized access to encryption keys, then attacks on confidentiality of the transmitted data will be prevented.
- Detection: In a number of cases, absolute protection is not feasible, but it is practical to detect security attacks. For example, there are intrusion detection systems designed to detect the presence of unauthorized individuals logged onto a system. Another example is detection of a denial of service attack, in which communications

or processing resources are consumed so that they are unavailable to legitimate users.

- Response: If security mechanisms detect an ongoing attack, such as a denial of service attack, the system may be able to respond in such a way as to halt the attack and prevent further damage.
- Recovery: An example of recovery is the use of backup systems, so that if data integrity is compromised, a prior, correct copy of the data can be reloaded.

1.4.3 Assurance and Evaluation

Those who are "consumers" of computer security services and mechanisms (e.g., system managers, vendors, customers, and end users) desire a belief that the security measures in place work as intended. That is, security consumers want to feel that the security infrastructure of their systems meet security requirements and enforce security policies. These considerations bring us to the concepts of assurance and evaluation.

Assurance as the degree of confidence one has that the security measures, both technical and operational, work as intended to protect the system and the information it processes. This encompasses both system design and system implementation. Thus, assurance deals with the questions, "Does the security system design meet its requirements?" and "Does the security system implementation meet its specifications?" Note that assurance is expressed as a degree of confidence, not in terms of a formal proof that a design or implementation is correct. The state of the art in proving designs and implementations is such that it is not possible to provide absolute proof. Much work has been done in developing formal models that define requirements and characterize designs and implementations, together with logical and mathematical techniques for addressing these issues. But assurance is still a matter of degree.

Evaluation is the process of examining a computer product or system with respect to certain criteria. Evaluation involves testing and may also involve formal analytic or mathematical techniques. The central thrust of work in this area is the development of evaluation criteria that can be applied to any security system (encompassing security services and mechanisms) and that are broadly supported for making product comparisons.

REFERENCES

1. A.V.R. Mayuri (2012), “Phishing Detection based on Visual-Similarity” Conference Proceedings from “International Conference on Network and Cyber Security - 2012” SRK Institute of Technology, Vijayawada, A.P.
2. Amit Sharma (2010), “Cyber Wars and National Security - A paradigm shift from Means to Ends” Proceedings from Conference on Cyber Security, “Emerging Cyber Threats & Challenges, (2010)” CII, Confederation of Indian Industry, Chennai.
3. Anderson, D., Reimers, K. and Barretto, C. (March 2014). Post-Secondary Education Network Security: Results of Addressing the End-User Challenge.publication date Mar 11, 2014 publication description INTED2014 (International Technology, Education, and Development Conference)
4. B., McDermott, E., & Geer, D. (2001). Information security is information risk management. In Proceedings of the 2001 Workshop on New Security Paradigms NSPW ‘01, (pp. 97 – 104). ACM.
5. B.G.Gupta (2010), “Security Convergence – Physical & Information” Proceedings from Conference on Cyber Security, “Emerging Cyber Threats & Challenges, (2010)” CII, Confederation of Indian Industry, Chennai.
6. Boritz, J. Efrim (2005). “IS Practitioners’ Views on Core Concepts of Information Integrity”. International Journal of Accounting Information Systems. Elsevier. 6 (4): 260–279.
7. Caldwell, Tracey (12 February 2013). “Risky business: why security awareness is crucial for employees”. The Guardian. Retrieved 8 October 2018.
8. Cmde Ashok Sawhney, (2010), “Cyber war in 21st Century - Emerging Security Challenge”, Proceedings from Conference on Cyber Security, “Emerging Cyber Threats & Challenges, (2010)” CII, Confederation of Indian Industry, Chennai.

9. Computer Security and Mobile Security Challenges. researchgate.net. 3 December 2015. Archived from the original on 12 October 2016. Retrieved 4 August 2016.
10. Hemavathy (2010), "Emerging Cyber Threats and Counter Measures for Protecting Defense Network" Proceedings from Conference on Cyber Security, "Emerging Cyber Threats & Challenges, (2010)" CII, Confederation of Indian Industry, Chennai.
11. James Greene (2012). "Intel Trusted Execution Technology: White Paper". Intel Corporation. Archived from the original on 11 June 2014. Retrieved 18 December 2013.
12. Krutz, Ronald L.; Russell Dean Vines (2003). The CISSP Prep Guide (Gold ed.). Indianapolis, IN: Wiley.
13. Layton, Timothy P. (2007). Information Security: Design, Implementation, Measurement, and Compliance. Boca Raton, FL: Auerbach publications.
14. Manasi Desai, Dharam Padia (2011) "Security Problems in Cloud Computing" Proceedings of ''ICT4U" – 46th National convention, Computer Society of India, Ahmedabad.
15. Peltier, Thomas R. (2002). Information Security Policies, Procedures, and Standards: guidelines for effective information security management. Boca Raton, FL: Auerbach publications.
16. Pipkin, D. (2000). Information security: Protecting the global enterprise. New York: Hewlett-Packard Company.
17. Schlienger, Thomas; Teufel, Stephanie (December 2003). "Information security culture - from analysis to change". South African Computer Society (SAICSIT). 2003 (31): 46–52.
18. Stevens, Tim (11 June 2018). "Global Cybersecurity: New Directions in Theory and Methods" (PDF). Politics and Governance. 6 (2): 1–4.
19. Studies prove once again that users are the weakest link in the security chain. CSO Online. 22 January 2014. Retrieved 8 October 2018.

CHAPTER 2

CRYPTOGRAPHY

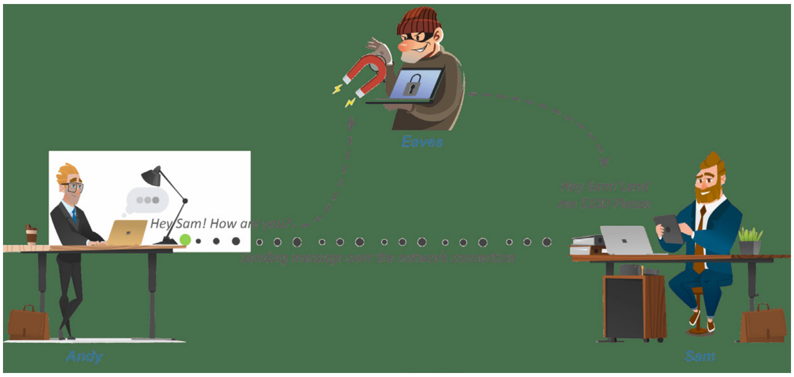
INTRODUCTION

Cryptography is a method of protecting information and communications through the use of codes, so that only those for whom the information is intended can read and process it. The prefix “crypt-” means “hidden” or “vault” -- and the suffix “-graphy” stands for “writing.”

In computer science, cryptography refers to secure information and communication techniques derived from mathematical concepts and a set of rule-based calculations called algorithms, to transform messages in ways that are hard to decipher. These deterministic algorithms are used for cryptographic key generation, digital signing, verification to protect data privacy, web browsing on the internet, and confidential communications such as credit card transactions and email.

2.1 MEANING OF CRYPTOGRAPHY

Cryptography is the study of secure communications techniques that allow only the sender and intended recipient of a message to view its contents. The term is derived from the Greek word kryptos, which means hidden. It is closely associated to encryption, which is the act of scrambling ordinary text into what's known as ciphertext and then back again upon arrival. In addition, cryptography also covers the obfuscation of information in images using techniques such as microdots or merging.


Eaves
Hey Sam! How are you?
Hey Sam! Lend me $100 Please
Sending message over the network connection
Andy
Sam

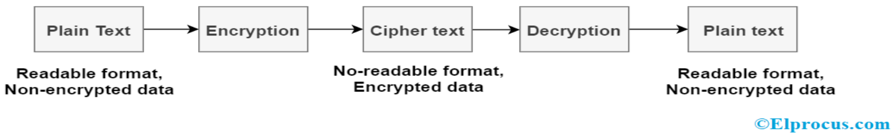
Cryptography is the method of transmitting secured data and communications via few codes so that only the destined person knows about the actual information that is transmitted. This form of process intercepts unauthorized accessibility for the data. So, in clear the name itself indicates that "crypt" refers to "hidden" to "writing". Encoding of information in cryptography follows mathematical hypotheses and few calculations described as algorithms. The encoded data is transmitted so that it makes it difficult to find the original data. These sets of rules are utilized in the procedures of digital signing, authentication to secure data, cryptographic key development and to safeguard all your financial transactions. Mostly, cryptography is followed by the organizations to go with the objectives of:

Privacy – The transmitted data should not be known by external parties except for the intended individual.

Reliability – the data cannot be modified in storage or transfer between the sender and the destined receiver having no kind of modification.

Non-repudiation – Once the data is transmitted, the sender has no chance to deny it in the later phases.

Authentication – Both the sender and receiver need to circumstantiate their own identities about the transmitted and received data.


Plain Text
Encryption
Cipher text
Decryption
Plain text
Readable format, Non-encrypted data
No-readable format, Encrypted data
Readable format, Non-encrypted data
©Elprocus.com

2.1.1 History of Cryptography

This is all very abstract, and a good way to understand the specifics of what we're talking about is to look at one of the earliest known forms of cryptography. It's known as the *Caesar cipher,* because Julius Caesar used it for his confidential correspondence; as his biographer Suetonius described it, "if he had anything confidential to say, he wrote it in cipher, that is, by so changing the order of the letters of the alphabet ... If anyone wishes to decipher these, and get at their meaning, he must substitute the fourth letter of the alphabet, namely D, for A, and so with the others."

Suetonius's description can be broken down into the two cryptographic elements we've discussed, the algorithm and the key. The algorithm here is simple: each letter is replaced by another letter from later in the alphabet. The key is how many letters later in the alphabet you need to go to create your ciphertext. It's three in the version of the cipher Suetonius describes, but obviously other variations are possible — with a key of four, A would become E, for instance.

A few things should be clear from this example. Encryption like this offers a fairly simple way to secretly send any message you like. Contrast that with a system of code phrases where, say, "Let's order pizza" means "I'm going to invade Gaul."

To translate that sort of code, people at both ends of the communication chain would need a book of code phrases, and you'd have no way to encode new phrases you hadn't thought of in advance. With the Caesar cipher, you can encrypt any message you can think of. The tricky part is that everyone communicating needs to know the algorithm and the key in advance, though it's much easier to safely pass on and keep that information than it would be with a complex code book.

The Caesar cipher is what's known as a substitution cipher, because each letter is substituted with another one; other variations on this, then, would substitute letter blocks or whole words. For most of history, cryptography consisted of various substitution ciphers deployed to keep government and military communications secure.

Medieval Arab mathematicians pushed the science forward, particularly the art of decryption — once researchers realized that certain letters in a given language are more common than others, it becomes easier to recognize patterns, for instance.

But most pre-modern encryption is incredibly simple by modern standards, for the obvious reason that, before the advent of computers, it was difficult to perform mathematical transformations quickly enough to make encryption or decryption worthwhile.

In fact, the development of computers and advances in cryptography went hand in hand. Charles Babbage, whose idea for the Difference Engine presaged modern computers, was also interested in cryptography. During World War II, the Germans used the electromechanical Enigma machine to encrypt messages — and, famously, Alan Turing led a team in Britain that developed a similar machine to break the code, in the process laying some of the groundwork for the first modern computers. Cryptography got radically more complex as computers became available, but remained the province of spies and generals for several more decades. However, that began to change in the 1960s.

2.1.2 Types of Cryptography

Cryptography is further classified into three different categories:

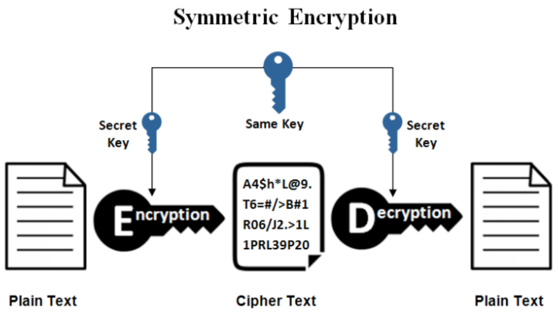
- Symmetric Key Cryptography (Private/Secret Key Cryptography)
- Asymmetric Key Cryptography (Public Key Cryptography)
- Hash Function

Symmetric Key Cryptography

Symmetric key cryptography is a type of cryptography in which the single common key is used by both sender and receiver for the purpose of encryption and decryption of a message. This system is also called private or secret key cryptography and AES (Advanced Encryption System) is the most widely uses symmetric key cryptography.

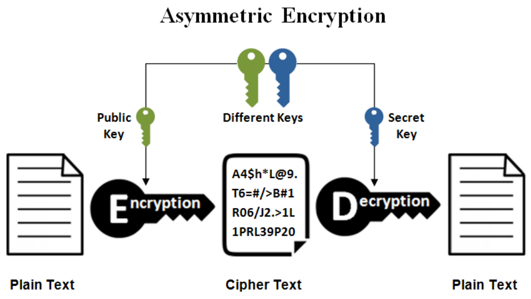
The symmetric key system has one major drawback that the two parties must somehow exchange the key in a secure way as there is only one single key for encryption as well as decryption process.

Types: AES (Advanced Encryption Standard), DES, Triple DES, RC2, RC4, RC5, IDEA, Blowfish, Stream cipher, Block cipher, etc. are the types of symmetric key cryptography.



Asymmetric Key Cryptography

Asymmetric Key Cryptography is completely different and a more secure approach than symmetric key cryptography. In this system, every user uses two keys or a pair of keys (private key and public key) for encryption and decryption process. Private key is kept as a secret with every user and public key is distributed over the network so if anyone wants to send message to any user can use those public keys.



Either of the key can be used to encrypt the message and the one left is used for decryption purpose. Asymmetric key cryptography is also known as public key cryptography and is more secure than symmetric key. RSA is the most popular and widely used asymmetric algorithm.

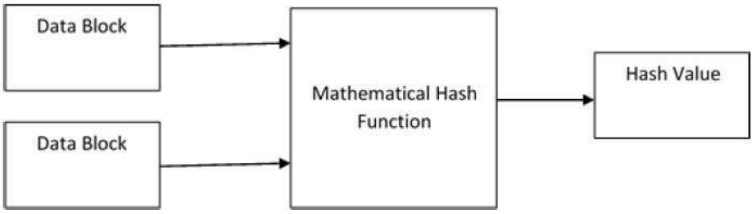
Types: RSA, DSA, PKCs, Elliptic Curve techniques, etc. are the common types of asymmetric key cryptography.

Hash Function

A Hash function is a cryptography algorithm that takes input of arbitrary length and gives the output in fixed length. The hash function is also considered as a mathematical equation that takes seed (numeric input) and produce the output that is called hash or message digest. This system operates in one-way manner and does not require any key. Also, it is considered as the building blocks of modern cryptography.

The hash function works in a way that it operates on two blocks of fixed length binary data and then generate a hash code. There are different rounds of hashing functions and each round takes an input of combination of most recent block and the output of the last round.

Types: Some popular hash functions are Message Digest 5 (MD5), SHA (Secure Hash Algorithm), RIPEMD, and Whirlpool. MD5 is the most commonly used hash function to encrypt and protect your passwords and private data.



2.2 PUBLIC-KEY CRYPTOGRAPHY

Public-key cryptography is a radical departure from all that has gone before. Right up to modern times all cryptographic systems have been based on the elementary tools of substitution and permutation. However, public-key algorithms are based on mathematical functions and are asymmetric in nature, involving the use of two keys, as opposed to conventional single key encryption. Several misconceptions are held about p-k:

1. That p-k encryption is more secure from cryptanalysis than conventional encryption. In fact the security of any system depends on key length and the computational work involved in breaking the cipher.
2. That p-k encryption has superseded single key encryption. This is unlikely due to the increased processing power required.
3. That key management is trivial with public key cryptography, this is not correct.

2.2.1 Principles of Public-Key Cryptosystems

The concept of P-K evolved from an attempt to solve two problems, key distribution and the development of digital signatures. In 1976 Whitfield Diffie and Martin Hellman achieved great success in developing the conceptual framework. For conventional encryption the same key is used for encryption and decryption. This is not a necessary condition. Instead it is possible to develop a cryptographic system that relies on one key for encryption and a different but related key for decryption. Furthermore these algorithms have the following important characteristic:

- It is computationally infeasible to determine the decryption key given only knowledge of the algorithm and the encryption key.

In addition, some algorithms such as RSA, also exhibits the following characteristics:

- Either of the two related keys can be used for encryption, with the other used for decryption.

1. Each system generates a pair of keys.
2. Each system publishes its encryption key (public key) keeping its companion key private.
3. If A wishes to send a message to B it encrypts the message using B's public key.
4. When B receives the message, it decrypts the message using its private key. No one else can decrypt the message because only B knows its private key.

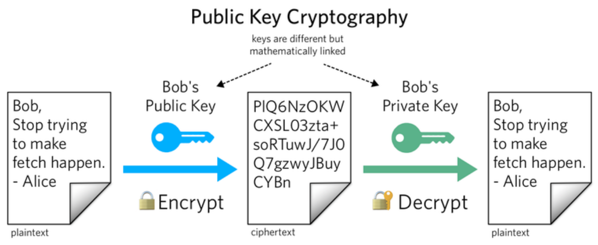

Public Key Cryptography
keys are different but mathematically linked
Bob's Public Key
Bob's Private Key
Bob, Stop trying to make fetch happen. - Alice
plaintext
Encrypt
PlQ6NzOKW CXSL03zta+ soRTuwJ/7J0 Q7gzwyJBuy CYBn
ciphertext
Decrypt
Bob, Stop trying to make fetch happen. - Alice
plaintext

Figure 1: Public Key Cryptography.

Considering P-K in more detail we have a source A that produces plaintext X destined for B (figure 2). B generates a pair of keys KU_b (a public key) and KR_b (a private key). With X and KU_b as inputs, A forms the cipher text Y:

$$Y = E_{KU_b}(X)$$

The intended receiver B is able to invert the transformation with his private key:

$X = D_{KR_b}(Y).$

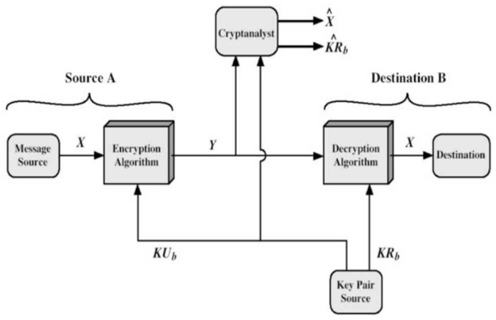


Figure 2: Public Key Cryptography: Secrecy.

Authentication

As mentioned, either key may be used for encryption with the other used for subsequent decryption. This facilitates a different form of scheme as shown in figure 3. In this case A prepares a message to B using his private key to encrypt and B can decrypt it using A's public key.

$$Y = E_{KR_a}(X)$$
$$X = D_{KU_a}(Y).$$

As the message was prepared using A's private key it could only have come from A therefore the entire message serves as a digital signature. It should be noted that this scheme does not provide confidentiality because everyone has access to A's public key. Also the scheme is not efficient because B must maintain/store both the cipher text (as proof of authenticity) and the decoded plaintext (for practical use of the document). A more efficient way of achieving the same result is to encrypt a small block of bits that are a function of the document. This block, called an authenticator, must have the property that it is infeasible to change the document without changing the authenticator. If the authenticator is encrypted using the senders

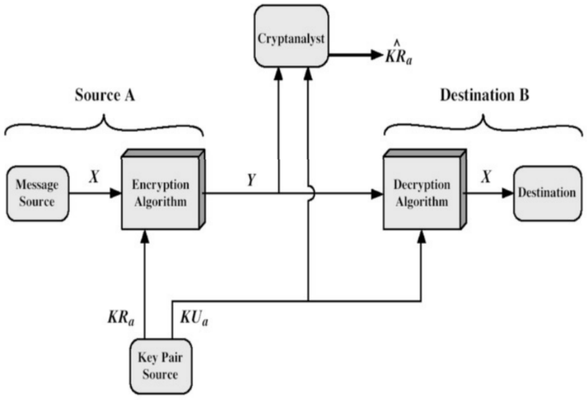


Figure 3: Public key cryptography: authentication.

Confidentiality and Authentication If both are required; the double use of the public key scheme (figure 4) facilitates this. Here

$$Z = E_{KU_b}[E_{KR_a}(X)]$$
$$X = D_{KU_a}[D_{KR_b}(Z)]$$

In this case the message is first encrypted using the sender's private key, providing the digital signature. Then a second encryption is performed using the receiver's public key, which delivers confidentiality. The disadvantage with this scheme is that the public key algorithm which is complex must be used four times.

2.2.2 Applications for P-K Cryptosystems

In broad terms, we can classify the use of public-key cryptosystems into three categories:

- Encryption/decryption: where the sender encrypts the message with the receivers public key.
- Digital signature: where the sender "signs" a message with his private key.
- Key exchange: several approaches later.

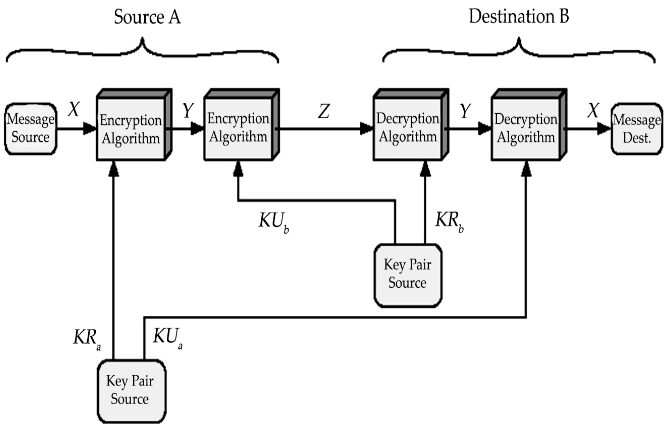


Figure 4: Public key cryptography: secrecy and authentication.

2.2.3 Requirements of the Algorithm

The requirements of any P-K system:

1. It is computationally easy for party B to generate a key pair (public (KU) and private (KR)).

2. It is computationally easy for sender A knowing KU_b and the message to be encrypted to generate the corresponding cipher text $C = E_{KU_b}(M)$.
3. It is computationally easy for the receiver B to decrypt the resulting ciphertext using his private key (KR_b) to recover the original message. $M = D_{KR_b}(C) = D_{KR_b}[E_{KU_b}(M)]$.
4. It is computationally infeasible for an opponent, knowing the public key KU_b, to determine the private key KR_b.
5. It is computationally infeasible for an opponent, knowing KU_b and C to recover the plaintext message M.
6. A sixth requirement that, although useful, is not necessary for all public-key applications - the encryption and decryption can be applied in either order: $M = E_{KU_b}[D_{KR_b}(M)] = D_{KU_b}[E_{KR_b}(M)]$

These are formidable requirements as is evidenced by the fact that only one algorithm (RSA) has received widespread acceptance in over 20 years. The requirements boil down to the need for a trapdoor one-way function. A one-way function is a function that maps a domain into a range such that every function value has a unique inverse, with the condition that the calculation of the function is easy whereas the calculation of the inverse is infeasible:

$$\begin{array}{lll} Y & = f(X) & \text{easy} \\ X & = f^{-1}(Y) & \text{infeasible} \end{array}$$

"Easy" is defined to mean a problem that can be solved in polynomial time as a function of input length (n). For example, the time to compute is proportional to n^a where a is *a* fixed constant. "Infeasible" is not as well defined however. Generally we can say that if the effort to solve is greater than polynomial time the problem is infeasible, e.g. if time to compute is proportional to 2^n. Trapdoor one-way functions are a family of invertible functions f_k such that $Y = f_k(X)$ is easy if k and X known, $X = f_k(Y)$ is easy if k and Y are known, and $X = f_k^{-1}(Y)$ is infeasible if Y is known but k is not known. The development of a practical public-key scheme depends on the discovery of a suitable trapdoor one-way function.

The Knapsack Algorithm

Many algorithms have been proposed for P-K, and have subsequently been broken. The most famous of these was proposed by Ralph Merkle as follows. The problem deals with determining which of a set of objects are in a container, say a knapsack. Of the list of say six objects of different weights shown below, which subset is in the knapsack if it weighs S?

Object 1	455 g
Object 2	341 g
Object 3	284 g
Object 4	132 g
Object 5	82 g
Object 6	56 g

Given that the weight of the knapsack is S = 821 grams, the problem is to determine which of the items are in the knapsack. The problem shown here is simple but when the number of items is increased (> 100) it becomes computationally infeasible. So what we have is six different objects with six different weights. The knapsack weighs nothing itself but with a selected number of objects in it weighs (say) 821 grams. Which objects does it contain? Merkle's contribution was to show how to turn the knapsack problem into a scheme for encryption and decryption. In other words how to incorporate "trapdoor" information which enabled the easy solution of the knapsack problem. Suppose we wish to send messages in blocks of n bits. We define the following:

- Cargo vector: $a = (a_1, a_2, \ldots, a_n)$, where a_i is an integer.
- Plaintext message block $x = (x_1, x_2, \ldots, x_n)$, where x_i is a binary digit.
- Corresponding cipher text S:

$$S = \mathbf{a} \cdot \mathbf{x} = \sum_{i=1}^{n} (a_i x_i).$$

The vector a is considered to be a list of potential elements to be put into the knapsack with each vector element equal to each weight of the element. The message block x is considered to be a selection of elements of the cargo vector in the knapsack. Each element is set equal to 1 if the corresponding element is in the knapsack and 0 if it is not. The product S is simply the sum of the selected item's weights (i.e. the weight of the contents of the knapsack).

As an example lets take a cargo vector as follows:

a = (455, 341, 284, 132, 82, 56)

x = (x_1, x_2, x_3, x_4, x_5, x_6) a six bit binary number

S = 821

For encryption a is used as the public key. The person sending the message x performs S = a · x and sends S as the cipher text. The receiving party must recover x from S and a. Two requirements are as follows:

1. That there be a unique inverse for each value of S. For example if S = 3 and a = (1, 3, 2, 5) then the problem would have two solutions, x = (1, 0, 1, 0) and x = (0, 1, 0, 0). The value of a must be chosen so that each combination of elements yields a unique value of S.
2. That decryption is hard in general but easy if special knowledge is available.

For large values of n the knapsack problem is hard in general. If however we impose the condition that each element of a is larger than the sum of the preceding elements we have:

$$a_i > \sum_{j=1}^{i-1} a_j \qquad 1 < i < n$$

This is known as a super increasing vector and in this case the solution is easy. For example, consider the vector a′ = (171, 197, 459, 1191, 2410) which satisfies the condition. Suppose we have S′ = a′ · x′ = 3798. Because 3798 > 2410, a_5 must be included (x_5 = *1*) because without a5 all the other elements cannot contribute enough to add up to 3798 (or 2410). Now consider 3798 − 2410 = 1388. The number 1388 is bigger than 1191 so a4 must be included (x_4 = *1*). Continuing1 in this fashion we find that x_3 = *0*, x_2 = *1* and x_1 = *0*. What Merkle did was to tie an easy super increasing knapsack problem to a hard general knapsack problem. Suppose we choose an easy knapsack vector a$'$ with n elements. Also select two integers' *m* and ω such that m is greater than the sum of the elements, and ω is relatively prime to *m*, that is:

$$m > \sum_{i=1}^{n} a_i \qquad gcd(w, m) = 1$$

Now, we construct a hard knapsack vector, a, by multiplying an easy vector a' by ω (mod *m*):

$$\mathbf{a} = \omega \mathbf{a}' \pmod{m}$$

The vector a will in general not be super increasing and therefore can be used to construct hard knapsack problems. However, knowledge of ω and m enables the conversion of this hard knapsack problem to an easy one. To see this, first observe that since ω and m are relatively prime, there exists a unique multiplicative inverse ω^{-1}, modulo m. Therefore:

$$\omega^{-1}\mathbf{a} = \mathbf{a}' \pmod{m}.$$

We can now state the knapsack scheme. The ingredients are as follows:

1. a$'$, a super increasing vector (private, chosen).
2. m, an integer larger than the sum of all a_j's (private, chosen).
3. ω, an integer relatively prime to *m* (private, chosen).
4. ω^{-1}, the inverse of ω, modulo m (private, calculated).
5. a, equal to ωa$'$ (mod m) (public, calculated).

The private key consists of the triple (ω^{-1}, m, a$'$) and the public key consists of the value of a.

Suppose user A has published his public key a and that user B wishes to send a message x to A. B calculates the sum S = a · x. The determination of x given S and a is difficult so this is a secure transmission. However, on receipt, user A is able to decrypt easily. Defining $S' = \omega^{-1}S$ (mod m) we have the following:

$$\begin{aligned} S &= \mathbf{a} \cdot \mathbf{x} = \omega \mathbf{a}' \cdot \mathbf{x} \\ S' &= \omega^{-1} S \pmod{m} \\ &= \omega^{-1} \omega \mathbf{a}' \cdot \mathbf{x} \pmod{m} \\ &= \mathbf{a}' \cdot \mathbf{x} \end{aligned}$$

Thus we have converted the hard problem of finding x given S into the easy problem of finding x given S' and a'.

For example, given the plaintext message x = (0, 1, 0, 0, 1, 0, 1, 1), user B computes a · x = 818. User A first computes S 0 = ω^{-1}S (mod m) = 415, and then solves the easy knapsack problem to recover x = (0, 1, 0, 0, 1, 0, 1, 1).

2.2.4 Cryptography Benefits and Drawbacks

Nowadays, the networks have gone global and information has taken the digital form of bits and bytes. Critical information now gets stored, processed and transmitted in digital form on computer systems and open communication channels.

Since information plays such a vital role, adversaries are targeting the computer systems and open communication channels to either steal the sensitive information or to disrupt the critical information system.

Modern cryptography provides a robust set of techniques to ensure that the malevolent intentions of the adversary are thwarted while ensuring the legitimate users get access to information.

2.3 TRANSPOSITION CIPHERS AND SUBSTITUTION CIPHERS

From the encryption algorithm point of view, there are two main techniques we used to implement in the secret key cryptography (symmetric cipher) system: Substitution cipher and Transposition cipher. Substitution ciphers replace bits, characters, or blocks of characters with substitution. Transposition ciphers rearrange bits or characters in the data. We now describe some details about the two kinds of cipher and simply introduce some examples that we use very often in the two kinds of cipher.

2.3.1 Substitution Techniques

Substitution technique is one that the letters in the plaintext will be replaced by other letters or by numbers or symbols. [Caesar Cipher] The earliest use of substitution cipher is also the simplest one that is proposed by Julius Caesar, called Caesar Cipher. The Caesar Cipher works with replacing each letter with the letter standing three places further down of the alphabet order. For example:

plaintext:	a	b	c	d	e	f	g	h	w	x	y	z
ciphertext:	e	f	g	h	i	j	k	l	z	a	b	c

So if the plaintext is "meet me after the party". The ciphertext would be "phhw ph diwhu wkh sduwb".

plaintext:	meet	me	after	the	party
ciphertext:	phhw	ph	diwhu	wkh	sduwb

If we assign each letter a number from 0 to 25(from A to Z). Take the Ciphertext as C, Encryption as E, and plaintext as P. Then we can describe the Caesar Cipher as below

$$C=E(p)=(p+3)\bmod(26) \qquad (1)$$

A shift could be any amount, so the general Caesar algorithm is

$$C=E(p)=(p+k)\bmod(26) \qquad (2)$$

where k takes on a value in the range from 1 to 25. And the decryption algorithm is simply

$$p=D(C)=(C-k)\bmod(26) \qquad (3$$

If it is known that a given ciphertext is a Caesar cipher, then a brute-force cryptanalysis will be easily performed. Just try all the 25 for the possible value of k. In this example, there are three reasons for us to use the brute-force cryptanalysis. First is that the encryption and the decryption algorithms are known. Second is that there are only 25 keys to try. Third is that the language of the plaintext is known and easily recognizable.

For general cases, we always assume that the first condition is held, that is the algorithms of encryption and decryption are always known by the enemy who want to break the cipher. What really makes the brute-force attack impractical is that most of the algorithms use a large number of keys, that is, the second condition. For example, the triple DES algorithm uses a 168-bit key which makes people who choose to use the brute-force attacking way wasting resources or time. And the third condition is also important. If the language of the plaintext is unknown, we do not have any idea to recognize that if the key we try is right even in the trial that is right.

[Polyalphabetic cipher]

Simple substitution ciphers like Caesar cipher use a single mapping from plaintext to ciphertext letters, that is the same plaintext will have the same ciphertext. This characteristic is always not good in cryptography from the security point of view. Polyalphabetic cipher solves this problem by using multiple substitutions. Image a cipher disk with two circles (outer and inner circle) and they are movable between each other.Every time we randomly turn around the inner circle, we will get a response pair from each alphabet. Then we record where the &(or any sign different from alphabets and numbers) sign stand. That is the simple way to produce a substitution cipher which works and avoid the single mapping from plaintext to ciphertext problem.

2.3.2 Transposition Techniques

Transposition technique is achieved by performing some kind of permutation on the plaintext letters. It is very simple to realize this kind of cipher. We can do it by the example. If the plaintext is “meet me after the party”, we can rearrange it by this way:

```
m   e   m   a   t   r   h   p   r   y
  e   t   e   f   e   t   e   a   t
```

So we get the plaintext and the ciphertext like this:

plaintext: meet me after the party
ciphertext: mematrhpryetefeteat

[Columnar transposition]

Another simple transposition cipher is called Columnar transposition. If the plaintext is "data encryption", we will compose the sentence into a 3*5 matrix. For example:

key:	4	1	2	3	5
plaintext :	d	a	t	a	
	e	n	c	r	y
	p	t	i	o	n

ciphertext: anttciarodep yn

Of course, the transposition cipher can be made more secure by performing more than one stage of transposition. For example, doing the Columnar transposition 2 or 3 times and it will efficiently to increase the security of this cipher.

2.4 BLOCK CIPHER AND STREAM CIPHER

We can even more separate symmetric cipher to two kinds of cipher as block cipher and stream cipher by the encryption basic sense. In this report we pay more attention to the block cipher, but we also give some stream cipher examples.

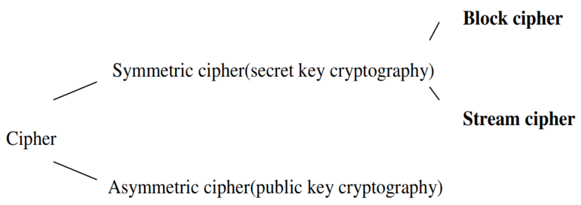

Block cipher
Symmetric cipher(secret key cryptography)
Stream cipher
Cipher
Asymmetric cipher(public key cryptography)

Figure 5. Block cipher and Stream cipher.

The most different part between the block cipher and the stream cipher is that the block cipher encrypts the fixed size of the input data. On the other hand, stream cipher

Block Cipher

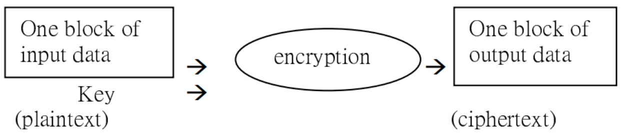


Figure 6. Block cipher scheme.

Let M be a plaintext message. A block cipher breaks M into successive blocks M_1 , M_2 , …… and encrypt each Mk with the same key K; that is,

$$E_k(M) = E(M_1)E(M_2)...... \qquad (4)$$

Typical size of block cipher block size is 64bits, 128bits or larger. Older cipher usually had the smaller size. Considering of the security, the larger the block size has, the safer the data is. Because each bits in the original data influences the every single output bit. And with aspect of processing speed, it is the same that we hope that the block size much larger. One of the advantages of the block cipher is the fast speed. The drawback of the block cipher is that we must fit the block size, or we cannot do block cipher encryption. Sometimes we have to add additional redundant to fit the block size to do encryption. And this is kind of wasting resource.

Stream Cipher

Stream cipher is different from block cipher that stream cipher break message M into successive characters or bitsm_1 , m_2 , …… and encrypt each m_k with the ith element k_i of a key stream K= k_1 k_2 ……; that is,

$$E_k(M) = E_{k_1}(m_1)E_{k_2}(m_2)...... \quad (5)$$

The stream cipher produces key stream by using a key instead of dealing with block data. The key stream is often used to do XOR with plaintext and the results could be used to do encryption. We describe the XOR algorithm as followed.

Table 2. XOR-operation

Input 1	Input 2	XOR output
0	0	0
0	1	1
1	0	1
1	1	0

2.5 RSA CRYPTOSYSTEM

This cryptosystem is one the initial system. It remains most employed cryptosystem even today. The system was invented by three scholars Ron Rivest, Adi Shamir, and Len Adleman and hence, it is termed as RSA cryptosystem.

We will see two aspects of the RSA cryptosystem, firstly generation of key pair and secondly encryption-decryption algorithms.

Generation of RSA Key Pair

Each person or a party who desires to participate in communication using encryption needs to generate a pair of keys, namely public key and private key. The process followed in the generation of keys is described below:

Generate the RSA Modulus (n)

- Select two large primes, p and q.
- Calculate n=p*q. For strong unbreakable encryption, let n be a large number, typically a minimum of 512 bits.

Find Derived Number (e)

- Number e must be greater than 1 and less than (p − 1)(q − 1).
- There must be no common factor for e and (p − 1)(q − 1) except for 1. In other words two numbers e and (p − 1)(q − 1) are coprime.

Form the Public Key

- The pair of numbers (n, e) form the RSA public key and is made public.
- Interestingly, though n is part of the public key, difficulty in factorizing a large prime number ensures that attacker cannot find in finite time the two primes (p & q) used to obtain n. This is strength of RSA.

Generate the Private Key

- Private Key d is calculated from p, q, and e. For given n and e, there is unique number d.
- Number d is the inverse of e modulo (p - 1)(q – 1). This means that d is the number less than (p - 1)(q - 1) such that when multiplied by e, it is equal to 1 modulo (p - 1)(q - 1).
- This relationship is written mathematically as follows:

$ed = 1 \bmod (p - 1)(q - 1)$

The Extended Euclidean Algorithm takes p, q, and e as input and gives d as output.

Example

An example of generating RSA Key pair is given below. (For ease of understanding, the primes p & q taken here are small values. Practically, these values are very high).

- Let two primes be p = 7 and q = 13. Thus, modulus n = pq = 7 x 13 = 91.
- Select e = 5, which is a valid choice since there is no number that is common factor of 5 and (p − 1)(q − 1) = 6 × 12 = 72, except for 1.
- The pair of numbers (n, e) = (91, 5) forms the public key and can be made available to anyone whom we wish to be able to send us encrypted messages.
- Input p = 7, q = 13, and e = 5 to the Extended Euclidean Algorithm. The output will be d = 29.
- Check that the d calculated is correct by computing

  de = 29 × 5 = 145 = 1 mod 72
- Hence, public key is (91, 5) and private keys is (91, 29).

2.5.1 RSA Analysis

The security of RSA depends on the strengths of two separate functions. The RSA cryptosystem is most popular public-key cryptosystem strength of which is based on the practical difficulty of factoring the very large numbers.

Encryption Function: It is considered as a one-way function of converting plaintext into cipher text and it can be reversed only with the knowledge of private key d.

Key Generation: The difficulty of determining a private key from an RSA public key is equivalent to factoring the modulus n. An attacker thus cannot use knowledge of an RSA public key to determine an RSA private key unless he can factor n. It is also a one way function, going from p & q values to modulus n is easy but reverse is not possible. If either of these two functions are proved non one-way, then RSA will be broken. In fact, if a technique for factoring efficiently is developed then RSA will no longer be safe. The strength of RSA encryption drastically goes down against attacks if the number p and q are not large primes and/ or chosen public key e is a small number.

2.5.2 Encryption and Decryption in RSA Public-key Cryptosystem

Encryption is the process of transforming information so it is unintelligible to anyone but the intended recipient. Decryption is the process of decoding encrypted information. A cryptographic algorithm, also called a cipher, is a mathematical function used for encryption or decryption. Usually, two related functions are used, one for encryption and the other for decryption.

With most modern cryptography, the ability to keep encrypted information secret is based not on the cryptographic algorithm, which is widely known, but on a number called a key that must be used with the algorithm to produce an encrypted result or to decrypt encrypted information. Decryption with the correct key is simple. Decryption without the correct key is very difficult, if not impossible.

Symmetric-Key Encryption

With symmetric-key encryption, the encryption key can be calculated from the decryption key and vice versa. With most symmetric algorithms, the same key is used for both encryption and decryption, as shown in Figure 7, "Symmetric-Key Encryption".

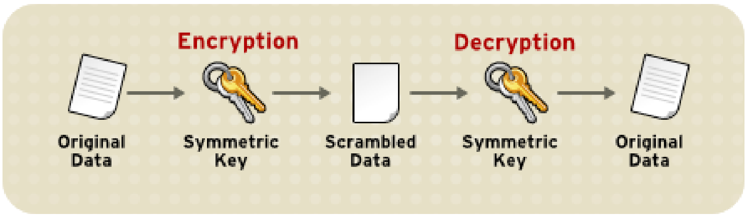


Figure 7: Symmetric-key encryption.

Implementations of symmetric-key encryption can be highly efficient, so that users do not experience any significant time delay as a result of the encryption and decryption. Symmetric-key encryption also provides a degree of authentication, since information encrypted with one symmetric key cannot be decrypted with any other symmetric key. Thus, as long as the symmetric key is kept secret by the two parties using it to encrypt communications, each party can be sure that it is communicating with the other as long as the decrypted messages continue to make sense.

Symmetric-key encryption is effective only if the symmetric key is kept secret by the two parties involved. If anyone else discovers the key, it affects both confidentiality and authentication. A person with an unauthorized symmetric key not only can decrypt messages sent with that key, but can encrypt new messages and send them as if they came from one of the legitimate parties using the key.

Symmetric-key encryption plays an important role in SSL communication, which is widely used for authentication, tamper detection, and encryption over TCP/IP networks. SSL also uses techniques of public-key encryption.

Public-Key Encryption

Public-key encryption (also called asymmetric encryption) involves a pair of keys, a public key and a private key, associated with an entity. Each public key is published, and the corresponding private key is kept secret. Data encrypted with a public key can be decrypted only with the corresponding private key. Figure 8, "Public-Key Encryption" shows a simplified view of the way public-key encryption works.

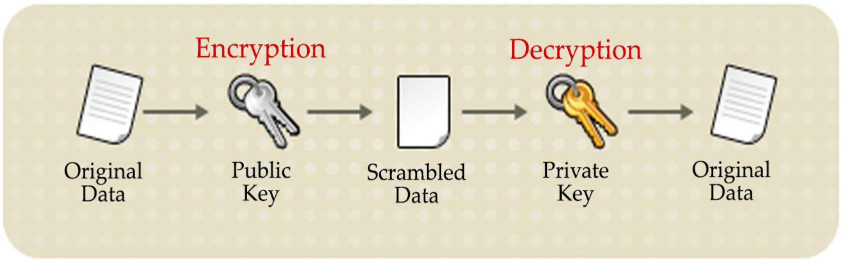


Figure 8: Public-key encryption.

The scheme shown in Figure 8, "Public-Key Encryption" allows public keys to be freely distributed, while only authorized people are able to read data encrypted using this key. In general, to send encrypted data, the data is encrypted with that person's public key, and the person receiving the encrypted data decrypts it with the corresponding private key.

Compared with symmetric-key encryption, public-key encryption requires more processing and may not be feasible for encrypting and decrypting large amounts of data. However, it is possible to use public-key encryption to send a symmetric key, which can then be used to encrypt additional data. This is the approach used by the SSL/TLS protocols.

The reverse of the scheme shown in Figure 8 "Public-Key Encryption" also works: data encrypted with a private key can be decrypted only with the corresponding public key. This is not a recommended practice to encrypt sensitive data, however, because it means that anyone with the public key, which is by definition published, could decrypt the data. Nevertheless, private-key encryption is useful because it means the private key can be used to sign data with a digital signature, an important requirement for electronic commerce and other commercial applications of cryptography.

Key Length and Encryption Strength

Breaking an encryption algorithm is basically finding the key to the access the encrypted data in plain text. For symmetric algorithms, breaking the algorithm usually means trying to determine the key used to encrypt the text. For a public key algorithm, breaking the algorithm usually means acquiring the shared secret information between two recipients.

One method of breaking a symmetric algorithm is to simply try every key within the full algorithm until the right key is found. For public key algorithms, since half of the key pair is publicly known, the other half (private key) can be derived using published, though complex, mathematical calculations. Manually finding the key to break an algorithm is called a brute force attack.

Breaking an algorithm introduces the risk of intercepting, or even impersonating and fraudulently verifying, private information.

The key strength of an algorithm is determined by finding the fastest method to break the algorithm and comparing it to a brute force attack.

For symmetric keys, encryption strength is often described in terms of the size or length of the keys used to perform the encryption: longer keys generally provide stronger encryption. Key length is measured in bits.

An encryption key is considered full strength if the best known attack to break the key is no faster than a brute force attempt to test every key possibility.

Different types of algorithms particularly public key algorithms may require different key lengths to achieve the same level of encryption strength as a symmetric-key cipher. The RSA cipher can use only a subset of all possible values for a key of a given length, due to the nature of the mathematical problem on which it is based. Other ciphers, such as those used for symmetric-key encryption, can use all possible values for a key of a given length. More possible matching options means more security.

Because it is relatively trivial to break an RSA key, an RSA public-key encryption cipher must have a very long key at least 1024 bits to be considered cryptographically strong. On the other hand, symmetric-key ciphers are reckoned to be equivalently strong using a much shorter key length, as little as 80 bits for most algorithms.

2.5.3 RSA Signature Scheme

The RSA public-key cryptosystem can be used for both encryption and signatures. Each user has three integers e, d and n, $n = pq$ with p and q large primes. For the key pair (e, d), $ed \equiv 1 \pmod{\varphi(n)}$ must be satisfied. If sender A wants to send signed message c corresponding to message m to receiver B, A signs it using A’s private key, computing c ≡ m^{dA} (mod n_A). First A computes ϕ(nA) ≡ lcm (pA − 1, qA − 1)

where lcm stands for the least common multiple. The sender A selects his own key pair (e_A, d_A) such that eA•dA ≡ 1 (mod ϕ(nA))

The modulus n_A and the public key e_A are published., Figure 9 illustrates the RSA signature scheme.

Example

Choose p = 11 and q = 17. Then n = pq = 187.

$$\text{Compute } \varphi(n) = 1\text{ cm }(p-1, q-1)$$
$$= 1\text{ cm }(10, 16) = 80$$

Select $e_A = 27$. Then $e_A d_A \equiv 1 \pmod{\varphi(n_A)}$

$27 d_A \equiv 1 \pmod{80}$

$d_A = 3$

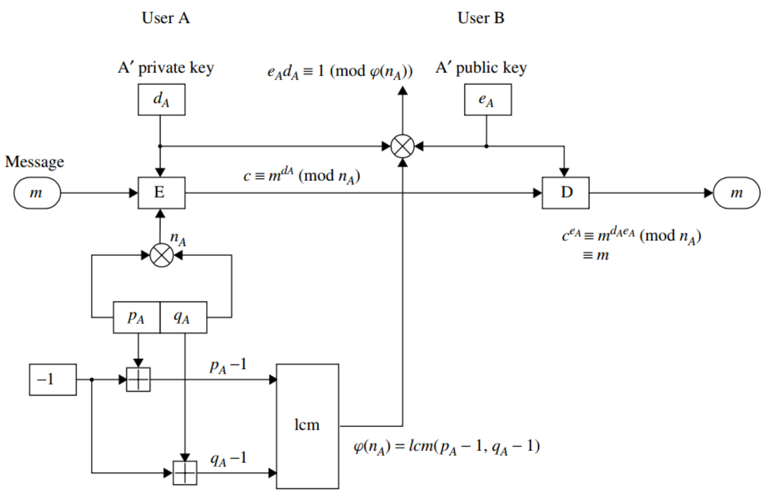

User A
User B
A′ private key
$e_A d_A \equiv 1 \pmod{\varphi(n_A)}$
A′ public key
d_A
e_A
Message
m
E
$c \equiv m^{dA} \pmod{n_A}$
D
m
n_A
$c^{e_A} \equiv m^{d_A e_A} \pmod{n_A}$
$\equiv m$
p_A
q_A
−1
$p_A - 1$
lcm
$q_A - 1$
$\varphi(n_A) = lcm(p_A - 1, q_A - 1)$

Figure 9: The RSA signature scheme

Suppose m = 55. Then the signed message is

c ≡ m^{dA} (mod 187)

$\equiv 553 \pmod{187} \equiv 132$

The message will be recreated as:

$m \equiv c^{eA} \pmod{n}$

$\equiv 132^{27} \pmod{187} \equiv 55$

Thus, the message m is accepted as authentic.

Next, consider a case where the message is much longer. The larger m requires more computation in signing and verification steps. Therefore, it is better to compute the message digest using a appropriate hash function, for example, the SHA-1 algorithm. Signing the message digest rather than the message often improves the efficiency of the process because the message digest is usually much smaller than the message. When the message is assumed to be m = 75 139, the message digest h of m is computed using the SHA-1 algorithm as follows:

$h \equiv H(m) \pmod{n}$

$\equiv H(75\,139) \pmod{187}$

$\equiv$ 86a0aab5631e729b0730757b0770947307d9f597

$\equiv$ 768587753333627872847426508024461003561962698135 (mod 187) (decimal)

The message digest h is then computed as:

$h \equiv H(75\,139) \pmod{187} \equiv 11$

Signing h with A's private key d_A produces:

$c \equiv h^{dA} \pmod{n}$

$\equiv 113 \pmod{187} \equiv 22$

Thus, the signature verification proceeds as follows:

$h \equiv c^{eA} \pmod{n}$

$\equiv 22^{27} \pmod{187} \equiv 11$

which shows that verification is accomplished.

In hardware, RSA is about 1000 times slower than DES. RSA is also implemented in smartcards, but these implementations are slower. DES is about 100 times faster than RSA. However, RSA will never reach the speed of symmetric cipher algorithms. It is known that the security of RSA depends on the problem of factoring large numbers. To find the private key from the public key e and the modulus n, one has to factor n. Currently, n must be larger than a 129 decimal digit modulus. Easy methods to break RSA have not yet been found. A brute-force attack is even less efficient than trying to factor n. RSA encryption and signature verification are faster if you use a low value for e, but can be insecure.

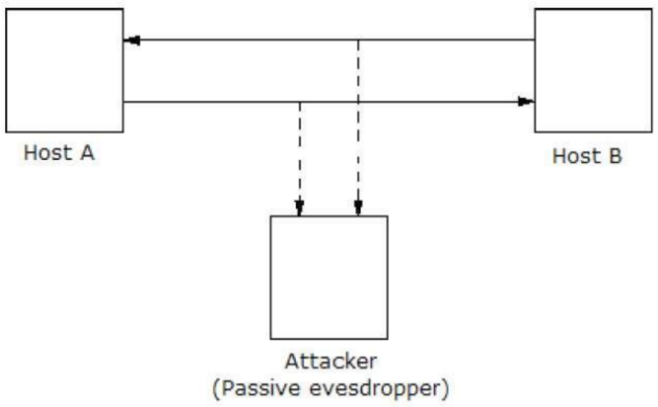
2.5.4 Attacks on Cryptosystems

In the present era, not only business but almost all the aspects of human life are driven by information. Hence, it has become imperative to protect useful information from malicious activities such as attacks. Let us consider the types of attacks to which information is typically subjected to.

Attacks are typically categorized based on the action performed by the attacker. An attack, thus, can be passive or active.

Passive Attacks

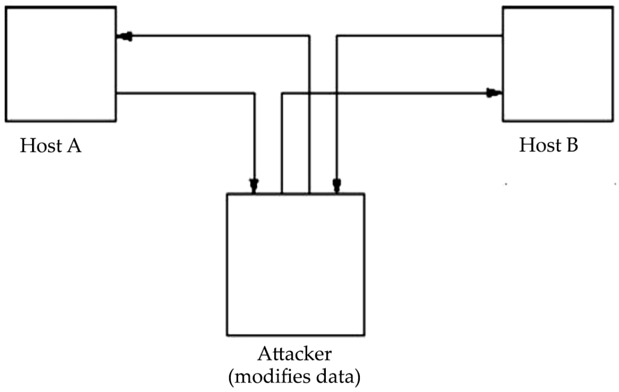
The main goal of a passive attack is to obtain unauthorized access to the information. For example, actions such as intercepting and eavesdropping on the communication channel can be regarded as passive attack. These actions are passive in nature, as they neither affect information nor disrupt the communication channel. A passive attack is often seen as stealing information. The only difference in stealing physical goods and stealing information is that theft of data still leaves the owner in possession of that data. Passive information attack is thus more dangerous than stealing of goods, as information theft may go unnoticed by the owner.



Active Attacks

An active attack involves changing the information in some way by conducting some process on the information. For example,

- Modifying the information in an unauthorized manner.
- Initiating unintended or unauthorized transmission of information.
- Alteration of authentication data such as originator name or timestamp associated with information
- Unauthorized deletion of data.
- Denial of access to information for legitimate users (denial of service).



Cryptography provides many tools and techniques for implementing cryptosystems capable of preventing most of the attacks described above.

Assumptions of Attacker

Let us see the prevailing environment around cryptosystems followed by the types of attacks employed to break these systems:

Environment around Cryptosystem

While considering possible attacks on the cryptosystem, it is necessary to know the cryptosystems environment. The attacker's assumptions and knowledge about the environment decides his capabilities.

In cryptography, the following three assumptions are made about the security environment and attacker's capabilities.

Details of the Encryption Scheme

The design of a cryptosystem is based on the following two cryptography algorithms

Public Algorithms: With this option, all the details of the algorithm are in the public domain, known to everyone.

Proprietary algorithms: The details of the algorithm are only known by the system designers and users.

In case of proprietary algorithms, security is ensured through obscurity. Private algorithms may not be the strongest algorithms as they are developed in-house and may not be extensively investigated for weakness.

Secondly, they allow communication among closed group only. Hence they are not suitable for modern communication where people communicate with large number of known or unknown entities. The algorithm is preferred to be public with strength of encryption lying in the key.

Thus, the first assumption about security environment is that the encryption algorithm is known to the attacker.

Availability of Cipher text

We know that once the plaintext is encrypted into cipher text, it is put on unsecure public channel (say email) for transmission. Thus, the attacker can obviously assume that it has access to the cipher text generated by the cryptosystem.

Availability of Plaintext and Cipher text

This assumption is not as obvious as other. However, there may be situations where an attacker can have access to plaintext and corresponding cipher text. Some such possible circumstances are:

- The attacker influences the sender to convert plaintext of his choice and obtains the cipher text.
- The receiver may divulge the plaintext to the attacker inadvertently. The attacker has access to corresponding cipher text gathered from open channel.
- In a public-key cryptosystem, the encryption key is in open domain and is known to any potential attacker. Using this key, he can generate pairs of corresponding plaintexts and cipher texts.

Cryptographic Attacks

The basic intention of an attacker is to break a cryptosystem and to find the plaintext from the cipher text. To obtain the plaintext, the attacker only needs to find out the secret decryption key, as the algorithm is already in public domain.

Hence, he applies maximum effort towards finding out the secret key used in the cryptosystem. Once the attacker is able to determine the key, the attacked system is considered as broken or compromised.

Based on the methodology used, attacks on cryptosystems are categorized as follows:

Cipher text Only Attacks (COA): In this method, the attacker has access to a set of cipher text(s). He does not have access to corresponding plaintext. COA is said to be successful when the corresponding plaintext can be determined from a given set of cipher text. Occasionally, the encryption key can be determined from this attack. Modern cryptosystems are guarded against cipher text-only attacks.

Known Plaintext Attack (KPA): In this method, the attacker knows the plaintext for some parts of the cipher text. The task is to decrypt the rest of the cipher text using this information. This may be done by determining the key or via some other method. The best example of this attack is linear cryptanalysis against block ciphers.

Chosen Plaintext Attack (CPA): In this method, the attacker has the text of his choice encrypted. So he has the cipher text-plaintext pair of his choice. This simplifies his task of determining the encryption key. An example of this attack is differential cryptanalysis applied against block ciphers as well as hash functions. A popular public key cryptosystem, RSA is also vulnerable to chosen-plaintext attacks.

Dictionary Attack: This attack has many variants, all of which involve compiling a 'dictionary'. In simplest method of this attack, attacker builds a dictionary of cipher texts and corresponding plaintexts that he has learnt over a period of time. In future, when an attacker gets the cipher text, he refers the dictionary to find the corresponding plaintext.

Brute Force Attack (BFA): In this method, the attacker tries to determine the key by attempting all possible keys. If the key is 8 bits long, then the number of possible keys is $2^8 = 256$. The attacker knows the cipher text and the algorithm, now he attempts all the 256 keys one by one for decryption. The time to complete the attack would be very high if the key is long.

Birthday Attack: This attack is a variant of brute-force technique. It is used against the cryptographic hash function. When students in a class are asked about their birthdays, the answer is one of the possible 365 dates. Let us assume the first student's birthdate is 3rd Aug. Then to find the next student whose birthdate is 3rd Aug, we need to enquire 1.25*◎$\sqrt{365} \approx 25$ students.

Similarly, if the hash function produces 64 bit hash values, the possible hash values are 1.8×10^{19}. By repeatedly evaluating the function for different inputs, the same output is expected to be obtained after about 5.1×10^{9} random inputs.

If the attacker is able to find two different inputs that give the same hash value, it is a collision and that hash function is said to be broken.

Man in Middle Attack (MIM): The targets of this attack are mostly public key cryptosystems where key exchange is involved before communication takes place.

- Host A wants to communicate to host B, hence requests public key of B.
- An attacker intercepts this request and sends his public key instead.
- Thus, whatever host A sends to host B, the attacker is able to read.
- In order to maintain communication, the attacker re-encrypts the data after reading with his public key and sends to B.
- The attacker sends his public key as A's public key so that B takes it as if it is taking it from A.

Side Channel Attack (SCA): This type of attack is not against any particular type of cryptosystem or algorithm. Instead, it is launched to exploit the weakness in physical implementation of the cryptosystem.

Timing Attacks: They exploit the fact that different computations take different times to compute on processor. By measuring such timings, it is be possible to know about a particular computation the processor is carrying out. For example, if the encryption takes a longer time, it indicates that the secret key is long.

Power Analysis Attacks: These attacks are similar to timing attacks except that the amount of power consumption is used to obtain information about the nature of the underlying computations.

Fault analysis Attacks: In these attacks, errors are induced in the cryptosystem and the attacker studies the resulting output for useful information.

Practicality of Attacks

The attacks on cryptosystems described here are highly academic, as majority of them come from the academic community. In fact, many academic attacks involve quite unrealistic assumptions about environment as well as the capabilities of the attacker.

REFERENCES

1. Alex Biryukov, Adi Shamir, and David Wagner. Real time cryptanalysis of A5/1 on a PC. In FSE: Fast Software Encryption, pages 1–18. Springer, 2000.
2. ANSI X9.42-2003. Public Key Cryptography for the Financial Services Industry: Agreement of Symmetric Keys Using Discrete Logarithm Cryptography. Technical report, American Bankers Association, 2003.
3. ANSI X9.62-2001. Elliptic Curve Key Agreement and Key Transport Protocols. Technical report, American Bankers Association, 2001.
4. Carlisle Adams and Steve Lloyd. Understanding PKI: Concepts, Standards, and Deployment Considerations. Addison-Wesley Longman Publishing, Boston, MA, USA, 2002.
5. Dan Boneh and Matthew Franklin. Identity-based encryption from the Weil pairing. SIAM J. Comput., 32(3):586–615, 2003.
6. Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen. Post-Quantum Cryptography. Springer, 2009.
7. Daniel V. Bailey and Christof Paar. Efficient arithmetic in finite field extensions with application in elliptic curve cryptography. Journal of Cryptology, 14, 2001.
8. Hirsch, Frederick J. "SSL/TLS Strong Encryption: An Introduction". Apache HTTP Server. Retrieved 17 April 2013. The first two sections contain a very good introduction to public-key cryptography.
9. Mavroeidis, Vasileios, and Kamer Vishi, "The Impact of Quantum Computing on Present Cryptography", International Journal of Advanced Computer Science and Applications, 31 March 2018
10. Paar, Christof; Pelzl, Jan; Preneel, Bart (2010). Understanding Cryptography: A Textbook for Students and Practitioners.

Springer.

11. Robinson, Sara (June 2003). “Still Guarding Secrets after Years of Attacks, RSA Earns Accolades for its Founders” (PDF). SIAM News. 36 (5).
12. Sawer, Patrick (11 March 2016). “The unsung genius who secured Britain’s computer defences and paved the way for safe online shopping”. The Telegraph.

CHAPTER 3

PROGRAM SECURITY

INTRODUCTION

A program security flaw is an undesired program behaviour caused by a program vulnerability. Early idea was to attack the finished program to reveal faults, and then to patch the corresp. errors. Experience shows that this is not effective, and just tends to introduce new faults (and errors)! More modern approach is to use careful specification and compare behaviour with the expected.

Protecting programs is at the heart of computer security because programs constitute so much of a computing system (the operating system, device drivers, the network infrastructure, database management systems and other applications, even executable commands on web pages). For now, we call all these pieces of code "programs."

So we need to ask two important questions:

- How do we keep programs free from flaws?
- How do we protect computing resources against programs that contain flaws?

3.1 SECURE PROGRAMS

Consider what we mean when we say that a program is "secure." Security implies some degree of trust that the program enforces expected confidentiality, integrity, and availability. From the point of view of a program or a programmer, how can we look at a software component or code fragment and assess its security? This question is, of course, similar to the problem of assessing software quality in general. One way to assess security or quality is to ask people to name the characteristics of software that contribute to its overall security. However, we are likely to get different answers from different people. This difference occurs because the importance of the characteristics depends on who is analyzing the software. For example, one person may decide that code is secure because it takes too long to break through its security controls. And someone else may decide code is secure if it has run for a period of time with no apparent failures. But a third person may decide that any potential fault in meeting security requirements makes code insecure.

An assessment of security can also be influenced by someone's general perspective on software quality. For example, if your manager's idea of quality is conformance to specifications, then she might consider the code secure if it meets security requirements, whether or not the requirements are complete or correct. This security view played a role when a major computer manufacturer delivered all its machines with keyed locks, since a keyed lock was written in the requirements. But the machines were not secure, because all locks were configured to use the same key! Thus, another view of security is fitness for purpose; in this view, the manufacturer clearly had room for improvement.

In general, practitioners often look at quantity and types of faults for evidence of a product's quality (or lack of it). For example, developers track the number of faults found in requirements, design, and code inspections and use them as indicators of the likely quality of the final product.

3.1.1 Fixing Faults

One approach to judging quality in security has been fixing faults. You might argue that a module in which 100 faults were discovered and fixed is better than another in which only 20 faults were discovered and fixed, suggesting that more rigorous analysis and testing had led to the finding of the larger number of faults. Au contraire, challenges your friend: a piece of software with 100 discovered faults is inherently full of problems and could clearly have hundreds more waiting to appear. Your friend's opinion is confirmed by the software testing literature; software that has many faults early on is likely to have many others still waiting to be found.

Early work in computer security was based on the paradigm of "penetrate and patch," in which analysts searched for and repaired faults. Often, a top-quality "tiger team" would be convened to test a system's security by attempting to cause it to fail. The test was considered to be a "proof" of security; if the system withstood the attacks, it was considered secure. Unfortunately, far too often the proof became a counterexample, in which not just one but several serious security problems were uncovered. The problem discovery in turn led to a rapid effort to "patch" the system to repair or restore the security. However, the patch efforts were largely useless, making the system less secure rather than more secure because they frequently introduced new faults. There are at least four reasons why.

The pressure to repair a specific problem encouraged a narrow focus on the fault itself and not on its context. In particular, the analysts paid attention to the immediate cause of the failure and not to the underlying design or requirements faults.

- The fault often had nonobvious side effects in places other than the immediate area of the fault.
- Fixing one problem often caused a failure somewhere else, or the patch addressed the problem in only one place, not in other related places.
- The fault could not be fixed properly because system functionality or performance would suffer

3.1.2 Unexpected Behavior

The inadequacies of penetrate-and-patch led researchers to seek a better way to be confident that code meets its security requirements. One way to do that is to compare the requirements with the behavior. That is, to understand program security, we can examine programs to see whether they behave as their designers intended or users expected. We call such unexpected behavior a program security flaw; it is inappropriate program behavior caused by a program vulnerability. Unfortunately, the terminology in the computer security field is not consistent with the IEEE standard; the terms "vulnerability" and "flaw" do not map directly to the characterization of faults and failures. A flaw can be either a fault or failure, and a vulnerability usually describes a class of flaws, such as a buffer overflow. In spite of the inconsistency, it is important for us to remember that we must view vulnerabilities and flaws from two perspectives, cause and effect, so that we see what fault caused the problem and what failure (if any) is visible to the user. For example, a Trojan horse may have been injected in a piece of code a flaw exploiting a vulnerability but the user may not yet have seen the Trojan horse's malicious behavior. Thus, we must address program security flaws from inside and outside, to find causes not only of existing failures but also of incipient ones. Moreover, it is not enough just to identify these problems. We must also determine how to prevent harm caused by possible flaws.

Program security flaws can derive from any kind of software fault. That is, they cover everything from a misunderstanding of program requirements to a one-character error in coding or even typing. The flaws can result from problems in a single code component or from the failure of several programs or program pieces to interact compatibly through a shared interface. The security flaws can reflect code that was intentionally designed or coded to be malicious or code that was simply developed in a sloppy or misguided way. Thus, it makes sense to divide program flaws into two separate logical categories: inadvertent human errors versus malicious, intentionally induced flaws.

These categories help us understand some ways to prevent the inadvertent and intentional insertion of flaws into future code, but we still have to address their effects, regardless of intention. That is, in the words of Sancho Panza in Man of La Mancha, "it doesn't matter whether the stone hits the pitcher or the pitcher hits the stone, it's going to be bad for the pitcher." An inadvertent error can cause just as much harm to users and their organizations as can an intentionally induced flaw. Furthermore, a system attack often exploits an unintentional security flaw to perform intentional damage. From reading the popular press, you might conclude that intentional security incidents (called cyber attacks) are the biggest security threat today. In fact, plain, unintentional human errors are more numerous and cause much more damage.

Regrettably, we do not have techniques to eliminate or address all program security flaws. Security is fundamentally hard, security often conflicts with usefulness and performance, there is no ""silver bullet" to achieve security effortlessly, and false security solutions impede real progress toward more secure programming. There are two reasons for this distressing situation.

- Program controls apply at the level of the individual program and programmer. When we test a system, we try to make sure that the functionality prescribed in the requirements is implemented in the code. That is, we take a "should do" checklist and verify that the code does what it is supposed to do. However, security is also about preventing certain actions: a "shouldn't do" list. A system shouldn't do anything not on its "should do" list. It is almost impossible to ensure that a program does precisely what its designer or user intended, and nothing more. Regardless of designer or programmer intent, in a large and complex system, the pieces that have to fit together properly interact in an unmanageably large number of ways. We are forced to examine and test the code for typical or likely cases; we cannot exhaustively test every state and data combination to verify a system's behavior. So sheer size and complexity preclude total flaw prevention or mediation. Programmers intending to implant malicious code can take advantage of this incompleteness and hide some flaws successfully,

- Programming and software engineering techniques change and evolve far more rapidly than do computer security techniques. So we often find ourselves trying to secure last year's technology while software developers are rapidly adopting today's and next year's technology.

Still, the situation is far from bleak. Computer security has much to offer to program security. By understanding what can go wrong and how to protect against it, we can devise techniques and tools to secure most computer applications.

3.1.3 Types of Flaws

To aid our understanding of the problems and their prevention or correction, we can define categories that distinguish one kind of problem from another. For example, a taxonomy of program flaws, dividing them first into intentional and inadvertent flaws. They further divide intentional flaws into malicious and nonmalicious ones. In the taxonomy, the inadvertent flaws fall into six categories:

- validation error (incomplete or inconsistent): permission checks
- domain error: controlled access to data
- serialization and aliasing: program flow order
- inadequate identification and authentication: basis for authorization

- boundary condition violation: failure on first or last case
- other exploitable logic errors

3.2 NONMALICIOUS PROGRAM ERRORS

Being human, programmers and other developers make many mistakes, most of which are unintentional and nonmalicious. Many such errors cause program malfunctions but do not lead to more serious security vulnerabilities. However, a few classes of errors have plagued programmers and security professionals for decades, and there is no reason to believe they will disappear. In this section we consider three classic error types that have enabled many recent security breaches. We explain each type, why it is relevant to security, and how it can be prevented or mitigated.

3.2.1 Buffer Overflows

A buffer overflow is the computing equivalent of trying to pour two liters of water into a one-liter pitcher: Some water is going to spill out and make a mess. And in computing, what a mess these errors have made!

Definition

A buffer (or array or string) is a space in which data can be held. A buffer resides in memory. Because memory is finite, a buffer's capacity is finite. For this reason, in many programming languages the programmer must declare the buffer's maximum size so that the compiler can set aside that amount of space.

Let us look at an example to see how buffer overflows can happen. Suppose a C language program contains the declaration:

char sample[10];

The compiler sets aside 10 bytes to store this buffer, one byte for each of the 10 elements of the array, sample[0] tHRough sample[9].

Now we execute the statement:

```
sample[10] = 'B';
```

The subscript is out of bounds (that is, it does not fall between 0 and 9), so we have a problem. The nicest outcome (from a security perspective) is for the compiler to detect the problem and mark the error during compilation. However, if the statement were

```
sample[i] = 'B';
```

we could not identify the problem until i was set during execution to a too-big subscript. It would be useful if, during execution, the system produced an error message warning of a subscript out of bounds. Unfortunately, in some languages, buffer sizes do not have to be predefined, so there is no way to detect an out-of-bounds error. More importantly, the code needed to check each subscript against its potential maximum value takes time and space during execution, and the resources are applied to catch a problem that occurs relatively infrequently. Even if the compiler were careful in analyzing the buffer declaration and use, this same problem can be caused with pointers, for which there is no reasonable way to define a proper limit. Thus, some compilers do not generate the code to check for exceeding bounds.

Let us examine this problem more closely. It is important to recognize that the potential overflow causes a serious problem only in some instances. The problem's occurrence depends on what is adjacent to the array sample. For example, suppose each of the ten elements of the array sample is filled with the letter A and the erroneous reference uses the letter B, as follows:

```
for (i=0; i<=9; i++)
        sample[i] = 'A';
sample[10] = 'B'
```

All program and data elements are in memory during execution, sharing space with the operating system, other code, and resident routines. So there are four cases to consider in deciding where the 'B' goes, as shown in Figure 1. If the extra character overflows into the user's data space, it simply overwrites an existing variable value (or it may be written into an as-yet unused location), perhaps affecting the program's result, but affecting no other program or data.

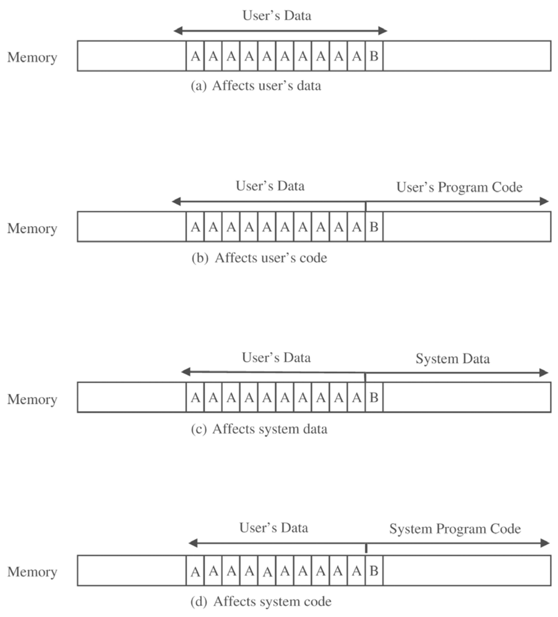


Figure 1: Places Where a Buffer Can Overflow.

In the second case, the 'B' goes into the user's program area. If it overlays an already executed instruction (which will not be executed again), the user should perceive no effect. If it overlays an instruction that is not yet executed, the machine will try to execute an instruction with operation code 0x42, the internal code for the character 'B'. If there is no instruction with operation code 0x42, the system will halt on an illegal instruction exception. Otherwise, the machine will use subsequent bytes as if they were the rest of the instruction, with success or failure depending on the meaning of the contents. Again, only the user is likely to experience an effect.

The most interesting cases occur when the system owns the space immediately after the array that overflows. Spilling over into system data or code areas produces similar results to those for the user's space: computing with a faulty value or trying to execute an improper operation.

3.2.2 Security Implication

We consider program flaws from unintentional or nonmalicious causes. Remember, however, that even if a flaw came from an honest mistake, the flaw can still cause serious harm. A malicious attacker can exploit these flaws.

Let us suppose that a malicious person understands the damage that can be done by a buffer overflow; that is, we are dealing with more than simply a normal, errant programmer. The malicious programmer looks at the four cases illustrated in Figure 1 and thinks deviously about the last two: What data values could the attacker insert just after the buffer to cause mischief or damage, and what planned instruction codes could the system be forced to execute? There are many possible answers, some of which are more malevolent than others. Here, we present two buffer overflow attacks that are used frequently.

First, the attacker may replace code in the system space. Remember that every program is invoked by the operating system and that the operating system may run with higher privileges than those of a regular program. Thus, if the attacker can gain control by masquerading as the operating system, the attacker can execute many commands in a powerful role. Therefore, by replacing a few instructions right after returning from his or her own procedure, the attacker regains control from the operating system, possibly with raised privileges. If the buffer overflows into system code space, the attacker merely inserts overflow data that correspond to the machine code for instructions.

On the other hand, the attacker may make use of the stack pointer or the return register. Subprocedure calls are handled with a stack, a data structure in which the most recent item inserted is the next one removed (last arrived, first served). This structure works well because procedure calls can be nested, with each return causing control to transfer back to the immediately preceding routine at its point of execution. Each time a procedure is called, its parameters, the return address (the address immediately after its call), and other local values are pushed onto a stack. An old stack pointer is also pushed onto the stack, and a stack pointer register is reloaded with the address of these new values. Control is then transferred to the subprocedure.

As the subprocedure executes, it fetches parameters that it finds by using the address pointed to by the stack pointer. Typically, the stack pointer is a register in the processor. Therefore, by causing an overflow into the stack, the attacker can change either the old stack pointer (changing the context for the calling procedure) or the return address (causing control to transfer where the attacker wants when the subprocedure returns). Changing the context or return address allows the attacker to redirect execution to a block of code the attacker wants.

In both these cases, a little experimentation is needed to determine where the overflow is and how to control it. But the work to be done is relatively smallprobably a day or two for a competent analyst. These buffer overflows are carefully explained in a paper of the famed l0pht computer security group. Buffer overflows ten years after Mudge and found that, far from being a minor aspect of attack, buffer overflows have been a very significant attack vector and have spawned several other new attack types.

An alternative style of buffer overflow occurs when parameter values are passed into a routine, especially when the parameters are passed to a web server on the Internet. Parameters are passed in the URL line, with a syntax similar to

http://www.somesite.com/subpage/userinput.asp?parm1=(808)555-1212 &parm2=2009Jan17

In this example, the page userinput receives two parameters, parm1 with value (808)555-1212 (perhaps a U.S. telephone number) and parm2 with value 2009Jan17 (perhaps a date). The web browser on the caller's machine will accept values from a user who probably completes fields on a form. The browser encodes those values and transmits them back to the server's web site.

The attacker might question what the server would do with a really long telephone number, say, one with 500 or 1000 digits. But, you say, no telephone in the world has such a number; that is probably exactly what the developer thought, so the developer may have allocated 15 or 20 bytes for an expected maximum length telephone number. Will the program crash with 500 digits? And if it crashes, can it be made to crash in a predictable and usable way? Passing a very long string to a web server is a slight variation on the classic buffer overflow, but no less effective.

Buffer overflows have existed almost as long as higher-level programming languages with arrays. For a long time they were simply a minor annoyance to programmers and users, a cause of errors and sometimes even system crashes. Rather recently, attackers have used them as vehicles to cause first a system crash and then a controlled failure with a serious security implication. The large number of security vulnerabilities based on buffer overflows shows that developers must pay more attention now to what had previously been thought to be just a minor annoyance.

3.2.3 Incomplete Mediation

Incomplete mediation is another security problem that has been with us for decades. Attackers are exploiting it to cause security problems.

Definition

Consider the example of the previous section:

http://www.somesite.com/subpage/userinput.asp?parm1=(808)555-1212 &parm2=2009Jan17

The two parameters look like a telephone number and a date. Probably the client's (user's) web browser enters those two values in their specified format for easy processing on the server's side.

What would happen if parm2 were submitted as 1800Jan01? Or 1800Feb30? Or 2048Min32? Or 1Aardvark2Many?

Something would likely fail. As with buffer overflows, one possibility is that the system would fail catastrophically, with a routine's failing on a data type error as it tried to handle a month named "Min" or even a year (like 1800) that was out of range. Another possibility is that the receiving program would continue to execute but would generate a very wrong result. (For example, imagine the amount of interest due today on a billing error with a start date of 1 Jan 1800.) Then again, the processing server might have a default condition, deciding to treat 1Aardvark2Many as 3 July 1947. The possibilities are endless.

One way to address the potential problems is to try to anticipate them. For instance, the programmer in the examples above may have written code to check for correctness on the client's side (that is, the user's browser). The client program can search for and screen out errors. Or, to prevent the use of nonsense data, the program can restrict choices only to valid ones. For example, the program supplying the parameters might have solicited them by using a drop-down box or choice list from which only the twelve conventional months would have been possible choices. Similarly, the year could have been tested to ensure that the value was between 1995 and 2015, and date numbers would have to have been appropriate for the months in which they occur (no 30th of February, for example). Using these verification techniques, the programmer may have felt well insulated from the possible problems a careless or malicious user could cause.

However, the program is still vulnerable. By packing the result into the return URL, the programmer left these data fields in a place the user can access (and modify). In particular, the user could edit the URL line, change any parameter values, and resend the line. On the server side, there is no way for the server to tell if the response line came from the client's browser or as a result of the user's editing the URL directly. We say in this case that the data values are not completely mediated: The sensitive data (namely, the parameter values) are in an exposed, uncontrolled condition.

3.2.4 Security Implication

Incomplete mediation is easy to exploit, but it has been exercised less often than buffer overflows. Nevertheless, unchecked data values represent a serious potential vulnerability.

To demonstrate this flaw's security implications, we use a real example; only the name of the vendor has been changed to protect the guilty. Things, Inc., was a very large, international vendor of consumer products, called Objects. The company was ready to sell its Objects through a web site, using what appeared to be a standard e-commerce application. The management at Things decided to let some of its in-house developers produce the web site so that its customers could order Objects directly from the web.

To accompany the web site, Things developed a complete price list of its Objects, including pictures, descriptions, and drop-down menus for size, shape, color, scent, and any other properties. For example, a customer on the web could choose to buy 20 of part number 555A Objects. If the price of one such part were $10, the web server would correctly compute the price of the 20 parts to be $200. Then the customer could decide whether to have the Objects shipped by boat, by ground transportation, or sent electronically. If the customer were to choose boat delivery, the customer's web browser would complete a form with parameters like these:

http://www.things.com/order.asp?custID=101&part=555A&qy=20 &price =10&ship=boat&shipcost=5&total=205

So far, so good; everything in the parameter passage looks correct. But this procedure leaves the parameter statement open for malicious tampering. Things should not need to pass the price of the items back to itself as an input parameter; presumably Things knows how much its Objects cost, and they are unlikely to change dramatically since the time the price was quoted a few screens earlier.

A malicious attacker may decide to exploit this peculiarity by supplying instead the following URL, where the price has been reduced from $205 to $25:

http://www.things.com/order.asp?custID=101&part=555A&qy=20 &price =1&ship=boat&shipcost=5&total=25

Surprise! It worked. The attacker could have ordered Objects from Things in any quantity at any price. And yes, this code was running on the web site for a while before the problem was detected. From a security perspective, the most serious concern about this flaw was the length of time that it could have run undetected. Had the whole world suddenly made a rush to Things's web site and bought Objects at a fraction of their price, Things probably would have noticed. But Things is large enough that it would never have detected a few customers a day choosing prices that were similar to (but smaller than) the real price, say 30 percent off. The e-commerce division would have shown a slightly smaller profit than other divisions, but the difference probably would not have been enough to raise anyone's eyebrows; the vulnerability could have gone unnoticed for years. Fortunately, Things hired a consultant to do a routine review of its code, and the consultant found the error quickly.

This web program design flaw is easy to imagine in other web settings. Those of us interested in security must ask ourselves how many similar problems are there in running code today? And how will those vulnerabilities ever be found?

3.2.5 Time-of-Check to Time-of-Use Errors

The third programming flaw we investigate involves synchronization. To improve efficiency, modern processors and operating systems usually change the order in which instructions and procedures are executed. In particular, instructions that appear to be adjacent may not actually be executed immediately after each other, either because of intentionally changed order or because of the effects of other processes in concurrent execution.

Definition

Access control is a fundamental part of computer security; we want to make sure that only those who should access an object are allowed that access. Every requested access must be governed by an access policy stating who is allowed access to what; then the request must be mediated by an access-policy-enforcement agent. But an incomplete mediation problem occurs when access is not checked universally. The time-of-check to time-of-use (TOCTTOU) flaw concerns mediation that is performed with a "bait and switch" in the middle. It is also known as a serialization or synchronization flaw.

To understand the nature of this flaw, consider a person's buying a sculpture that costs $100. The buyer removes five $20 bills from a wallet, carefully counts them in front of the seller, and lays them on the table. Then the seller turns around to write a receipt. While the seller's back is turned, the buyer takes back one $20 bill. When the seller turns around, the buyer hands over the stack of bills, takes the receipt, and leaves with the sculpture. Between the time the security was checked (counting the bills) and the access (exchanging the sculpture for the bills), a condition changed: What was checked is no longer valid when the object (that is, the sculpture) is accessed.

A similar situation can occur with computing systems. Suppose a request to access a file were presented as a data structure, with the name of the file and the mode of access presented in the structure.

The data structure is essentially a "work ticket," requiring a stamp of authorization; once authorized, it is put on a queue of things to be done. Normally the access control mediator receives the data structure, determines whether the access should be allowed, and either rejects the access and stops or allows the access and forwards the data structure to the file handler for processing.

To carry out this authorization sequence, the access control mediator would have to look up the file name (and the user identity and any other relevant parameters) in tables. The mediator could compare the names in the table to the file name in the data structure to determine whether access is appropriate. More likely, the mediator would copy the file name into its own local storage area and compare from there. Comparing from the copy leaves the data structure in the user's area, under the user's control.

It is at this point that the incomplete mediation flaw can be exploited. While the mediator is checking access rights for the file my file, the user could change the file name descriptor to your file. Having read the work ticket once, the mediator would not be expected to reread the ticket before approving it; the mediator would approve the access and send the now-modified descriptor to the file handler.

The problem is called a time-of-check to time-of-use flaw because it exploits the delay between the two times. That is, between the time the access was checked and the time the result of the check was used, a change occurred, invalidating the result of the check.

3.2.6 Security Implication

The security implication here is pretty clear: Checking one action and performing another is an example of ineffective access control. We must be wary whenever a time lag or loss of control occurs, making sure that there is no way to corrupt the check's results during that interval.

Fortunately, there are ways to prevent exploitation of the time lag. One way is to ensure that critical parameters are not exposed during any loss of control. The access checking software must own the request data until the requested action is complete. Another way is to ensure serial integrity; that is, to allow no interruption (loss of control) during the validation. Or the validation routine can initially copy data from the user's space to the routine's area out of the user's reach and perform validation checks on the copy. Finally, the validation routine can seal the request data with a checksum to detect modification.

3.2.7 Combinations of Nonmalicious Program Flaws

These three vulnerabilities are bad enough when each is considered on its own. But perhaps the worst aspect of all three flaws is that they can be used together as one step in a multistep attack. An attacker may not be content with causing a buffer overflow. Instead the attacker may begin a three-pronged attack by using a buffer overflow to disrupt all execution of arbitrary code on a machine. At the same time, the attacker may exploit a time-of-check to time-of-use flaw to add a new user ID to the system. The attacker then logs in as the new user and exploits an incomplete mediation flaw to obtain privileged status, and so forth. The clever attacker uses flaws as common building blocks to build a complex attack. For this reason, we must know about and protect against even simple flaws. Unfortunately, these kinds of flaws are widespread and dangerous. As we see in the next section, innocuous-seeming program flaws can be exploited by malicious attackers to plant intentionally harmful code.

3.3 VIRUSES AND OTHER MALICIOUS CODE

By themselves, programs are seldom security threats. The programs operate on data, taking action only when data and state changes trigger it. Much of the work done by a program is invisible to users who are not likely to be aware of any malicious activity. For instance, when was the last time you saw a bit? Do you know in what form a document file is stored? If you know a document resides somewhere on a disk, can you find it? Can you tell if a game program does anything in addition to its expected interaction with you? Which files are modified by a word processor when you create a document? Which programs execute when you start your computer or open a web page? Most users cannot answer these questions. However, since users usually do not see computer data directly, malicious people can make programs serve as vehicles to access and change data and other programs. Let us look at the possible effects of malicious code and then examine in detail several kinds of programs that can be used for interception or modification of data.

3.3.1 Why Worry About Malicious Code?

None of us like the unexpected, especially in our programs. Malicious code behaves in unexpected ways, thanks to a malicious programmer's intention. We think of the malicious code as lurking inside our system: all or some of a program that we are running or even a nasty part of a separate program that somehow attaches itself to another (good) program.

How can such a situation arise? When you last installed a major software package, such as a word processor, a statistical package, or a plug-in from the Internet, you ran one command, typically called INSTALL or SETUP. From there, the installation program took control, creating some files, writing in other files, deleting data and files, and perhaps renaming a few that it would change. A few minutes and a quite a few disk accesses later, you had plenty of new code and data, all set up for you with a minimum of human intervention. Other than the general descriptions on the box, in documentation files, or on web pages, you had absolutely no idea exactly what "gifts" you had received. You hoped all you received was good, and it probably was. The same uncertainty exists when you unknowingly download an application, such as a Java applet or an ActiveX control, while viewing a web site. Thousands or even millions of bytes of programs and data are transferred, and hundreds of modifications may be made to your existing files, all occurring without your explicit consent or knowledge.

3.3.2 Malicious Code Can Do Much (Harm)

Malicious code can do anything any other program can, such as writing a message on a computer screen, stopping a running program, generating a sound, or erasing a stored file. Or malicious code can do nothing at all right now; it can be planted to lie dormant, undetected, until some event triggers the code to act.

The trigger can be a time or date, an interval (for example, after 30 minutes), an event (for example, when a particular program is executed), a condition (for example, when communication occurs on a network interface), a count (for example, the fifth time something happens), some combination of these, or a random situation. In fact, malicious code can do different things each time, or nothing most of the time with something dramatic on occasion. In general, malicious code can act with all the predictability of a two-year-old child: We know in general what two-year-olds do, we may even know what a specific two-year-old often does in certain situations, but two-year-olds have an amazing capacity to do the unexpected.

Malicious code runs under the user's authority. Thus, malicious code can touch everything the user can touch, and in the same ways. Users typically have complete control over their own program code and data files; they can read, write, modify, append, and even delete them. And well they should. But malicious code can do the same, without the user's permission or even knowledge.

3.3.3 Malicious Code Has Been Around a Long Time

The popular literature and press continue to highlight the effects of malicious code as if it were a relatively recent phenomenon. It is not. Cohen [COH84] is sometimes credited with the discovery of viruses, but in fact Cohen gave a name to a phenomenon known long before. For example, Thompson, in his 1984 Turing Award lecture, "Reflections on Trusting Trust" [THO84], described code that can be passed by a compiler. vulnerabilities, and program security flaws, especially intentional ones. What is new about malicious code is the number of distinct instances and copies that have appeared and the speed with which exploit code appears.

So malicious code is still around, and its effects are more pervasive. It is important for us to learn what it looks like and how it works so that we can take steps to prevent it from doing damage or at least mediate its effects. How can malicious code take control of a system? How can it lodge in a system? How does malicious code spread? How can it be recognized? How can it be detected? How can it be stopped? How can it be prevented?

3.3.4 Kinds of Malicious Code

Malicious code or rogue program

Malicious code or rogue program is the general name for unanticipated or undesired effects in programs or program parts, caused by an agent intent on damage. This definition excludes unintentional errors, although they can also have a serious negative effect. This definition also excludes coincidence, in which two benign programs combine for a negative effect. The agent is the writer of the program or the person who causes its distribution. By this definition, most faults found in software inspections, reviews, and testing do not qualify as malicious code, because we think of them as unintentional. However, keep in mind as you read this chapter that unintentional faults can in fact invoke the same responses as intentional malevolence; a benign cause can still lead to a disastrous effect.

You are likely to have been affected by a virus at one time or another, either because your computer was infected by one or because you could not access an infected system while its administrators were cleaning up the mess one made. In fact, your virus might actually have been a worm: The terminology of malicious code is sometimes used imprecisely. A virus is a program that can replicate itself and pass on malicious code to other nonmalicious programs by modifying them. The term "virus" was coined because the affected program acts like a biological virus: It infects other healthy subjects by attaching itself to the program and either destroying it or coexisting with it. Because viruses are insidious, we cannot assume that a clean program yesterday is still clean today. Moreover, a good program can be modified to include a copy of the virus program, so the infected good program itself begins to act as a virus, infecting other programs. The infection usually spreads at a geometric rate, eventually overtaking an entire computing system and spreading to all other connected systems.

A virus can be either transient or resident. A transient virus has a life that depends on the life of its host; the virus runs when its attached program executes and terminates when its attached program ends. (During its execution, the transient virus may spread its infection to other programs.) A resident virus locates itself in memory; then it can remain active or be activated as a stand-alone program, even after its attached program ends.

Trojan Horse

A Trojan horse is malicious code that, in addition to its primary effect, has a second, nonobvious malicious effect.[1] As an example of a computer Trojan horse, consider a login script that solicits a user's identification and password, passes the identification information on to the rest of the system for login processing, but also retains a copy of the information for later, malicious use. In this example, the user sees only the login occurring as expected, so there is no evident reason to suspect that any other action took place.

Logic Bomb

A logic bomb is a class of malicious code that "detonates" or goes off when a specified condition occurs. A time bomb is a logic bomb whose trigger is a time or date.

Trapdoor or backdoor

A trapdoor or backdoor is a feature in a program by which someone can access the program other than by the obvious, direct call, perhaps with special privileges. For instance, an automated bank teller program might allow anyone entering the number 990099 on the keypad to process the log of everyone's transactions at that machine. In this example, the trapdoor could be intentional, for maintenance purposes, or it could be an illicit way for the implementer to wipe out any record of a crime.

Worm

A worm is a program that spreads copies of itself through a network. Shock and Hupp [SHO82] are apparently the first to describe a worm, which, interestingly, was for nonmalicious purposes. The primary difference between a worm and a virus is that a worm operates through networks, and a virus can spread through any medium (but usually uses copied program or data files). Additionally, the worm spreads copies of itself as a stand-alone program, whereas the virus spreads copies of itself as a program that attaches to or embeds in other programs.

A rabbit as a virus or worm that self-replicates without bound, with the intention of exhausting some computing resource. A rabbit might create copies of itself and store them on disk in an effort to completely fill the disk, for example.

These definitions match current careful usage. The distinctions among these terms are small, and often the terms are confused, especially in the popular press. The term "virus" is often used to refer to any piece of malicious code. Furthermore, two or more forms of malicious code can be combined to produce a third kind of problem. For instance, a virus can be a time bomb if the viral code that is spreading will trigger an event after a period of time has passed. The kinds of malicious code are summarized in Table 1.

Table 1: Types of Malicious Code

Code Type	Characteristics
Virus	Attaches itself to program and propagates copies of itself to other programs
Trojan horse	Contains unexpected, additional functionality
Logic bomb	Triggers action when condition occurs
Time bomb	Triggers action when specified time occurs
Trapdoor	Allows unauthorized access to functionality
Worm	Propagates copies of itself through a network
Rabbit	Replicates itself without limit to exhaust resources

Because "virus" is the popular name given to all forms of malicious code and because fuzzy lines exist between different kinds of malicious code, we are not too restrictive in the following discussion. We want to look at how malicious code spreads, how it is activated, and what effect it can have. A virus is a convenient term for mobile malicious code, so in the following sections we use the term "virus" almost exclusively. The points made apply also to other forms of malicious code.

3.3.5 How Viruses Attach

A printed copy of a virus does nothing and threatens no one. Even executable virus code sitting on a disk does nothing. What triggers a virus to start replicating? For a virus to do its malicious work and spread itself, it must be activated by being executed. Fortunately for virus writers but unfortunately for the rest of us, there are many ways to ensure that programs will be executed on a running computer.

For example, recall the SETUP program that you initiate on your computer. It may call dozens or hundreds of other programs, some on the distribution medium, some already residing on the computer, some in memory. If any one of these programs contains a virus, the virus code could be activated. Let us see how. Suppose the virus code were in a program on the distribution medium, such as a CD; when executed, the virus could install itself on a permanent storage medium (typically, a hard disk) and also in any and all executing programs in memory. Human intervention is necessary to start the process; a human being puts the virus on the distribution medium, and perhaps another initiates the execution of the program to which the virus is attached. (It is possible for execution to occur without human intervention, though, such as when execution is triggered by a date or the passage of a certain amount of time.) After that, no human intervention is needed; the virus can spread by itself.

A more common means of virus activation is as an attachment to an e-mail message. In this attack, the virus writer tries to convince the victim (the recipient of the e-mail message) to open the attachment. Once the viral attachment is opened, the activated virus can do its work. Some modern e-mail handlers, in a drive to "help" the receiver (victim), automatically open attachments as soon as the receiver opens the body of the e-mail message. The virus can be executable code embedded in an executable attachment, but other types of files are equally dangerous. For example, objects such as graphics or photo images can contain code to be executed by an editor, so they can be transmission agents for viruses. In general, it is safer to force users to open files on their own rather than automatically; it is a bad idea for programs to perform potentially security-relevant actions without a user's consent. However, ease-of-use often trumps security, so programs such as browsers, e-mail handlers, and viewers often "helpfully" open files without asking the user first.

3.3.6 Appended Viruses

A program virus attaches itself to a program; then, whenever the program is run, the virus is activated. This kind of attachment is usually easy to program.

In the simplest case, a virus inserts a copy of itself into the executable program file before the first executable instruction. Then, all the virus instructions execute first; after the last virus instruction, control flows naturally to what used to be the first program instruction. Such a situation is shown in Figure 2.

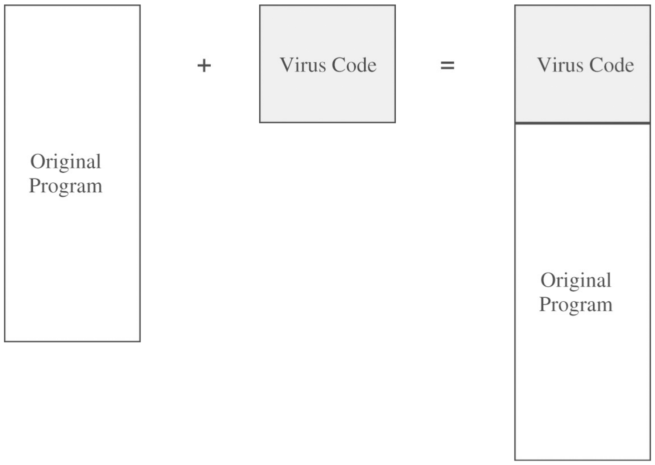


Figure 2: Virus Appended to a Program.

This kind of attachment is simple and usually effective. The virus writer does not need to know anything about the program to which the virus will attach, and often the attached program simply serves as a carrier for the virus. The virus performs its task and then transfers to the original program. Typically, the user is unaware of the effect of the virus if the original program still does all that it used to. Most viruses attach in this manner.

3.3.7 Viruses That Surround a Program

An alternative to the attachment is a virus that runs the original program but has control before and after its execution. For example, a virus writer might want to prevent the virus from being detected. If the virus is stored on disk, its presence will be given away by its file name, or its size will affect the amount of space used on the disk. The virus writer might arrange for the virus to attach itself to the program that constructs the listing of files on the disk. If the virus regains control after the listing program has generated the listing but before the listing is displayed or printed, the virus could eliminate its entry from the listing and falsify space counts so that it appears not to exist. A surrounding virus is shown in Figure 3.

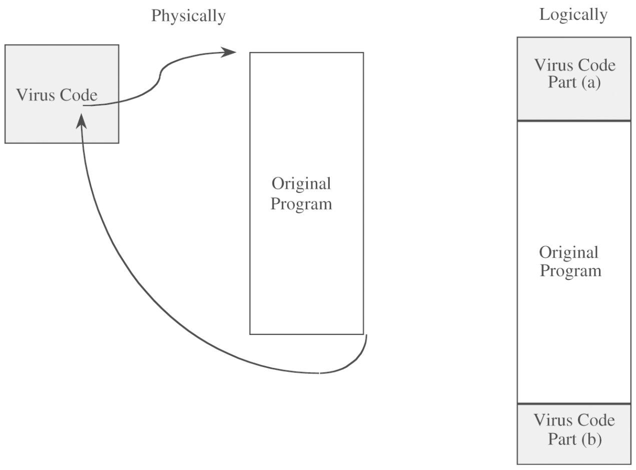


Figure 3: Virus Surrounding a Program.

3.3.8 Integrated Viruses and Replacements

A third situation occurs when the virus replaces some of its target, integrating itself into the original code of the target. Such a situation is shown in Figure 4. Clearly, the virus writer has to know the exact structure of the original program to know where to insert which pieces of the virus.

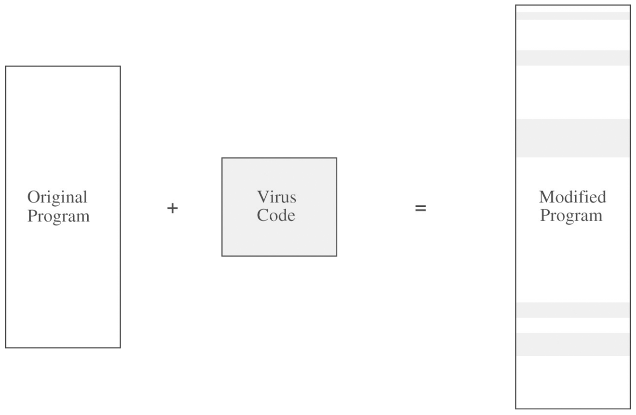


Figure 4: Virus Integrated into a Program.

Finally, the virus can replace the entire target, either mimicking the effect of the target or ignoring the expected effect of the target and performing only the virus effect. In this case, the user is most likely to perceive the loss of the original program.

3.3.9 Document Viruses

Currently, the most popular virus type is what we call the document virus, which is implemented within a formatted document, such as a written document, a database, a slide presentation, a picture, or a spreadsheet. These documents are highly structured files that contain both data (words or numbers) and commands (such as formulas, formatting controls, links). The commands are part of a rich programming language, including macros, variables and procedures, file accesses, and even system calls. The writer of a document virus uses any of the features of the programming language to perform malicious actions.

The ordinary user usually sees only the content of the document (its text or data), so the virus writer simply includes the virus in the commands part of the document, as in the integrated program virus.

3.3.10 How Viruses Gain Control

The virus (V) has to be invoked instead of the target (T). Essentially, the virus either has to seem to be T, saying effectively "I am T" or the virus has to push T out of the way and become a substitute for T, saying effectively "Call me instead of T." A more blatant virus can simply say "invoke me [you fool]."

The virus can assume T's name by replacing (or joining to) T's code in a file structure; this invocation technique is most appropriate for ordinary programs. The virus can overwrite T in storage (simply replacing the copy of T in storage, for example). Alternatively, the virus can change the pointers in the file table so that the virus is located instead of T whenever T is accessed through the file system. These two cases are shown in Figure 5.

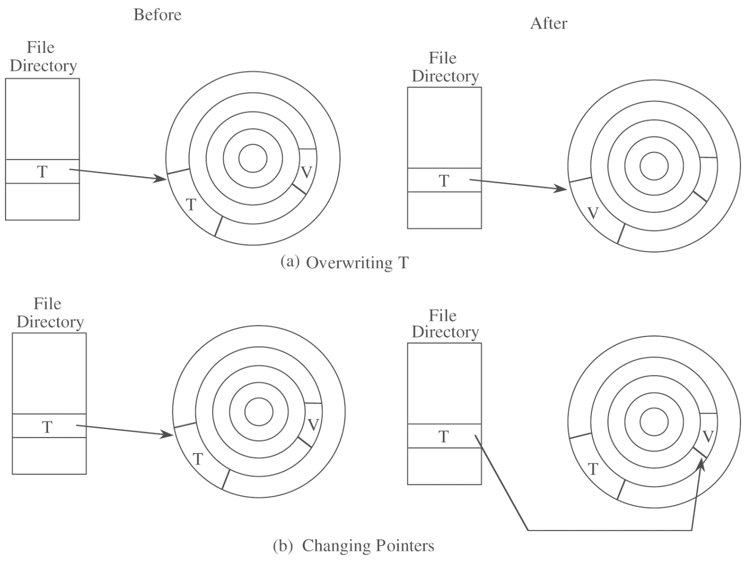


Figure 5: Virus Completely Replacing a Program.

The virus can supplant T by altering the sequence that would have invoked T to now invoke the virus V; this invocation can be used to replace parts of the resident operating system by modifying pointers to those resident parts, such as the table of handlers for different kinds of interrupts.

3.3.11 Homes for Viruses

The virus writer may find these qualities appealing in a virus:

- It is hard to detect.
- It is not easily destroyed or deactivated.
- It spreads infection widely.
- It can reinfect its home program or other programs.
- It is easy to create.
- It is machine independent and operating system independent.

Few viruses meet all these criteria. The virus writer chooses from these objectives when deciding what the virus will do and where it will reside.

Just a few years ago, the challenge for the virus writer was to write code that would be executed repeatedly so that the virus could multiply. Now, however, one execution is enough to ensure widespread distribution. Many viruses are transmitted by e-mail, using either of two routes. In the first case, some virus writers generate a new e-mail message to all addresses in the victim's address book. These new messages contain a copy of the virus so that it propagates widely. Often the message is a brief, chatty, nonspecific message that would encourage the new recipient to open the attachment from a friend (the first recipient). For example, the subject line or message body may read "I thought you might enjoy this picture from our vacation." In the second case, the virus writer can leave the infected file for the victim to forward unknowingly. If the virus's effect is not immediately obvious, the victim may pass the infected file unwittingly to other victims.

Let us look more closely at the issue of viral residence.

One-Time Execution

The majority of viruses today execute only once, spreading their infection and causing their effect in that one execution. A virus often arrives as an e-mail attachment of a document virus. It is executed just by being opened.

Boot Sector Viruses

A special case of virus attachment, but formerly a fairly popular one, is the so-called boot sector virus. When a computer is started, control begins with firmware that determines which hardware components are present, tests them, and transfers control to an operating system. A given hardware platform can run many different operating systems, so the operating system is not coded in firmware but is instead invoked dynamically, perhaps even by a user's choice, after the hardware test.

The operating system is software stored on disk. Code copies the operating system from disk to memory and transfers control to it; this copying is called the bootstrap (often boot) load because the operating system figuratively pulls itself into memory by its bootstraps. The firmware does its control transfer by reading a fixed number of bytes from a fixed location on the disk (called the boot sector) to a fixed address in memory and then jumping to that address (which will turn out to contain the first instruction of the bootstrap loader). The bootstrap loader then reads into memory the rest of the operating system from disk. To run a different operating system, the user just inserts a disk with the new operating system and a bootstrap loader. When the user reboots from this new disk, the loader there brings in and runs another operating system. This same scheme is used for personal computers, workstations, and large mainframes.

To allow for change, expansion, and uncertainty, hardware designers reserve a large amount of space for the bootstrap load. The boot sector on a PC is slightly less than 512 bytes, but since the loader will be larger than that, the hardware designers support "chaining," in which each block of the bootstrap is chained to (contains the disk location of) the next block. This chaining allows big bootstraps but also simplifies the installation of a virus. The virus writer simply breaks the chain at any point, inserts a pointer to the virus code to be executed, and reconnects the chain after the virus has been installed. This situation is shown in Figure 6.

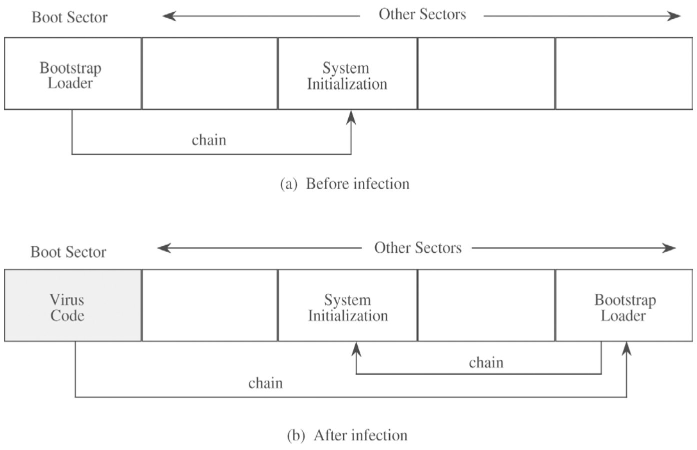


Figure 6: Boot Sector Virus Relocating Code.

The boot sector is an especially appealing place to house a virus. The virus gains control very early in the boot process, before most detection tools are active, so that it can avoid, or at least complicate, detection. The files in the boot area are crucial parts of the operating system. Consequently, to keep users from accidentally modifying or deleting them with disastrous results, the operating system makes them "invisible" by not showing them as part of a normal listing of stored files, preventing their deletion. Thus, the virus code is not readily noticed by users.

Memory-Resident Viruses

Some parts of the operating system and most user programs execute, terminate, and disappear, with their space in memory being available for anything executed later. For very frequently used parts of the operating system and for a few specialized user programs, it would take too long to reload the program each time it was needed. Such code remains in memory and is called "resident" code. Examples of resident code are the routine that interprets keys pressed on the keyboard, the code that handles error conditions that arise during a program's execution, or a program that acts like an alarm clock, sounding a signal at a time the user determines. Resident routines are sometimes called TSRs or “terminate and stay resident” routines.

Virus writers also like to attach viruses to resident code because the resident code is activated many times while the machine is running. Each time the resident code runs, the virus does too. Once activated, the virus can look for and infect uninfected carriers. For example, after activation, a boot sector virus might attach itself to a piece of resident code. Then, each time the virus was activated it might check whether any removable disk in a disk drive was infected and, if not, infect it. In this way the virus could spread its infection to all removable disks used during the computing session.

A virus can also modify the operating system’s table of programs to run. On a Windows machine the registry is the table of all critical system information, including programs to run at startup. If the virus gains control once, it can insert a registry entry so that it will be reinvoked each time the system restarts. In this way, even if the user notices and deletes the executing copy of the virus from memory, the virus will return on the next system restart.

3.3.12 Other Homes for Viruses

A virus that does not take up residence in one of these cozy establishments has to fend more for itself. But that is not to say that the virus will go homeless.

One popular home for a virus is an application program. Many applications, such as word processors and spreadsheets, have a “macro” feature, by which a user can record a series of commands and repeat them with one invocation. Such programs also provide a “startup macro” that is executed every time the application is executed. A virus writer can create a virus macro that adds itself to the startup directives for the application. It also then embeds a copy of itself in data files so that the infection spreads to anyone receiving one or more of those files.

Libraries are also excellent places for malicious code to reside. Because libraries are used by many programs, the code in them will have a broad effect. Additionally, libraries are often shared among users and transmitted from one user to another, a practice that spreads the infection. Finally, executing code in a library can pass on the viral infection to other transmission media. Compilers, loaders, linkers, runtime monitors, runtime debuggers, and even virus control programs are good candidates for hosting viruses because they are widely shared.

3.3.13 Virus Signatures

A virus cannot be completely invisible. Code must be stored somewhere, and the code must be in memory to execute. Moreover, the virus executes in a particular way, using certain methods to spread. Each of these characteristics yields a telltale pattern, called a signature, that can be found by a program that looks for it. The virus's signature is important for creating a program, called a virus scanner, that can detect and, in some cases, remove viruses. The scanner searches memory and long-term storage, monitoring execution and watching for the telltale signatures of viruses. For example, a scanner looking for signs of the Code Red worm can look for a pattern containing the following characters:

```
/default.ida?NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
%u9090%u6858%ucbd3
%u7801%u9090%u6858%ucdb3%u7801%u9090%u6858
%ucbd3%u7801%u9090
%u9090%u8190%u00c3%u0003%ub00%u531b%u53ff
%u0078%u0000%u00=a
HTTP/1.0
```

When the scanner recognizes a known virus's pattern, it can then block the virus, inform the user, and deactivate or remove the virus. However, a virus scanner is effective only if it has been kept up to date with the latest information on current viruses.

3.3.14 Storage Patterns

Most viruses attach to programs that are stored on media such as disks. The attached virus piece is invariant, so the start of the virus code becomes a detectable signature. The attached piece is always located at the same position relative to its attached file. For example, the virus might always be at the beginning, 400 bytes from the top, or at the bottom of the infected file. Most likely, the virus will be at the beginning of the file because the virus writer wants to obtain control of execution before the bona fide code of the infected program is in charge. In the simplest case, the virus code sits at the top of the program, and the entire virus does its malicious duty before the normal code is invoked. In other cases, the virus infection consists of only a handful of instructions that point or jump to other, more detailed instructions elsewhere. For example, the infected code may consist of condition testing and a jump or call to a separate virus module. In either case, the code to which control is transferred will also have a recognizable pattern. Both of these situations are shown in Figure 7.

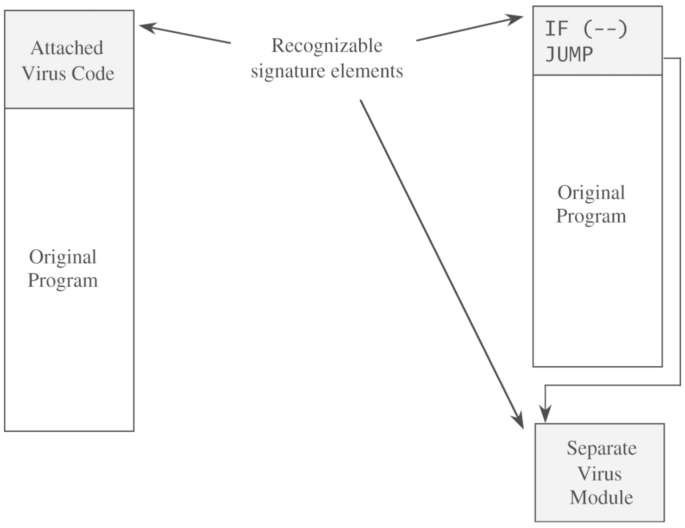


Figure 7: Recognizable Patterns in Viruses.

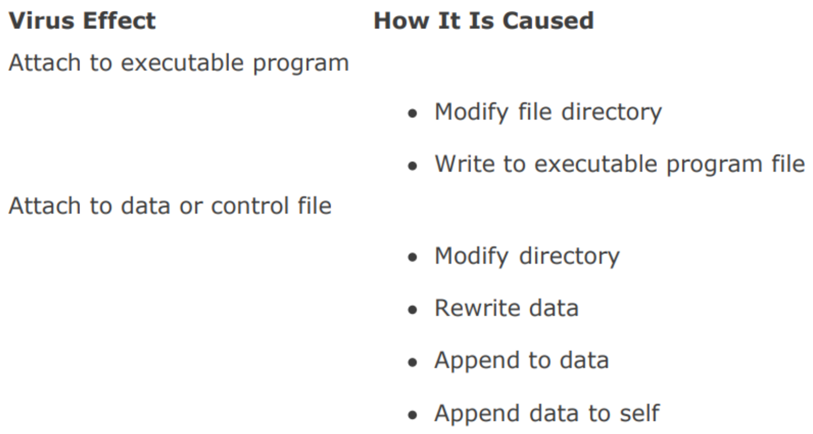
A virus may attach itself to a file, in which case the file's size grows. Or the virus may obliterate all or part of the underlying program, in which case the program's size does not change but the program's functioning will be impaired. The virus writer has to choose one of these detectable effects.

The virus scanner can use a code or checksum to detect changes to a file. It can also look for suspicious patterns, such as a JUMP instruction as the first instruction of a system program (in case the virus has positioned itself at the bottom of the file but is to be executed first, as in Figure 7).

3.3.15 Execution Patterns

A virus writer may want a virus to do several things at the same time, namely, spread infection, avoid detection, and cause harm. These goals are shown in Table 7, along with ways each goal can be addressed. Unfortunately, many of these behaviors are perfectly normal and might otherwise go undetected. For instance, one goal is modifying the file directory; many normal programs create files, delete files, and write to storage media. Thus, no key signals point to the presence of a virus.

Table 2: Virus Effects and Causes.



Virus Effect	How It Is Caused
Attach to executable program	• Modify file directory • Write to executable program file
Attach to data or control file	• Modify directory • Rewrite data • Append to data • Append data to self

Remain in memory	• Intercept interrupt by modifying interrupt handler address table
	• Load self in nontransient memory area
Infect disks	• Intercept interrupt
	• Intercept operating system call (to format disk, for example)
	• Modify system file
	• Modify ordinary executable program
Conceal self	• Intercept system calls that would reveal self and falsify result
	• Classify self as "hidden" file
Spread infection	• Infect boot sector
	• Infect systems program
	• Infect ordinary program
	• Infect data ordinary program reads to control its execution
Prevent deactivation	• Activate before deactivating program and block deactivation
	• Store copy to reinfect after deactivation

Most virus writers seek to avoid detection for themselves and their creations. Because a disk’s boot sector is not visible to normal operations (for example, the contents of the boot sector do not show on a directory listing), many virus writers hide their code there. A resident virus can monitor disk accesses and fake the result of a disk operation that would show the virus hidden in a boot sector by showing the data that should have been in the boot sector (which the virus has moved elsewhere).

There are no limits to the harm a virus can cause. On the modest end, the virus might do nothing; some writers create viruses just to show they can do it. Or the virus can be relatively benign, displaying a message on the screen, sounding the buzzer, or playing music. From there, the problems can escalate. One virus can erase files, another an entire disk; one virus can prevent a computer from booting, and another can prevent writing to disk. The damage is bounded only by the creativity of the virus's author.

3.3.16 Transmission Patterns

A virus is effective only if it has some means of transmission from one location to another. As we have already seen, viruses can travel during the boot process by attaching to an executable file or traveling within data files. The travel itself occurs during execution of an already infected program. Since a virus can execute any instructions a program can, virus travel is not confined to any single medium or execution pattern. For example, a virus can arrive on a disk or from a network connection, travel during its host's execution to a hard disk boot sector, reemerge next time the host computer is booted, and remain in memory to infect other disks as they are accessed.

3.3.17 Polymorphic Viruses

The virus signature may be the most reliable way for a virus scanner to identify a virus. If a particular virus always begins with the string 47F0F00E08 (in hexadecimal) and has string 00113FFF located at word 12, it is unlikely that other programs or data files will have these exact characteristics. For longer signatures, the probability of a correct match increases.

If the virus scanner will always look for those strings, then the clever virus writer can cause something other than those strings to be in those positions. Many instructions cause no effect, such as adding 0 to a number, comparing a number to itself, or jumping to the next instruction. These instructions, sometimes called noops, can be sprinkled into a piece of code to distort any pattern. For example, the virus could have two alternative but equivalent beginning words; after being installed, the virus will choose one of the two words for its initial word. Then, a virus scanner would have to look for both patterns. A virus that can change its appearance is called a polymorphic virus. (Poly means "many" and morph means "form.")

A two-form polymorphic virus can be handled easily as two independent viruses. Therefore, the virus writer intent on preventing detection of the virus will want either a large or an unlimited number of forms so that the number of possible forms is too large for a virus scanner to search for. Simply embedding a random number or string at a fixed place in the executable version of a virus is not sufficient, because the signature of the virus is just the constant code excluding the random part. A polymorphic virus has to randomly reposition all parts of itself and randomly change all fixed data. Thus, instead of containing the fixed (and therefore searchable) string "HA! INFECTED BY A VIRUS," a polymorphic virus has to change even that pattern sometimes.

Trivially, assume a virus writer has 100 bytes of code and 50 bytes of data. To make two virus instances different, the writer might distribute the first version as 100 bytes of code followed by all 50 bytes of data. A second version could be 99 bytes of code, a jump instruction, 50 bytes of data, and the last byte of code. Other versions are 98 code bytes jumping to the last two, 97 and three, and so forth. Just by moving pieces around, the virus writer can create enough different appearances to fool simple virus scanners. Once the scanner writers became aware of these kinds of tricks, however, they refined their signature definitions.

A simple variety of polymorphic virus uses encryption under various keys to make the stored form of the virus different. These are sometimes called encrypting viruses. This type of virus must contain three distinct parts: a decryption key, the (encrypted) object code of the virus, and the (unencrypted) object code of the decryption routine. For these viruses, the decryption routine itself, or a call to a decryption library routine, must be in the clear so that becomes the signature.

To avoid detection, not every copy of a polymorphic virus has to differ from every other copy. If the virus changes occasionally, not every copy will match a signature of every other copy.

3.3.18 The Source of Viruses

Since a virus can be rather small, its code can be “hidden” inside other larger and more complicated programs. Two hundred lines of a virus could be separated into one hundred packets of two lines of code and a jump each; these one hundred packets could be easily hidden inside a compiler, a database manager, a file manager, or some other large utility.

Virus discovery could be aided by a procedure to determine if two programs are equivalent. However, theoretical results in computing are very discouraging when it comes to the complexity of the equivalence problem. The general question “Are these two programs equivalent?” is undecidable (although that question can be answered for many specific pairs of programs). Even ignoring the general undecidability problem, two modules may produce subtly different results that mayor may notbe security relevant. One may run faster, or the first may use a temporary file for workspace whereas the second performs all its computations in memory. These differences could be benign, or they could be a marker of an infection. Therefore, we are unlikely to develop a screening program that can separate infected modules from uninfected ones.

Although the general is dismaying, the particular is not. If we know that a particular virus may infect a computing system, we can check for it and detect it if it is there. Having found the virus, however, we are left with the task of cleansing the system of it. Removing the virus in a running system requires being able to detect and eliminate its instances faster than it can spread.

3.3.19 Prevention of Virus Infection

The only way to prevent the infection of a virus is not to receive executable code from an infected source. This philosophy used to be easy to follow because it was easy to tell if a file was executable or not. For example, on PCs, a .exe extension was a clear sign that the file was executable. However, as we have noted, today's files are more complex, and a seemingly nonexecutable file may have some executable code buried deep within it. For example, a word processor may have commands within the document file; as we noted earlier, these commands, called macros, make it easy for the user to do complex or repetitive things. But they are really executable code embedded in the context of the document. Similarly, spreadsheets, presentation slides, other office- or business-related files, and even media files can contain code or scripts that can be executed in various waysand thereby harbor viruses. And, as we have seen, the applications that run or use these files may try to be helpful by automatically invoking the executable code, whether you want it run or not! Against the principles of good security, e-mail handlers can be set to automatically open (without performing access control) attachments or embedded code for the recipient, so your e-mail message can have animated bears dancing across the top.

Another approach virus writers have used is a little-known feature in the Microsoft file design. Although a file with a .doc extension is expected to be a Word document, in fact, the true document type is hidden in a field at the start of the file. This convenience ostensibly helps a user who inadvertently names a Word document with a .ppt (Power-Point) or any other extension. In some cases, the operating system will try to open the associated application but, if that fails, the system will switch to the application of the hidden file type. So, the virus writer creates an executable file, names it with an inappropriate extension, and sends it to the victim, describing it is as a picture or a necessary code add-in or something else desirable. The unwitting recipient opens the file and, without intending to, executes the malicious code.

More recently, executable code has been hidden in files containing large data sets, such as pictures or read-only documents. These bits of viral code are not easily detected by virus scanners and certainly not by the human eye. For example, a file containing a photograph may be highly granular; if every sixteenth bit is part of a command string that can be executed, then the virus is very difficult to detect.

Because you cannot always know which sources are infected, you should assume that any outside source is infected. Fortunately, you know when you are receiving code from an outside source; unfortunately, it is not feasible to cut off all contact with the outside world.

In their interesting paper comparing computer virus transmission with human disease transmission, Individuals' efforts to keep their computers free from viruses lead to communities that are generally free from viruses because members of the community have little (electronic) contact with the outside world. In this case, transmission is contained not because of limited contact but because of limited contact outside the community. Governments, for military or diplomatic secrets, often run disconnected network communities. The trick seems to be in choosing one's community prudently. However, as use of the Internet and the World Wide Web increases, such separation is almost impossible to maintain.

Nevertheless, there are several techniques for building a reasonably safe community for electronic contact, including the following:

- *Use only commercial software acquired from reliable, well-established vendors.* There is always a chance that you might receive a virus from a large manufacturer with a name everyone would recognize. However, such enterprises have significant reputations that could be seriously damaged by even one bad incident, so they go to some degree of trouble to keep their products virus-free and to patch any problem-causing code right away. Similarly, software distribution companies will be careful about products they handle.

- *Test all new software on an isolated computer.* If you must use software from a questionable source, test the software first on a computer that is not connected to a network and contains no sensitive or important data. Run the software and look for unexpected behavior, even simple behavior such as unexplained figures on the screen. Test the computer with a copy of an up-to-date virus scanner created before the suspect program is run. Only if the program passes these tests should you install it on a less isolated machine.
- *Open attachments only when you know them to be safe.* What constitutes "safe" is up to you, as you have probably already learned in this chapter. Certainly, an attachment from an unknown source is of questionable safety. You might also distrust an attachment from a known source but with a peculiar message.
- *Make a recoverable system image and store it safely.* If your system does become infected, this clean version will let you reboot securely because it overwrites the corrupted system files with clean copies. For this reason, you must keep the image write-protected during reboot. Prepare this image now, before infection; after infection it is too late. For safety, prepare an extra copy of the safe boot image.
- *Make and retain backup copies of executable system files.* This way, in the event of a virus infection, you can remove infected files and reinstall from the clean backup copies (stored in a secure, offline location, of course). Also make and retain backups of important data files that might contain infectable code; such files include word-processor documents, spreadsheets, slide presentations, pictures, sound files, and databases. Keep these backups on inexpensive media, such as CDs or DVDs so that you can keep old backups for a long time. In case you find an infection, you want to be able to start from a clean backupthat is, one taken before the infection.

- *Use virus detectors (often called virus scanners) regularly and update them daily.* Many of the available virus detectors can both detect and eliminate infection from viruses. Several scanners are better than one because one may detect the viruses that others miss. Because scanners search for virus signatures, they are constantly being revised as new viruses are discovered. New virus signature files or new versions of scanners are distributed frequently; often, you can request automatic downloads from the vendor's web site. Keep your detector's signature file up to date.

3.3.20 Truths and Misconceptions about Viruses

Because viruses often have a dramatic impact on the computer-using community, they are often highlighted in the press, particularly in the business section. However, there is much misinformation in circulation about viruses. Let us examine some of the popular claims about them.

- *Viruses can infect only Microsoft Windows systems. False.* Among students and office workers, PCs running Windows are popular computers, and there may be more people writing software (and viruses) for them than for any other kind of processor. Thus, the PC is most frequently the target when someone decides to write a virus. However, the principles of virus attachment and infection apply equally to other processors, including Macintosh computers, Unix and Linux workstations, and mainframe computers. Cell phones and PDAs are now also virus targets. In fact, no writeable stored-program computer is immune to possible virus attack. This situation means that all devices containing computer code, including automobiles, airplanes, microwave ovens, radios, televisions, voting machines, and radiation therapy machines have the potential for being infected by a virus.

- *Viruses can modify "hidden" or "read-only" files. True.* We may try to protect files by using two operating system mechanisms. First, we can make a file a hidden file so that a user or program listing all files on a storage device will not see the file's name. Second, we can apply a read-only protection to the file so that the user cannot change the file's contents. However, each of these protections is applied by software, and virus software can override the native software's protection. Moreover, software protection is layered, with the operating system providing the most elementary protection. If a secure operating system obtains control before a virus contaminator has executed, the operating system can prevent contamination as long as it blocks the attacks the virus will make.
- *Viruses can appear only in data files, or only in Word documents, or only in programs. False.* What are data? What is an executable file? The distinction between these two concepts is not always clear, because a data file can control how a program executes and even cause a program to execute. Sometimes a data file lists steps to be taken by the program that reads the data, and these steps can include executing a program. For example, some applications contain a configuration file whose data are exactly such steps. Similarly, word-processing document files may contain startup commands to execute when the document is opened; these startup commands can contain malicious code. Although, strictly speaking, a virus can activate and spread only when a program executes, in fact, data files are acted on by programs. Clever virus writers have been able to make data control files that cause programs to do many things, including pass along copies of the virus to other data files.
- *Viruses spread only on disks or only through e-mail. False.* File-sharing is often done as one user provides a copy of a file to another user by writing the file on a transportable disk. However, any means of electronic file transfer

will work. A file can be placed in a network's library or posted on a bulletin board. It can be attached to an e-mail message or made available for download from a web site. Any mechanism for sharing filesof programs, data, documents, and so forthcan be used to transfer a virus.

- *Viruses cannot remain in memory after a complete power off/power on reboot. True, but . . .* If a virus is resident in memory, the virus is lost when the memory loses power. That is, computer memory (RAM) is volatile, so all contents are deleted when power is lost.[2] However, viruses written to disk certainly can remain through a reboot cycle. Thus, you can receive a virus infection, the virus can be written to disk (or to network storage), you can turn the machine off and back on, and the virus can be reactivated during the reboot. Boot sector viruses gain control when a machine reboots (whether it is a hardware or software reboot), so a boot sector virus may remain through a reboot cycle because it activates immediately when a reboot has completed.
- *Viruses cannot infect hardware. True.* Viruses can infect only things they can modify; memory, executable files, and data are the primary targets. If hardware contains writeable storage (socalled firmware) that can be accessed under program control, that storage is subject to virus attack. There have been a few instances of firmware viruses. Because a virus can control hardware that is subject to program control, it may seem as if a hardware device has been infected by a virus, but it is really the software driving the hardware that has been infected. Viruses can also exercise hardware in any way a program can. Thus, for example, a virus could cause a disk to loop incessantly, moving to the innermost track then the outermost and back again to the innermost.
- *Viruses can be malevolent, benign, or benevolent. True.* Not all viruses are bad. For example, a virus might locate uninfected programs, compress them so that they occupy less memory, and insert a copy of a routine that

decompresses the program when its execution begins. At the same time, the virus is spreading the compression function to other programs. This virus could substantially reduce the amount of storage required for stored programs, possibly by up to 50 percent. However, the compression would be done at the request of the virus, not at the request, or even knowledge, of the program owner.

REFERENCES

1. Andreas M. Antonopoulos, *Mastering Bitcoin: Unlocking Digital Cryptocurrencies* (Dec 2014, O›Reilly; 2/e 2017). First edition free online.
2. Applied cryptography and network security: Menezes, van Oorschot and Vanstone, *Handbook of Applied Cryptography* (1996, CRC Press; 2001 with corrections), free online for personal use.
3. Bruce Schneier. *Secrets and Lies: Digital Security in a Networked World* (2000, Wiley).
4. Dieter Gollmann, *Computer Security, 3/e* (2011, Wiley).
5. Goodrich and Tamassia, *Introduction to Computer Security* (2010, Addison-Wesley).
6. Kaufman, Perlman and Speciner, *Network Security: Private Communications in a Public World, second edition* (2003, Prentice Hall).
7. Keith M. Martin, *Everyday Cryptography* (2017, 2/e; Oxford University Press).
8. Mark Stamp, *Information Security: Principles and Practice, 2/e* (2011, Wiley).
9. Matt Bishop, *Computer Security: Art and Science* (2002, Addison-Wesley). Shorter version which «omits much of the mathematical formalism»: *Introduction to Computer Security* (2005, Addison-Wesley).
10. Operating system security: Trent Jaeger, Operating System Security (2008, Morgan and Claypool).
11. Paul van Oorschot, Computer Security and the Internet: Tools and Jewels (2020, Springer). Personal use copy freely available on author's web site.
12. Pfleeger and Pfleeger, *Security in Computing, 4/e* (2007, Prentice Hall).
13. Saltzer and Kaashoek, *Principles of Computer System Design* (2009, Morgan Kaufmann). Free online chapters include (pdf) Ch.11: Information Security.

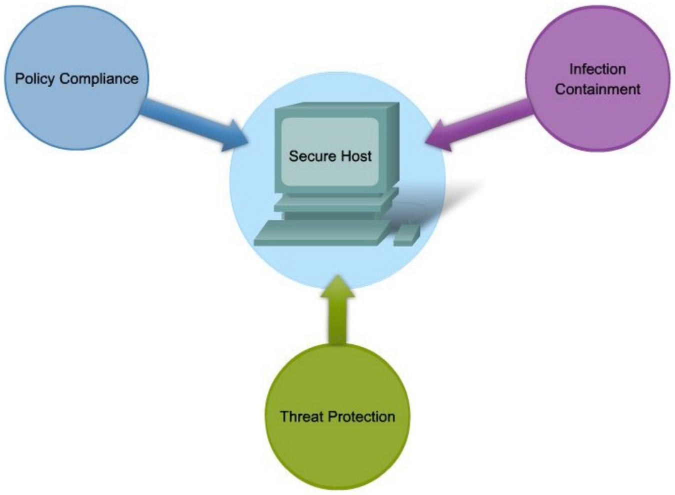
14. Security in the real-life systems (including anecdotes): Ross Anderson, *Security Engineering: A Guide to Building Dependable Distributed Systems, 2/e* (2008, Wiley). The first edition (2001) is available free online.
15. Smith and Marchesini, *The Craft of System Security* (2007, Addison-Wesley).
16. Smith, *Elementary Information Security* (2011, Jones & Bartlett Learning).
17. Stallings and Brown, *Computer Security: Principles and Practice, 3/e* (2014, Prentice Hall).
18. Wenliang Du, *Computer Security: A Hands-on Approach* (2017, self-published). Updated May 2019.
19. William Stallings, *Cryptography and Network Security: Principles and Practice, 5/e* (2010, Prentice Hall). Relative to this book›s 4th edition, the network security components and an extra chapter on SNMP are also packaged as Stallings› *Network Security Essentials: Applications and Standards, 3/e* (2007, Prentice Hall).

CHAPTER 4

PROTECTION AND SECURITY IN OPERATING SYSTEM

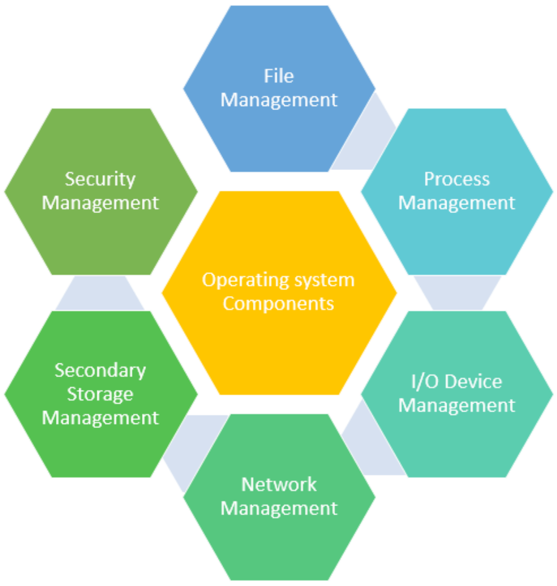
INTRODUCTION

Protection and security requires that computer resources such as CPU, software's, memory etc. are protected. This extends to the operating system as well as the data in the system. This can be done by ensuring integrity, confidentiality and availability in the operating system. The system must be protect against unauthorized access, viruses, worms etc. During the past few years we have been observing a significant increase in the number of people who use computers to perform tasks where security is important. Typical such applications, that are of interest to general public and can be expected to be used on home computers, include Internet banking, e-government applications, electronic signature creation and verification applications. We will use the term security-critical application to denote such applications. Organizations use information systems to store and process confidential business data and personal data. Unauthorized access to information stored or processed by all of the mentioned applications (and many others) can often cause a substantial loss to the affected person or organization. It is usually the user's responsibility to protect the sensitive data. However, common users are not information security experts and can only follow some guidelines given to them. Even that is usually possible only if the guidelines are simple enough. While a larger organization can dedicate some computers to security-critical applications and protect them against unauthorized access, modification or software installation, it can hardly be expected in a home environment.



4.1 SECURITY IN COMMON OPERATING SYSTEMS

In the thesis, we have analyzed two groups of currently common desktop operating systems for personal computers. The first group consists of Microsoft Windows 2000/ XP Professional/Vista, and the second one consists of the Linux operating systems. We have used the security functional requirements classes specified in the international standard ISO/IEC 15408 to organize the analysis of the security functions of the considered operating systems. Operating systems in both groups use hardware resources to protect themselves against manipulation by processes, and to protect the processes against each other. They implement security functions for security audit records generation and review, user data protection, user identification and authentication, security management, limited trusted path and trusted channels, and other security requirements. As far as access control is concerned, operating systems in both groups implement discretionary access control mechanisms, and some of them (Linux, Windows Vista) optionally provide different sorts of (partial) mandatory access control mechanisms.



4.1.1 Security Problems of Common Operating Systems

We have identified several common security problems that are not addressed sufficiently by the considered operating systems if potentially malicious applications are to be used simultaneusly with security-critical ones.

Abuse of privileges by an administrator

Linux operating systems not not apply discretionary access control to the processes running on behalf of the user with ID 0 (root). A system administrator can, by default, run any program on behalf of this user. The administrator can, therefore, manipulate with data of any user, and can also modify parts of the operating system and applications, e.g. to capture sensitive input such as passwords, PINs. The SELinux security module can be used with a carefully specified security policy and with a suitable separation of duty among more people to lower the risks of abuse of privileges by an administrator. In Windows operating systems, the users can limit the access of administrators to their data. However, this feature is insufficient because it can be circumvented by abusing the privileges for backup operators. The protection of confidentiality of the data can be improved using encryption, but the administrators are still able to overcome the encryption by designating a special user able to decrypt all files.

To sum up, the system administrators (and other users with special privileges) effectively control the entire system. Their trustworthiness is therefore very important for the overall security of the operating systems.

Too many processes with high privileges

Another common problem is that many processes providing various services run on behalf of privileged users (e.g. root on Linux, or special system user on Windows). Many of these processes do need the privileges, but there are also many of them that need only a small subset of the privileges. Programs contain various flaws that may allow attackers to execute arbitrary code with the privileges of the exploited process. When such flawed program runs within a process with an administrator's privileges, it may allow an unidentified attacker to abuse the administrator's privileges.

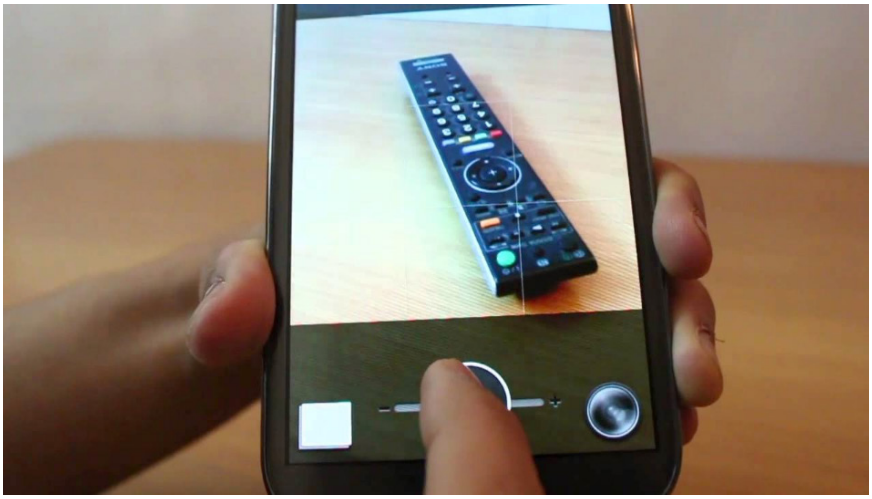
Abuse of privileges of an ordinary user

A currently very common problem is the abuse of privileges of an ordinary user by malicious applications. Many users do not realize that the programs they have obtained from untrusted sources (such as the huge number of web pages) can, on top of (or instead of) the declared activities, perform any operations, including malicious ones, while abusing the user's privileges

Another problem is that applications often contain flaws that allow a code embedded in a specially crafted document to be executed as a result of the application's processing of the document. When the user uses such application to process a document obtained from an untrusted source, the risk is similar to that of directly running a program from an untrusted source. This problem becomes even more significant when we consider common web browsers or e-mail clients that, to be more userfriendly, automatically run various applications to process documents in web pages or e-mail attachments, often without asking the user whether or not to do so. This is probably the most common way of spreading computer viruses and Trojans nowadays. The lack of awareness of users also helps the attackers to use this way to abuse the users' privileges. The users often trust unauthenticated information. A typical example is the e-mail address of the sender of an e-mail message – it is trivial to forge while many users, seeing a known e-mail address, believe that the message must have been received from a person they trust. The message often comes from a virus, Trojan or another malware that attempts to exploit a security flaw in an application the recipient is expected to use to process the message.

Direct manipulation with the hardware

Direct manipulation with the hardware while it is not controlled by the operating system also presents a nonnegligible possibility of breaking security of the considered operating systems. If an attacker can physically manipulate with a computer, he or she can modify the contents of its hard disks, and thus modify any data, applications or parts of the operating system. Even though the operating systems support digital signature verification for system components or for software packages during installation, the signature verification depends on data stored on the disk that may be subject to unauthorized modification via direct hardware manipulation. The system can also be modified not to perform the verification at all. This problem cannot be, in general, solved at the operating system layer without booting the system from a medium that can be trusted not to have been modified in an unauthorized way (e.g. a physically protected read-only medium or a file the authenticity of which is verified before using it by the computer's firmware and hardware).



4.1.2 Existing Partial Solutions

Some of the problems mentioned above can be partially solved using the features provided by the considered operating systems. The problem with processes with too high privileges can, in some cases, be solved by minimizing the privileges to the minimal required set. It is, however, not always sufficient or possible.

The problem with abusing the user's privileges can be sometimes solved by increasing the awareness of the users, and by strict separation of the security-critical activities from the risky ones (such as web browsing or e-mail processing). When a user uses different accounts for different purposes, he or she can set the access control lists in the way that the programs running with the user's identity for the risky operations cannot interfere with the sensitive data that are to be accessed only by processes running with the other user's identity.

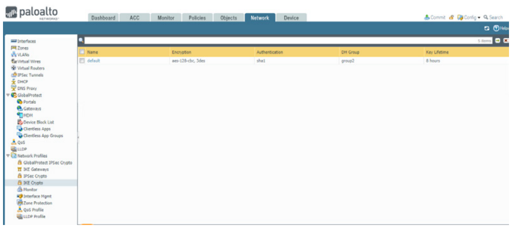
There have been many projects for the Linux operating systems implementing various security mechanisms (most often a sort of mandatory access control) to effectively solve some of the problems. They finally resulted in the inclusion of Linux Security Module (LSM) framework in the Linux kernel, and in the acceptance of SELinux module as a standard part of the kernel. SELinux can be used to protect confidentiality using BellLaPadula based multi-level security policy, and using any policy specified in terms of domain and type enforcement mechanism. The latter is nowadays used by several Linux distributions (e.g. RedHat, Debian) to limit the impact of exploiting flaws in applications on Linux servers. Attempts to use a strict SELinux policy on desktop systems have failed due to too diverse requirements of desktop systems [11, 2], and they have resulted in usage of so called targeted policy that constrains many of the system services and server processes but leaves the user-started processes unconfined in a single domain. This way the user's data are not protected against malicious code started by the user, either directly or indirectly via a flawed application and a malicious document.

Windows operating systems also brought several interesting attempts to solve the problems. Windows Vista introduces two new security features that are worth mentioning. One of them is called Mandatory integrity control (MIC). Filesystem objects and processes are assigned integrity levels, and a process can only modify objects with the same or lower integrity level than the process's integrity level. In fact, one half of the standard Biba model rules are used. It can also be configured to enforce one half of the standard Bell-LaPadula model – a process can only read from objects with the same or lower level. Because MIC only implements a half of the standard rules, it lacks the provable security properties of Bell-LaPadula and Biba models. Its primary use was to protect the user's data and programs against a malicious code executed by a flawed web browser. It does not, however, prevent other processes from reading (and acting upon) malicious data that have been downloaded from untrusted sources.

Another new security feature introduced in Windows Vista is User Account Control (UAC). It deals with the, well known and unfortunate, fact that many users on desktop systems use accounts with administrator privileges (either to overcome problems with some software or just out of a sort of laziness) for their normal computer usage. This leads to a situation that even a flawed web browser or e-mail client can perform operations that are restricted to administrators, and it effectively makes the access control mechanisms ineffective. UAC works by disabling some of the special privileges normally given to administrators and prompting the user for a permission to grant these privileges to the process that attempts to perform an operation that requires the privileges. This way, the processes cannot perform the privileged operations without the user knowing about it. They can still, however, perform any operations that do not require the privileges dedicated to administrators.

4.1.3 Existing Protection Profiles

The security requirements for IT products are often specified in the form of a protection profile (PP) according to the international standard ISO/IEC 15408. In the thesis, we have analyzed several existing protection profiles for operating systems. In the PP registry, several protection profiles for operating systems are registered. Two of them, Controlled access protection profile (CAPP) and Labeled security protection profile (LSPP), are for general use. A few others are specifically tailored to the needs of the Department of Defense of the U.S.A. for the classified information processing, but they are not suitable for general home/office use.



Name	Encryption	Authentication	DH Group	Key Lifetime
default	aes-128-cbc, 3des	sha1	group2	8 hours

The security objectives and the functional security requirements of the CAPP do not cover protection against abuse of an administrator's privileges – a trustworthy administrator is assumed. No protection against malicious code executing with a user's privileges is provided because all access control decisions are based on the user's identity regardless of the program being executed. The user has thus no way of preventing a malicious program from accessing any data accessible to the user. Even if the user attempts to restrict his/her own access rights to an object, the malicious program running on the user's behalf can grant the access rights to the user (or to any other user).

The LSPP improves the protection of confidentiality of data in the environments where information is/may be classified in the Bell-LaPadula way. When a process (subject) is executed at a security level, it cannot read from objects with a higher security level (thus containing more sensitive information), and it cannot write to objects with a lower security level. For example, if communication objects connecting to untrusted external systems are classified at the lowest level, a malicious program running at a higher level cannot send sensitive information to the external systems, and a malicious process operating at the lowest level (and thus able to communicate with the external system) can read no sensitive information (contained in an object with a higher level).

The LSPP, however, contains no improvements regarding integrity protection. A malicious program running at the lowest security level can still cause damage to valuable data stored on the system. As we will show later, the integrity protection can be of equal, or sometimes even higher importance that the confidentiality protection. The LSPP, just like the CAPP, does not cover the protection against abuse of an administrator's privileges – it assumes a trustworthy administrator.

4.2 GOALS, OBJECTIVES AND METHODS

The problems mentioned in the previous section have given rise to the primary goals of the thesis:

- designing a suitable security model for an operating system supporting secure use of security-critical applications alongside untrusted and potentially malicious applications,
- creating a protection profile, compliant to the ISO/IEC

15408 standard, for a general purpose operating system supporting such use, utilizing the security model.

In order to achieve the goals, we have set the following objectives to fulfil in the thesis:

- Identify and categorize typical applications and identify the protection requirements.
- Specify the data classification scheme.
- Specify the security model.
- Formulate and prove security properties of the model.
- Create the protection profile.

In order to specify the security model we have used a simplified model of an operating system consisting of active entities – subjects (processes) performing operations on passive entities – objects (files, directories, communication objects, processes, . . .). We started with read, write, create and delete operations, and we extended the set of operations later to cover a more realistic operating system. We have modelled access control and information flow control using logical functions operating on subjects and objects and yielding true or false depending on whether the operation is permitted or not.

The structure of a protection profile is specified by the international standard ISO/IEC 15408. We have used the standard to write our protection profile.

4.3 THREATS TO PROTECTION AND SECURITY

A threat is a program that is malicious in nature and leads to harmful effects for the system. Some of the common threats that occur in a system are –

4.3.1 Virus

Viruses are generally small snippets of code embedded in a system. They are very dangerous and can corrupt files, destroy data, crash systems etc. They can also spread further by replicating themselves as required.

4.3.2 Trojan Horse

A trojan horse can secretly access the login details of a system. Then a malicious user can use these to enter the system as a harmless being and wreak havoc.

4.3.3 Trap Door

A trap door is a security breach that may be present in a system without the knowledge of the users. It can be exploited to harm the data or files in a system by malicious people.

4.3.4 Worm

A worm can destroy a system by using its resources to extreme levels. It can generate multiple copies which claim all the resources and don't allow any other processes to access them. A worm can shut down a whole network in this way.

4.3.5 Denial of Service

These type of attacks do not allow the legitimate users to access a system. It overwhelms the system with requests so it is overwhelmed and cannot work properly for other user.

4.4 PROTECTION AND SECURITY METHODS

The different methods that may provide protect and security for different computer systems are –

4.4.1 Authentication

This deals with identifying each user in the system and making sure they are who they claim to be. The operating system makes sure that all the users are authenticated before they access the system. The different ways to make sure that the users are authentic are:

- Username/ Password

Each user has a distinct username and password combination and they need to enter it correctly before they can access the system.

- User Key/ User Card

The users need to punch a card into the card slot or use they individual key on a keypad to access the system.

- User Attribute Identification

Different user attribute identifications that can be used are fingerprint, eye retina etc. These are unique for each user and are compared with the existing samples in the database. The user can only access the system if there is a match.

4.4.2 One Time Password

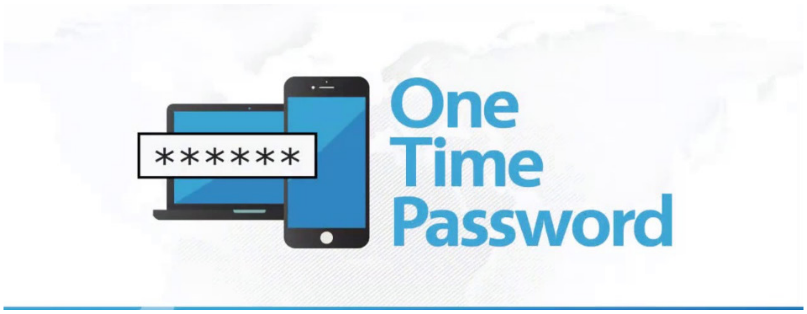
These passwords provide a lot of security for authentication purposes. A onetime password can be generated exclusively for a login every time a user wants to enter the system. It cannot be used more than once. The various ways a onetime password can be implemented are −

- Random Numbers

The system can ask for numbers that correspond to alphabets that are pre-arranged. This combination can be changed each time a login is required.

- Secret Key

A hardware device can create a secret key related to the user id for login. This key can change each time.



4.4.3 Classes of Applications Considered

We have considered the typical applications used on personal computers in the home and small office environment. We have identified several classes according to the security requirements for their data.

Malicious applications

A special class of applications is the class of malicious applications. These are applications that have been intentionally programmed to perform malicious activities. The typical examples are computer viruses, worms, Trojan horses and other kinds of so called malware. They can be downloaded from the Internet by the user, received as an attachment of an e-mail message, or a vulnerable application may be turned into a malicious one by processing malicious data. The user is usually unaware of the fact that a particular application is malicious. It has to be assumed that the malicious applications do anything not prevented by the operating system or the environment of the computer (e.g. a network firewall).

Local applications

The class of local applications contains the applications that are used to process data stored in a local file system. These applications generally do not need network access to perform their tasks. The typical examples are text processors, spreadsheets, presentation software, graphic editors, ... Local applications are used to process data with varying requirements regarding the confidentiality and integrity protection. If they process malicious data, they may become malicious due to programming errors.

Sensitive web access

Web browsers are often used to access remote services that process data requiring confidentiality and/or integrity protection. A typical example is an Internet-banking system. It provides access to financial information; it allows the user to submit transaction orders to the bank, etc. It also processes authentication data (e.g. passwords). All such data may be considered confidential by the user, and therefore, are to be adequately protected. The confidentiality and the integrity of the data during their transmission is usually protected by means of cryptography. Cryptography is usually also used to provide authentication of the remote system. But the data is also to be protected while stored in the memory or in a file on the local computer. Consider an instance of a web browser used for general Internet access. It may have processed some malicious data, a and therefore, it may have become a malicious application exporting everything to an attacker. If the instance of the web browser is later used to access an Internet banking system, all the confidential information may leak.

Digital signature creation

A digital signature creation application needs access to the private key. The private key is a very sensitive piece of information the confidentiality of which has to be protected. The integrity of the private key has to be protected as well because its modification can lead not only to the loss of ability to create correct digital signatures, but also to the leak of information that is sufficient to compute the corresponding private key in certain cases.

Digital signature verification

A digital signature verification application needs access to the public key. The public key requires no confidentiality protection, but it does require integrity protection. If attackers were able to modify the public key used to verify a digital signature, they would be able to create a digitally signed document that would pass the signature verification process.

Data encryption

A data encryption application using asymmetric cryptography needs access to the public key of the receiver of the data. As mentioned above, the public key requires no confidentiality protection, but it does require integrity protection. In the case of encryption, if the public key were modified by an attacker, the attacker would be able to decrypt the encrypted data instead of the intended receiver. The encrypted output of a data encryption application may be transmitted via communication channels that do not provide confidentiality protection even if the confidentiality of the original data is to be protected.

4.4.4 Security Model

Objects

It can be seen in the examples in the previous section that we have to deal with data with varying requirements regarding the confidentiality and integrity protection. As far as the confidentiality is concerned, we can classify the data into three basic categories:

- public data,
- normal data – C-normal,
- data that are sensitive regarding their confidentiality – C-sensitive.

The public data require no confidentiality protection. They may be freely transmitted via communication channels and/or to remote systems that provide no confidentiality protection. An example of public data is the data downloaded from public Internet. The normal data are to be protected by means of discretionary access control against unauthorized reading by other users than the owner of the data. The C-sensitive data are the data that their owner (a user) wishes to remain unreadable to the others regardless of the software the user uses, and even if the users makes some mistakes (such as setting wrong access rights for discretionary access control). Examples of C-sensitive data are private and secret keys, passwords for Internet banking, etc. As far as the integrity (or trustworthiness) of data is concerned, we can also classify the data into three basic categories:

- potentially malicious data,
- normal data – I-normal,
- data that are sensitive regarding their integrity – I-sensitive.

The requirements of the integrity protection of data is tightly coupled to the trustworthiness of the data. The trustworthiness of data can be thought of as a metric of how reliable the data are. If some data can be modified by anyone, they cannot be trusted not to contain wrong or malicious information. If some data are to be relied on, their integrity has to be protected.

The potentially malicious data require no integrity protection, and can neither be trusted to contain valid information, nor can be trusted not to contain malicious content. The normal data is to be protected by means of discretionary access control against unauthorized modification by other users that the owner of the data.

The I-sensitive data are the data that their owner wishes to remain unmodified by the others regardless of the software the user uses, and even if the users makes some mistakes. The I-sensitive data are to be modifiable only under special conditions upon their owner's request. A special category of I-sensitive data is the category of the shared system files such as the programs, the libraries, various system-wide configuration files, the user database . . . Some of these files may be modifiable by the designated system administrator, some of them should be even more restricted.

The number of the confidentiality and integrity categories may be higher in real systems.

A common approach to ensuring the confidentiality and/or the integrity of information in systems that deal with data classified into several confidentiality/integrity levels, is to define an information flow policy, and then to enforce the policy. In order to enforce an information flow policy, subjects are divided into two categories – trusted and untrusted. A trusted subject is a subject that is trusted to enforce the information flow policy (with exceptions) by itself; an untrusted subject is a subject that is not trusted to enforce the policy by itself, and therefore the policy has to be enforced on the subject's operations by the system.

A typical information flow policy protecting confidentiality (e.g. one based on Bell-LaPadula model) states that a subject operating at a confidentiality level CS may only read from an object with a confidentiality level COr if $CS \geq COr$, and may only write to an object with a confidentiality level COw if $CS \leq COw$. If a subject is to be able to read from a more confidential object, and to write to a less confidential object, it has to be a trusted subject.

A typical information flow policy protecting integrity (e.g. one based on Biba model) states that a subject operating at an integrity level IS may only read from an object with an integrity level IOr if $IS \leq IOr$, and may only write to an object with an integrity level IOw if $IS \geq IOw$. Only a trusted subject can read from an object with a lower integrity level, and write to an object with a higher integrity level.

The problem with the division of subjects into the two categories is that it would lead to the need of too many trusted subjects in the home and office environment.

We will divide subjects into three categories:

- untrusted subjects,
- partially trusted subjects, and
- trusted subjects.

An untrusted subject is a subject that is not trusted to enforce the information flow policy. It is assumed to perform any operations on any objects unless it is prevented from doing so by the operating system. A trusted subject is a subject that is trusted to enforce the information flow policy by itself. A trusted subject may be used to perform tasks than require violation of the policy under conditions that are verified by the trusted subject. A trusted subject can, therefore, be used to implement an exception to the policy. A partially trusted subject is a subject that is trusted to enforce the information flow policy regarding a specific set of objects, but not trusted to enforce the information flow policy regarding any other objects. In other words, a trusted subject is

- trusted not to transfer information from a defined set of objects (designated inputs) at a higher confidentiality level to a defined set of objects (designated outputs) at a lower confidentiality level in a way other than the intended one, and
- trusted not to transfer information from a defined set of objects (designated inputs) at a lower integrity level to a defined set of objects (designated outputs) at a higher integrity level in a way other than the intended one, but
- not trusted not to transfer information between any other objects.

The sets of designated inputs and outputs regarding confidentiality are distinct from the sets regarding integrity. Any of the sets may be empty. A partially trusted subject, like a trusted one, can be used to implement an exception to the policy, because it can violate the policy (and it is trusted to do it only in an intended way). The most important difference between trusted and partially trusted subjects is in the level of trust. While trusted subjects are completely trusted to behave correctly, partially trusted subjects are only trusted not to abuse the possibility of the information flow violating the policy between a defined set of input objects and a defined set of output objects.

Information Flow Policy

Having specified the objects and the subjects and their classification, we can formulate the information flow policy to protect the confidentiality and the integrity of the information stored in, or transferred via the objects. We will first specify the policy objectives in an informal way, and then we will define the policy formally.

In accordance with the classification of objects, the information flow policy has the following objectives:

- Prevent reading of C-sensitive objects by subjects of other users than the owner of the object.
- Prevent modification of I-sensitive objects by subjects of other users than the owner of the object.
- Prevent information passing from objects with a higher confidentiality level3 to objects with a lower confidentiality level by untrusted subjects with the exception stated below.
- Allow the user to explicitly allow a subject to read a C-normal object on per request basis. The user's approval in such case must be obtained via a mechanism independent on the subject. The idea of this objective is to allow the user to perform operations such as submitting a C-normal document to a remote system, that is not trusted to process C-normal data in general and is considered a public object with respect to our classification scheme, without the need to reclassify the document first (a therefore to expose its content to any subject). Because this approach is very prone to the user's mistakes, it should be limited to C-normal objects and not applicable to C-sensitive objects.
- Prevent information passing from objects with a lower integrity level4 to objects with a higher integrity level by untrusted subjects.
- Allow the user to specify the maximal integrity level for each subject and prevent the subject from writing to objects with a higher integrity level. The idea of this objective is to prevent modification of objects with a high integrity level unless required by the user.
- Allow the user to define four sets of special input and output objects (two sets for confidentiality protection

and two sets for integrity protection) and two special confidentiality levels (for reading and writing respectively) and two special integrity levels associated with the sets for each partially trusted subject, and apply the same rules to partially trusted subjects with the following exceptions:

(a) Allow a partially trusted subject to transfer in formation from an object O_{in} with a confidentiality level c_{in} to an object O_{out} with a confidentiality level $c_{out} < c_{in}$ if the object O_{in} is in the input set for confidentiality protection, the object O_{out} is in the output set for confidentiality protection, c_{in} is at most the special confidentiality level for reading, and c_{out} is at least the special confidentiality level for writing

(b) Allow a partially trusted subject to transfer information from an object O_{in} with an integrity level i_{in} to an object O_{out} with an integrity level $i_{out} > i_{in}$ if the object O_{in} is in the input set for integrity protection, the object O_{out} is in the output set for integrity protection, i_{in} is at least the special integrity level for reading, and i_{out} is at most the special integrity level for writing.

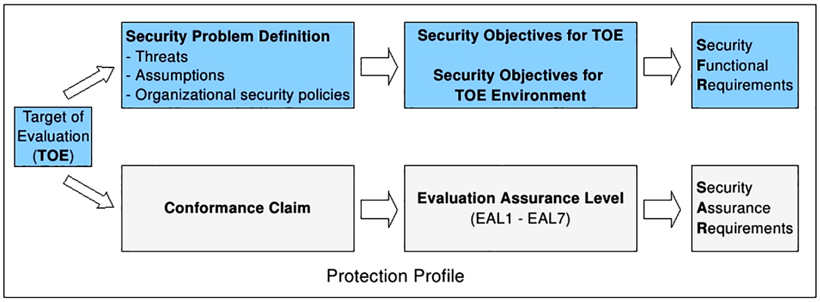
4.5 PROTECTION PROFILE OVERVIEW

To fulfil one of the goals of the thesis, we have created a protection profile, compliant to the ISO/IEC 15408 standard, for a general purpose operating system suitable for using security-critical applications alongside potentially malicious ones. Our protection profile has been designed to support our new security model in addition to the common discretionary access control policy supported by the common existing operating systems. Our protection profile specifies several assumptions about the security environment of the operating system, namely about the hardware and its physical surroundings:

- the hardware supports access control for memory regions and peripheral devices,

- the processor(s) restrict the use of privileged instructions to the operating system,
- the hardware is physically protected against modification,
- and the internal communication paths (such as system buses) are protected against unauthorized monitoring and tampering with.

Protection Profile Overview


Target of Evaluation (TOE)
Security Problem Definition
- Threats
- Assumptions
- Organizational security policies
Security Objectives for TOE
Security Objectives for TOE Environment
Security Functional Requirements
Conformance Claim
Evaluation Assurance Level (EAL1 - EAL7)
Security Assurance Requirements
Protection Profile

The hardware assumptions are consistent with the currently common PC hardware. The other two assumptions are to be upheld by the environment, most commonly using physical security mechanisms. It is also assumed that an operating system compliant with the protection profile is constructed as a modification of an existing operating system. On one hand this limits the choice of security functional requirements and, most notably, security assurance requirements; on the other hand it allows a compliant operating system to be used in practice by utilizing existing applications and hardware support.

The objectives of the protection profile can be summarized as follows:

- restricting access to identified, authenticated, and authorized users only,
- enforcing a discretionary access control policy based on user identities,

- enforcing confidentiality and integrity protection according to our new security model,
- cryptographic confidentiality and integrity protection of stored data,
- auditing of security related events,
- residual information protection,
- system management restricted to authorized system administrators,
- limiting the ability of an authorized administrator to abuse his/her privileges,
- and the feasibility of construction of a compliant operating system by modifying an existing operating system while preserving its compatibility with existing applications.

The security functional requirements specified in the protection profile can be summarized as follows:

- Security Audit (FAU) – the protection profile requires that the operating system is able to generate audit records for the listed events, that the authorized audit administrators are able to review, sort and search the audit records, and configure the set of audited events, and that the audit records are protected against unauthorized modification and deletion.
- User Data Protection (FDP) – the protection profile requires that the operating system enforces the discretionary access control policy and our information flow policy based on the user identity and group membership and other security attributes associated with subjects and objects. It also requires removal of any residual information from resources upon their allocation. Support for cryptographic protection of integrity and confidentiality of stored data is also required.
- Identification and Authentication (FIA) – the protection profile specifies security attributes to be associated with each user and the user's processes. It allows minimal

strength requirements for authentication data to be specified. It requires a successful identification and authentication of an authorized user before performing any other action on the user's behalf. It requires re-authentication before performing several specific security-critical actions.

- Security Managements (FMT) – the protection profile requires the management of the security attributes and data to be restricted to the authorized users acting in specific roles, and in compliance with the information flow and access control policies. The protection profiles defines several security roles and splits the privileges and responsibilities among them in order to minimize the ability of a single individual (e.g. a system administrator) to abuse his/her privileges. It also supports authorization of securitycritical operations by multiple administrators.
  - Protection of TOE Security Functions (FPT) – the protection profile requires the operating system to protect itself against tampering by unauthorized subjects, as well as to separate the security domains of subjects. The protection profile requires that the security functions must always be invoked before other operations within the scope of control can be performed, i.e. it may not be possible to bypass the security functions. The protection profile also requires the operating system to perform tests during start-up to verify the crucial assumptions about the hardware.
  - Resource Utilization (FRU) – the protection profile requires the operating system to enforce the maximum quotas on disk space used by an individual user.
  - TOE Access (FTA) – the protection profile requires the operating system to support session locking on the user's request as well as automatically after a specified period of inactivity. It also requires

the operating system to maintain and display information about successful and unsuccessful session establishments.

- Trusted Path/Channel (FTP) – the protection profile requires the operating system to provide a trusted path between the user and itself for specified securitycritical interactions including (re-) authentication, authentication data management, special security attributes management,
- Cryptographic Support (FCS) – the protection profile specifies requirements for cryptographic key generation and destructions, and for cryptographic operations.

The security assurance requirements specified in the protection profile are based on EAL3 augmented with the addition of requirements for detection of modification during delivery of the operating system.

Discussion

In the thesis, we discuss the benefits of our model. We present several usage examples based on the considered classes of applications. These examples can also be found in [13]. We also compare our new security model and our protection profile to several other projects and to the mentioned existing protection profiles.

4.5.1 Our New Model vs. MIC in Windows Vista

MIC can be used to prevent a potentially malicious application running at a low integrity level from modifying data at a higher integrity level. Unlike our new model, however, it does not prevent an untrusted application running at a higher integrity level from reading (potentially malicious) data at a lower level. If the application contains a flaw that can be exploited by processing malicious data, the user can, e.g. by an accident, use it to read and process malicious data stored at the low integrity level, and thus turn the application to a malicious one that can spoil data at its (higher) integrity level.

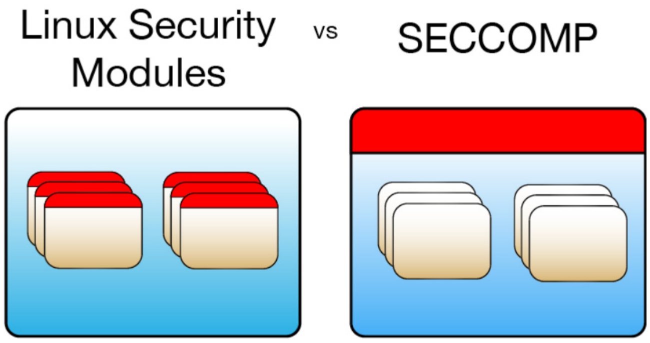
MIC also allows confidentiality protection to be turned on. Unlike in our new model, the confidentiality and integrity levels in MIC are not independent, the same level is used for both. It can be used to prevent a potentially malicious application running at a low level from reading (and also from modifying) data at a higher level. Unlike our new model, it does not prevent an application running at a higher level (and thus capable of reading data at that level) from writing to objects at a lower level. Any application is thus able to export any data that it can read to external untrusted systems, or to store them to a low-level file.

MIC is a useful feature that allows a careful user to use a web browser or to test a potentially malicious application without the risk that they will modify (and optionally also read) any data classified at a higher level. The user must be careful enough, however, not to open any files classified at the low level in another application running at a higher level unless the application is trusted not to misbehave upon reading the data. The integrity and confidentiality protection provided by our new security model is definitely stronger than that of MIC, and it has provable security properties similar to those of Bell-LaPadula and Biba models.

4.5.2 Our New Model vs. SELinux

SELinux, by implementing a flavour of domain and type enforcement, provides a very flexible security mechanism that can be used to enforce a wide range of security policies. We show in the thesis that it can be used to implement our new model as well. The commonly used SELinux policies[11, 2] define unique types and domains for many typical system services, server applications, and their data, and strictly restrict the set of operations the processes running within these domains are allowed to perform. The strict version of the policy is suitable for servers but causes problems on desktop systems because the restrictions are too strict to be accepted by users. The targeted version of the policy, that restricts many system services but leaves the applications started by the user in a single, unconfined domain, is suitable for desktops. It prevents a flawed system service program from accessing data it does not have to be able to access.

It does not, however, prevent user-started applications from accessing the user's data, perhaps except for some specially designated sensitive data (such as private keys) accessible only to certain applications. The targeted policy could be combined with our new model to provide combined benefits of both. The targeted policy provides more rigorous restrictions for specific services while our new model can be used to protect the ordinary users' data.



4.5.3 Our Protection Profile vs. CAPP and LSPP

Our protection profile, the CAPP, and the LSPP are designed for general purpose operating systems. We will now discuss the benefits of our protection profile when compared to those two. All three protection profiles have some common objectives, such as restricting access to authorized users only, audit, discretionary access control policy enforcement, residual information protection and restricting system management to authorized administrators only. They also use similar security functional requirements to fulfil these objectives. We will, therefore, concentrate on the objectives that make the protection profiles different.

There are three new, important objectives in our protection profile when compared to the CAPP or the LSPP:

- enforcing confidentiality and integrity protection according to our new security model,
- cryptographic confidentiality and integrity protection of stored data,
- and limiting the ability of an authorized administrator to abuse his/her privileges.

The first is intended to address the problem of abusing the privileges of a user (including an administrator) by malicious code (whether executing as a standalone malicious applications, or as a part of a flawed application). Our new model has provable security properties that ensure that no untrusted applications can cause information to flow from objects at a higher confidentiality and/or lower integrity level to objects at a lower confidentiality and/or higher integrity level. The user can, thus, run a malicious application as an untrusted subject without the risk that it could cause an information leak or spoiling. Applications that need to be able to override this restriction can often be run as partially trusted subjects where the amount of trust can be very limited. We expect this approach to minimize the risk resulting from flaws in such applications.

The second new objective addresses the problem of direct manipulation with storage devices. Data encryption provides for confidentiality protection of the stored data against manipulation by means that are not under control of the operating system. Cryptographic integrity protection can prevent unnoticed unauthorized modification of the stored data including the operating system itself. Our protection profile is missing the assumption of a trustworthy administrator (this assumption is present in both the CAPP and the LSPP). The third objective addresses the problem of privilege abuse by an administrator. This is not addressed by the other two protection profiles. In our protection profile, it is addressed by separating the privileges and responsibilities for various aspects of system management (such as general system management, security management, audit management), and by defining distinct security roles for such activities. When the roles are assigned to different individuals, no single person can cause an unnoticed security policy violation. For environments, where this problem is a major concern, our protection profile supports multiple independent authorizations for critical operations. When compared to the LSPP's Bell-LaPadula style confidentiality protection, our protection profile addresses the integrity protection as well as the confidentiality protection. The concept of the partially trusted subjects, that we have introduced, is also a convenient way of minimizing the amount of trust given to applications that have to be able to violate the rules for untrusted subjects.

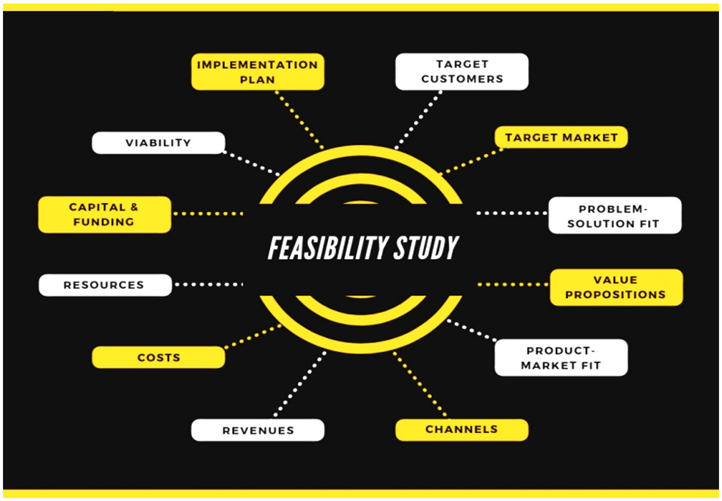
Benefits Summary

The key benefits of our new security model are summarized in the table 1. While we have based our new model on the ideas of Bell-LaPadula and Biba models, we have introduced significant improvements. Unlike others (e.g. MIC), we use independent confidentiality and integrity levels, and, most notably, we have introduced partially trusted subjects. The introduction of partially trusted subjects allows us to achieve strong security properties with minimal trust in correct behaviour of the partially trusted subjects.

Our protection profile, unlike the well-known and often used protection profiles addresses the issues of privilege abuse by a system administrator, includes cryptographic protection against direct storage media manipulation, and provides for both confidentiality and integrity protection according to our new security model.

Feasibility of Implementation

The thesis includes a feasibility study to show that it is feasible to modify a real, existing operating system to comply with our protection profile. We have chosen Linux operating systems as the base. We compare the security functions of Linux operating systems with the security functional requirements of our protection profile, and identify the missing features. We suggest two ways of implementing our new security model – the major missing feature – in Linux. In one approach we suggest using Linux Security Module framework present in the Linux kernel. In the other one we show how a SELinux policy can be created to enforce the rules of our new security model.



The goals of our thesis were to design a suitable security model and to create a protection profile for a general purpose operating system for home and office environment suitable to support the use of security-critical applications alongside untrusted and potentially malicious applications. We have presented several typical examples of applications used in the target environment, analyzed the security requirements and properties of the data involved, and designed a security model with a formally defined information flow policy to protect the confidentiality and the integrity of the data. In the thesis, we have formally proved important security properties of the model. We have created a new protection profile utilizing our new security model. We have discussed the benefits of our new security model and protection profile for security. We have compared them to other projects and protection profiles. Finally, we have discussed possible ways to modify Linux operating system to comply with our protection profile. It seems to be feasible to make the needed modifications in a near future. Having said that, we believe we have managed to fulfil the goals.

Table 1: Summary of the benefits of our new security model

	our new model	Bell-LaPadula	Biba	MIC
confidentiality protection	✓	✓		✓
integrity protection	✓		✓	✓
prevents untrusted subjects from causing information leak/spoiling	✓	✓	✓	
prevents partially trusted subjects from causing arbitrary information leak/spoiling	✓			

We can see the following tasks that might follow the thesis:

- Implementation – attempt to implement the suggested modifications. This is currently a work in progress of two of our students, and we expect to have the first results in June, 2010.
- Usability testing – find out whether the resulting system can be used without unacceptable discomfort for the users.
- Compare the different possible ways to make the needed modifications in terms of performance, usability, compatibility, extensibility.

REFERENCES

1. Domain and type enforcement. http://www.cs.wm.edu/~hallyn/dte/. [cit. 2004].
2. F. Coker and R. Coker. Taking advantage of selinux in red hat enterprise linux. Red Hat Magazine, (6), 2005. [cit. 2009].
3. Fedora selinux project : discussion of policies. http://fedoraproject.org/wiki/SELinux/Policies. [cit. 2009].
4. H. F. Tipton and M. Krause, editors. Information Security Management Handbook. CRC Press LLC, 5th edition, 2004. ISBN 0-8493-1997-8.
5. ISO/IEC 15408:2005, common criteria for information technology security evaluation : part 1 introduction and general model. http://www.commoncriteriaportal.org/.
6. ISO/IEC 15408:2005, common criteria for information technology security evaluation : part 2 security functional requirements. http://www.commoncriteriaportal.org/.
7. ISO/IEC 15408:2005, common criteria for information technology security evaluation : part 3 security assurance requirements. http://www.commoncriteriaportal.org/.
8. J. Janácek. A security model for an operating system for ˇ security-critical applications in small office and home environment. Communications : Scientific Letters of the University of Žilina, 11(3):5–10, 2009. ISSN 1335-4205.
9. J. Janácek. Bezpe ˇ cnostʹ opera ˇ cných systémov : písomná ˇ castʹ ˇ dizertacnej skúšky. ˇ http: //www.dcs.fmph.uniba.sk/~janacek/BezpecnostOS.pdf.
10. J. Janácek. Mandatory access control for small office and home ˇ environment. In Informaˇcné Technológie – Aplikácie a Teória : Zborník príspevkov prezentovaných na pracovnom seminári ITAT, pages 27–34, Sena, 2009. PONT s.r.o. ˇ
11. J. Janácek. Two dimensional labelled security model with ˇ partially trusted subjects and its enforcement using selinux dte mechanism. In Networked Digital Technologies, Part I,

volume 87 of Communications in Computer and Information Science. Springer, 2010. ISBN 978-3-642-14291-8, to appear.

12. Linux intrusion detection system. http://www.lids.org/. [cit. 2004].
13. M. Russinovich. Inside windows vista user account control. http://technet.microsoft.com/en-us/magazine/2007. 06.uac. aspx. [cit. 2009].
14. Medusa DS9. http://medusa.fornax.sk/. [cit. 2004]. [8] Role set based access control. http://www.rsbac.org/. [cit. 2004].
15. NSA. Information systems security organization: Controlled access protection profile : version 1.d. http://www. commoncriteriaportal.org/files/ppfiles/capp.pdf, 1999.
16. NSA. Information systems security organization: Labeled security protection profile : version 1.b. http://www. commoncriteriaportal.org/files/ppfiles/lspp.pdf, 1999.
17. S. Riley. Mandatory integrity control in windows vista. http:// blogs.technet.com/steriley/archive/2006/ 07/21/442870.aspx. [cit. 2009].
18. Security enhanced linux. http://www.nsa.gov/selinux/. [cit. 2004]. [10] D. E. Bell and L. J. La Padula. Secure computer system: unified exposition and multics interpretation : technical report. http: //csrc.nist.gov/publications/history/ bell76.pdf, 1976.

CHAPTER 5

DESIGNING TRUSTED OPERATING SYSTEMS

INTRODUCTION

Operating systems are the prime providers of security in computing systems. They support many programming capabilities, permit multiprogramming and sharing of resources, and enforce restrictions on program and user behavior. Because they have such power, operating systems are also targets for attack, because breaking through the defenses of an operating system gives access to the secrets of computing systems.

Trusted Operating System (TOS) generally refers to an operating system that provides sufficient support for multilevel security and evidence of correctness to meet a particular set of government requirements.

The most common set of criteria for trusted operating system design is the Common Criteria combined with the Security Functional Requirements (SFRs) for Labeled Security Protection Profile (LSPP) and mandatory access control (MAC). The Common Criteria is the result of a multi-year effort by the governments of the U.S., Canada, United Kingdom, France, Germany, the Netherlands and other countries to develop a harmonized security criteria for IT products.

5.1 TRUSTED SYSTEM

A trusted system is a system that is relied upon to a specified extent to enforce a specified security policy. This is equivalent to saying that a trusted system is one whose failure would break a security policy (if a policy exists that the trusted system is trusted to enforce). The meaning of the word "trust" is critical, as it does not carry the meaning that might be expected in everyday usage. A system trusted by a user, is one that the user feels safe to use, and trusts to do tasks without secretly executing harmful or unauthorized programs; while trusted computing refers to whether programs can trust the platform to be unmodified from that expected, whether or not those programs are innocent, malicious or execute tasks that are undesired by the user.

Trusted system can also be seen as level base security system where protection is provided and handled according to different levels. This is commonly found in military, where information is categorized as unclassified (U), confidential(C), Secret(S), Top secret(TS) and beyond. These also enforces the policies of No-read up and No-write down.

We studied these four services:

- memory protection
- file protection
- general object access control
- user authentication

The four major underpinnings of a trusted operating system:

- **Policy**. Every system can be described by its requirements: statements of what the system should do and how it should do it. An operating system's security requirements are a set of well-defined, consistent, and implementable rules that have been clearly and unambiguously expressed. If the operating system is implemented to meet these requirements, it meets the user's expectations. To ensure that the requirements are clear, consistent, and effective, the operating system usually follows a stated security policy: a set of rules that lay out what is to be secured and why.
- **Model**. To create a trusted operating system, the designers must be confident that the proposed system will meet its requirements while protecting appropriate objects and relationships. They usually begin by constructing a model of the environment to be secured. The model is actually a representation of the policy the operating system will enforce. Designers compare the model with the system requirements to make sure that the overall system functions are not compromised or degraded by the security needs. Then, they study different ways of enforcing that security.
- **Design**. After having selected a security model, designers choose a means to implement it. Thus, the design involves both what the trusted operating system is (that is, its intended functionality) and how it is to be constructed (its implementation).
- **Trust**. Because the operating system plays a central role in enforcing security, we (as developers and users) seek some basis (assurance) for believing that it will meet our expectations. Our trust in the system is rooted in two aspects: features (the operating system has all the necessary functionality needed to enforce the expected security policy) and assurance (the operating system has been implemented in such a way that we have

confidence it will enforce the security policy correctly and effectively).

To trust any program, we base our trust on rigorous analysis and testing, looking for certain key characteristics:

- Functional correctness. The program does what it is supposed to, and it works correctly.
- Enforcement of integrity. Even if presented erroneous commands or commands from unauthorized users, the program maintains the correctness of the data with which it has contact.
- Limited privilege: The program is allowed to access secure data, but the access is minimized and neither the access rights nor the data are passed along to other untrusted programs or back to an untrusted caller.
- Appropriate confidence level. The program has been examined and rated at a degree of trust appropriate for the kind of data and environment in which it is to be used.

Trusted software is often used as a safe way for general users to access sensitive data. Trusted programs are used to perform limited (safe) operations for users without allowing the users to have direct access to sensitive data.

Security professionals prefer to speak of trusted instead of secure operating systems. A trusted system connotes one that meets the intended security requirements, is of high enough quality, and justifies the user's confidence in that quality. That is, trust is perceived by the system's receiver or user, not by its developer, designer, or manufacturer. As a user, you may not be able to evaluate that trust directly. You may trust the design, a professional evaluation, or the opinion of a valued colleague. But in the end, it is your responsibility to sanction the degree of trust you require.

It is important to realize that there can be degrees of trust; unlike security, trust is not a dichotomy. For example, you trust certain friends with deep secrets, but you trust others only to give you the time of day. Trust is a characteristic that often grows over time, in accordance with evidence and experience. For instance, banks increase their trust in borrowers as the borrowers repay loans as expected; borrowers with good trust (credit) records can borrow larger amounts. Finally, trust is earned, not claimed or conferred. The comparison in Table 1 highlights some of these distinctions.

Table 1. Qualities of Security and Trustedness

Secure	**Trusted**
Either-or: Something either is or is not secure.	Graded: There are degrees of "trust-worthiness."
Property of presenter	Property of receiver
Asserted based on product charac-teristics	Judged based on evidence and analysis
Absolute: not qualified as to how used, where, when, or by whom	Relative: viewed in context of use
A goal	A characteristic

The adjective trusted appears many times, as in trusted process (a process that can affect system security, or a process whose incorrect or malicious execution is capable of violating system security policy), trusted product (an evaluated and approved product), trusted software (the software portion of a system that can be relied upon to enforce security policy), trusted computing base (the set of all protection mechanisms within a computing system, including hardware, firmware, and software, that together enforce a unified security policy over a product or system), or trusted system (a system that employs sufficient hardware and software integrity measures to allow its use for processing sensitive information). These definitions are paraphrased from. Common to these definitions are the concepts of

- enforcement of security policy
- sufficiency of measures and mechanisms
- evaluation

In studying trusted operating systems, we examine closely what makes them trustworthy.

5.2 SECURITY POLICIES

A security policy is a statement of the security we expect the system to enforce. An operating system (or any other piece of a trusted system) can be trusted only in relation to its security policy; that is, to the security needs the system is expected to satisfy.

5.2.1 Military Security Policy

Military security policy is based on protecting classified information. Each piece of information is ranked at a particular sensitivity level, such as unclassified, restricted, confidential, secret, or top secret. The ranks or levels form a hierarchy, and they reflect an increasing order of sensitivity, as shown in Figure 1. That is, the information at a given level is more sensitive than the information in the level below it and less sensitive than in the level above it. For example, restricted information is more sensitive than unclassified but less sensitive than confidential. We can denote the sensitivity of an object O by rank_O.

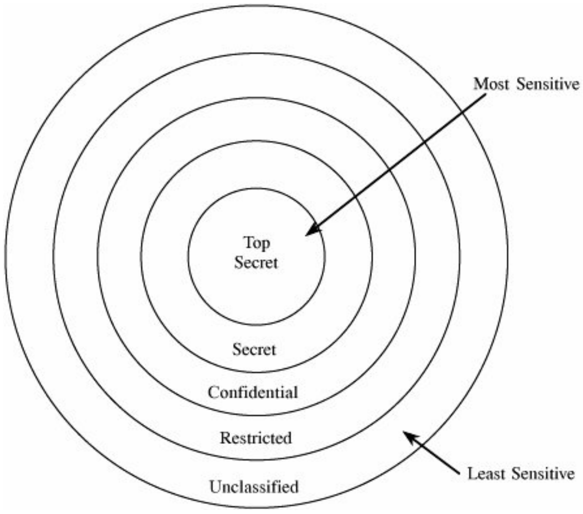


Figure 1. Hierarchy of Sensitivities.

Information access is limited by the need-to-know rule: Access to sensitive data is allowed only to subjects who need to know those data to perform their jobs. Each piece of classified information may be associated with one or more projects, called compartments, describing the subject matter of the information. For example, the alpha project may use secret information, as may the beta project, but staff on alpha do not need access to the information on beta. In other words, both projects use secret information, but each is restricted to only the secret information needed for its particular project. In this way, compartments help enforce need-to-know restrictions so that people obtain access only to information that is relevant to their jobs. A compartment may include information at only one sensitivity level, or it may cover information at several sensitivity levels. The relationship between compartments and sensitivity levels is shown in Figure 2.

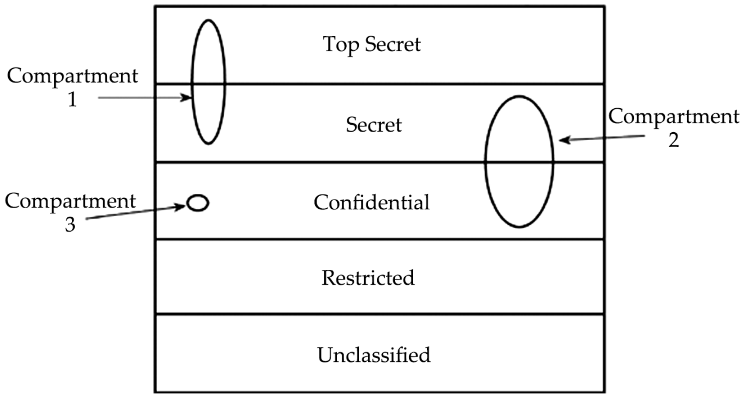


Figure 2. Compartments and Sensitivity Levels.

We can assign names to identify the compartments, such as snowshoe, crypto, and Sweden. A single piece of information can be coded with zero, one, two, or more compartment names, depending on the categories to which it relates. The association of information and compartments is shown in Figure 3. For example, one piece of information may be a list of publications on cryptography, whereas another may describe development of snowshoes in Sweden. The compartment of this first piece of information is {crypto}; the second is {snowshoe, Sweden}.

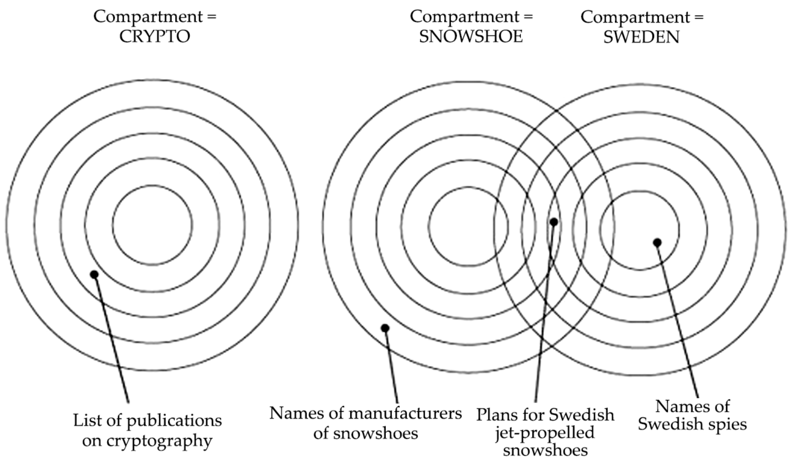

Compartment = CRYPTO
Compartment = SNOWSHOE
Compartment = SWEDEN
List of publications on cryptography
Names of manufacturers of snowshoes
Plans for Swedish jet-propelled snowshoes
Names of Swedish spies

Figure 3. Association of Information and Compartments.

The combination <rank; compartments> is called the class or classification of a piece of information. By designating information in this way, we can enforce need-to-know both by security level and by topic.

A person seeking access to sensitive information must be cleared. A clearance is an indication that a person is trusted to access information up to a certain level of sensitivity and that the person needs to know certain categories of sensitive information. The clearance of a subject is expressed as a combination <rank; compartments>. This combination has the same form as the classification of a piece of information.

Now we introduce a relation ≤, called dominance, on the sets of sensitive objects and subjects. For a subject o,

$s \leq o$ if and only if

$\text{rank}_S \leq \text{rank}_O$ and

$\text{compartments}_S \subseteq \text{compartments}_O$

We say that o dominates s (or s is dominated by o) if s ≤ is the opposite. Dominance is used to limit the sensitivity and content of information a subject can access. A subject can read an object only if

- the clearance level of the subject is at least as high as that of the information and
- the subject has a need to know about all compartments for which the information is classified

These conditions are equivalent to saying that the subject dominates the object.

To see how the dominance relation works, consider the concentric circles in Figure 3. According to the relationships depicted there, information classified as <secret;{Sweden}> could be read by someone cleared for access to <top secret;{Sweden}> or <secret;{Sweden, crypto}>, but not by someone with a <top secret;{crypto}> clearance or someone cleared for <confidential;{Sweden}> or <secret;{France}>.

Military security enforces both sensitivity requirements and need-to-know requirements. Sensitivity requirements are known as **hierarchical** requirements because they reflect the hierarchy of sensitivity levels; need-to-know restrictions are **nonhierarchical** because compartments do not necessarily reflect a hierarchical structure. This combinational model is appropriate for a setting in which access is rigidly controlled by a central authority. Someone, often called a security officer, controls clearances and classifications, which are not generally up to individuals to alter.

5.2.2 Commercial Security Policies

Commercial enterprises have significant security concerns. They worry that industrial espionage will reveal information to competitors about new products under development. Likewise, corporations are often eager to protect information about the details of corporate finance. So even though the commercial world is usually less rigidly and less hierarchically structured than the military world, we still find many of the same concepts in commercial security policies. For example, a large organization, such as a corporation or a university, may be divided into groups or departments, each responsible for a number of disjoint projects.

There may also be some corporate-level responsibilities, such as accounting and personnel activities. Data items at any level may have different degrees of sensitivity, such as public, proprietary, or internal; here, the names may vary among organizations, and no universal hierarchy applies.

Let us assume that public information is less sensitive than proprietary, which in turn is less sensitive than internal. Projects and departments tend to be fairly well separated, with some overlap as people work on two or more projects. Corporate-level responsibilities tend to overlie projects and departments, as people throughout the corporation may need accounting or personnel data. However, even corporate data may have degrees of sensitivity. Projects themselves may introduce a degree of sensitivity: Staff members on project old-standby have no need to know about project new-product, while staff members on new-product may have access to all data on old-standby. For these reasons, a commercial layout of data might look like Figure 4.

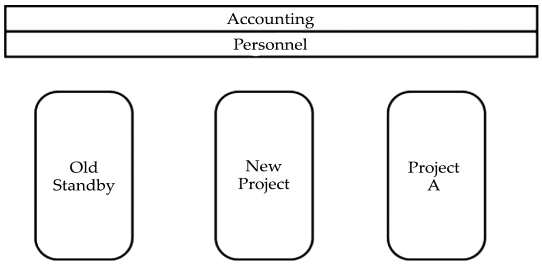


Figure 4. Commercial View of Sensitive Information.

Two significant differences exist between commercial and military information security. First, outside the military, there is usually no formalized notion of clearances: A person working on a commercial project does not require approval for project MARS access by a central security officer. Typically, an employee is not conferred a different degree of trust by being allowed access to internal data. Second, because there is no formal concept of a clearance, the rules for allowing access are less regularized. For example, if a senior manager decides that a person needs access to a piece of MARS internal data, the manager will instruct someone to allow the access, either one-time or continuing. Thus, there is no dominance function for most commercial information access because there is no formal concept of a commercial clearance.

So far, much of our discussion has focused only on read access, which addresses confidentiality in security. In fact, this narrow view holds true for much of the existing work in computer security. However, integrity and availability are at least as important as confidentiality in many instances. Policies for integrity and availability are significantly less well formulated than those for confidentiality, in both military and commercial realms. In the two examples that follow, we explore some instances of integrity concerns.

ClarkWilson Commercial Security Policy

In many commercial applications, integrity can be at least as important as confidentiality. The correctness of accounting records, the accuracy of legal work, and the proper timing of medical treatments are the essence of their fields. Clark and Wilson proposed a policy for what they call *well-formed transactions*, which they assert are as important in their field as is confidentiality in a military realm.

To see why, consider a company that orders and pays for materials. A representation of the procurement process might be this:

- A purchasing clerk creates an order for a supply, sending copies of the order to both the supplier and the receiving department.
- The supplier ships the goods, which arrive at the receiving department. A receiving clerk checks the delivery, ensures that the correct quantity of the right item has been received, and signs a delivery form. The delivery form and the original order go to the accounting department.

- The supplier sends an invoice to the accounting department. An accounting clerk compares the invoice with the original order (as to price and other terms) and the delivery form (as to quantity and item) and issues a check to the supplier.

The sequence of activities is important. A receiving clerk will not sign a delivery form without already having received a matching order (because suppliers should not be allowed to ship any quantities of any items they want and be paid), and an accounting clerk will not issue a check without already having received a matching order and delivery form (because suppliers should not be paid for goods not ordered or received). Furthermore, in most cases, both the order and the delivery form must be signed by authorized individuals. Performing the steps in order, performing exactly the steps listed, and authenticating the individuals who perform the steps constitute a well-formed transaction. The goal of the ClarkWilson policy is to maintain consistency between the internal data and the external (users') expectations of those data.

Clark and Wilson present their policy in terms of constrained data items, which are processed by transformation procedures. A transformation procedure is like a monitor in that it performs only particular operations on specific kinds of data items; these data items are manipulated only by transformation procedures. The transformation procedures maintain the integrity of the data items by validating the processing to be performed. Clark and Wilson propose defining the policy in terms of access triples: <userID, TP_i, {CDI_j, CDI_k, ...}>, combining a transformation procedure, one or more constrained data items, and the identification of a user who is authorized to operate on those data items by means of the transaction procedure.

Separation of Duty

A second commercial security policy involves separation of responsibility. Clark and Wilson raised this issue in their analysis of commercial security requirements, and Lee and Nash and Poland added to the concept.

To see how it works, we continue our example of a small company ordering goods. In the company, several people might be authorized to issue orders, receive goods, and write checks. However, we would not want the same person to issue the order, receive the goods, and write the check, because there is potential for abuse. Therefore, we might want to establish a policy that specifies that three separate individuals issue the order, receive the goods, and write the check, even though any of the three might be authorized to do any of these tasks. This required division of responsibilities is called separation of duty.

Separation of duty is commonly accomplished manually by means of dual signatures. Clark and Wilson triples are "stateless," meaning that a triple does not have a context of prior operations; triples are incapable of passing control information to other triples. Thus, if one person is authorized to perform operations TP_1 and TP_2, the Clark and Wilson triples cannot prevent the same person from performing both TP_1 and TP_2 on a given data item. However, it is quite easy to implement distinctness if it is stated as a policy requirement.

Chinese Wall Security Policy

Brewer and Nash defined a security policy called the Chinese Wall that reflects certain commercial needs for information access protection. The security requirements reflect issues relevant to those people in legal, medical, investment, or accounting firms who might be subject to conflict of interest. A conflict of interest exists when a person in one company can obtain sensitive information about people, products, or services in competing companies.

The security policy builds on three levels of abstraction.

- **Objects**. At the lowest level are elementary objects, such as files. Each file contains information concerning only one company.

- **Company groups**. At the next level, all objects concerning a particular company are grouped together.
- **Conflict classes**. At the highest level, all groups of objects for competing companies are clustered.

With this model, each object belongs to a unique company group, and each company group is contained in a unique conflict class. A conflict class may contain one or more company groups. For example, suppose you are an advertising company with clients in several fields: chocolate companies, banks, and airlines. You might want to store data on chocolate companies Suchard and Cadbury; on banks Citicorp, Deutsche Bank, and Credit Lyonnais; and on airline SAS. You want to prevent your employees from inadvertently revealing information to a client about that client's competitors, so you establish the rule that no employee will know sensitive information about competing companies. Using the Chinese Wall hierarchy, you would form six company groups (one for each company) and three conflict classes: {Suchard, Cadbury}, {Citicorp, Deutsche Bank, Credit Lyonnais}, and {SAS}.

The hierarchy guides a simple access control policy: A person can access any information as long as that person has never accessed information from a different company in the same conflict class. That is, access is allowed if either the object requested is in the same company group as an object that has previously been accessed or the object requested belongs to a conflict class that has never before been accessed. In our example, initially you can access any objects. Suppose you read from a file on Suchard. A subsequent request for access to any bank or to SAS would be granted, but a request to access Cadbury files would be denied. Your next access, of SAS data, does not affect future accesses. But if you then access a file on Credit Lyonnais, you will be blocked from future accesses to Deutsche Bank or Citicorp. From that point on, as shown in Figure 5, you can access objects only concerning Suchard, SAS, Credit Lyonnais, or a newly defined conflict class.

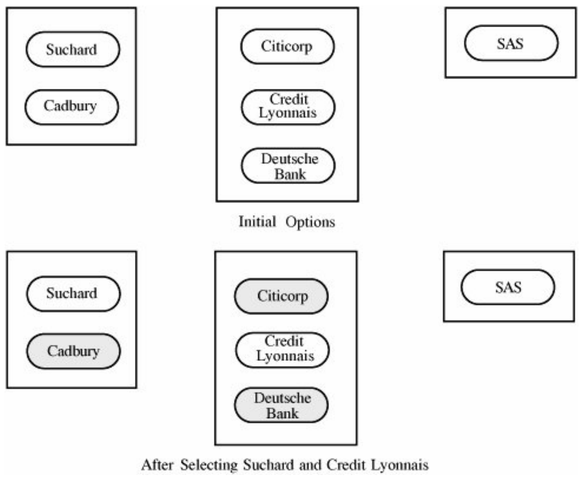


Figure 5. Chinese Wall Security Policy.

The Chinese Wall is a commercially inspired confidentiality policy. It is unlike most other commercial policies, which focus on integrity. It is also interesting because access permissions change dynamically: As a subject accesses some objects, other objects that would previously have been accessible are subsequently denied.

5.3 MODELS OF SECURITY

In security and elsewhere, models are often used to describe, study, or analyze a particular situation or relationship. McLean gives a good overview of models for security. In particular, security models are used to

- test a particular policy for completeness and consistency
- document a policy
- help conceptualize and design an implementation
- check whether an implementation meets its requirements

We assume that some access control policy dictates whether a given user can access a particular object. We also assume that this policy is established outside any model. That is, a policy decision determines whether a specific user should have access to a specific object; the model is only a mechanism that enforces that policy. Thus, we begin studying models by considering simple ways to control access by one user.

5.3.1 Multilevel Security

Ideally, we want to build a model to represent a range of sensitivities and to reflect the need to separate subjects rigorously from objects to which they should not have access. For instance, consider an election and the sensitivity of data involved in the voting process. The names of the candidates are probably not sensitive. If the results have not yet been released, the name of the winner is somewhat sensitive. If one candidate received an embarrassingly low number of votes, the vote count may be more sensitive. Finally, the way a particular individual voted is extremely sensitive. Users can also be ranked by the degree of sensitivity of information to which they can have access.

For obvious reasons, the military has developed extensive procedures for securing information. A generalization of the military model of information security has also been adopted as a model of data security within an operating system. Bell and La Padula were first to describe the properties of the military model in mathematical notation, and Denning first formalized the structure of this model. In 2005, Bell returned to the original model to highlight its contribution to computer security. He observed that the model demonstrated the need to understand security requirements before beginning system design, build security into not onto the system, develop a security toolbox, and design the system to protect itself. The generalized model is called the lattice model of security because its elements form a mathematical structure called a lattice.

Lattice Model of Access Security

The military security model is representative of a more general scheme, called a lattice. The dominance relation ≤ defined in the military model is the relation for the lattice. The relation ≤ is transitive and antisymmetric. The largest element of the lattice is the classification <all compartments>, and the smallest element is <unclassified; no compartments>; these two elements respectively dominate and are dominated by all elements. Therefore, the military model is a lattice.

Many other structures are lattices. For example, we noted earlier that a commercial security policy may contain data sensitivities such as public, proprietary, and internal, with the natural ordering that public data are less sensitive than proprietary, which are less sensitive than internal. These three levels also form a lattice.

Security specialists have chosen to base security systems on a lattice because it naturally represents increasing degrees. A security system designed to implement lattice models can be used in a military environment. However, it can also be used in commercial environments with different labels for the degrees of sensitivity. Thus, lattice representation of sensitivity levels applies to many computing situations.

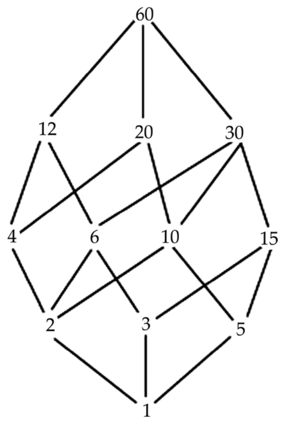


Figure 6. Sample Lattice.

BellLa Padula Confidentiality Model

The Bell and La Padula model is a formal description of the allowable paths of information flow in a secure system. The model's goal is to identify allowable communication when maintaining secrecy is important. The model has been used to define security requirements for systems concurrently handling data at different sensitivity levels. This model is a formalization of the military security policy and was central to the U.S. Department of Defense's evaluation criteria.

We are interested in secure information flows because they describe acceptable connections between subjects and objects of different levels of sensitivity. One purpose for security-level analysis is to enable us to construct systems that can perform concurrent computation on data at two different sensitivity levels. For example, we may want to use one machine for top-secret and confidential data at the same time. The programs processing top-secret data would be prevented from leaking top-secret data to the confidential data, and the confidential users would be prevented from accessing the top-secret data. Thus, the BellLa Padula model is useful as the basis for the design of systems that handle data of multiple sensitivities.

To understand how the BellLa Padula model works, consider a security system with the following properties. The system covers a set of subjects S and a set of objects O. Each subject s in S and each object o in O has a fixed security class C(s) and C(o) (denoting clearance and classification level). The security classes are ordered by a relation ≤. (Note: The classes may form a lattice, even though the BellLa Padula model can apply to even less restricted cases.)

Two properties characterize the secure flow of information.

Simple Security Property

A subject s may have read access to an object o only if C(o) ≤ Cs).

In the military model, this property says that the security class (clearance) of someone receiving a piece of information must be at least as high as the class (classification) of the information.

**-Property (called the "Star Property")*

A subject s who has read access to an object o may have write access to an object p only if C(o) ≤ Cp).

In the military model, this property says that the contents of a sensitive object can be written only to objects at least as high.

In the military model, one interpretation of the *-property is that a person obtaining information at one level may pass that information along only to people at levels no lower than the level of the information. The *-property prevents **write-down**, which occurs when a subject with access to high-level data transfers that data by writing it to a low-level object.

Literally, the*-property requires that a person receiving information at one level not talk with people cleared at levels lower than the level of the informationnot even about the weather! This example points out that this property is stronger than necessary to ensure security; the same is also true in computing systems. The BellLa Padula model is extremely conservative: It ensures security even at the expense of usability or other properties.

The implications of these two properties are shown in Figure 7. The classifications of subjects (represented by squares) and objects (represented by circles) are indicated by their positions: As the classification of an item increases, it is shown higher in the figure. The flow of information is generally horizontal (to and from the same level) and upward (from lower levels to higher). A downward flow is acceptable only if the highly cleared subject does not pass any high-sensitivity data to the lower-sensitivity object.

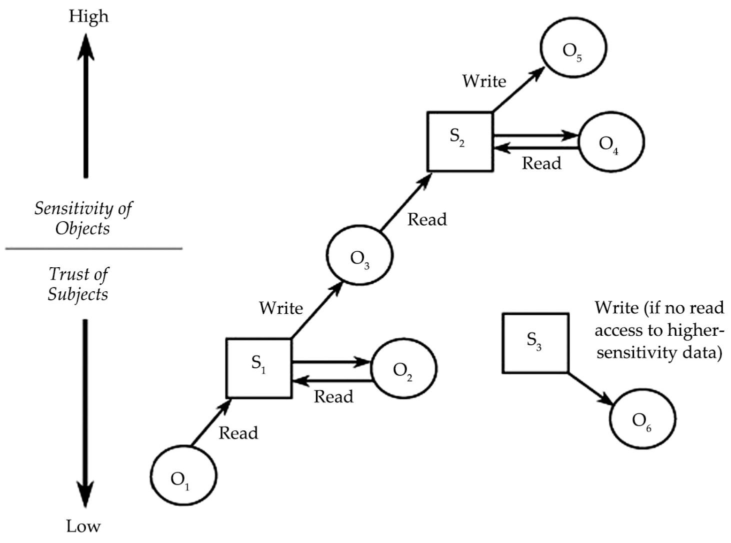


Figure 7. Secure Flow of Information.

For computing systems, downward flow of information is difficult because a computer program cannot readily distinguish between having read a piece of information and having read a piece of information that influenced what was later written. (McLean, in work related to Goguen and Meseguer, presents an interesting counter to the *-property of Bell and La Padula. He suggests considering noninterference, which can be loosely described as tracing the effects of inputs on outputs. If we can trace all output effects, we can determine conclusively whether a particular low-level output was "contaminated" with high-level input.)

Biba Integrity Model

The BellLa Padula model applies only to secrecy of information: The model identifies paths that could lead to inappropriate disclosure of information. However, the integrity of data is important, too. Biba constructed a model for preventing inappropriate modification of data.

The Biba model is the counterpart (sometimes called the dual) of the BellLa Padula model. Biba defines "integrity levels," which are analogous to the sensitivity levels of the BellLa Padula model. Subjects and objects are ordered by an integrity classification scheme, denoted I(s) and I(o). The properties are

- **Simple Integrity Property.** Subject s can modify (have write access to) object o only if I(s) ≥ Io)
- **Integrity *-Property.** If subject s has read access to object o with integrity level I(o), s can have write access to object p only if I(o) ≥ Ip)

These two rules cover untrustworthy information in a natural way. Suppose John is known to be untruthful sometimes. If John can create or modify a document, other people should distrust the truth of the statements in that document. Thus, an untrusted subject who has write access to an object reduces the integrity of that object. Similarly, people are rightly skeptical of a report based on unsound evidence. The low integrity of a source object implies low integrity for any object based on the source object.

This model addresses the integrity issue that the BellLa Padula model ignores. However, in doing so, the Biba model ignores secrecy. Secrecy-based security systems have been much more fully studied than have integrity-based systems. The current trend is to join secrecy and integrity concerns in security systems, although no widely accepted formal models achieve this compromise.

5.3.2 Models Proving Theoretical Limitations of Security Systems

Models are also useful for demonstrating the feasibility of an approach. Consider the security properties that we want a system to have. We want to build a model that tells us (before we invest in design, code, and testing) whether the properties can actually be achieved. This new class of models is based on the general theory of computability, which you may have studied in your computer science classes. Computability helps us determine decidability: If we pose a question, we want to know if we will ever be able to decide what the answer is. The results of these computability-based models show us the limitations of abstract security systems.

GrahamDenning Model

Lampson and Graham and Denning introduced the concept of a formal system of protection rules. Graham and Denning constructed a model having generic protection properties. This model forms the basis for two later models of security systems.

The GrahamDenning model operates on a set of subjects S, a set of objects O, a set of rights R, and an access control matrix A. The matrix has one row for each subject and one column for each subject and each object. The rights of a subject on another subject or an object are shown by the contents of an element of the matrix. For each object, one subject designated the "owner" has special rights; for each subject, another subject designated the "controller" has special rights.

The GrahamDenning model has eight primitive protection rights. These rights are phrased as commands that can be issued by subjects, with effects on other subjects or objects.

- Create object allows the commanding subject to introduce a new object to the system.
- Create subject, delete object, and delete subject have the similar effect of creating or destroying a subject or object.
- Read access right allows a subject to determine the current access rights of a subject to an object.
- Grant access right allows the owner of an object to convey any access rights for an object to another subject.
- Delete access right allows a subject to delete a right of another subject for an object, provided that the deleting subject either is the owner of the object or controls the subject from which access should be deleted.
- Transfer access right allows a subject to transfer one of its rights for an object to another subject. Each right can be transferable or nontransferable. If a subject receives a

transferable right, the subject can then transfer that right (either transferable or not) to other subjects. If a subject receives a nontransferable right, it can use the right but cannot transfer that right to other subjects.

These rules are shown in Table 2, which shows prerequisite conditions for executing each command and its effect. The access control matrix is A [s,o], where s is a subject and o is an object. The subject executing each command is denoted x. A transferable right is denoted r*; a nontransferable right is written r.

Table 2. Protection System Commands.

Command	Precondition	Effect
Create object o		Add column for o in A; place owner in A[x,o]
Create subject s		Add row for s in A; place control in A[x,s]
Delete object o	Owner in A[x,o]	Delete column o
Delete subject s	Control in A[x,s]	Delete row s
Read access right of s on o	Control in A[x,s] or owner in A[x,o]	Copy A[s,o] to x
Delete access right r of s on o	Control in A[x,s] or owner in A[x,o]	Remove r from A[s,o]
Grant access right r to s on o	Owner in A[x,o]	Add r to A[s,o]
Transfer access right r or r* to s on o	r* in A[x,o]	Add r or r* to A[s,o]

This set of rules provides the properties necessary to model the access control mechanisms of a protection system. For example, this mechanism can represent a reference monitor or a system of sharing between two untrustworthy, mutually suspicious subsystems.

HarrisonRuzzoUllman Results

Harrison, Ruzzo, and Ullman proposed a variation on the GrahamDenning model. This revised model answered several questions concerning the kinds of protection a given system can offer. Suppose you are about to use a particular operating system and you want to know if a given user can ever be granted a certain kind of access. For example, you may be establishing protection levels in Windows or MVS. You set up the access controls and then ask whether user X will ever have access to object Y. The three researchers developed their model so that we might be able to answer questions like this one.

The HarrisonRuzzoUllman model (called the HRU model) is based on **commands**, where each command involves **conditions** and **primitive operations**. The structure of a command is as follows.

```
command $name(o_1, o_2, \dots, o_k)$
    if      $r_1$ in $A[s_1, o_1]$ and
            $r_2$ in $A[s_2, o_2]$ and
                  …
            $r_m$ in $A[s_m, o_m]$
    then
            $op_1$
            $op_2$
            …
            $op_n$
end
```

This command is structured like a procedure, with parameters o_1 through o_k. The notation of the HRU model is slightly different from the GrahamDenning model; in HRU every subject is an object, too. Thus, the columns of the access control matrix are all the subjects and all the objects that are not subjects. For this reason, all the parameters of a command are labeled o, although they could be either subjects or nonsubject objects. Each r is a generic right, as in the GrahamDenning model. Each op is a primitive operation, defined in the following list. The access matrix is shown in Table 3.

Table 3. Access Matrix in HRU Model.

Objects						
Subjects	**S_1**	**S_2**	**S_3**	**O_1**	**O_2**	**O_3**
S_1	Control	Own, Suspend, Resume		Own	Own	Read, Propagate
S_2		Control			Extend	Own
S_3			Control	Read, Write	Write	Read

The primitive operations op, similar to those of the GrahamDenning model, are as follows:

- create subject s
- create object o
- destroy subject s
- destroy object o
- enter right r into A[s,o]
- delete right r from A[s,o]

The interpretations of these operations are what their names imply. A **protection system** is a set of subjects, objects, rights, and commands.

Harrison et al. demonstrate that these operations are adequate to model several examples of protection systems, including the Unix protection mechanism and an indirect access mode introduced by Graham and Denning. Thus, like the GrahamDenning model, the HRU model can represent "reasonable" interpretations of protection.

Two important results derived by Harrison et al. have major implications for designers of protection systems.

The first result from HRU indicates that, In the modeled system, in which commands are restricted to a single operation each, it is possible to decide whether a given subject can ever obtain a particular right to an object.

Therefore, we can decide (that is, we can know in advance) whether a low-level subject can ever obtain read access to a high-level object, for example.

The second result is less encouraging. Harrison et al. show that, If commands are not restricted to one operation each, it is not always decidable whether a given protection system can confer a given right.

Thus, we cannot determine in general whether a subject can obtain a particular right to an object.

As an example, consider protection in the Unix operating system. The Unix protection scheme is relatively simple; other protection systems are more complex. Because the Unix protection scheme requires more than one operation per command in the HRU model, there can be no general procedure to determine whether a certain access right can be given to a subject.

The HRU result is important but bleak. In fact, the HRU result can be extended. There may be an algorithm to decide the access right question for a particular collection of protection systems, but even an infinite number of algorithms cannot decide the access right question for all protection systems. However, the negative results do not say that no decision process exists for any protection system. In fact, for certain specific protection systems, it is decidable whether a given access right can be conferred.

TakeGrant Systems

One final model of a protection system is the **takegrant** system, introduced by Jones and expanded by Lipton and Snyder.

This model has only four primitive operations: create, revoke, take, and grant. Create and revoke are similar to operations from the GrahamDenning and HRU models; take and grant are new types of operations. These operations are presented most naturally through the use of graphs.

As in other systems, let S be a set of subjects and O be a set of objects; objects can be either active (subjects) or passive (nonsubject objects). Let R be a set of rights. Each subject or object is denoted by a node of a graph; the rights of a particular subject to a particular object are denoted by a labeled, directed edge from the subject to the object. Figure 8 shows an example of subject, object, and rights.

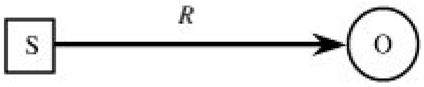


Figure 8. Subject, Object, and Rights.

Let s be the subject performing each of the operations. The four operations are defined as follows. The effects of these operations are shown in Figure 9.

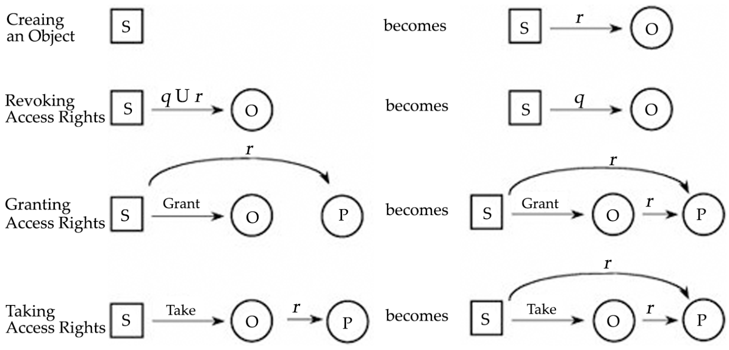


Figure 9. Creating an Object; Revoking, Granting, and Taking Access Rights.

- Create(o,r). A new node with label o is added to the graph. From s to o is a directed edge with label r, denoting the rights of s on o.
- Revoke(o,r). The rights r are revoked from s on o. The edge from s to o was labeled q U q. Informally, we say that s can revoke its rights to do r on o.
- Grant(o,p,r). Subject s grants to o access rights r on p. A specific right is grant. Subject s can grant to o access rights r on p only if s has grant rights on o and s has r rights on p. Informally, s can grant (share) any of its

rights with o, as long as s has the right to grant privileges to o. An edge from o to p is added, with label r.

- Take(o,p,r). Subject s takes from o access rights r on p. A specific right is take. Subject s can take from o access rights r on p only if s has take right on o and o has r rights on p. Informally, s can take any rights o has, as long as s has the right to take privileges from o. An edge from s to p is added, with label r.

This set of operations is even shorter than the operations of either of the two previous models. However, take and grant are more complex rights.

Snyder shows that in this system certain protection questions are decidable; furthermore, they are decidable in reasonable (less than exponential) time. In, Snyder considers two questions:

- Can we decide whether a given subject can share an object with another subject?
- Can we decide whether a given subject can steal access to an object from another subject?

Clearly, these are important questions to answer about a protection system, for they show whether the access control mechanisms are secure against unauthorized disclosure.

The answer to Snyder's first question is yes. Sharing can occur only if several other subjects together have the desired access to the object and the first subject is connected to each of the group of other subjects by a path of edges having a particular form. An algorithm that detects sharability runs in time proportional to the size of the graph of the particular case.

Snyder also answers the second question affirmatively, in a situation heavily dependent on the ability to share. Thus, an algorithm can decide whether access can be stolen by direct appeal to the algorithm to decide sharability.

Landwehr points out that the takegrant model assumes the worst about users: If a user can grant access rights, the model assumes that the user will. Suppose a user can create a file and grant access to it to everyone. In that situation, every user could allow access to every object by every other user. This worst-case assumption limits the applicability of the model to situations of controlled sharing of information. In general, however, the takegrant model is useful because it identifies conditions under which a user can obtain access to an object.

5.4 TRUSTED OPERATING SYSTEM DESIGN

Operating systems by themselves (regardless of their security constraints) are very difficult to design. They handle many duties, are subject to interruptions and context switches, and must minimize overhead so as not to slow user computations and interactions. Adding the responsibility for security enforcement to the operating system substantially increases the difficulty of designing an operating system.

5.4.1 Trusted System Design Elements

That security considerations pervade the design and structure of operating systems implies two things. First, an operating system controls the interaction between subjects and objects, so security must be considered in every aspect of its design. That is, the operating system design must include definitions of which objects will be protected in what way, which subjects will have access and at what levels, and so on. There must be a clear mapping from the security requirements to the design, so that all developers can see how the two relate. Moreover, once a section of the operating system has been designed, it must be checked to see that the degree of security that it is supposed to enforce or provide has actually been designed correctly. This checking can be done in many ways, including formal reviews or simulations. Again, a mapping is necessary, this time from the requirements to design to tests so that developers can affirm that each aspect of operating system security has been tested and shown to work correctly.

Second, because security appears in every part of an operating system, its design and implementation cannot be left fuzzy or vague until the rest of the system is working and being tested. It is extremely hard to retrofit security features to an operating system designed with inadequate security. Leaving an operating system's security to the last minute is much like trying to install plumbing or wiring in a house whose foundation is set, structure defined, and walls already up and painted; not only must you destroy most of what you have built, but you may also find that the general structure can no longer accommodate all that is needed (and so some has to be left out or compromised). Thus, security must be an essential part of the initial design of a trusted operating system. Indeed, the security considerations may shape many of the other design decisions, especially for a system with complex and constraining security requirements. For the same reasons, the security and other design principles must be carried throughout implementation, testing, and maintenance.

Good design principles are always good for security, as we have noted above. But several important design principles are quite particular to security and essential for building a solid, trusted operating system. These principles have been articulated well by Saltzer and Saltzer and Schroeder:

- *Least privilege.* Each user and each program should operate by using the fewest privileges possible. In this way, the damage from an inadvertent or malicious attack is minimized.
- *Economy of mechanism.* The design of the protection system should be small, simple, and straightforward. Such a protection system can be carefully analyzed, exhaustively tested, perhaps verified, and relied on.
- *Open design.* The protection mechanism must not depend on the ignorance of potential attackers; the mechanism should be public, depending on secrecy of relatively few key items, such as a password table. An open design is also available for extensive public scrutiny, thereby providing independent confirmation of the design

security.

- *Complete mediation.* Every access attempt must be checked. Both direct access attempts (requests) and attempts to circumvent the access checking mechanism should be considered, and the mechanism should be positioned so that it cannot be circumvented.
- *Permission based.* The default condition should be denial of access. A conservative designer identifies the items that should be accessible, rather than those that should not.
- *Separation of privilege.* Ideally, access to objects should depend on more than one condition, such as user authentication plus a cryptographic key. In this way, someone who defeats one protection system will not have complete access.
- *Least common mechanism.* Shared objects provide potential channels for information flow. Systems employing physical or logical separation reduce the risk from sharing.
- *Ease of use.* If a protection mechanism is easy to use, it is unlikely to be avoided.

Although these design principles were suggested several decades ago, they are as accurate now as they were when originally written. The principles have been used repeatedly and successfully in the design and implementation of numerous trusted systems. More importantly, when security problems have been found in operating systems in the past, they almost always derive from failure to abide by one or more of these principles.

5.4.2 Security Features of Ordinary Operating Systems

A multiprogramming operating system performs several functions that relate to security. To see how, examine Figure 10, which illustrates how an operating system interacts with users, provides services, and allocates resources.

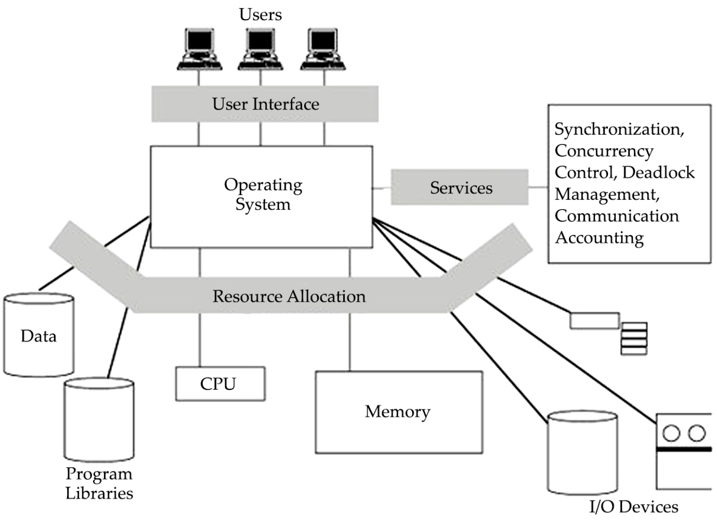


Figure 10. Overview of an Operating System's Functions.

We can see that the system addresses several particular functions that involve computer security:

- *User authentication.* The operating system must identify each user who requests access and must ascertain that the user is actually who he or she purports to be. The most common authentication mechanism is password comparison.
- *Memory protection.* Each user's program must run in a portion of memory protected against unauthorized accesses. The protection will certainly prevent outsiders' accesses, and it may also control a user's own access to restricted parts of the program space. Differential security, such as read, write, and execute, may be applied to parts of a user's memory space. Memory protection is usually performed by hardware mechanisms, such as paging or segmentation.
- File and I/O device access control. The operating system must protect user and system files from access by unauthorized users. Similarly, I/O device use must be protected. Data protection is usually achieved by table lookup, as with an access control matrix.

- Allocation and access control to general objects. Users need general objects, such as constructs to permit concurrency and allow synchronization. However, access to these objects must be controlled so that one user does not have a negative effect on other users. Again, table lookup is the common means by which this protection is provided.
- *Enforced sharing*. Resources should be made available to users as appropriate. Sharing brings about the need to guarantee integrity and consistency. Table lookup, combined with integrity controls such as monitors or transaction processors, is often used to support controlled sharing.
- Guaranteed *fair service.* All users expect CPU usage and other service to be provided so that no user is indefinitely starved from receiving service. Hardware clocks combine with scheduling disciplines to provide fairness. Hardware facilities and data tables combine to provide control.
- *Interprocess communication* and synchronization. Executing processes sometimes need to communicate with other processes or to synchronize their accesses to shared resources. Operating systems provide these services by acting as a bridge between processes, responding to process requests for asynchronous communication with other processes or synchronization. Interprocess communication is mediated by access control tables.
- Protected operating system protection data. The operating system must maintain data by which it can enforce security. Obviously if these data are not protected against unauthorized access (read, modify, and delete), the operating system cannot provide enforcement. Various techniques, including encryption, hardware control, and isolation, support isolation of operating system protection data.

5.4.3 Security Features of Trusted Operating Systems

Unlike regular operating systems, trusted systems incorporate technology to address both features and assurance. The design of a trusted system is delicate, involving selection of an appropriate and consistent set of features together with an appropriate degree of assurance that the features have been assembled and implemented correctly. Figure 11 illustrates how a trusted operating system differs from an ordinary one. Compare it with Figure 10. Notice how objects are accompanied or surrounded by an access control mechanism, offering far more protection and separation than does a conventional operating system. In addition, memory is separated by user, and data and program libraries have controlled sharing and separation.

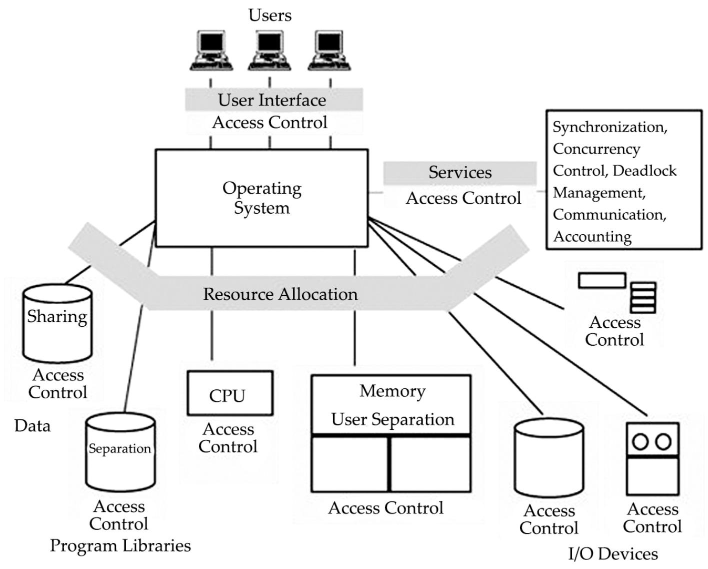


Figure 11. Security Functions of a Trusted Operating System.

The key features of a trusted operating system including

- user identification and authentication
- mandatory access control

- discretionary access control
- object reuse protection
- complete mediation
- trusted path
- audit
- audit log reduction
- intrusion detection

We consider each of these features in turn.

Identification and Authentication

Identification is at the root of much of computer security. We must be able to tell who is requesting access to an object, and we must be able to verify the subject's identity. As we see shortly, most access control, whether mandatory or discretionary, is based on accurate identification. Thus, identification involves two steps: finding out who the access requester is and verifying that the requester is indeed who he/she/it claims to be. That is, we want to establish an identity and then authenticate or verify that identity. Trusted operating systems require secure identification of individuals, and each individual must be uniquely identified.

Mandatory and Discretionary Access Control

Mandatory access control (MAC) means that access control policy decisions are made beyond the control of the individual owner of an object. A central authority determines what information is to be accessible by whom, and the user cannot change access rights. An example of MAC occurs in military security, where an individual data owner does not decide who has a top-secret clearance; neither can the owner change the classification of an object from top secret to secret.

By contrast, discretionary access control (DAC), as its name implies, leaves a certain amount of access control to the discretion of the object's owner or to anyone else who is authorized to control the object's access. The owner can determine who should have access rights to an object and what those rights should be. Commercial environments typically use DAC to allow anyone in a designated group, and sometimes additional named individuals, to change access. For example, a corporation might establish access controls so that the accounting group can have access to personnel files. But the corporation may also allow Ana and Jose to access those files, too, in their roles as directors of the Inspector General's office. Typically, DAC access rights can change dynamically. The owner of the accounting file may add Renee and remove Walter from the list of allowed accessors, as business needs dictate.

MAC and DAC can both be applied to the same object. MAC has precedence over DAC, meaning that of all those who are approved for MAC access, only those who also pass DAC will actually be allowed to access the object. For example, a file may be classified secret, meaning that only people cleared for secret access can potentially access the file. But of those millions of people granted secret access by the government, only people on project "deer park" or in the "environmental" group or at location "Fort Hamilton" are actually allowed access.

Object Reuse Protection

One way that a computing system maintains its efficiency is to reuse objects. The operating system controls resource allocation, and as a resource is freed for use by other users or programs, the operating system permits the next user or program to access the resource. But reusable objects must be carefully controlled, lest they create a serious vulnerability. To see why, consider what happens when a new file is created. Usually, space for the file comes from a pool of freed, previously used space on a disk or other storage device. Released space is returned to the pool "dirty," that is, still containing the data from the previous user. Because most users would write to a file before trying to read from it, the new user's data obliterate the previous owner's, so there is no inappropriate disclosure of the previous user's information. However, a malicious user may claim a large amount of disk space and then scavenge for sensitive data. This kind of attack is called **object reuse**. The problem is not limited to disk; it can occur with main memory, processor registers and storage, other magnetic media (such as disks and tapes), or any other reusable storage medium.

To prevent object reuse leakage, operating systems clear (that is, overwrite) all space to be reassigned before allowing the next user to have access to it. Magnetic media are particularly vulnerable to this threat. Very precise and expensive equipment can sometimes separate the most recent data from the data previously recorded, from the data before that, and so forth. This threat, called magnetic remanence. In any case, the operating system must take responsibility for "cleaning" the resource before permitting access to it.

Complete Mediation

For mandatory or discretionary access control to be effective, all accesses must be controlled. It is insufficient to control access only to files if the attack will acquire access through memory or an outside port or a network or a covert channel. The design and implementation difficulty of a trusted operating system rises significantly as more paths for access must be controlled. Highly trusted operating systems perform complete mediation, meaning that all accesses are checked.

Trusted Path

One way for a malicious user to gain inappropriate access is to "spoof" users, making them think they are communicating with a legitimate security enforcement system when in fact their keystrokes and commands are being intercepted and analyzed. For example, a malicious spoofer may place a phony user ID and password system between the user and the legitimate system. As the illegal system queries the user for identification information, the spoofer captures the real user ID and password; the spoofer can use these bona fide entry data to access the system later on, probably with malicious intent. Thus, for critical operations such as setting a password or changing access permissions, users want an unmistakable communication, called a trusted path, to ensure that they are supplying protected information only to a legitimate receiver. On some trusted systems, the user invokes a trusted path by pressing a unique key sequence that, by design, is intercepted directly by the security enforcement software; on other trusted systems, security-relevant changes can be made only at system startup, before any processes other than the security enforcement code run.

Accountability and Audit

A security-relevant action may be as simple as an individual access to an object, such as a file, or it may be as major as a change to the central access control database affecting all subsequent accesses. Accountability usually entails maintaining a log of security-relevant events that have occurred, listing each event and the person responsible for the addition, deletion, or change. This audit log must obviously be protected from outsiders, and every security-relevant event must be recorded.

Audit Log Reduction

Theoretically, the general notion of an audit log is appealing because it allows responsible parties to evaluate all actions that affect all protected elements of the system. But in practice an audit log may be too difficult to handle, owing to volume and analysis. To see why, consider what information would have to be collected and analyzed. In the extreme (such as where the data involved can affect a business' viability or a nation's security), we might argue that every modification or even each character read from a file is potentially security relevant; the modification could affect the integrity of data, or the single character could divulge the only really sensitive part of an entire file. And because the path of control through a program is affected by the data the program processes, the sequence of individual instructions is also potentially security relevant. If an audit record were to be created for every access to a single character from a file and for every instruction executed, the audit log would be enormous. (In fact, it would be impossible to audit every instruction, because then the audit commands themselves would have to be audited. In turn, these commands would be implemented by instructions that would have to be audited, and so on forever.)

In most trusted systems, the problem is simplified by an audit of only the opening (first access to) and closing of (last access to) files or similar objects. Similarly, objects such as individual memory locations, hardware registers, and instructions are not audited. Even with these restrictions, audit logs tend to be very large. Even a simple word processor may open fifty or more support modules (separate files) when it begins, it may create and delete a dozen or more temporary files during execution, and it may open many more drivers to handle specific tasks such as complex formatting or printing. Thus, one simple program can easily cause a hundred files to be opened and closed, and complex systems can cause thousands of files to be accessed in a relatively short time. On the other hand, some systems continuously read from or update a single file. A bank teller may process transactions against the general customer accounts file throughout the entire day; what is significant is not that the teller accessed the accounts file, but which entries in the file were accessed. Thus, audit at the level of file opening and closing is in some cases too much data and in other cases not enough to meet security needs.

A final difficulty is the "needle in a haystack" phenomenon. Even if the audit data could be limited to the right amount, typically many legitimate accesses and perhaps one attack will occur. Finding the one attack access out of a thousand legitimate accesses can be difficult. A corollary to this problem is the one of determining who or what does the analysis. Does the system administrator sit and analyze all data in the audit log? Or do the developers write a program to analyze the data? If the latter, how can we automatically recognize a pattern of unacceptable behavior? These issues are open questions being addressed not only by security specialists but also by experts in artificial intelligence and pattern recognition.

Intrusion Detection

Closely related to audit reduction is the ability to detect security lapses, ideally while they occur. As we have seen in the State Department example, there may well be too much information in the audit log for a human to analyze, but the computer can help correlate independent data. **Intrusion detection** software builds patterns of normal system usage, triggering an alarm any time the usage seems abnormal. After a decade of promising research results in intrusion detection, products are now commercially available. Some trusted operating systems include a primitive degree of intrusion detection software.

5.4.4 Kernelized Design

A kernel is the part of an operating system that performs the lowest-level functions. In standard operating system design, the kernel implements operations such as synchronization, interprocess communication, message passing, and interrupt handling. The kernel is also called a nucleus or core. The notion of designing an operating system around a kernel is described by Lampson and Sturgis and by Popek and Kline.

A security kernel is responsible for enforcing the security mechanisms of the entire operating system. The security kernel provides the security interfaces among the hardware, operating system, and other parts of the computing system. Typically, the operating system is designed so that the security kernel is contained within the operating system kernel. Security kernels are discussed in detail by Ames.

There are several good design reasons why security functions may be isolated in a security kernel.

- Coverage. Every access to a protected object must pass through the security kernel. In a system designed in this way, the operating system can use the security kernel to ensure that every access is checked.

- Separation. Isolating security mechanisms both from the rest of the operating system and from the user space makes it easier to protect those mechanisms from penetration by the operating system or the users.
- Unity. All security functions are performed by a single set of code, so it is easier to trace the cause of any problems that arise with these functions.
- Modifiability. Changes to the security mechanisms are easier to make and easier to test.
- Compactness. Because it performs only security functions, the security kernel is likely to be relatively small.
- Verifiability. Being relatively small, the security kernel can be analyzed rigorously. For example, formal methods can be used to ensure that all security situations (such as states and state changes) have been covered by the design.

Notice the similarity between these advantages and the design goals of operating systems that we described earlier. These characteristics also depend in many ways on modularity.

On the other hand, implementing a security kernel may degrade system performance because the kernel adds yet another layer of interface between user programs and operating system resources. Moreover, the presence of a kernel does not guarantee that it contains all security functions or that it has been implemented correctly. And in some cases a security kernel can be quite large.

How do we balance these positive and negative aspects of using a security kernel? The design and usefulness of a security kernel depend somewhat on the overall approach to the operating system's design. There are many design choices, each of which falls into one of two types: Either the kernel is designed as an addition to the operating system, or it is the basis of the entire operating system. Let us look more closely at each design choice.

Reference Monitor

The most important part of a security kernel is the reference monitor, the portion that controls accesses to objects. A reference monitor is not necessarily a single piece of code; rather, it is the collection of access controls for devices, files, memory, interprocess communication, and other kinds of objects. As shown in Figure 12, a reference monitor acts like a brick wall around the operating system or trusted software.

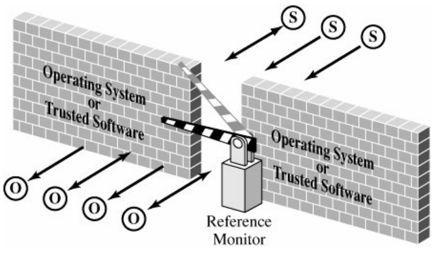


Figure 12. Reference Monitor.

A reference monitor must be

- **tamperproof**, that is, impossible to weaken or disable
- **unbypassable**, that is, always invoked when access to any object is required
- **analyzable**, that is, small enough to be subjected to analysis and testing, the completeness of which can be ensured

A reference monitor can control access effectively only if it cannot be modified or circumvented by a rogue process, and it is the single point through which all access requests must pass. Furthermore, the reference monitor must function correctly if it is to fulfill its crucial role in enforcing security. Because the likelihood of correct behavior decreases as the complexity and size of a program increase, the best assurance of correct policy enforcement is to build a small, simple, understandable reference monitor.

The reference monitor is not the only security mechanism of a trusted operating system. Other parts of the security suite include audit, identification, and authentication processing, as well as the setting of enforcement parameters, such as who the allowable subjects are and which objects they are allowed to access. These other security parts interact with the reference monitor, receiving data from the reference monitor or providing it with the data it needs to operate. The reference monitor concept has been used for many trusted operating systems and also for smaller pieces of trusted software. The validity of this concept is well supported both in research and in practice.

Trusted Computing Base

The trusted computing base, or TCB, is the name we give to everything in the trusted operating system necessary to enforce the security policy. Alternatively, we say that the TCB consists of the parts of the trusted operating system on which we depend for correct enforcement of policy. We can think of the TCB as a coherent whole in the following way. Suppose you divide a trusted operating system into the parts that are in the TCB and those that are not, and you allow the most skillful malicious programmers to write all the non-TCB parts. Since the TCB handles all the security, there is nothing the malicious non-TCB parts can do to impair the correct security policy enforcement of the TCB. This definition gives you a sense that the TCB forms the fortress-like shell that protects whatever in the system needs protection. But the analogy also clarifies the meaning of trusted in trusted operating system: Our trust in the security of the whole system depends on the TCB.

It is easy to see that it is essential for the TCB to be both correct and complete. Thus, to understand how to design a good TCB, we focus on the division between the TCB and non-TCB elements of the operating system and spend our effort on ensuring the correctness of the TCB.

TCB Functions

Just what constitutes the TCB? We can answer this question by listing system elements on which security enforcement could depend:

- hardware, including processors, memory, registers, and I/O devices
- some notion of processes, so that we can separate and protect security-critical processes
- primitive files, such as the security access control database and identification/authentication data
- protected memory, so that the reference monitor can be protected against tampering
- some interprocess communication, so that different parts of the TCB can pass data to and activate other parts. For example, the reference monitor can invoke and pass data securely to the audit routine.

It may seem as if this list encompasses most of the operating system, but in fact the TCB is only a small subset. For example, although the TCB requires access to files of enforcement data, it does not need an entire file structure of hierarchical directories, virtual devices, indexed files, and multidevice files. Thus, it might contain a primitive file manager to handle only the small, simple files needed for the TCB. The more complex file manager to provide externally visible files could be outside the TCB. Figure 13 shows a typical division into TCB and non-TCB sections.

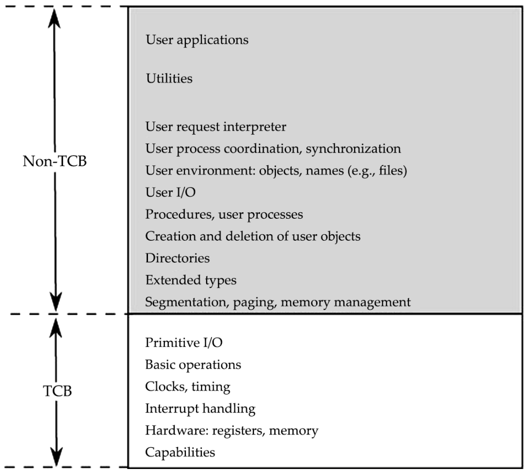

Non-TCB
TCB
User applications
Utilities
User request interpreter
User process coordination, synchronization
User environment: objects, names (e.g., files)
User I/O
Procedures, user processes
Creation and deletion of user objects
Directories
Extended types
Segmentation, paging, memory management
Primitive I/O
Basic operations
Clocks, timing
Interrupt handling
Hardware: registers, memory
Capabilities

Figure 13. TCB and Non-TCB Code.

The TCB, which must maintain the secrecy and integrity of each domain, monitors four basic interactions.

- *Process activation.* In a multiprogramming environment, activation and deactivation of processes occur frequently. Changing from one process to another requires a complete change of registers, relocation maps, file access lists, process status information, and other pointers, much of which is security-sensitive information.
- *Execution domain switching.* Processes running in one domain often invoke processes in other domains to obtain more sensitive data or services.
- *Memory protection.* Because each domain includes code and data stored in memory, the TCB must monitor memory references to ensure secrecy and integrity for each domain.
- I/O operation. In some systems, software is involved with each character transferred in an I/O operation. This software connects a user program in the outermost domain to an I/O device in the innermost (hardware) domain. Thus, I/O operations can cross all domains.

TCB Design

The division of the operating system into TCB and non-TCB aspects is convenient for designers and developers because it means that all security-relevant code is located in one (logical) part. But the distinction is more than just logical. To ensure that the security enforcement cannot be affected by non-TCB code, TCB code must run in some protected state that distinguishes it. Thus, the structuring into TCB and non-TCB must be done consciously. However, once this structuring has been done, code outside the TCB can be changed at will, without affecting the TCB's ability to enforce security. This ability to change helps developers because it means that major sections of the operating systemutilities, device drivers, user interface managers, and the likecan be revised or replaced any time; only the TCB code must be controlled more carefully. Finally, for anyone evaluating the security of a trusted operating system, a division into TCB and non-TCB simplifies evaluation substantially because non-TCB code need not be considered.

TCB Implementation

Security-related activities are likely to be performed in different places. Security is potentially related to every memory access, every I/O operation, every file or program access, every initiation or termination of a user, and every interprocess communication. In modular operating systems, these separate activities can be handled in independent modules. Each of these separate modules, then, has both security-related and other functions.

Collecting all security functions into the TCB may destroy the modularity of an existing operating system. A unified TCB may also be too large to be analyzed easily. Nevertheless, a designer may decide to separate the security functions of an existing operating system, creating a security kernel. This form of kernel is depicted in Figure 14.

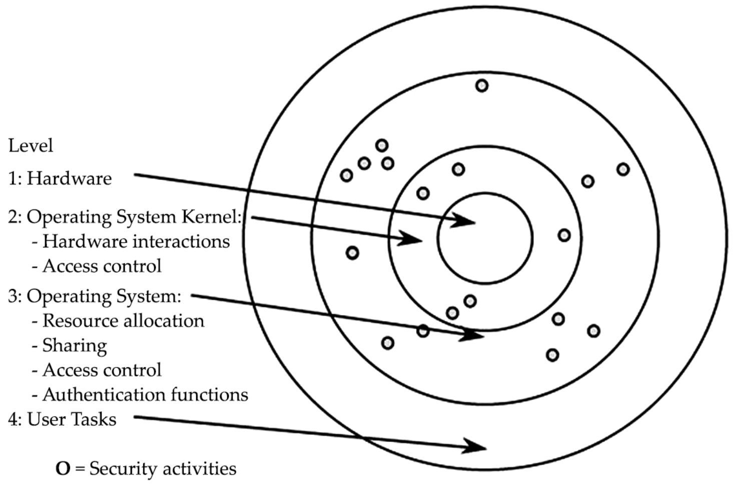


Figure 14. Combined Security Kernel/Operating System.

A more sensible approach is to design the security kernel first and then design the operating system around it. This technique was used by Honeywell in the design of a prototype for its secure operating system, Scomp. That system contained only twenty modules to perform the primitive security functions, and it consisted of fewer than 1,000 lines of higher-level-language source code. Once the actual security kernel of Scomp was built, its functions grew to contain approximately 10,000 lines of code.

In a security-based design, the security kernel forms an interface layer, just atop system hardware. The security kernel monitors all operating system hardware accesses and performs all protection functions. The security kernel, which relies on support from hardware, allows the operating system itself to handle most functions not related to security. In this way, the security kernel can be small and efficient. As a byproduct of this partitioning, computing systems have at least three execution domains: security kernel, operating system, and user. See Figure 15.

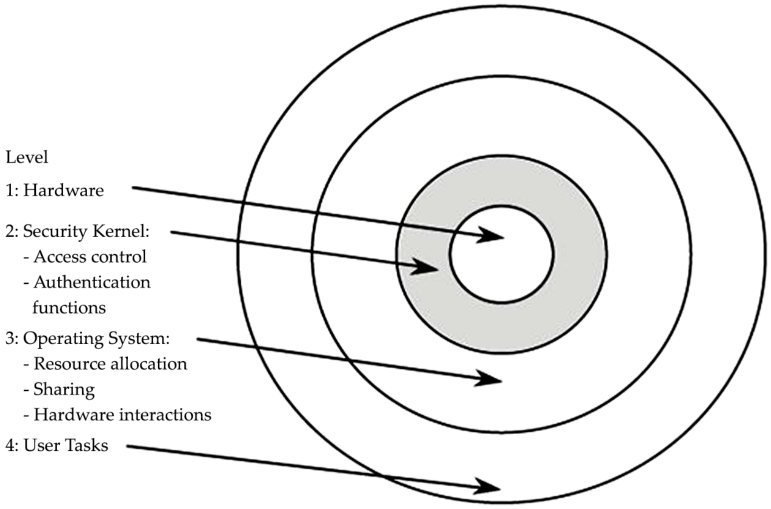


Figure 15. Separate Security Kernel.

5.4.5 Separation/Isolation

Rushby and Randell list four ways to separate one process from others: physical, temporal, cryptographic, and logical separation. With **physical separation**, two different processes use two different hardware facilities. For example, sensitive computation may be performed on a reserved computing system; nonsensitive tasks are run on a public system. Hardware separation offers several attractive features, including support for multiple independent threads of execution, memory protection, mediation of I/O, and at least three different degrees of execution privilege. **Temporal separation** occurs when different processes are run at different times. For instance, some military systems run nonsensitive jobs between 8:00 a.m. and noon, with sensitive computation only from noon to 5:00 p.m. Encryption is used for **cryptographic separation**, so two different processes can be run at the same time because unauthorized users cannot access sensitive data in a readable form. **Logical separation**, also called isolation, is provided when a process such as a reference monitor separates one user's objects from those of another user. Secure computing systems have been built with each of these forms of separation.

Multiprogramming operating systems should isolate each user from all others, allowing only carefully controlled interactions between the users. Most operating systems are designed to provide a single environment for all. In other words, one copy of the operating system is available for use by many users, as shown in Figure 16. The operating system is often separated into two distinct pieces, located at the highest and lowest addresses of memory.

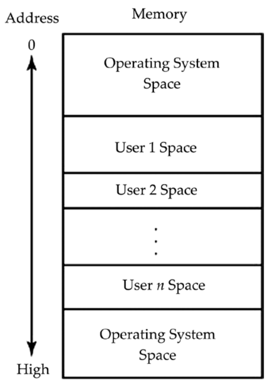


Figure 16. Conventional Multiuser Operating System Memory.

5.4.6 Virtualization

Virtualization is a powerful tool for trusted system designers because it allows users to access complex objects in a carefully controlled manner. By **virtualization** we mean that the operating system emulates or simulates a collection of a computer system's resources. We say that a **virtual machine** is a collection of real or simulated hardware facilities: a [central] processor that runs an instruction set, an amount of directly addressable storage, and some I/O devices. These facilities support the execution of programs.

Obviously, virtual resources must be supported by real hardware or software, but the real resources do not have to be the same as the simulated ones. There are many examples of this type of simulation. For instance, printers are often simulated on direct access devices for sharing in multiuser environments. Several small disks can be simulated with one large one. With demand paging, some noncontiguous memory can support a much larger contiguous virtual memory space. And it is common even on PCs to simulate space on slower disks with faster memory. In these ways, the operating system provides the virtual resource to the user, while the security kernel precisely controls user accesses.

Multiple Virtual Memory Spaces

The IBM MVS/ESA operating system uses virtualization to provide logical separation that gives the user the impression of physical separation. IBM MVS/ESA is a paging system such that each user's logical address space is separated from that of other users by the page mapping mechanism. Additionally, MVS/ESA includes the operating system in each user's logical address space, so a user runs on what seems to be a complete, separate machine.

Most paging systems present to a user only the user's virtual address space; the operating system is outside the user's virtual addressing space. However, the operating system is part of the logical space of each MVS/ESA user. Therefore, to the user MVS/ESA seems like a single-user system, as shown in Figure 17.

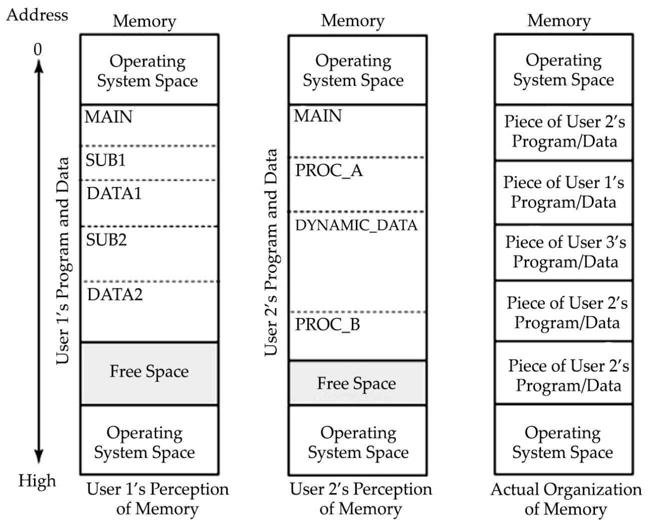

Address
0
High
User 1's Program and Data
Memory
Operating System Space
MAIN
SUB1
DATA1
SUB2
DATA2
Free Space
Operating System Space
User 1's Perception of Memory
Memory
Operating System Space
MAIN
PROC_A
DYNAMIC_DATA
PROC_B
Free Space
Operating System Space
User 2's Program and Data
User 2's Perception of Memory
Memory
Operating System Space
Piece of User 2's Program/Data
Piece of User 1's Program/Data
Piece of User 3's Program/Data
Piece of User 2's Program/Data
Piece of User 2's Program/Data
Operating System Space
Actual Organization of Memory

Figure 17. Multiple Virtual Addressing Spaces.

A primary advantage of MVS/ESA is memory management. Each user's virtual memory space can be as large as total addressable memory, in excess of 16 million bytes. And protection is a second advantage of this representation of memory. Because each user's logical address space includes the operating system, the user's perception is of running on a separate machine, which could even be true.

5.4.7 Virtual Machines

The IBM Processor Resources/System Manager (PR/SM) system provides a level of protection that is stronger still. A conventional operating system has hardware facilities and devices that are under the direct control of the operating system, as shown in Figure 18. PR/SM provides an entire virtual machine to each user, so that each user not only has logical memory but also has logical I/O devices, logical files, and other logical resources. PR/SM performs this feat by strictly separating resources.

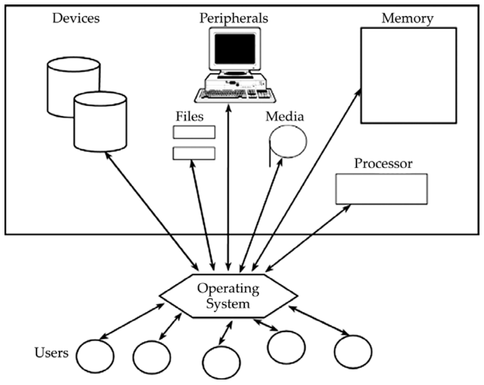


Figure 18. Conventional Operating System.

The PR/SM system is a natural extension of the concept of virtual memory. Virtual memory gives the user a memory space that is logically separated from real memory; a virtual memory space is usually larger than real memory, as well. A virtual machine gives the user a full set of hardware features; that is, a complete machine that may be substantially different from the real machine. These virtual hardware resources are also logically separated from those of other users. The relationship of virtual machines to real ones is shown in Figure 19.

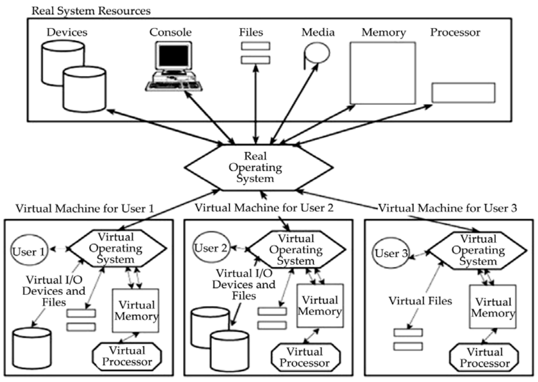


Figure 19. Virtual Machine.

Both MVS/ESA and PR/SM improve the isolation of each user from other users and from the hardware of the system. Of course, this added complexity increases the overhead incurred with these levels of translation and protection.

5.4.8 Layered Design

As described previously, a kernelized operating system consists of at least four levels: hardware, kernel, operating system, and user. Each of these layers can include sublayers. For example, in, the kernel has five distinct layers. At the user level, it is not uncommon to have quasi system programs, such as database managers or graphical user interface shells, that constitute separate layers of security themselves.

Layered Trust

The layered view of a secure operating system can be depicted as a series of concentric circles, with the most sensitive operations in the innermost layers. Then, the trustworthiness and access rights of a process can be judged by the process's proximity to the center: The more trusted processes are closer to the center. But we can also depict the trusted operating system in layers as a stack, with the security functions closest to the hardware. Such a system is shown in Figure 20.

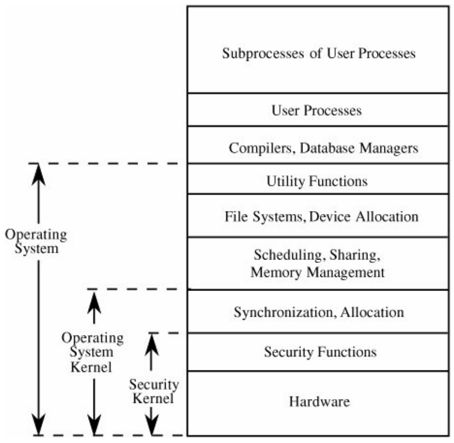

Subprocesses of User Processes
User Processes
Compilers, Database Managers
Utility Functions
File Systems, Device Allocation
Scheduling, Sharing, Memory Management
Synchronization, Allocation
Security Functions
Hardware
Operating System
Operating System Kernel
Security Kernel

Figure 20. Layered Operating System.

In this design, some activities related to protection functions are performed outside the security kernel. For example, user authentication may include accessing a password table, challenging the user to supply a password, verifying the correctness of the password, and so forth. The disadvantage of performing all these operations inside the security kernel is that some of the operations (such as formatting the userterminal interaction and searching for the user in a table of known users) do not warrant high security.

Alternatively, we can implement a single logical function in several different modules; we call this a layered design. Trustworthiness and access rights are the basis of the layering. In other words, a single function may be performed by a set of modules operating in different layers, as shown in Figure 21. The modules of each layer perform operations of a certain degree of sensitivity.

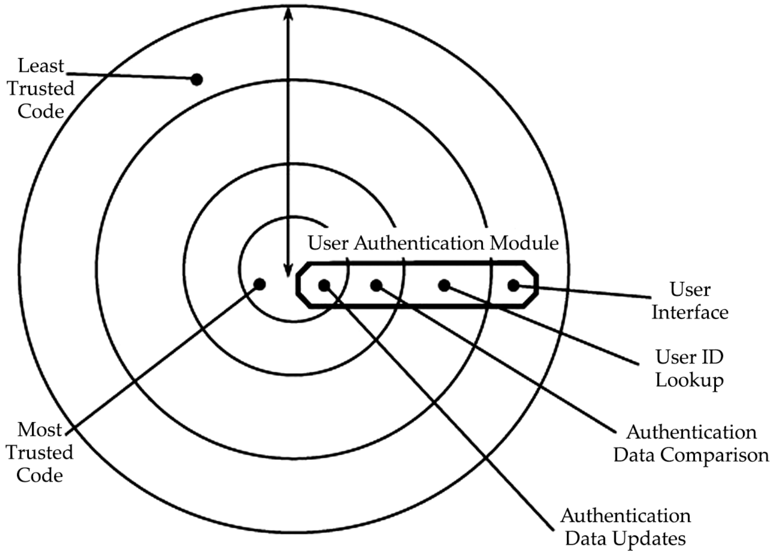

Least Trusted Code
User Authentication Module
User Interface
User ID Lookup
Authentication Data Comparison
Authentication Data Updates
Most Trusted Code

Figure 21. Modules Operating In Different Layers.

Neumann describes the layered structure used for the Provably Secure Operating System (PSOS). As shown in Table 4, some lower-level layers present some or all of their functionality to higher levels, but each layer properly encapsulates those things below itself.

Table 4. PSOS Design Hierarchy.

Level	Function	Hidden by Level	Visible to User
16	User request interpreter		Yes
15	User environments and name spaces		Yes
14	User I/O		Yes
13	Procedure records		Yes
12	User processes and visible I/O		Yes
11	Creation and deletion of user objects		Yes
10	Directories	11	Partially
9	Extended types	11	Partially
8	Segments	11	Partially
7	Paging	8	No
6	System processes and I/O	12	No
5	Primitive I/O	6	No
4	Arithmetic and other basic operations		Yes
3	Clocks	6	No
2	Interrupts	6	No
1	Registers and addressable memory	7	Partially
0	Capabilities		Yes

A layered approach is another way to achieve encapsulation. Layering is recognized as a good operating system design. Each layer uses the more central layers as services, and each layer provides a certain level of functionality to the layers farther out. In this way, we can "peel off" each layer and still have a logically complete system with less functionality. Layering presents a good example of how to trade off and balance design characteristics.

Another justification for layering is damage control. To see why, consider Neumann's two examples of risk, shown in Tables 5 and 6. In a conventional, nonhierarchically designed system, any problemhardware failure, software flaw, or unexpected condition, even in a supposedly non-security-relevant portioncan cause disaster because the effect of the problem is unbounded and because the system's design means that we cannot be confident that any given function has no (indirect) security effect.

Table 5. Conventionally (Nonhierarchically) Designed System.

Level	Functions	Risk
All	Noncritical functions	Disaster possible
All	Less critical functions	Disaster possible
All	Most critical functions	Disaster possible

Table 6. Hierarchically Designed System.

Level	Functions	Risk
2	Noncritical functions	Few disasters likely from noncritical software
1	Less critical functions	Some failures possible from less critical functions, but because of separation, effect limited
0	Most critical functions	Disasters possible but unlikely if system simple enough to be analyzed extensively

By contrast, as shown in Table 6, hierarchical structuring has two benefits:

- Hierarchical structuring permits identification of the most critical parts, which can then be analyzed intensely for correctness, so the number of problems should be smaller.
- Isolation limits effects of problems to the hierarchical levels at and above the point of the problem, so the effects of many problems should be confined.

These design propertiesthe kernel, separation, isolation, and hierarchical structurehave been the basis for many trustworthy system prototypes. They have stood the test of time as best design and implementation practices.

REFERENCES

1. Arpaci-Dusseau, Remzi; Arpaci-Dusseau, Andrea (2015). Operating Systems: Three Easy Pieces.
2. Author, Guest (7 February 2020). "User Authentication Methods & Technologies to Prevent Breach". ID R&D. Retrieved 8 November 2020.
3. Bic, Lubomur F.; Shaw, Alan C. (2003). Operating Systems. Pearson: Prentice Hall.
4. Bishop, Matt (2004). Computer security: art and science. Addison-Wesley.
5. Daly, Christopher. (2004). A Trust Framework for the DoD Network-Centric Enterprise Services (NCES) Environment, IBM Corp., 2004.
6. Deitel, Harvey M.; Deitel, Paul; Choffnes, David (25 December 2015). Operating Systems. Pearson/Prentice Hall. ISBN 978-0-13-092641-8.
7. Eugene Schultz, E. (2007). "Risks due to convergence of physical security systems and information technology environments". Information Security Technical Report. 12 (2): 80–84. doi:10.1016/j.istr.2007.06.001.
8. Federal Financial Institutions Examination Council (2008). "Authentication in an Internet Banking Environment" (PDF). Archived (PDF) from the original on 5 May 2010. Retrieved 31 December 2009.
9. Graham Pulford (17 October 2007). High-Security Mechanical Locks: An Encyclopedic Reference. Butterworth-Heinemann. pp. 76–. ISBN 978-0-08-055586-7.
10. Leva, Alberto; Maggio, Martina; Papadopoulos, Alessandro Vittorio; Terraneo, Federico (2013). Control-based Operating System Design. IET. ISBN 978-1-84919-609-3.
11. Newman, Robert (2010). Security and access control using biometric technologies. Boston, Mass.: Course Technology. ISBN 978-1-4354-9667-5. OCLC 535966830.
12. Niemelä, Harri (2011). "The study of business opportunities

and value add of NFC applications in security". theseus.fi. Retrieved 22 March 2019.

13. O'Brien, J. A., & Marakas, G. M.(2011). Management Information Systems. 10e. McGraw-Hill Irwin.
14. P. A. Loscocco, S. D. Smalley, Meeting Critical Security Objectives with Security-Enhanced Linux Proceedings of the 2001 Ottawa Linux Symposium.
15. Pereira, Henrique G. G.; Fong, Philip W. L. (2019). "SEPD: An Access Control Model for Resource Sharing in an IoT Environment". Computer Security – ESORICS 2019. Lecture Notes in Computer Science. Springer International Publishing. 11736: 195–216. doi:10.1007/978-3-030-29962-0_10. ISBN 978-3-030-29961-3.
16. Rhodes, Brian (2019). "Designing Access Control Guide". ipvm.com. Retrieved 1 October 2019.
17. Robert N. M. Watson. "A decade of OS access-control extensibility". Commun. ACM 56, 2 (February 2013), 52–63.
18. Schapranow, Matthieu-P. (2014). Real-time Security Extensions for EPCglobal Networks. Springer. ISBN 978-3-642-36342-9.
19. Silberschatz, Avi; Galvin, Peter; Gagne, Greg (2008). Operating Systems Concepts. John Wiley & Sons. ISBN 978-0-470-12872-5.

CHAPTER 6

DATABASE AND DATA MINING SECURITY

INTRODUCTION

A database is an organized collection of data, generally stored and accessed electronically from a computer system. Where databases are more complex they are often developed using formal design and modeling techniques. The database management system (DBMS) is the software that interacts with end users, applications, and the database itself to capture and analyze the data. The DBMS software additionally encompasses the core facilities provided to administer the database. The sum total of the database, the DBMS and the associated applications can be referred to as a “database system”. Often the term “database” is also used to loosely refer to any of the DBMS, the database system or an application associated with the database.

Ensuring the integrity of computer networks, both in relation to security and with regard to the institutional life of the nation in general, is a growing concern. Security and defense networks, proprietary research, intellectual property, and data based market mechanisms that depend on unimpeded and undistorted access, can all be severely compromised by malicious intrusions. Data mining has many applications in security including in national security (e.g., surveillance) as well as in cyber security (e.g., virus detection).

Protecting data is at the heart of many secure systems, and many users (people, programs, or systems) rely on a database management system (DBMS) to manage the protection. The security of database management systems, as an example of how application security can be designed and implemented for a specific task. There is substantial current interest in DBMS security because databases are newer than programming and operating systems. Databases are essential to many business and government organizations, holding data that reflect the organization's core competencies. Often, when business processes are reengineered to make them more effective and more in tune with new or revised goals, one of the first systems to receive careful scrutiny is the set of databases supporting the business processes. Thus, databases are more than software-related repositories. Their organization and contents are considered valuable corporate assets that must be carefully protected.

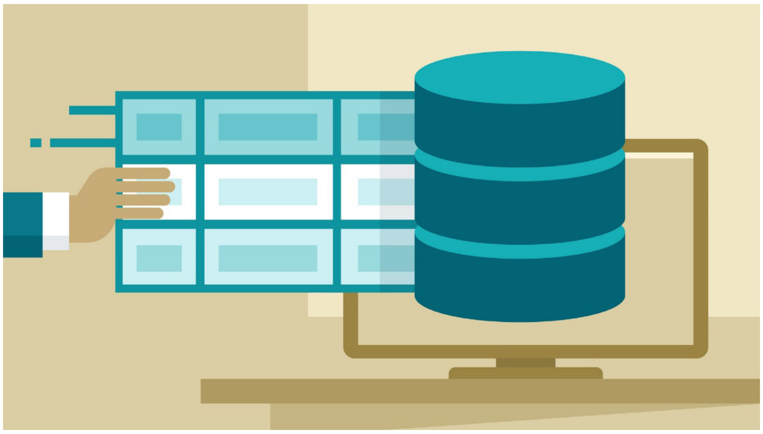
However, the protection provided by database management systems has had mixed results. Over time, we have improved our understanding of database security problems, and several good controls have been developed. But, as you will see, there are still more security concerns for which there are no available controls.

Then we consider the security requirements for database management systems. Two major security problems integrity and secrecy are explained in a database context. Two major (but related) database security problems, the inference problem and the multilevel problem. Both problems are complex, and there are no immediate solutions. However, by understanding the problems, we become more sensitive to ways of reducing potential threats to the data.

6.1 CONCEPT OF DATABASE

A database is a collection of data and a set of rules that organize the data by specifying certain relationships among the data. Through these rules, the user describes a logical format for the data. The data items are stored in a file, but the precise physical format of the file is of no concern to the user. A database administrator is a person who defines the rules that organize the data and also controls who should have access to what parts of the data. The user interacts with the database through a program called a database manager or a database management system (DBMS), informally known as a front end.

A database is stored as a file or a set of files. The information in these files may be broken down into records, each of which consists of one or more fields. Fields are the basic units of data storage, and each field typically contains information pertaining to one aspect or attribute of the entity described by the database. Records are also organized into tables that include information about relationships between its various fields. Although database is applied loosely to any collection of information in computer files, a database in the strict sense provides cross-referencing capabilities. Using keywords and various sorting commands, users can rapidly search, rearrange, group, and select the fields in many records to retrieve or create reports on particular aggregates of data.



Database records and files must be organized to allow retrieval of the information. Queries are the main way users retrieve database information. The power of a DBMS comes from its ability to define new relationships from the basic ones given by the tables and to use them to get responses to queries.

6.1.1 Components of Databases

The database file consists of records, each of which contains one related group of data. As shown in the example in Table 1, a record in a name and address file consists of one name and address. Each record contains fields or elements, the elementary data items themselves. The fields in the name and address record are NAME, ADDRESS, CITY, STATE, and ZIP (where ZIP is the U.S. postal code). This database can be viewed as a two-dimensional table, where a record is a row and each field of a record is an element of the table.

Table 1: Example of a Database.

ADAMS	212 Market St.	Columbus	OH	43210
BENCHLY	501 Union St.	Chicago	IL	60603
CARTER	411 Elm St.	Columbus	OH	43210

Not every database is easily represented as a single, compact table. The database in Figure 1 logically consists of three files with possibly different uses. These three files could be represented as one large table, but that depiction may not improve the utility of or access to the data.

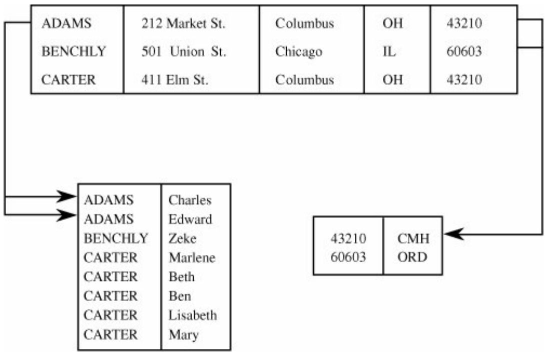


Figure 1: Related Parts of a Database.

The logical structure of a database is called a schema. A particular user may have access to only part of the database, called a subschema. The overall schema of the database in Figure 1 is detailed in Table 2. The three separate blocks of the figure are examples of subschemas, although other subschemas of this database can be defined. We can use schemas and subschemas to present to users only those elements they wish or need to see. For example, if Table 1 represents the employees at a company, the subschema on the lower left can list employee names without revealing personal information such as home address.

Table 2: Schema of Database Shown in Figure 1.

Name	**First**	**Address**	**City**	**State**	**Zip**	**Airport**
ADAMS	Charles	212 Market St.	Columbus	OH	43210	CMH

ADAMS	Edward	212 Market St.	Columbus	OH	43210	CMH
BENCH-LY	Zeke	501 Union St.	Chicago	IL	60603	ORD
CARTER	Mar-lene	411 Elm St.	Columbus	OH	43210	CMH
CARTER	Beth	411 Elm St.	Columbus	OH	43210	CMH
CARTER	Ben	411 Elm St.	Columbus	OH	43210	CMH
CARTER	Lisa-beth	411 Elm St.	Columbus	OH	43210	CMH
CARTER	Mary	411 Elm St.	Columbus	OH	43210	CMH

The rules of a database identify the columns with names. The name of each column is called an attribute of the database. A relation is a set of columns. For example, using the database in Table 2, we see that NAMEZIP is a relation formed by taking the NAME and ZIP columns, as shown in Table 3. The relation specifies clusters of related data values in much the same way that the relation "mother of" specifies a relationship among pairs of humans. In this example, each cluster contains a pair of elements, a NAME and a ZIP. Other relations can have more columns, so each cluster may be a triple, a 4-tuple, or an n-tuple (for some value n) of elements.

Table 3: Relation in a Database.

Name	**Zip**
ADAMS	43210
BENCHLY	60603
CARTER	43210

Queries

Users interact with database managers through commands to the DBMS that retrieve, modify, add, or delete fields and records of the database. A command is called a query. Database management systems have precise rules of syntax for queries. Most query languages use an English-like notation, and many are based on SQL, a structured query language originally developed by IBM. For example, the query

SELECT NAME = ‘ADAMS’

retrieves all records having the value ADAMS in the NAME field.

The result of executing a query is a subschema. One way to form a subschema of a database is by selecting records meeting certain conditions. For example, we might select records in which ZIP=43210, producing the result shown in Table 4.

Table 4: Result of Select Query.

Name	First	Address	City	State	Zip	Airport
ADAMS	Charles	212 Market St.	Columbus	OH	43210	CMH
ADAMS	Edward	212 Market St.	Columbus	OH	43210	CMH
CARTER	Marlene	411 Elm St.	Columbus	OH	43210	CMH
CARTER	Beth	411 Elm St.	Columbus	OH	43210	CMH
CARTER	Ben	411 Elm St.	Columbus	OH	43210	CMH
CARTER	Lisabeth	411 Elm St.	Columbus	OH	43210	CMH
CARTER	Mary	411 Elm St.	Columbus	OH	43210	CMH

Other, more complex, selection criteria are possible, with logical operators such as and ($\wedge$) and), and comparisons such as

SELECT (ZIP=’43210’) $\wedge$ (NAME=’ADAMS)

After having selected records, we may project these records onto one or more attributes. The select operation identifies certain rows from the database, and a project operation extracts the values from certain fields (columns) of those records. The result of a select-project operation is the set of values of specified attributes for the selected records. For example, we might select records meeting the condition ZIP=43210 and project the results onto the attributes NAME and FIRST, as in Table 5. The result is the list of first and last names of people whose addresses have zip code 43210.

Table 5: Results of Select-Project Query.

ADAMS	Charles
ADAMS	Edward
CARTER	Marlene

CARTER	Beth
CARTER	Ben
CARTER	Lisabeth
CARTER	Mary

Notice that we do not have to project onto the same attribute(s) on which the selection is done. For example, we can build a query using ZIP and NAME but project the result onto FIRST:

SHOW FIRST WHERE (ZIP='43210') $\wedge$ (NAME='ADAMS)

The result would be a list of the first names of people whose last names are ADAMS and ZIP is 43210.

We can also merge two subschema on a common element by using a join query. The result of this operation is a subschema whose records have the same value for the common element. For example, Figure 2 shows that the subschema NAMEZIP and the subschema ZIPAIRPORT can be joined on the common field ZIP to produce the subschema NAMEAIRPORT.

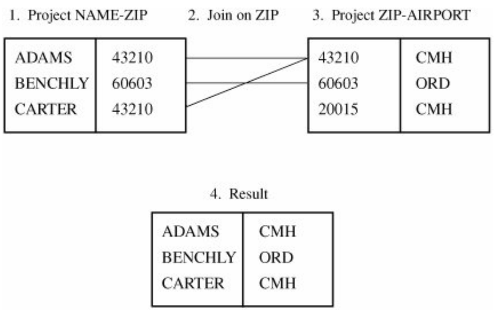


Figure 2: Results of Select-Project-Join Query.

6.1.2 Advantages of Using Databases

The logical idea behind a database is this: A database is a single collection of data, stored and maintained at one central location, to which many people have access as needed. However, the actual implementation may involve some other physical storage arrangement or access. The essence of a good database is that the users are unaware of the physical arrangements; the unified logical arrangement is all they see. As a result, a database offers many advantages over a simple file system:

- shared access, so that many users can use one common, centralized set of data
- minimal redundancy, so that individual users do not have to collect and maintain their own sets of data
- data consistency, so that a change to a data value affects all users of the data value
- data integrity, so that data values are protected against accidental or malicious undesirable changes
- controlled access, so that only authorized users are allowed to view or to modify data values

A DBMS is designed to provide these advantages efficiently. However, as often happens, the objectives can conflict with each other. In particular, as we shall see, security interests can conflict with performance. This clash is not surprising because measures taken to enforce security often increase the computing system's size or complexity. What is surprising, though, is that security interests may also reduce the system's ability to provide data to users by limiting certain queries that would otherwise seem innocuous.

6.2 SECURITY REQUIREMENTS

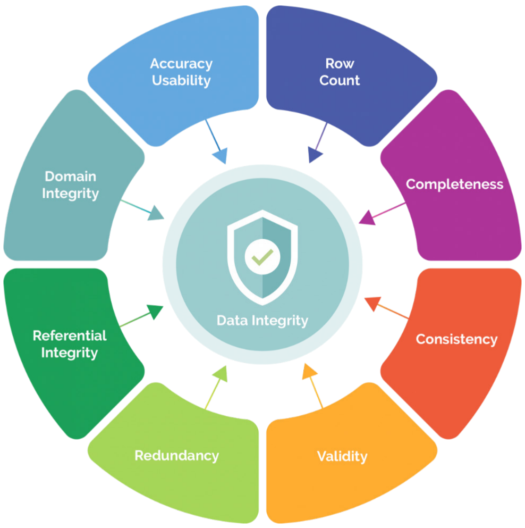
The basic security requirements of database systems are not unlike those of other computing systems we have studied. The basic problems access control, exclusion of spurious data, authentication of users, and reliability have appeared in many contexts so far in this book. Following is a list of requirements for database security.

- *Physical database integrity*. The data of a database are immune to physical problems, such as power failures, and someone can reconstruct the database if it is destroyed through a catastrophe.

- *Logical database integrity*. The structure of the database is preserved. With logical integrity of a database, a modification to the value of one field does not affect other fields, for example.
- *Element integrity*. The data contained in each element are accurate.
- *Auditability*. It is possible to track who or what has accessed (or modified) the elements in the database.
- *Access control*. A user is allowed to access only authorized data, and different users can be restricted to different modes of access (such as read or write).
- *User authentication*. Every user is positively identified, both for the audit trail and for permission to access certain data.
- *Availability*. Users can access the database in general and all the data for which they are authorized.

6.2.1 Integrity of the Database

If a database is to serve as a central repository of data, users must be able to trust the accuracy of the data values. This condition implies that the database administrator must be assured that updates are performed only by authorized individuals. It also implies that the data must be protected from corruption, either by an outside illegal program action or by an outside force such as fire or a power failure. Two situations can affect the integrity of a database: when the whole database is damaged (as happens, for example, if its storage medium is damaged) or when individual data items are unreadable. Integrity of the database as a whole is the responsibility of the DBMS, the operating system, and the (human) computing system manager. From the perspective of the operating system and the computing system manager, databases and DBMSs are files and programs, respectively. Therefore, one way of protecting the database as a whole is to regularly back up all files on the system. These periodic backups can be adequate controls against catastrophic failure.



Sometimes it is important to be able to reconstruct the database at the point of a failure. For instance, when the power fails suddenly, a bank's clients may be in the middle of making transactions or students may be in the midst of registering online for their classes. In these cases, we want to be able to restore the systems to a stable point without forcing users to redo their recently completed transactions. To handle these situations, the DBMS must maintain a log of transactions. For example, suppose the banking system is designed so that a message is generated in a log (electronic or paper or both) each time a transaction is processed. In the event of a system failure, the system can obtain accurate account balances by reverting to a backup copy of the database and reprocessing all later transactions from the log.

6.2.2 Element Integrity

The integrity of database elements is their correctness or accuracy. Ultimately, authorized users are responsible for entering correct data into databases. However, users and programs make mistakes collecting data, computing results, and entering values. Therefore, DBMSs sometimes take special action to help catch errors as they are made and to correct errors after they are inserted.

This corrective action can be taken in three ways. First, the DBMS can apply field checks, activities that test for appropriate values in a position. A field might be required to be numeric, an uppercase letter, or one of a set of acceptable characters. The check ensures that a value falls within specified bounds or is not greater than the sum of the values in two other fields. These checks prevent simple errors as the data are entered. (Sidebar 6-1 demonstrates the importance of element integrity.)

A second integrity action is provided by access control. To see why, consider life without databases. Data files may contain data from several sources, and redundant data may be stored in several different places. For example, a student's home address may be stored in many different campus files: at class registration, for dining hall privileges, at the bookstore, and in the financial aid office. Indeed, the student may not even be aware that each separate office has the address on file. If the student moves from one residence to another, each of the separate files requires correction. Without a database, there are several risks to the data's integrity. First, at a given time, there could be some data files with the old address (they have not yet been updated) and some simultaneously with the new address (they have already been updated). Second, there is always the possibility that the data fields were changed incorrectly, again leading to files with incorrect information. Third, there may be files of which the student is unaware, so he or she does not know to notify the file owner about updating the address information. These problems are solved by databases. They enable collection and control of this data at one central source, ensuring the student and users of having the correct address.

The third means of providing database integrity is maintaining a change log for the database. A change log lists every change made to the database; it contains both original and modified values. Using this log, a database administrator can undo any changes that were made in error.

6.2.3 Auditability

For some applications it may be desirable to generate an audit record of all access (read or write) to a database. Such a record can help to maintain the database's integrity, or at least to discover after the fact who had affected which values and when. A second advantage, as we see later, is that users can access protected data incrementally; that is, no single access reveals protected data, but a set of sequential accesses viewed together reveals the data, much like discovering the clues in a detective novel. In this case, an audit trail can identify which clues a user has already been given, as a guide to whether to tell the user more.

Audited events in operating systems are actions like open file or call procedure; they are seldom as specific as write record 3 or execute instruction I. To be useful for maintaining integrity, database audit trails should include accesses at the record, field, and even element levels. This detail is prohibitive for most database applications.

Furthermore, it is possible for a record to be accessed but not reported to a user, as when the user performs a select operation. (Accessing a record or an element without transferring to the user the data received is called the pass-through problem.) Also, you can determine the values of some elements without accessing them directly. (For example, you can ask for the average salary in a group of employees when you know the number of employees in the group is only one.) Thus, a log of all records accessed directly may both overstate and understate what a user actually knows.

6.2.4 Access Control

Databases are often separated logically by user access privileges. For example, all users can be granted access to general data, but only the personnel department can obtain salary data and only the marketing department can obtain sales data. Databases are very useful because they centralize the storage and maintenance of data. Limited access is both a responsibility and a benefit of this centralization.

The database administrator specifies who should be allowed access to which data, at the view, relation, field, record, or even element level. The DBMS must enforce this policy, granting access to all specified data or no access where prohibited. Furthermore, the number of modes of access can be many. A user or program may have the right to read, change, delete, or append to a value, add or delete entire fields or records, or reorganize the entire database.

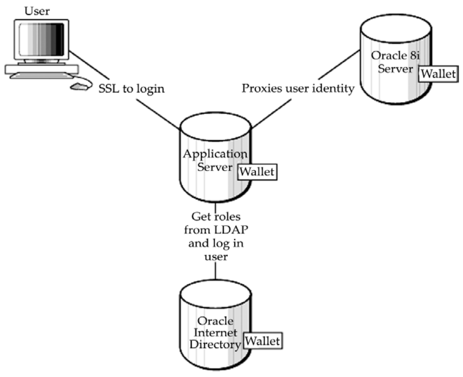
Superficially, access control for a database seems like access control for operating systems or any other component of a computing system. However, the database problem is more complicated. Operating system objects, such as files, are unrelated items, whereas records, fields, and elements are related. Although a user cannot determine the contents of one file by reading others, a user might be able to determine one data element just by reading others. The problem of obtaining data values from others is called inference.

It is important to notice that you can access data by inference without needing direct access to the secure object itself. Restricting inference may mean prohibiting certain paths to prevent possible inferences. However, restricting access to control inference also limits queries from users who do not intend unauthorized access to values. Moreover, attempts to check requested accesses for possible unacceptable inferences may actually degrade the DBMS's performance.

Finally, size or granularity is different between operating system objects and database objects. An access control list of several hundred files is much easier to implement than an access control list for a database with several hundred files of perhaps a hundred fields each. Size affects the efficiency of processing.

6.2.5 User Authentication

The DBMS can require rigorous user authentication. For example, a DBMS might insist that a user pass both specific password and time-of-day checks. This authentication supplements the authentication performed by the operating system. Typically, the DBMS runs as an application program on top of the operating system. This system design means that there is no trusted path from the DBMS to the operating system, so the DBMS must be suspicious of any data it receives, including user authentication. Thus, the DBMS is forced to do its own authentication.



6.2.6 Availability

A DBMS has aspects of both a program and a system. It is a program that uses other hardware and software resources, yet to many users it is the only application run. Users often take the DBMS for granted, employing it as an essential tool with which to perform particular tasks. But when the system is not availablebusy serving other users or down to be repaired or upgradedthe users are very aware of a DBMS's unavailability. For example, two users may request the same record, and the DBMS must arbitrate; one user is bound to be denied access for a while. Or the DBMS may withhold unprotected data to avoid revealing protected data, leaving the requesting user unhappy. Problems like these result in high availability requirements for a DBMS.

6.2.7 Integrity/Confidentiality/Availability

The three aspects of computer securityintegrity, confidentiality, and availabilityclearly relate to database management systems. As we have described, integrity applies to the individual elements of a database as well as to the database as a whole. Thus, integrity is a major concern in the design of database management systems.

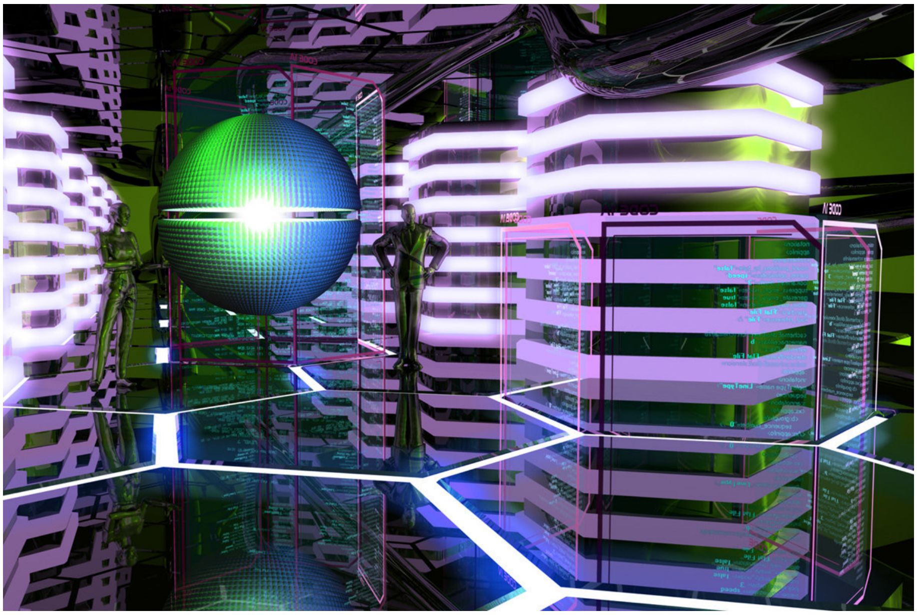
Confidentiality is a key issue with databases because of the inference problem, whereby a user can access sensitive data indirectly.

Finally, availability is important because of the shared access motivation underlying database development. However, availability conflicts with confidentiality.

6.3 RELIABILITY AND INTEGRITY

Databases amalgamate data from many sources, and users expect a DBMS to provide access to the data in a reliable way. When software engineers say that software has reliability, they mean that the software runs for very long periods of time without failing. Users certainly expect a DBMS to be reliable, since the data usually are key to business or organizational needs. Moreover, users entrust their data to a DBMS and rightly expect it to protect the data from loss or damage. Concerns for reliability and integrity are general security issues, but they are more apparent with databases.

However, the controls we consider are not absolute: No control can prevent an authorized user from inadvertently entering an acceptable but incorrect value.



Database concerns about reliability and integrity can be viewed from three dimensions:

- *Database integrity*: concern that the database as a whole is protected against damage, as from the failure of a disk drive or the corruption of the master database index. These concerns are addressed by operating system integrity controls and recovery procedures.
- *Element integrity*: concern that the value of a specific data element is written or changed only by authorized users. Proper access controls protect a database from corruption by unauthorized users.

- *Element accuracy*: concern that only correct values are written into the elements of a database. Checks on the values of elements can help prevent insertion of improper values. Also, constraint conditions can detect incorrect values.

6.3.1 Protection Features from the Operating System

A responsible system administrator backs up the files of a database periodically along with other user files. The files are protected during normal execution against outside access by the operating system's standard access control facilities. Finally, the operating system performs certain integrity checks for all data as a part of normal read and write operations for I/O devices. These controls provide basic security for databases, but the database manager must enhance them.

6.3.2 Two-Phase Update

A serious problem for a database manager is the failure of the computing system in the middle of modifying data. If the data item to be modified was a long field, half of the field might show the new value, while the other half would contain the old. Even if errors of this type were spotted easily (which they are not), a more subtle problem occurs when several fields are updated and no single field appears to be in obvious error. DBMSs, uses a two-phase update.

Update Technique

During the first phase, called the intent phase, the DBMS gathers the resources it needs to perform the update. It may gather data, create dummy records, open files, lock out other users, and calculate final answers; in short, it does everything to prepare for the update, but it makes no changes to the database. The first phase is repeatable an unlimited number of times because it takes no permanent action. If the system fails during execution of the first phase, no harm is done because all these steps can be restarted and repeated after the system resumes processing.

The last event of the first phase, called committing, involves the writing of a commit flag to the database. The commit flag means that the DBMS has passed the point of no return: After committing, the DBMS begins making permanent changes.

The second phase makes the permanent changes. During the second phase, no actions from before the commit can be repeated, but the update activities of phase two can also be repeated as often as needed. If the system fails during the second phase, the database may contain incomplete data, but the system can repair these data by performing all activities of the second phase. After the second phase has been completed, the database is again complete.

6.3.3 Redundancy/Internal Consistency

Many DBMSs maintain additional information to detect internal inconsistencies in data. The additional information ranges from a few check bits to duplicate or shadow fields, depending on the importance of the data.

Error Detection and Correction Codes

One form of redundancy is error detection and correction codes, such as parity bits, Hamming codes, and cyclic redundancy checks. These codes can be applied to single fields, records, or the entire database. Each time a data item is placed in the database, the appropriate check codes are computed and stored; each time a data item is retrieved, a similar check code is computed and compared to the stored value. If the values are unequal, they signify to the DBMS that an error has occurred in the database. Some of these codes point out the place of the error; others show precisely what the correct value should be. The more information provided, the more space required to store the codes.

Shadow Fields

Entire attributes or entire records can be duplicated in a database. If the data are irreproducible, this second copy can provide an immediate replacement if an error is detected. Obviously, redundant fields require substantial storage space.

6.3.4 Recovery

In addition to these error correction processes, a DBMS can maintain a log of user accesses, particularly changes. In the event of a failure, the database is reloaded from a backup copy and all later changes are then applied from the audit log.

6.3.5 Concurrency/Consistency

Database systems are often multiuser systems. Accesses by two users sharing the same database must be constrained so that neither interferes with the other. Simple locking is done by the DBMS. If two users attempt to read the same data item, there is no conflict because both obtain the same value.

If both users try to modify the same data items, we often assume that there is no conflict because each knows what to write; the value to be written does not depend on the previous value of the data item. However, this supposition is not quite accurate.

To see how concurrent modification can get us into trouble, suppose that the database consists of seat reservations for a particular airline flight. Agent A, booking a seat for passenger Mock, submits a query to find which seats are still available. The agent knows that Mock prefers a right aisle seat, and the agent finds that seats 5D, 11D, and 14D are open. At the same time, Agent B is trying to book seats for a family of three traveling together. In response to a query, the database indicates that 8ABC and 11DEF are the two remaining groups of three adjacent unassigned seats. Agent A submits the update command

SELECT (SEAT-NO = '11D') ASSIGN 'MOCK,E' TO PASSENGER-NAME

while Agent B submits the update sequence

SELECT (SEAT-NO='11D') ASSIGN 'EHLERS,P' TO PASSENGER-NAME

as well as commands for seats 11E and 11F. Then two passengers have been booked into the same seat (which would be uncomfortable, to say the least).

Both agents have acted properly: Each sought a list of empty seats, chose one seat from the list, and updated the database to show to whom the seat was assigned. The difficulty in this situation is the time delay between reading a value from the database and writing a modification of that value. During the delay time, another user has accessed the same data.

To resolve this problem, a DBMS treats the entire queryupdate cycle as a single atomic operation. The command from the agent must now resemble "read the current value of seat PASSENGER-NAME for seat 11D; if it is 'UNASSIGNED', modify it to 'MOCK,E' (or 'EHLERS,P')." The readmodify cycle must be completed as an uninterrupted item without allowing any other users access to the PASSENGER-NAME field for seat 11D. The second agent's request to book would not be considered until after the first agent's had been completed; at that time, the value of PASSENGERNAME would no longer be 'UNASSIGNED'.

A final problem in concurrent access is readwrite. Suppose one user is updating a value when a second user wishes to read it. If the read is done while the write is in progress, the reader may receive data that are only partially updated. Consequently, the DBMS locks any read requests until a write has been completed.

6.3.6 Monitors

The monitor is the unit of a DBMS responsible for the structural integrity of the database. A monitor can check values being entered to ensure their consistency with the rest of the database or with characteristics of the particular field. For example, a monitor might reject alphabetic characters for a numeric field. We discuss several forms of monitors.

Range Comparisons

A range comparison monitor tests each new value to ensure that the value is within an acceptable range. If the data value is outside the range, it is rejected and not entered into the database. For example, the range of dates might be 131, "/," 112, "/," 19002099. An even more sophisticated range check might limit the day portion to 130 for months with 30 days, or it might take into account leap year for February.

Range comparisons are also convenient for numeric quantities. For example, a salary field might be limited to $200,000, or the size of a house might be constrained to be between 500 and 5,000 square feet. Range constraints can also apply to other data having a predictable form.

Range comparisons can be used to ensure the internal consistency of a database. When used in this manner, comparisons are made between two database elements. For example, a grade level from K8 would be acceptable if the record described a student at an elementary school, whereas only 912 would be acceptable for a record of a student in high school. Similarly, a person could be assigned a job qualification score of 75100 only if the person had completed college or had had at least ten years of work experience. Filters or patterns are more general types of data form checks. These can be used to verify that an automobile plate is two letters followed by four digits, or the sum of all digits of a credit card number is a multiple of 9.

Checks of these types can control the data allowed in the database. They can also be used to test existing values for reasonableness. If you suspect that the data in a database have been corrupted, a range check of all records could identify those having suspicious values.

State Constraints

State constraints describe the condition of the entire database. At no time should the database values violate these constraints. Phrased differently, if these constraints are not met, some value of the database is in error.

On two-phase updates, we saw how to use a commit flag, which is set at the start of the commit phase and cleared at the completion of the commit phase. The commit flag can be considered a state constraint because it is used at the end of every transaction for which the commit flag is not set. A process to reset the commit flags in the event of a failure after a commit phase. In this way, the status of the commit flag is an integrity constraint on the database.

For another example of a state constraint, consider a database of employees' classifications. At any time, at most one employee is classified as "president." Furthermore, each employee has an employee number different from that of every other employee. If a mechanical or software failure causes portions of the database file to be duplicated, one of these uniqueness constraints might be violated. By testing the state of the database, the DBMS could identify records with duplicate employee numbers or two records classified as "president."

Transition Constraints

State constraints describe the state of a correct database. Transition constraints describe conditions necessary before changes can be applied to a database. For example, before a new employee can be added to the database, there must be a position number in the database with status "vacant." (That is, an empty slot must exist.) Furthermore, after the employee is added, exactly one slot must be changed from "vacant" to the number of the new employee.

Simple range checks and filters can be implemented within most database management systems. However, the more sophisticated state and transition constraints can require special procedures for testing. Such user-written procedures are invoked by the DBMS each time an action must be checked.

6.4 PROPOSALS FOR MULTILEVEL SECURITY

Implementing multilevel security for databases is difficult, probably more so than in operating systems, because of the small granularity of the items being controlled. We study approaches to multilevel security for databases.

Separation

As we have already seen, separation is necessary to limit access. We study mechanisms to implement separation in databases. Then, we see how these mechanisms can help to implement multilevel security for databases.

Partitioning

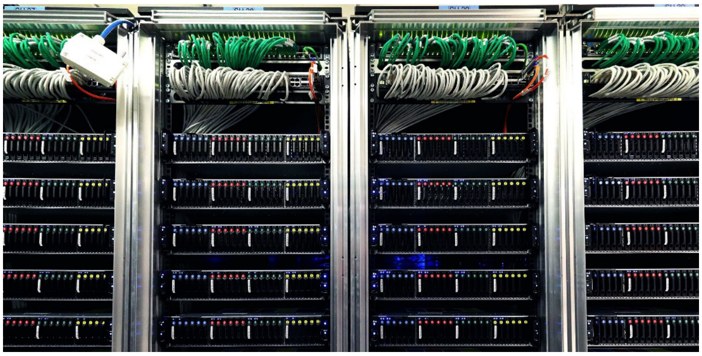
The obvious control for multilevel databases is partitioning. The database is divided into separate databases, each at its own level of sensitivity. This approach is similar to maintaining separate files in separate file cabinets.

This control destroys a basic advantage of databases: elimination of redundancy and improved accuracy through having only one field to update. Furthermore, it does not address the problem of a high-level user who needs access to some low-level data combined with high-level data.

Nevertheless, because of the difficulty of establishing, maintaining, and using multilevel databases, many users with data of mixed sensitivities handle their data by using separate, isolated databases.

Encryption

If sensitive data are encrypted, a user who accidentally receives them cannot interpret the data. Thus, each level of sensitive data can be stored in a table encrypted under a key unique to the level of sensitivity. But encryption has certain disadvantages.



First, a user can mount a chosen plaintext attack. Suppose party affiliation of REP or DEM is stored in encrypted form in each record. A user who achieves access to these encrypted fields can easily decrypt them by creating a new record with party=DEM and comparing the resulting encrypted version to that element in all other records. Worse, if authentication data are encrypted, the malicious user can substitute the encrypted form of his or her own data for that of any other user. Not only does this provide access for the malicious user, but it also excludes the legitimate user whose authentication data have been changed to that of the malicious user. These possibilities are shown in Figures 3 and 4.

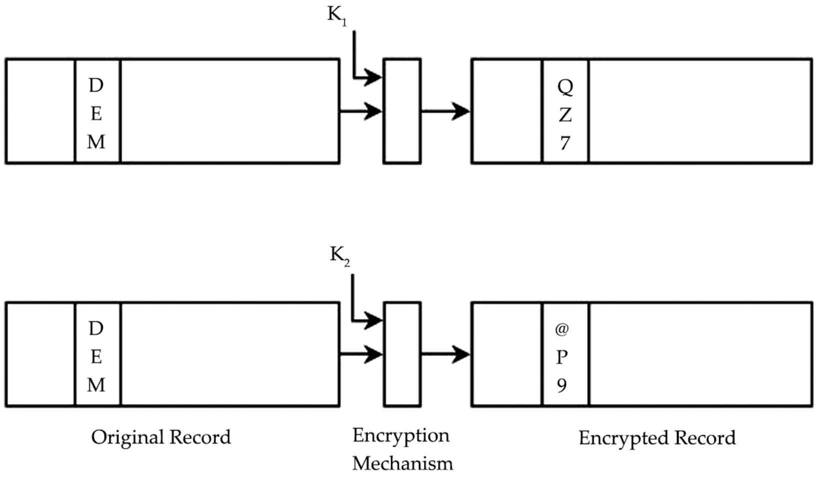


Figure 3: Cryptographic Separation: Different Encryption Keys.

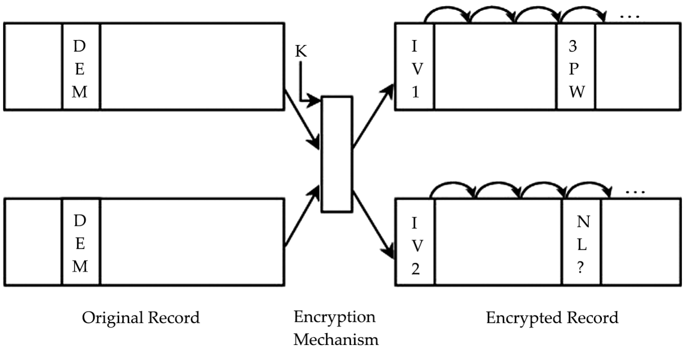


Figure 4: Cryptographic Separation: Block Chaining.

Using a different encryption key for each record overcomes these defects. Each record's fields can be encrypted with a different key, or all fields of a record can be cryptographically linked, as with cipher block chaining.

The disadvantage, then, is that each field must be decrypted when users perform standard database operations such as "select all records with SALARY > 10,000." Decrypting the SALARY field, even on rejected records, increases the time to process a query. Thus, encryption is not often used to implement separation in databases.

Integrity Lock

The lock is a way to provide both integrity and limited access for a database. The operation was nicknamed "spray paint" because each element is figuratively painted with a color that denotes its sensitivity. The coloring is maintained with the element, not in a master database table.

A model of the basic integrity lock is shown in Figure 5. As illustrated, each apparent data item consists of three pieces: the actual data item itself, a sensitivity label, and a checksum. The sensitivity label defines the sensitivity of the data, and the checksum is computed across both data and sensitivity label to prevent unauthorized modification of the data item or its label. The actual data item is stored in plaintext, for efficiency because the DBMS may need to examine many fields when selecting records to match a query.

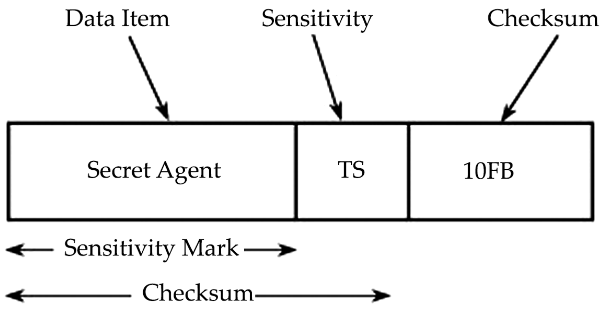


Figure 5: Integrity Lock.

The sensitivity label should be

- unforgeable, so that a malicious subject cannot create a new sensitivity level for an element
- unique, so that a malicious subject cannot copy a sensitivity level from another element
- concealed, so that a malicious subject cannot even determine the sensitivity level of an arbitrary element

The third piece of the integrity lock for a field is an error-detecting code, called a cryptographic checksum. To guarantee that a data value or its sensitivity classification has not been changed, this checksum must be unique for a given element, and must contain both the element's data value and something to tie that value to a particular position in the database. As shown in Figure 6, an appropriate cryptographic checksum includes something unique to the record (the record number), something unique to this data field within the record (the field attribute name), the value of this element, and the sensitivity classification of the element. These four components guard against anyone's changing, copying, or moving the data. The checksum can be computed with a strong encryption algorithm or hash function.

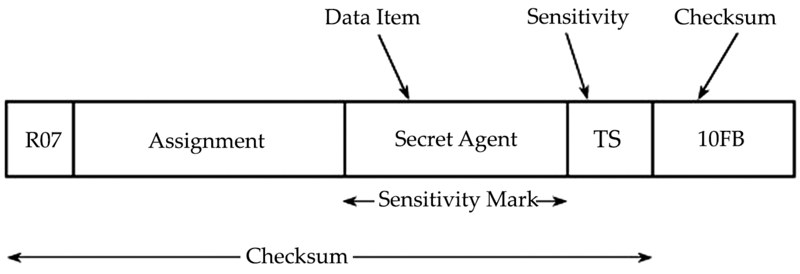


Figure 6: Cryptographic Checksum.

Sensitivity Lock

The sensitivity lock shown in Figure 7 was designed by Graubert and Kramer [GRA84b] to meet these principles. A sensitivity lock is a combination of a unique identifier (such as the record number) and the sensitivity level. Because the identifier is unique, each lock relates to one particular record. Many different elements will have the same sensitivity level. A malicious subject should not be able to identify two elements having identical sensitivity levels or identical data values just by looking at the sensitivity level portion of the lock. Because of the encryption, the lock's contents, especially the sensitivity level, are concealed from plain view. Thus, the lock is associated with one specific record, and it protects the secrecy of the sensitivity level of that record.

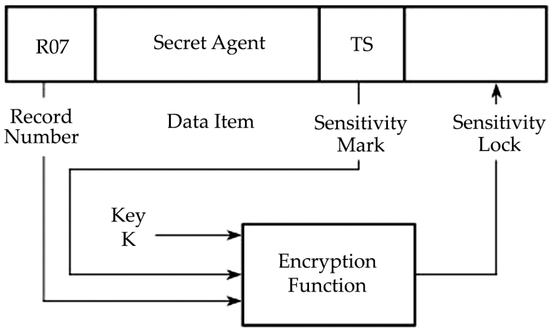


Figure 7: Sensitivity Lock.

6.4.1 Designs of Multilevel Secure Databases

These designs show the tradeoffs among efficiency, flexibility, simplicity, and trustworthiness.

Integrity Lock

The integrity lock DBMS was invented as a short-term solution to the security problem for multilevel databases. The intention was to be able to use any (untrusted) database manager with a trusted procedure that handles access control. The sensitive data were obliterated or concealed with encryption that protected both a data item and its sensitivity. In this way, only the access procedure would need to be trusted because only it would be able to achieve or grant access to sensitive data. The structure of such a system is shown in Figure 8.

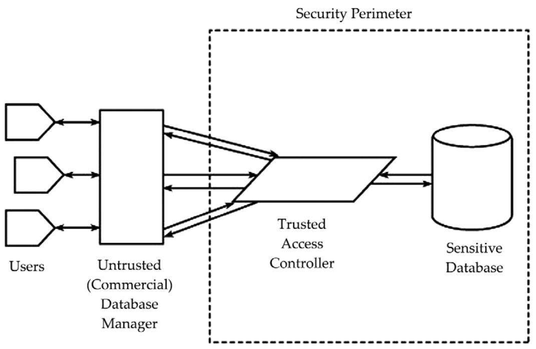

Security Perimeter
Users
Untrusted (Commercial) Database Manager
Trusted Access Controller
Sensitive Database

Figure 8: Trusted Database Manager.

The efficiency of integrity locks is a serious drawback. The space needed for storing an element must be expanded to contain the sensitivity label. Because there are several pieces in the label and one label for every element, the space required is significant.

Problematic, too, is the processing time efficiency of an integrity lock. The sensitivity label must be decoded every time a data element is passed to the user to verify that the user's access is allowable. Also, each time a value is written or modified, the label must be recomputed. Thus, substantial processing time is consumed. If the database file can be sufficiently protected, the data values of the individual elements can be left in plaintext. That approach benefits select and project queries across sensitive fields because an element need not be decrypted just to determine whether it should be selected.

A final difficulty with this approach is that the untrusted database manager sees all data, so it is subject to Trojan horse attacks by which data can be leaked through covert channels.

Trusted Front End

The model of a trusted front-end process is shown in Figure 9. This approach, originated by Hinke and Schaefer, recognizes that many DBMSs have been built and put into use without consideration of multilevel security. Staff members are already trained in using these DBMSs, and they may in fact use them frequently. The front-end concept takes advantage of existing tools and expertise, enhancing the security of these existing systems with minimal change to the system. The interaction between a user, a trusted front end, and a DBMS involves the following steps.

- A user identifies himself or herself to the front end; the front end authenticates the user's identity.
- The user issues a query to the front end.
- The front end verifies the user's authorization to data.
- The front end issues a query to the database manager.
- The database manager performs I/O access, interacting with low-level access control to achieve access to actual data.
- The database manager returns the result of the query to the trusted front end.
- The front end analyzes the sensitivity levels of the data items in the result and selects those items consistent with the user's security level.
- The front end transmits selected data to the untrusted front end for formatting.
- The untrusted front end transmits formatted data to the user.

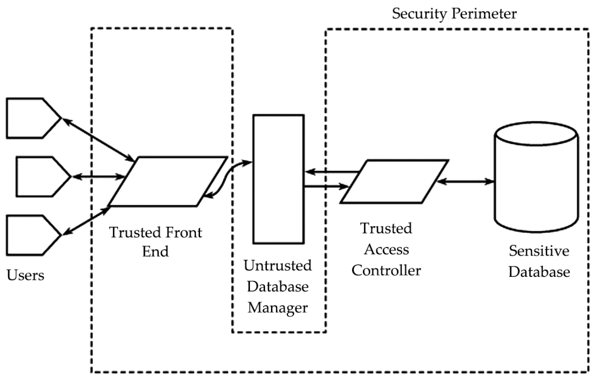


Figure 9: Trusted Front End.

The trusted front end serves as a one-way filter, screening out results the user should not be able to access. But the scheme is inefficient because potentially much data is retrieved and then discarded as inappropriate for the user.

Commutative Filters

The notion of a commutative filter was proposed by Denning as a simplification of the trusted interface to the DBMS. Essentially, the filter screens the user's request, reformatting it if necessary, so that only data of an appropriate sensitivity level are returned to the user.

A commutative filter is a process that forms an interface between the user and a DBMS. However, unlike the trusted front end, the filter tries to capitalize on the efficiency of most DBMSs. The filter reformats the query so that the database manager does as much of the work as possible, screening out many unacceptable records. The filter then provides a second screening to select only data to which the user has access.

Filters can be used for security at the record, attribute, or element level.

When used at the record level, the filter requests desired data plus cryptographic checksum information; it then verifies the accuracy and accessibility of data to be passed to the user.

At the attribute level, the filter checks whether all attributes in the user's query are accessible to the user and, if so, passes the query to the database manager. On return, it deletes all fields to which the user has no access rights.

At the element level, the system requests desired data plus cryptographic checksum information. When these are returned, it checks the classification level of every element of every record retrieved against the user's level.

Suppose a group of physicists in Washington works on very sensitive projects, so the current user should not be allowed to access the physicists' names in the database. This restriction presents a problem with this query:

retrieve NAME where ((OCCUP=PHYSICIST) (CITY=WASHDC))

Suppose, too, that the current user is prohibited from knowing anything about any people in Moscow. Using a conventional DBMS, the query might access all records, and the DBMS would then pass the results on to the user. However, as we have seen, the user might be able to infer things about Moscow employees or Washington physicists working on secret projects without even accessing those fields directly.

The commutative filter re-forms the original query in a trustable way so that sensitive information is never extracted from the database. Our sample query would become

retrieve NAME where ((OCCUP=PHYSICIST) (CITY=WASHDC)) from all records R where (NAME-SECRECY-LEVEL (R) USER-SECRECY-LEVEL) (OCCUP-SECRECY-LEVEL (R) USER-SECRECY-LEVEL) (CITY-SECRECY-LEVEL (R) USER-SECRECY-LEVEL))

The filter works by restricting the query to the DBMS and then restricting the results before they are returned to the user. In this instance, the filter would request NAME, NAME-SECRECY-LEVEL, OCCUP, OCCUP-SECRECY-LEVEL, CITY, and CITY-SECRECY-LEVEL values and would then filter and return to the user only those fields and items that are of a secrecy level acceptable for the user. Although even this simple query becomes complicated because of the added terms, these terms are all added by the front-end filter, invisible to the user.

An example of this query filtering in operation is shown in Figure 10. The advantage of the commutative filter is that it allows query selection, some optimization, and some subquery handling to be done by the DBMS. This delegation of duties keeps the size of the security filter small, reduces redundancy between it and the DBMS, and improves the overall efficiency of the system.

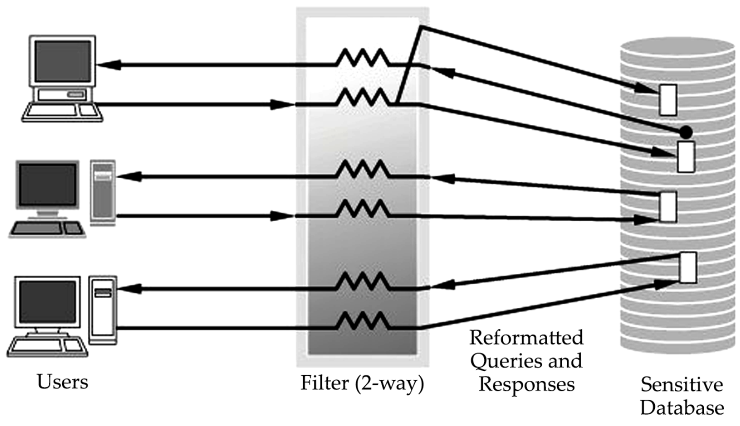


Figure 10: Commutative Filters.

Distributed Databases

The distributed or federated database is a fourth design for a secure multilevel database. In this case, a trusted front end controls access to two unmodified commercial DBMSs: one for all low-sensitivity data and one for all high-sensitivity data.

The front end takes a user's query and formulates single-level queries to the databases as appropriate. For a user cleared for high-sensitivity data, the front end submits queries to both the high- and low-sensitivity databases. But if the user is not cleared for high-sensitivity data, the front end submits a query to only the low-sensitivity database. If the result is obtained from either back-end database alone, the front end passes the result back to the user. If the result comes from both databases, the front end has to combine the results appropriately. For example, if the query is a join query having some high-sensitivity terms and some low, the front end has to perform the equivalent of a database join itself.

The distributed database design is not popular because the front end, which must be trusted, is complex, potentially including most of the functionality of a full DBMS itself. In addition, the design does not scale well to many degrees of sensitivity; each sensitivity level of data must be maintained in its own separate database.

Window/View

Traditionally, one of the advantages of using a DBMS for multiple users of different interests (but not necessarily different sensitivity levels) is the ability to create a different view for each user. That is, each user is restricted to a picture of the data reflecting only what the user needs to see. For example, the registrar may see only the class assignments and grades of each student at a university, not needing to see extracurricular activities or medical records. The university health clinic, on the other hand, needs medical records and drug-use information but not scores on standardized academic tests.

The notion of a window or a view can also be an organizing principle for multilevel database access. A window is a subset of a database, containing exactly the information that a user is entitled to access. Denning surveys the development of views for multilevel database security.

A view can represent a single user's subset database so that all of a user's queries access only that database. This subset guarantees that the user does not access values outside the permitted ones, because non-permitted values are not even in the user's database. The view is specified as a set of relations in the database, so the data in the view subset change as data change in the database.

Practical Issues

The multilevel security problem for databases has been studied since the 1970s. Several promising research results have been identified. However, as with trusted operating systems, the consumer demand has not been sufficient to support many products. Civilian users have not liked the inflexibility of the military multilevel security model, and there have been too few military users. Consequently, multilevel secure databases are primarily of research and historical interest.

The general concepts of multilevel databases are important. We do need to be able to separate data according to their degree of sensitivity. Similarly, we need ways of combining data of different sensitivities into one database. And these needs will only increase over time as larger databases contain more sensitive information, especially for privacy concerns.

6.5 DATA MINING

Databases are great repositories of data. More data are being collected and saved (partly because the cost per megabyte of storage has fallen from dollars a few years ago to fractions of cents today). Networks and the Internet allow sharing of databases by people and in ways previously unimagined. But to find needles of information in those vast fields of haystacks of data requires intelligent analyzing and querying of the data. Indeed, a whole specialization, called data mining, has emerged. In a largely automated way, data mining applications sort and search thorough data.

Data mining uses statistics, machine learning, mathematical models, pattern recognition, and other techniques to discover patterns and relations on large datasets. Data mining tools use association (one event often goes with another), sequences (one event often leads to another), classification (events exhibit patterns, for example coincidence), clustering (some items have similar characteristics), and forecasting (past events foretell future ones). The distinction between a database and a data mining application is becoming blurred; you can probably see how you could implement these techniques in ordinary database queries. Generally, database queries are manual, whereas data mining is more automatic. You could develop a database query to see what other products are bought by people who buy digital cameras and you might notice a preponderance of MP3 players in the result, but you would have to observe that relationship yourself. Data mining tools would present the significant relationships, not just between cameras and MP3 players, but also among bagels, airline tickets, and running shoes (if such a relationship existed). Humans have to analyze these correlations and determine what is significant.

Data mining presents probable relationships, but these are not necessarily cause-and-effect relationships. Suppose you analyzed data and found a correlation between sale of ice cream cones and death by drowning. You would not conclude that selling ice cream cones causes drowning (nor the converse). This distinction shows why humans must be involved in data mining to interpret the output: Only humans can discern that more variables are involved (for example, time of year or places where cones are sold).

Computer security gains from data mining. Data mining is widely used to analyze system data, for example, audit logs, to identify patterns related to attacks. Finding the precursors to an attack can help develop good prevention tools and techniques, and seeing the actions associated with an attack can help pinpoint vulnerabilities to control and damage that may have occurred.

However, we want to examine security problems involving data mining. Our now-familiar triad of confidentiality, integrity, and availability gives us clues to what these security issues are. Confidentiality concerns start with privacy but also include proprietary and commercially sensitive data and protecting the value of intellectual property: How do we control what is disclosed or derived? For integrity the important issue is correctness incorrect data are both useless and potentially damaging, but we need to investigate how to gauge and ensure correctness. The availability consideration relates to both performance and structure: Combining databases not originally designed to be combined affects whether results can be obtained in a timely manner or even at all.

6.5.1 Data Correctness and Integrity

"Connecting the dots" is a phrase currently in vogue: It refers to drawing conclusions from relationships between discrete bits of data. But before we can connect dots, we need to do two other important things: collect and correct them. Data storage and computer technology is making it possible to collect more dots than ever before. But if your name or address has ever appeared incorrectly on a mailing list, you know that not all collected dots are accurate.

Correcting Mistakes in Data

Data mining exacerbates this situation. Databases need unique keys to help with structure and searches. But different databases may not have shared keys, so they use some data field as if it were a key. In our example case, this shared data field might be the address, so now your neighbor's address is associated with cooking (even if your neighbor needs a recipe to make tea). Fortunately, this example is of little consequence.

Consider terrorists, however. A government's intelligence service collects data on suspicious activities. But the names of suspicious persons are foreign, written in a different alphabet. When transformed into the government's alphabet, the transformation is irregular: One agent writes "Doe," another "Do," and another "Dho." Trying to use these names as common keys is difficult at best. One approach is phonetic. You cluster terms that sound similar. In this case, however, you might bring in "Jo," "Cho," "Toe," and "Tsiao," too, thereby implicating innocent people in the terrorist search. Assuming a human analyst could correctly separate all these and wanted to correct the Doe/Do/Doh databases, there are still two problems. First, the analyst might not have access to the original databases held by other agencies. Even if the analyst could get to the originals, the analyst would probably never learn where else these original databases had already been copied.

One important goal of databases is to have a record in one place so that one correction serves all uses. With data mining, a result is an aggregate from multiple databases. There is no natural way to work backward from the result to the amalgamated databases to find and correct errors.

Using Comparable Data

Data semantics is another important consideration when mining for data. Consider two geographical databases with data on family income. Except one database has income by dollar, and the other has the data in thousands of dollars. Even if the field names are the same, combining the raw data would result in badly distorted statistics. Consider another attribute rated high/medium/low in one database and on a numerical scale of 1 to 5 in another. Should high/medium/low be treated as 1/3/5? Even if analysts use that transformation, computing with some 3-point and some 5-point precision reduces the quality of the results. Or how can you meaningfully combine one database that has a particular attribute with another that does not?

Eliminating False Matches

Coincidence is not correlation or causation; because two things occur together does not mean either causes the other. Data mining tries to highlight nonobvious connections in data, but data mining applications often use fuzzy logic to find these connections. These approaches will generate both false positives (false matches) and missed connections (false negatives). We need to be sensitive to the inherent inaccuracy of data mining approaches and guard against putting too much trust in the output of a data mining application just because “the computer said so.” Correctness of results and correct interpretation of those results are major security issues for data mining.

6.5.2 Availability of Data

Interoperability among distinct databases is a third security issue for data mining. As we just described, databases must have compatible structure and semantics to make data mining possible. Missing or incomparable data can make data mining results incorrect, so perhaps a better alternative is not to produce a result. But no result is not the same as a result of no correlation. As with single databases, data mining applications must deal with multiple sensitivities. Trying to combine databases on an attribute with more sensitive values can lead to no data and hence no matches.

REFERENCES

1. J. Vaidya, B. Shafiq, W. Fan, D. Mehmood and D. Lorenzi,“ A Random Decision Tree Framework for Privacy-Preserving Data Mining,” IEEE Transactions on Dependable and Secure Computing, vol. 11, no. 5, pp. 399–411, 2014.
2. C.-H. Yeh, G. Lee, and C.-Y. Lin, “Robust Laser Speckle Authentication System through Data Mining Techniques,” IEEE Transactions on Industrial Informatics, vol. 11, no. 2, pp. 505–512, 2015.
3. S. Khan, A. Sharma, A. S. Zamani, and A. Akhtar, “Data Mining for Security Purpose & its Solitude Suggestions,” International Journal of Technology Enhancements and Emerging Engineering Research, vol. 1, no. 7, pp. 1–4, 2012.
4. Venugopal K R, K G Srinivasa and L M Patnaik,“Soft Computing for Data Mining Applications,” Springer, 2009.
5. R. J. Bayardo and R. Agrawal, “Data Privacy Through Optimal k-Anonymization,” 21st International Conference on Data Engineering (ICDE’05), pp. 217–228, 2005.
6. M. Siddiqui, M. C. Wang, and J. Lee, “Detecting Internet Worms Using Data Mining Techniques,” Journal of Systemics, Cybernetics and Informatics, vol. 6, no. 6, pp. 48–53, 2009.
7. M. S. Abadeh, J. Habibi, and C. Lucas, “Intrusion Detection using a Fuzzy Genetics-Based Learning Algorithm,” Journal of Network and Computer Applications, vol. 30, no. 1, pp. 414–428, 2007.
8. P Deepa Shenoy, Srinivasa K G, Venugopal K R and L M Patnaik, “Dynamic Association Rule Mining using Genetic Algorithms,” Intelligent Data Analysis, vol. 9, no. 5, pp. 439–453, 2005.

CHAPTER 7

SECURITY IN NETWORKS

INTRODUCTION

Network security is a broad term that covers a multitude of technologies, devices and processes. In its simplest term, it is a set of rules and configurations designed to protect the integrity, confidentiality and accessibility of computer networks and data using both software and hardware technologies. Every organization, regardless of size, industry or infrastructure, requires a degree of network security solutions in place to protect it from the ever-growing landscape of cyber threats in the wild today.

Today's network architecture is complex and is faced with a threat environment that is always changing and attackers that are always trying to find and exploit vulnerabilities. These vulnerabilities can exist in a broad number of areas, including devices, data, applications, users and locations. For this reason, there are many network security management tools and applications in use today that address individual threats and exploits and also regulatory non-compliance. When just a few minutes of downtime can cause widespread disruption and massive damage to an organization's bottom line and reputation, it is essential that these protection measures are in place.

7.1 NETWORK SECURITY BASICS

Network security is the practice of preventing and protecting against unauthorized intrusion into corporate networks. As a philosophy, it complements endpoint security, which focuses on individual devices; network security instead focuses on how those devices interact, and on the connective tissue between them.

Definitions are fine as top-level statements of intent. But how do you lay out a plan for implementing that vision? Stephen Northcutt wrote a primer on the basics of network security for

CSOonline over a decade ago, but we feel strongly that his vision of the three phases of network security is still relevant and should be the underlying framework for your strategy. In his telling, network security consists of:

- **Protection**: You should configure your systems and networks as correctly as possible
- **Detection**: You must be able to identify when the configuration has changed or when some network traffic indicates a problem
- **Reaction:** After identifying problems quickly, you must respond to them and return to a safe state as rapidly as possible

This, in short, is a *defense in depth* strategy. If there's one common theme among security experts, it's that relying on one single line of defense is dangerous, because any single defensive tool can be defeated by a determined adversary. Your network isn't a line or a point: it's a territory, and even if an attacker has invaded part of it, you still have the resources to regroup and expel them, if you've organized your defense properly.

7.1.1 Principles of Network Security

There are three principles within the concept of network security—confidentiality, integrity, and availability—which together are sometimes referred to as the "CIA triad." A network can only be considered secure when it has all three elements in play simultaneously.

Confidentiality works to keep sensitive data protected and sequestered away from where it can be accessed by the average user. This goes hand-in-hand with the principle of availability, which seeks to ensure that data and resources are kept accessible for those who are authorized to access them. Challenges to availability can include DDoS attacks or equipment failure. The principle of integrity seeks to protect information from intentional or accidental changes in order to keep the data reliable, accurate, and trustworthy.

Every decision made regarding network security should be working to further at least one of these principles. This means that MSPs need to ask if each decision will ensure that data is kept confidential, that its integrity will be protected, and that it will be made more easily available to those with authorization to access it.

Why are these network security concepts so important? Cyberattacks are on the rise, with a recent report from Positive Technologies showing that government and healthcare organizations are becoming prime targets for hackers. The report also shows the goal of more than half of cybercrimes is data theft, and that financial gain was the motivation behind 42% of cyberattacks against individuals—and behind 30% of cyberattacks against organizations.

As our world becomes increasingly digitized, we rely more and more on the internet and networks to function. This in turn requires that the internet and networks provide us with reliable and secure service.

However, as more of our personal and sensitive data is stored in electronic repositories and archives, hackers are turning their attention to networked systems. For this reason, it is imperative that MSPs and security support personnel offer customers robust security systems that protect data from various threat vectors.

7.1.2 Network Security Methods

To implement this kind of defense in depth, there are a variety of specialized techniques and types of network security you will want to roll out. Cisco, a networking infrastructure company, uses the following schema to break down the different types of network security, and while some of it is informed by their product categories, it's a useful way to think about the different ways to secure a network.

- **Access control:** You should be able to block unauthorized users and devices from accessing your network. Users that are permitted network access should only be able to work with the limited set of resources for which they've been authorized.
- **Anti-malware:** Viruses, worms, and trojans by definition attempt to spread across a network, and can lurk dormant on infected machines for days or weeks. Your security effort should do its best to prevent initial infection and also root out malware that does make its way onto your network.
- **Application security:** Insecure applications are often the vectors by which attackers get access to your network. You need to employ hardware, software, and security processes to lock those apps down.
- **Behavioral analytics:** You should know what normal network behavior looks like so that you can spot anomalies or breaches as they happen.
- **Data loss prevention:** Human beings are inevitably the weakest security link. You need to implement technologies and processes to ensure that staffers don't deliberately or inadvertently send sensitive data outside the network.
- **Email security:** Phishing is one of the most common ways attackers gain access to a network. Email security tools can block both incoming attacks and outbound messages with sensitive data.
- **Firewalls:** Perhaps the granddaddy of the network security world, they follow the rules you define to permit or deny traffic at the border between your network and the internet, establishing a barrier between your trusted zone and the wild west outside. They don't preclude the need for a defense-in-depth strategy, but they're still a must-have.
- **Intrusion detection and prevention:** These systems scan network traffic to identify and block attacks, often by

correlating network activity signatures with databases of known attack techniques.

- **Mobile device and wireless security:** Wireless devices have all the potential security flaws of any other networked gadget — but also can connect to just about any wireless network anywhere, requiring extra scrutiny.
- **Network segmentation:** Software-defined segmentation puts network traffic into different classifications and makes enforcing security policies easier.
- **Security information and event management (SIEM):** These products aim to automatically pull together information from a variety of network tools to provide data you need to identify and respond to threats.
- **VPN:** A tool (typically based on IPsec or SSL) that authenticates the communication between a device and a secure network, creating a secure, encrypted "tunnel" across the open internet.
- **Web security:** You need to be able to control internal staff's web use in order to block web-based threats from using browsers as a vector to infect your network.

7.1.3 Network Security and The Cloud

More and more enterprises are offloading some of their computing needs to cloud service providers, creating hybrid infrastructures where their own internal network has to interoperate seamlessly — and securely — with servers hosted by third parties. Sometimes this infrastructure itself is a self-contained network, which can be either physical (several cloud servers working together) or virtual (multiple VM instances running together and "networking" with each other on a single physical server).

To handle the security aspects, many cloud vendors establish centralized security control policies on their own platform. However, the trick here is that those security systems won't always match up with your policies and procedures for your internal networks, and this mismatch can add to the workload for network security pros. There are a variety of tools and techniques available to you that can help ease some of this worry, but the truth is that this area is still in flux and the convenience of the cloud can mean network security headaches for you.

7.1.4 Network Security Software

To cover all those bases, you'll need a variety of software and hardware tools in your toolkit. Most venerable, as we've noted, is the firewall. The drumbeat has been to say that the days when a firewall was the sum total of your network security is long gone, with defense in depth needed to fight threats behind (and even in front of) the firewall. Indeed, it seems that one of the nicest things you can say about a firewall product in a review is that calling it a firewall is selling it short.

But firewalls can't be jettisoned entirely. They're properly one element in your hybrid defense-in-depth strategy. And as eSecurity Planet explains, there are a number of different firewall types, many of which map onto the different types of network security we covered earlier:

- Network firewalls
- Next-generation firewalls
- Web application firewalls
- Database firewalls
- Unified threat management
- Cloud firewalls
- Container firewalls
- Network segmentation firewalls

Beyond the firewall, a network security pro will deploy a number of tools to keep track of what's happening on their networks. Some of these tools are corporate products from big vendors, while others come in the form of free, open source utilities that sysadmins have been using since the early days of Unix. A great resource is SecTools.org, which maintains a charmingly Web 1.0 website that keeps constant track of the most popular network security tools, as voted on by users. Top categories include:

- Packet sniffers, which give deep insight into data traffic
- Vulnerability scanners like Nessus
- Intrusion detection and prevention software, like the legendary Snort
- Penetration testing software

That last category might raise some eyebrows — after all, what's penetration testing if not an attempt to hack into a network? But part of making sure you're locked down involves seeing how hard or easy it is to break in, and pros know it; ethical hacking is an important part of network security. That›s why you›ll see tools like Aircrack — which exists to sniff out wireless network security keys — alongside staid corporate offerings that cost tens of thousands of dollars on the SecTools.org list.

In an environment where you need to get many tools to work together, you might also want to deploy SIEM software, which we touched on above. SIEM products evolved from logging software, and analyze network data collected by a number of different tools to detect suspicious behavior on your network.

7.2 THREATS IN NETWORKS

Network threats are unlawful or malicious activities that intend to take advantage of network vulnerabilities. The goal is to breach, harm, or sabotage the information or data valuable to the company. Malicious actors also attack networks to gain unauthorized access and manipulate the same according to their intentions.

Regardless of the type of network security threat, there are different motives for executing network attacks and they are often malicious. Individuals, businesses, and nations have different reasons for executing an attack. The most common are hacktivism, extortion, cyber warfare, business feuds, and personal reasons.

The most common network security threats are Computer viruses, Computer worms, Trojan horse, SQL injection attack, DOS and DDOS attack, Rootkit, Rogue security software, Phishing, Adware and spyware, and Man-in-the-middle attacks. Computer viruses are the most common network threats for everyday internet users, with approximately 33% of PCs being affected by malware, most of which are viruses.

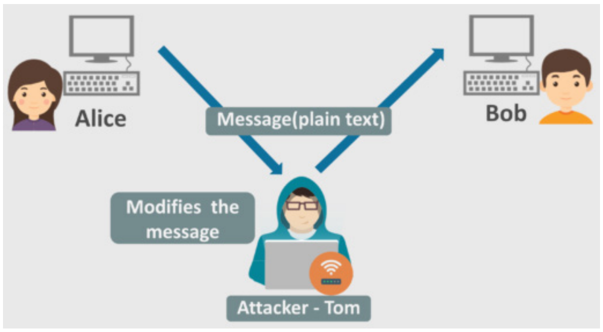
7.2.1 Network Security Attack

A *network attack* can be defined as any method, process, or means used to maliciously attempt to compromise network security. Network security is the process of preventing network attacks across a given network infrastructure, but the techniques and methods used by the attacker further distinguish whether the attack is an active cyber attack, a passive type attack, or some combination of the two.

Let's consider a simple network attack example to understand the difference between active and passive attack.

Active Attacks

An active attack is a network exploit in which attacker attempts to make changes to data on the target or data en route to the target.



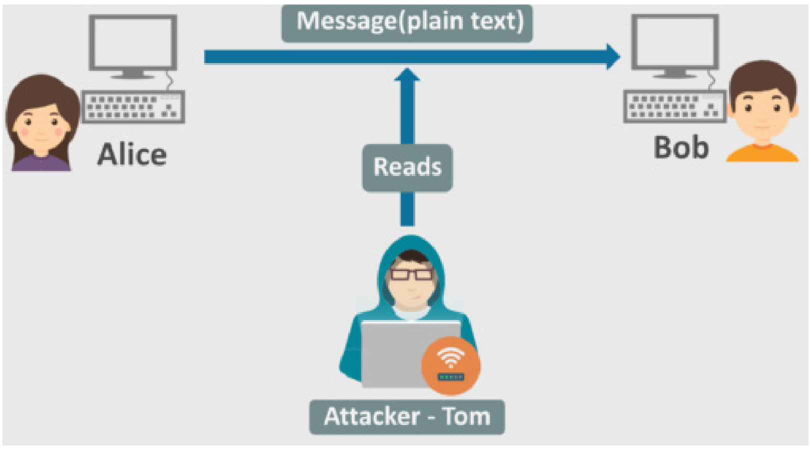
Meet Alice and Bob. Alice wants to communicate to Bob but distance is a problem. So, Alice sends an electronic mail to Bob via a network which is not secure against attacks. There is another person, Tom, who is on the same network as Alice and Bob. Now, as the data flow is open to everyone on that network, Tom alters some portion of an authorized message to produce an unauthorized effect. For example, a message meaning "Allow BOB to read confidential file X" is modified as "Allow Smith to read confidential file X".

Active network attacks are often aggressive, blatant attacks that victims immediately become aware of when they occur. Active attacks are highly malicious in nature, often locking out users, destroying memory or files, or forcefully gaining access to a targeted system or network.

Passive Attacks

A passive attack is a network attack in which a system is monitored and sometimes scanned for open ports and vulnerabilities, but does not affect system resources.

Let's consider the example we saw earlier:

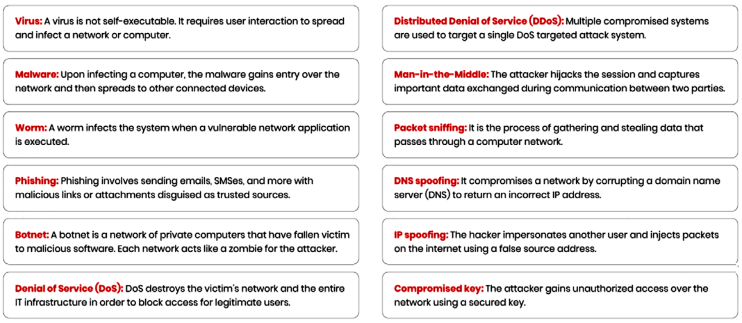


Alice sends an electronic mail to Bob via a network which is not secure against attacks. Tom, who is on the same network as Alice and Bob, monitors the data transfer that is taking place between Alice and Bob. Suppose, Alice sends some sensitive information like bank account details to Bob as plain text. Tom can easily access the data and use the data for malicious purposes.

So, the purpose of the passive attack is to gain access to the computer system or network and to collect data without detection.

So, network security includes implementing different hardware and software techniques necessary to guard underlying network architecture. With the proper network security in place, you can detect emerging threats before they infiltrate your network and compromise your data.

Active and passive network security attacks are further divided according to the methods used. The most prominent ones are:



Virus: A virus is not self-executable. It requires user interaction to spread and infect a network or computer.

Malware: Upon infecting a computer, the malware gains entry over the network and then spreads to other connected devices.

Worm: A worm infects the system when a vulnerable network application is executed.

Phishing: Phishing involves sending emails, SMSes, and more with malicious links or attachments disguised as trusted sources.

Botnet: A botnet is a network of private computers that have fallen victim to malicious software. Each network acts like a zombie for the attacker.

Denial of Service (DoS): DoS destroys the victim's network and the entire IT infrastructure in order to block access for legitimate users.

Distributed Denial of Service (DDoS): Multiple compromised systems are used to target a single DoS targeted attack system.

Man-in-the-Middle: The attacker hijacks the session and captures important data exchanged during communication between two parties.

Packet sniffing: It is the process of gathering and stealing data that passes through a computer network.

DNS spoofing: It compromises a network by corrupting a domain name server (DNS) to return an incorrect IP address.

IP spoofing: The hacker impersonates another user and injects packets on the internet using a false source address.

Compromised key: The attacker gains unauthorized access over the network using a secured key.

7.2.2 Identifying Your Network Security Threats

If you want to defend your network security effectively, you need a Certified Network Defender that can properly identify and mitigate the vulnerabilities within your network.

Enable your network visibility

The first step for preparing your network defender and other members of your security team to identify network threats and vulnerabilities is to enable your whole network visibility. The only way you can detect a threat is when it is visible. You can use the existing structures on your network devices to achieve visibility.

You can also design a strategic network diagram to exemplify your packet flows and the possible places where you can activate security procedures that will identify, categorize, and alleviate the threat.

Set up computer and network access

You need to construct your computer and network access to control who can access your network and the level of access they can have. Not every user should be given access to the whole network. Your network security policies will determine the appropriate ways to protect treasured assets, evaluate potential risks, lessen vulnerability channels, and craft a recovery plan in case of an incident.

Firewall configuration

Setting up a network firewall thwarts unauthorized access and internet-based attacks from dispersing into your computer networks. Your network firewall oversees the flow of computer data traffic permitted to traverse your network. They can also obstruct reconnaissance assaults, including IP scanning or port sweeps. Your internal firewall can restrict this, but you need to configure it.

Limit access to updates and installations

Malicious hackers can penetrate your computer network through out-of-date software for antivirus, operating systems, device drivers, firmware, and other endpoint mechanisms. Access control in network security is critical. Network defenders can mitigate the risk of random assaults by restricting the number of people who can install or update software. Your IT team should only be allowed to activate updates and installations only via their admin access.

7.3 NETWORK SECURITY CONTROLS

Network Security Controls are used to ensure the confidentiality, integrity, and availability of the network services. These security controls are either technical or administrative safeguards implemented to minimize the security risk. To reduce the risk of a network being compromised, an adequate network security requires implementing a proper combination of **network security** controls.

These network security controls include:

- Access Control
- Identification
- Authentication
- Authorization
- Accounting
- Cryptography
- Security Policy

These controls help organizations with implementing strategies for addressing network security concerns. The multiple layers of network security controls along with the network should be used to minimize the risks of attack or compromise. The overlapping use of these controls ensures defense in depth network security.

7.3.1 Access Control

Access control is a method for reducing the risk of data from being affected and to save the organization's crucial data by providing limited access of computer resources to users. The mechanism grants access to system resources to read, write, or execute to the user based on the access permissions and their associated roles. The crucial aspect of implementing access control is to maintain the integrity, confidentiality, and availability of the information.

An access control system includes:

- File permissions such as create, read, edit or delete
- Program permissions such as the right to execute a program
- Data rights such as the right to retrieve or update information in a database

There are two types of access controls:

Physical and logical. The physical access controls the access to buildings, physical IT assets, etc. The logical access controls the access to networks and data.

In general, access control provides essential services like authorization, identification, authentication, access permissions and accountability.

- Authorization determines the action a user can perform
- Identification and authentication identify and permit only authorized users to access the systems
- The access permissions determine approvals or permissions provided to a user to access a system and other resources
- Accountability categorizes the actions performed by a user

Access Control Terminology

The following terminologies are used to define access control on specific resources:

- **Subject:** A subject may be defined as a user or a process, which attempts to access the objects. Further, subjects are those entities that perform certain actions on the system.
- **Object:** An object is an explicit resource on which access restriction is imposed. The Access controls implemented on the objects further control the actions performed by the user. For example, files or hardware devices.

- **Reference Monitor:** It monitors the restrictions imposed according to certain access control rules. Reference monitor implements a set of rules on the ability of the subject to perform certain actions on the object.
- **Operation:** An operation is an action performed by the subject on the object. A user trying to delete a file is an example of an operation. Here, the user is the subject. Delete refers to the operation and file is the object.

Access Control Principles

Access control principles deal with restricting or allowing the access controls to users or processes. The principle includes the server receiving a request from the user and authenticating the user with the help of an Access Control Instruction (ACO). The server can either allow or deny the user to perform any actions like read, write, access files, etc.

Network Security Control is a part ***Certified Ethical Hacking v10(CEH v10)*** *training you learn the cyber security attacks and their impact.*

Access controls enable users to gain access to the entire directory, subtree of the directory and another specific set of entries and attribute values in the directory. It is possible to set permission values to a single user or a group of users. The directory and attribute values contain the access control instructions. Access control function uses an authorization database, maintained by the security admin, to check the authorization details of the requesting user.

Types of Access Control

Types of access control between how a subject can access an object. The policy for determining the mechanism uses access control technologies and security.

Discretionary Access Control (DAC)

Discretionary access controls determine the access controls taken by any possessor of an object in order to decide the access controls of the subjects on those objects. The other name for DAC is a need-to-know access model. It permits the user, who is granted access to information, to decide how to protect the information and the level of sharing desired. Access to files is restricted to users and groups based upon their identity and the groups to which the users belong.

Mandatory Access Control (MAC)

The mandatory access controls determine the usage and access policies of the users. Users can access a resource only if that particular user has the access rights to that resource. MAC finds its application in the data marked as highly confidential. The **network** administrators impose MAC, depending on the operating system and security kernel. It does not permit the end user to decide who can access the information, and does not permit the user to pass privileges to other users as the access could then be circumvented.

Role Based Access Control (RBAC)

In role based access control, the access permissions are available based on the access policies determined by the system. The access permissions are out of user control, which means that users cannot amend the access policies created by the system. Users Identification, Authentication, Authorization and Accounting

7.3.2 Identification

Identification deals with confirming the identity of a user, process, or device accessing the network. User identification is the most common technique used in authenticating the users in the network and applications. Users have a unique User ID, which helps in identifying them.

The authentication process includes verifying a user ID and a password. Users need to provide both the credentials in order to gain access to the network. The network administrators provide access controls and permissions to various other services depending on the user ID's.

Example: Username, Account Number, etc.

7.3.3 Authentication

Authentication refers to verifying the credentials provided by the user while attempting to connect to a network. Both wired and wireless networks perform authentication of users before allowing them to access the resources in the network. A typical user authentication consists of a user ID and a password. The other forms of authentication are authenticating a website using a digital certificate, comparing the product and the label associated with it.

Example: Password, PIN, etc.

7.3.4 Authorization

Authorization refers to the process of providing permission to access the resources or perform an action on the network. Network administrators can decide the access permissions of users on a multi-user system. They even decide the user privileges. The mechanism of authorization can allow the network administrator to create access permissions for users as well as verify the access permissions created for each user.

In logical terms, authorization succeeds authentication. But, the type of resources or perform an action on the network. Network administrators can decide the access permissions of users on a multi-user system. They even decide the user privileges. The mechanism of authorization can allow the network administrator to create access permissions for users as well as verify the access permissions created for each user.

In logical terms, authorization succeeds authentication. But, the type of authentication required for authorization varies. However, there are cases that do not require any authorization of the users requesting for a service.

Example: A user can only read the file but not write to or delete it.

7.3.5 Accounting

User accounting refers to tracking the actions performed by the user on a network. This includes verifying the files accessed by the user, functions like alteration or modification of the files or data. It keeps track of who, when, how the users access the network. It helps in identifying authorized and unauthorized actions.

7.4 FIREWALLS

A firewall is a network security device that monitors incoming and outgoing network traffic and decides whether to allow or block specific traffic based on a defined set of security rules.

Firewalls have been a first line of defense in network security for over 25 years. They establish a barrier between secured and controlled internal networks that can be trusted and untrusted outside networks, such as the Internet.

A firewall can be hardware, software, or both.

A Firewall is a necessary part of any security architecture and takes the guesswork out of host level protections and entrusts them to your network security device. Firewalls, and especially Next Generation Firewalls, focus on blocking malware and application-layer attacks, along with an integrated intrusion prevention system (IPS), these Next Generation Firewalls can react quickly and seamlessly to detect and react to outside attacks across the whole network. They can set policies to better defend your network and carry out quick assessments to detect invasive or suspicious activity, like malware, and shut it down.

7.4.1 Firewall History

Firewalls have existed since the late 1980's and started out as packet filters, which were networks set up to examine packets, or bytes, transferred between computers. Though packet filtering firewalls are still in use today, firewalls have come a long way as technology has developed throughout the decades.

- Gen 1 Virus
  - Generation 1, Late 1980's, virus attacks on stand-alone PC's affected all businesses and drove anti-virus products.
- Gen 2 Networks
  - Generation 2, Mid 1990's, attacks from the internet affected all business and drove creation of the firewall.
- Gen 3 Applications
  - Generation 3, Early 2000's, exploiting vulnerabilities in applications which affected most businesses and drove Intrusion Prevention Systems Products (IPS).
- Gen 4 Payload
  - Generation 4, Approx. 2010, rise of targeted, unknown, evasive, polymorphic attacks which affected most businesses and drove anti-bot and sandboxing products.

- Gen 5 Mega
  - Generation 5, Approx. 2017, large scale, multi-vector, mega attacks using advance attack tools and is driving advance threat prevention solutions.

Back in 1993, Check Point CEO Gil Shwed introduced the first stateful inspection firewall, FireWall-1. Fast forward twenty-seven years, and a firewall is still an organization's first line of defense against cyber attacks. Today's firewalls, including Next Generation Firewalls and Network Firewalls support a wide variety of functions and capabilities with built-in features, including:

- Network Threat Prevention
- Application and Identity-Based Control
- Hybrid Cloud Support
- Scalable Performance

7.4.2 Uses

Firewalls are used in both corporate and consumer settings. Modern organizations incorporate them into a security information and event management (SIEM) strategy along with other cybersecurity devices. They may be installed at an organization's network perimeter to guard against external threats, or within the network to create segmentation and guard against insider threats.

In addition to immediate threat defense, firewalls perform important logging and audit functions. They keep a record of events, which can be used by administrators to identify patterns and improve rule sets. Rules should be updated regularly to keep up with ever-evolving cybersecurity threats. Vendors discover new threats and develop patches to cover them as soon as possible.

In a single home network, a firewall can filter traffic and alert the user to intrusions. They are especially useful for always-on connections, like Digital Subscriber Line (DSL) or cable modem, because those connection types use static IP addresses. They are often used alongside to antivirus applications. Personal firewalls, unlike corporate ones, are usually a single product as opposed to a collection of various products. They may be software or a device with firewall firmware embedded. Hardware/firmware firewalls are often used for setting restrictions between in-home devices.

7.4.3 How does a firewall work?

A firewall decides which network traffic is allowed to pass through and which traffic is deemed dangerous. It essentially works by filtering out the good from the bad, or the trusted from the untrusted. However, before we go into detail, we must first understand the structure of web-based networks before explaining how a firewall operates to filter between them.

Firewalls are intended to secure the private networks and the endpoint devices within, known as network hosts.

Network hosts are devices that «talk» with other hosts on the network. They send and receive between internal networks, as well as outbound and inbound between external networks.

Your computers and other endpoint devices use networks to access the internet — and each other. However, the internet is segmented into sub-networks or 'subnets' for security and privacy.

The basic subnet segments are as follows:

- **External public networks** typically refer to the public/global internet or various extranets.
- **Internal private network** defines a home network, corporate intranets, and other «closed» networks.
- **Perimeter networks** detail border networks made of *bastion hosts* — computer hosts dedicated with hardened security that are ready to endure an external attack. As a secured buffer between internal and external networks, these can also be used to house any external-facing services provided by the internal network (i.e., servers for web, mail, FTP, VoIP, etc.). These are more

secure than external networks but less secure than the internal. *These are not always present in simpler networks like home networks but may often be used in organizational or national intranets.*

Screening routers are specialized gateway computers placed on a network to segment it. They are known as house firewalls on the network-level. The two most common segment models are the screened host firewall and the screened subnet firewall.

- **Screened host firewalls** use a single screening router between the external and internal networks, known as the choke router. These networks are the two subnets of this model.
- **Screened subnet firewalls** use two screening routers—one known as an *access router* between the external and perimeter network, and another labeled as the *choke router* between the perimeter and internal network. This creates three subnets, respectively.

As mentioned earlier, both the network perimeter and host machines themselves can house a firewall. To do this, it is placed between a single computer and its connection to a private network.

- **Network firewalls** involve the application of one or more firewalls between external networks and internal private networks. These regulate inbound and outbound network traffic, separating external public networks—like the global internet—from internal networks like home Wi-Fi networks, enterprise intranets, or national intranets. Network firewalls may come in the form of any of the following appliance types: dedicated hardware, software, and virtual.
- **Host firewalls** or ‹software firewalls› involve the use of firewalls on individual user devices and other private network endpoints as a barrier between devices within the network. These devices, or hosts, receive customized regulation of traffic to and from specific computer applications. Host firewalls may run on local devices as an operating system service or an endpoint security

application. Host firewalls can also dive deeper into web traffic, filtering based on HTTP and other networking protocols, allowing the management of what content arrives at your machine, rather than just where it comes from.

Network firewalls require configuration against a broad scope of connections, whereas host firewalls can be tailored to fit each machine's needs. However, host firewalls require more effort to customize, meaning that network-based are ideal for a sweeping control solution. But the use of both firewalls in both locations simultaneously is ideal for a multi-layer security system.

Filtering traffic via a firewall makes use of pre-set or dynamically learned rules for allowing and denying attempted connections. These rules are how a firewall regulates the web traffic flow through your private network and private computer devices. Regardless of type, all firewalls may filter by some blend of the following:

- **Source:** Where an attempted connection is being made from.
- **Destination:** Where an attempted connection is intended to go.
- **Contents:** What an attempted connection is trying to send.
- **Packet protocols:** What «language» an attempted connection is speaking to carry its message. Among the networking protocols that hosts use to «talk» with each other, TCP/IP is the primary protocol used to communicate across the internet and within intranet/sub-networks. Other standard protocols include IMCP and UDP.
- **Application protocols:** Common protocols include HTTP, Telnet, FTP, DNS, and SSH.

Source and destination are communicated by internet protocol (IP) addresses and ports. *IP addresses* are unique device names for each host. *Ports* are a sub-level of any given source and destination host device, similar to office rooms within a larger building. Ports are typically assigned specific purposes, so certain protocols and IP addresses utilizing uncommon ports or disabled ports can be a concern.

By using these identifiers, a firewall can decide if a data packet attempting a connection is to be discarded—silently or with an error reply to the sender—or forwarded.

7.4.4 What does firewall security do?

The concept of a network security firewall is meant to narrow the attack surface of a network to a single point of contact. Instead of every host on a network being directly exposed to the greater internet, all traffic must first contact the firewall. Since this also works in reverse, the firewall can filter and block non-permitted traffic, in or out. Also, firewalls are used to create an audit trail of attempted network connections for better security awareness.

Since traffic filtering can be a rule set established by owners of a private network, this creates custom use cases for firewalls. Popular use cases involve managing the following:

- **Infiltration from malicious actors:** Undesired connections from an oddly behaving source can be blocked. This can prevent eavesdropping and advanced persistent threats (APTs).
- **Parental controls:** Parents can block their children from viewing explicit web content.
- **Workplace web browsing restrictions:** Employers can prevent employees from using company networks to access unproductive services and content, such as social media.
- **Nationally controlled intranet:** National governments can block internal residents› access to web content and services that are potentially dissident to a nation›s leadership or its values.

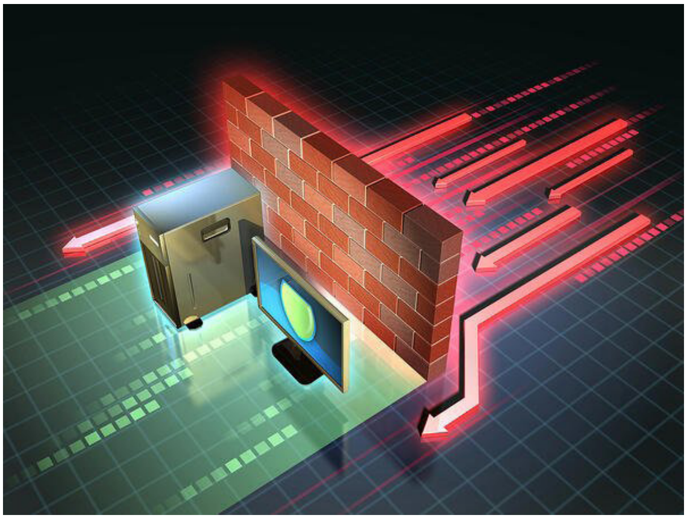
Notably, firewalls are not very effective at the following:

- **Identifying exploits of legitimate networking processes:** Firewalls do not anticipate human intent, so they cannot determine if a "legitimate" connection is intended for malicious purposes. For example, IP address fraud (IP spoofing) occurs because firewalls don't validate the source and destination IPs.
- **Prevent connections that do not pass through the firewall:** Network-level firewalls alone will not stop malicious internal activity. Internal firewalls such as host-based ones will need to be present in addition to the perimeter firewall, to partition your network and slow the movement of internal «fires.»
- **Provide adequate protection against viruses:** While connections carrying malicious code can be halted if not whitelisted, a connection deemed acceptable can still deliver these threats into your network. If a firewall overlooks a connection as a result of being misconfigured or exploited, an antivirus protection suite will still be needed to clean up any malware or viruses that enter.

7.4.5 Types of Firewalls

Firewall types can be divided into several different categories based on their general structure and method of operation. Here are eight types of firewalls:

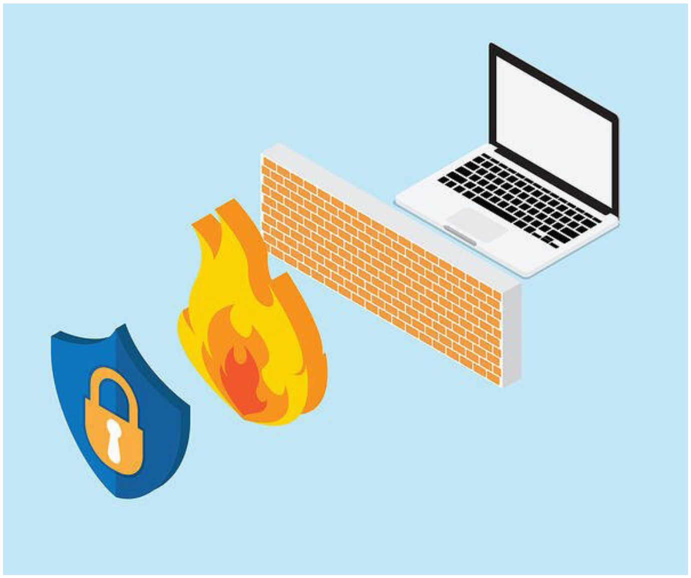
- Packet-filtering firewalls
- Circuit-level gateways
- Stateful inspection firewalls
- Application-level gateways (a.k.a. proxy firewalls)
- Next-gen firewalls
- Software firewalls
- Hardware firewalls
- Cloud firewalls



How do these firewalls work? And, which ones are the best for your business' cybersecurity needs?

Here are a few brief explainers:

Packet-Filtering Firewalls



As the most "basic" and oldest type of firewall architecture, packet-filtering firewalls basically create a checkpoint at a traffic router or switch. The firewall performs a simple check of the data packets coming through the router—inspecting information such as the destination and origination IP address, packet type, port number, and other surface-level information without opening up the packet to inspect its contents.

If the information packet doesn't pass the inspection, it is dropped.

The good thing about these firewalls is that they aren't very resource-intensive. This means they don't have a huge impact on system performance and are relatively simple. However, they're also relatively easy to bypass compared to firewalls with more robust inspection capabilities.

Circuit-Level Gateways

As another simplistic firewall type that is meant to quickly and easily approve or deny traffic without consuming significant computing resources, circuit-level gateways work by verifying the transmission control protocol (TCP) handshake. This TCP handshake check is designed to make sure that the session the packet is from is legitimate.

While extremely resource-efficient, these firewalls do not check the packet itself. So, if a packet held malware, but had the right TCP handshake, it would pass right through. This is why circuit-level gateways are not enough to protect your business by themselves.

Stateful Inspection Firewalls

These firewalls combine both packet inspection technology and TCP handshake verification to create a level of protection greater than either of the previous two architectures could provide alone.

However, these firewalls do put more of a strain on computing resources as well. This may slow down the transfer of legitimate packets compared to the other solutions.

Proxy Firewalls (Application-Level Gateways/Cloud Firewalls)

Proxy firewalls operate at the application layer to filter incoming traffic between your network and the traffic source—hence, the name "application-level gateway." These firewalls are delivered via a cloud-based solution or another proxy device. Rather than letting traffic connect directly, the proxy firewall first establishes a connection to the source of the traffic and inspects the incoming data packet.

This check is similar to the stateful inspection firewall in that it looks at both the packet and at the TCP handshake protocol. However, proxy firewalls may also perform deep-layer packet inspections, checking the actual contents of the information packet to verify that it contains no malware.

Once the check is complete, and the packet is approved to connect to the destination, the proxy sends it off. This creates an extra layer of separation between the "client" (the system where the packet originated) and the individual devices on your network—obscuring them to create additional anonymity and protection for your network.

If there's one drawback to proxy firewalls, it's that they can create significant slowdown because of the extra steps in the data packet transferal process.

Next-Generation Firewalls

Many of the most recently-released firewall products are being touted as "next-generation" architectures. However, there is not as much consensus on what makes a firewall truly next-gen.

Some common features of next-generation firewall architectures include deep-packet inspection (checking the actual contents of the data packet), TCP handshake checks, and surface-level packet inspection. Next-generation firewalls may include other technologies as well, such as intrusion prevention systems (IPSs) that work to automatically stop attacks against your network.

The issue is that there is no one definition of a next-generation firewall, so it's important to verify what specific capabilities such firewalls have before investing in one.

Software Firewalls

Software firewalls include any type of firewall that is installed on a local device rather than a separate piece of hardware (or a cloud server). The big benefit of a software firewall is that it's highly useful for creating defense in depth by isolating individual network endpoints from one another.

However, maintaining individual software firewalls on different devices can be difficult and time-consuming. Furthermore, not every device on a network may be compatible with a single software firewall, which may mean having to use several different software firewalls to cover every asset.

Hardware Firewalls

Hardware firewalls use a physical appliance that acts in a manner similar to a traffic router to intercept data packets and traffic requests before they're connected to the network's servers. Physical appliance-based firewalls like this excel at perimeter security by making sure malicious traffic from outside the network is intercepted before the company's network endpoints are exposed to risk.

The major weakness of a hardware-based firewall, however, is that it is often easy for insider attacks to bypass them. Also, the actual capabilities of a hardware firewall may vary depending on the manufacturer—some may have a more limited capacity to handle simultaneous connections than others, for example.

Cloud Firewalls

Whenever a cloud solution is used to deliver a firewall, it can be called a cloud firewall, or firewall-as-a-service (FaaS). Cloud firewalls are considered synonymous with proxy firewalls by many, since a cloud server is often used in a proxy firewall setup (though the proxy doesn't necessarily *have* to be on the cloud, it frequently is).

The big benefit of having cloud-based firewalls is that they are very easy to scale with your organization. As your needs grow, you can add additional capacity to the cloud server to filter larger traffic loads. Cloud firewalls, like hardware firewalls, excel at perimeter security.

7.4.6 Firewall Examples

In practice, a firewall has been a topic of both praise and controversy due to its real-world applications. While there is a decorated history of firewall accomplishments, this security type must be implemented correctly to avoid exploits. Additionally, firewalls have been known to be used in ethically questionable ways.

Great Firewall of China, internet censorship

Since 1998, China has had internal firewall frameworks in place to create its carefully monitored intranet. By nature, firewalls allow for the creation of a customized version of the global internet within a nation. They accomplish this by preventing select services and info from being used or accessed within this national intranet.

National surveillance and censorship allow for the ongoing suppression of free speech while maintaining its government's image. Furthermore, China's firewall allows its government to limit internet services to local companies. This makes control over things like search engines and email services much easier to regulate in favor of the government's goals.

Naturally, China has seen an ongoing internal protest against this censorship. The use of virtual private networks and proxies to get past the national firewall has allowed many to voice their dissatisfaction.

COVID-19 U.S. federal agency compromised due to remote work weaknesses

In 2020, a misconfigured firewall was just one of many security weaknesses that led to an anonymous United States federal agency›s breach.

It is believed that a nation-state actor exploited a series of vulnerabilities in the U.S. agency's cybersecurity. Among the many cited issues with their security, the firewall in-use had many outbound ports that were inappropriately open to traffic. Alongside being maintained poorly, the agency's network likely had new challenges with remote work. Once in the network, the attacker behaved in ways that show clear intent to move through any other open pathways to other agencies. This type of effort puts not only the infiltrated agency at risk of a security breach but many others as well.

U.S. power grid operator’s unpatched firewall exploited

In 2019, a United States power grid operations provider was impacted by a Denial-of-Service (DoS) vulnerability that hackers exploited. Firewalls on the perimeter network were stuck in a reboot exploit loop for roughly ten hours.

It was later deemed to be the result of a known-but-unpatched firmware vulnerability in the firewalls. A standard operating procedure for checking updates before implementation hadn’t been put into place yet causing delays in updates and an inevitable security issue. Fortunately, the security issue did not lead to any significant network penetration.

These events are another strong indicator of the importance of regular software updates. Without them, firewalls are yet another network security system that can be exploited.

7.4.7 How to Use Firewall Protection

Proper setup and maintenance of your firewall are essential to keep your network and devices protected.

Here are some tips to guide your firewall security practices:

- **Always update your firewalls as soon as possible:** Firmware patches keep your firewall updated against any newly discovered vulnerabilities. Personal and home firewall users can usually safely update immediately. Larger organizations may need to check configuration and compatibility across their network first. However, everyone should have processes in place to update promptly.
- **Use antivirus protection:** Firewalls alone are not designed to stop viruses and other infections. These

may get past firewall protections, and you'll need a security solution that's designed to disable and remove them. Kaspersky Total Security can protect you across your personal devices, and our many business security solutions can safeguard any network hosts you›ll seek to keep clean.

- **Limit accessible ports and hosts with a whitelist:** Default to connection denial for inbound traffic. Limit inbound and outbound connections to a strict whitelist of trusted IP addresses. Reduce user access privileges to necessities. It is easier to stay secure by enabling access when needed than to revoke and mitigate damage after an incident.
- **Segmented network:** Lateral movement by malicious actors is a clear danger that can be slowed by limiting cross-communication internally.
- **Have active network redundancies to avoid downtime:** Data backups for network hosts and other essential systems can prevent data loss and productivity during an incident.

7.5 INTRUSION DETECTION SYSTEMS

An Intrusion Detection System (IDS) is a system that monitors network traffic for suspicious activity and issues alerts when such activity is discovered. It is a software application that scans a network or a system for harmful activity or policy breaching. Any malicious venture or violation is normally reported either to an administrator or collected centrally using a security information and event management (SIEM) system. A SIEM system integrates outputs from multiple sources and uses alarm filtering techniques to differentiate malicious activity from false alarms.

Although intrusion detection systems monitor networks for potentially malicious activity, they are also disposed to false alarms. Hence, organizations need to fine-tune their IDS products when they first install them. It means properly setting up the intrusion detection systems to recognize what normal traffic on the network looks like as compared to malicious activity.

Intrusion prevention systems also monitor network packets inbound the system to check the malicious activities involved in it and at once sends the warning notifications.

7.5.1 Classification of Intrusion Detection Systems

Intrusion detection systems are designed to be deployed in different environments. And like many cybersecurity solutions, an IDS can either be host-based or network-based.

- **Network Intrusion Detection System (NIDS):** Network intrusion detection systems (NIDS) are set up at a planned point within the network to examine traffic from all devices on the network. It performs an observation of passing traffic on the entire subnet and matches the traffic that is passed on the subnets to the collection of known attacks. Once an attack is identified or abnormal behavior is observed, the alert can be sent to the administrator. An example of an NIDS is installing it on the subnet where firewalls are located in order to see if someone is trying crack the firewall.

- **Host Intrusion Detection System (HIDS):** Host intrusion detection systems (HIDS) run on independent hosts or devices on the network. A HIDS monitors the incoming and outgoing packets from the device only and will alert the administrator if suspicious or malicious activity is detected. It takes a snapshot of existing system files and compares it with the previous snapshot. If the analytical system files were edited or deleted, an alert is sent to the administrator to investigate. An example of HIDS usage can be seen on mission critical machines, which are not expected to change their layout.
- **Protocol-based Intrusion Detection System (PIDS):** Protocol-based intrusion detection system (PIDS) comprises of a system or agent that would consistently resides at the front end of a server, controlling and interpreting the protocol between a user/device and the server. It is trying to secure the web server by regularly monitoring the HTTPS protocol stream and accept the related HTTP protocol. As HTTPS is un-encrypted and before instantly entering its web presentation layer then this system would need to reside in this interface, between to use the HTTPS.
- **Application Protocol-based Intrusion Detection System (APIDS):** Application Protocol-based Intrusion Detection System (APIDS) is a system or agent that generally resides within a group of servers. It identifies the intrusions by monitoring and interpreting the communication on application specific protocols. For example, this would monitor the SQL protocol explicit to the middleware as it transacts with the database in the web server.

- **Hybrid Intrusion Detection System :** Hybrid intrusion detection system is made by the combination of two or more approaches of the intrusion detection system. In the hybrid intrusion detection system, host agent or system data is combined with network information to develop a complete view of the network system. Hybrid intrusion detection system is more effective in comparison to the other intrusion detection system. Prelude is an example of Hybrid IDS.

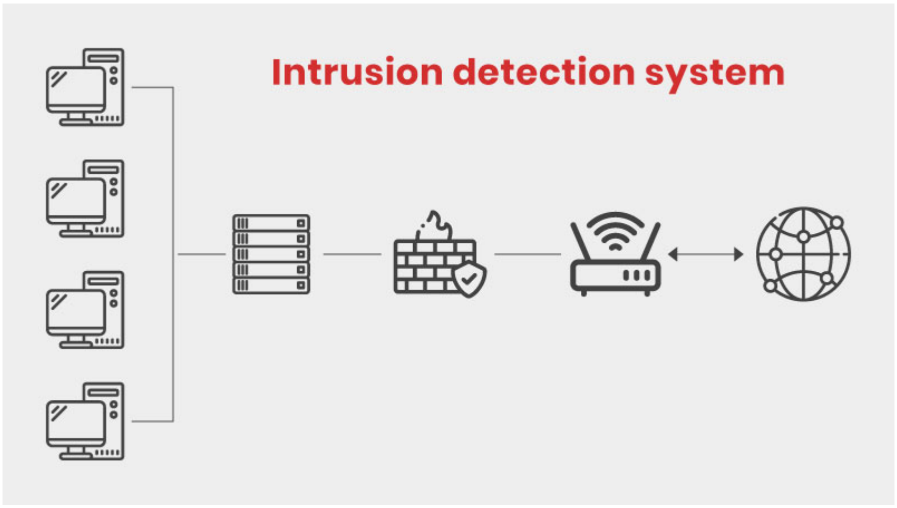
7.5.2 Detection Method of IDS Deployment

Beyond their deployment location, IDS solutions also differ in how they identify potential intrusions:

- **Signature Detection:** Signature-based IDS solutions use fingerprints of known threats to identify them. Once malware or other malicious content has been identified, a signature is generated and added to the list used by the IDS solution to test incoming content. This enables an IDS to achieve a high threat detection rate with no false positives because all alerts are generated based upon detection of known-malicious content. However, a signature-based IDS is limited to detecting known threats and is blind to zero-day vulnerabilities.
- **Anomaly Detection:** Anomaly-based IDS solutions build a model of the "normal" behavior of the protected system. All future behavior is compared to this model, and any anomalies are labeled as potential threats and generate alerts. While this approach can detect novel or zero-day threats, the difficulty of building an accurate model of "normal" behavior means that these systems must balance false positives (incorrect alerts) with false negatives (missed detections).
- **Hybrid Detection:** A hybrid IDS uses both signature-based and anomaly-based detection. This enables it to detect more potential attacks with a lower error rate than using either system in isolation.

7.5.3 What does an intrusion detection system do?

Intrusion detection systems use two methods: signature-based detection, which takes data activity and compares it to a signature or pattern in the signature database. Signature-based detection has a constraint whereby a new malicious activity that is not in the database is ignored. The other detection method is the statistical anomaly-based or behavior-based detection, which, unlike signature-based, detects any anomaly and gives alerts; hence it detects new types of attacks. It is referred to as an expert system as it learns what normal behavior in the system is.



7.5.4 Function of an Intrusion Detection System on A Network

Intrusion detection is a passive technology; it detects and acknowledges a problem but interrupt the flow of network traffic, Novak said. "As mentioned, the purpose is to find and alert on noteworthy traffic. An alert informs the IDS analyst that some interesting traffic has been observed. But it is after-the-fact because the traffic is not blocked or stopped in any way from reaching its destination."

Compare that to firewalls that block out known malware and intrusion prevention system (IPS) technology, which as the name describes, also blocks malicious traffic.

Although an IDS doesn't stop malware, cybersecurity experts said the technology still has a place in the modern enterprise.

"The functionality of what it does is still critically important," said Eric Hanselman, chief analyst with 451 Research. "The IDS piece itself is still relevant because at its core it's detecting an active attack."

However, cybersecurity experts said organizations usually don't buy and implement IDS as a standalone solution as they once did. Rather, they buy a suite of security capabilities or a security platform that has intrusion detection as one of many built-in capabilities.

Rob Clyde, board of directors vice chair ISACA, an association for IT governance professionals, and executive chair for the board at White Cloud Security Inc., agreed that intrusion detection is still a critical capability. But he said companies need to understand that an intrusion detection system requires maintenance and consider whether, and how, they'll support an IDS if they opt for it.

"Once you've gone down the path to say we're going to keep track of what's going on in our environment, you need someone to respond to alerts and incidents. Otherwise, why bother?" he said.

Given the work an IDS takes, he said smaller companies should have the capability but only as part of a larger suite of functions so they're not managing the IDS in addition to other standalone solutions. They should also consider working with a managed security service provider for their overall security requirements, as the provider due to scale can more efficiently respond to alerts. "They'll use machine learning or maybe AI and human effort to alert your staff to an incident or intrusion you truly have to worry about," he said.

"And at mid-size and larger companies, where you really need to know if someone is inside the network, you do want to have the additional layer, or additional layers, than just what's built into your firewall," he said.

7.5.5 Challenges of Managing an IDS

Intrusion detection systems do have several recognized management challenges that may be more work than an organization is willing or able to take on.

False positives

False positives (i.e., generating alerts when there is no real problem). "IDSs are notorious for generating false positives," Rexroad said, adding that alerts are generally are sent to a secondary analysis platform to help contend with this challenge.

This challenge also puts pressure on IT teams to continually update their IDSs with the right information to detect legitimate threats and to distinguish those real threats from allowable traffic.

It's no small task, experts said.

"IDS systems must be tuned by IT administrators to analyze the proper context and reduce false-positives. For example, there is little benefit to analyzing and providing alerts on internet activity for a server that is protected against known attacks. This would generate thousands of irrelevant alarms at the expense of raising meaningful alarms. Similarly, there are circumstances where perfectly valid activities may generate false alarms simply as a matter of probability," Rexroad said, noting that organizations often opt for a secondary analysis platform, such as a Security Incident & Event Management (SIEM) platform, to help with investigating alerts.

Staffing

Given the requirement for understanding context, an enterprise has to be ready to make any IDS fit its own unique needs, experts advised.

"What this means is that an IDS cannot be a one-size-fits all configuration to operate accurately and effectively. And, this requires a savvy IDS analyst to tailor the IDS for the interests and needs of a given site. And, knowledgeable trained system analysts are scarce," Novak added.

Missing a Legitimate Risk

"The trick with IDS is that you have to know what the attack is to be able to identify it. The IDS has always had the patient zero problem: You have to have found someone who got sick and died before you can identify it," Hanselman said.

IDS technology can also have trouble detecting malware with encrypted traffic, experts said. Additionally, the speed and distributed nature of incoming traffic can limit the effectiveness of an intrusion detection system in an enterprise.

"You might have an IDS that can handle 100 megabits of traffic but you might have 200 megabits coming at it or traffic gets distributed, so your IDS only sees one out of every three or four packets," Hanselman said.

7.5.6 The Future of Intrusion Detection Systems

Hanselman said those limitations still don't invalidate the value of an IDS as a function.

"No security tool is perfect. Different products have different blind spots, so the challenge is knowing those blind spots," he explained. "I continue to think that IDS will be with us for a long time to come. There's still that basic value in being able to identify specific hostile traffic on the wire."

However, experts said this has some organizations rethinking the need for an IDS – even though today implementing the technology remains a security best practice.

"This tuning and analysis requires a significant amount of effort based on the number of alerts received. An organization may not have the resources to manage all devices in this capacity. Other organizations may conduct a more comprehensive threat assessment and decide not to implement IDS devices," Rexroad said, adding that the high number of **IDS false positives** have some organizations opting against implementing IPSs as well for fear of blocking legitimate business transactions.

He said other organizations may decide to focus on more advanced protections at the internet gateway or use flow analysis from network devices in conjunction with log analysis from systems and applications to identify suspect events instead of using an IDS.

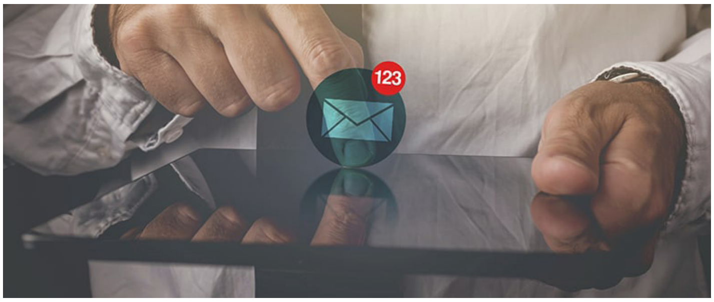
7.6 SECURE E-MAIL

Email security is a term for describing different procedures and techniques for protecting email accounts, content, and communication against unauthorized access, loss or compromise. Email is often used to spread malware, spam and phishing attacks. Attackers use deceptive messages to entice recipients to part with sensitive information, open attachments or click on hyperlinks that install malware on the victim's device. Email is also a common entry point for attackers looking to gain a foothold in an enterprise network and obtain valuable company data.

Email encryption involves encrypting, or disguising, the content of email messages to protect potentially sensitive information from being read by anyone other than intended recipients. Email encryption often includes authentication.

7.6.1 How Secure Is Email?

Email was designed to be as open and accessible as possible. It allows people in organizations to communicate with each other and with people in other organizations. The problem is that email is not secure. This allows attackers to use email as a way to cause problems in attempt to profit. Whether through spam campaigns, malware and phishing attacks, sophisticated targeted attacks, or business email compromise (BEC), attackers try to take advantage of the lack of security of email to carry out their actions. Since most organizations rely on email to do business, attackers exploit email in an attempt to steal sensitive information.



Because email is an open format, it can be viewed by anyone who can intercept it, causing email security concerns. This became an issue as organizations began sending confidential or sensitive information through email. An attacker could easily read the contents of an email by intercepting it. Over the years, organizations have been increasing email security measures to make it harder for attackers to get their hands on sensitive or confidential information.

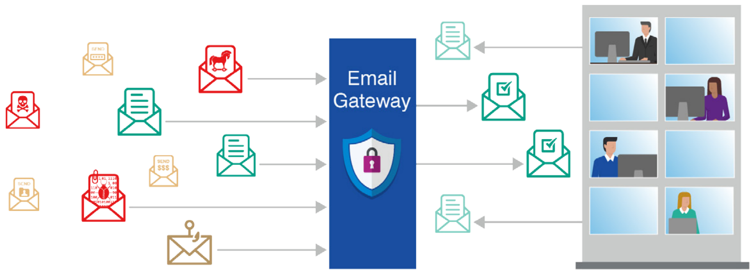
7.6.2 Email Security Policies

Because email is so critical in today's business world, organizations have established polices around how to handle this information flow. One of the first policies most organizations establish is around viewing the contents of emails flowing through their email servers. It's important to understand what is in the entire email in order to act appropriately. After these baseline policies are put into effect, an organization can enact various security policies on those emails.

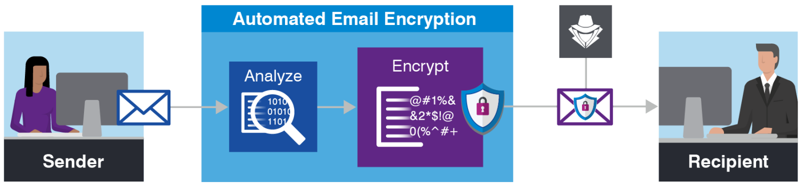
These email security policies can be as simple as removing all executable content from emails to more in-depth actions, like sending suspicious content to a sandboxing tool for detailed analysis. If security incidents are detected by these policies, the organization needs to have actionable intelligence about the scope of the attack. This will help determine what damage the attack may have caused. Once an organization has visibility into all the emails being sent, they can enforce email encryption policies to prevent sensitive email information from falling into the wrong hands.

7.6.3 Email Security Best Practices

One of the first best practices that organizations should put into effect is implementing a secure email gateway. An email gateway scans and processes all incoming and outgoing email and makes sure that threats are not allowed in. Because attacks are increasingly sophisticated, standard security measures, such as blocking known bad file attachments, are no longer effective. A better solution is to deploy a secure email gateway that uses a multi-layered approach.



It's also important to deploy an automated email encryption solution as a best practice. This solution should be able to analyze all outbound email traffic to determine whether the material is sensitive. If the content is sensitive, it needs to be encrypted before it is emailed to the intended recipient. This will prevent attackers from viewing emails, even if they were to intercept them.



Training employees on appropriate email usage and knowing what is a good and bad email is also an important best practice for email security. Users may receive a malicious email that slips through the secure email gateway, so it's critical that they understand what to look for. Most often they are exposed to phishing attacks, which have telltale signs. Training helps employees spot and report on these types of emails.

7.6.4 Email Security Tools

A secure email gateway, deployed either on-premises or in the cloud, should offer multi-layered protection from unwanted, malicious and BEC email; granular visibility; and business continuity for organizations of all sizes. These controls enable security teams to have confidence that they can secure users from email threats and maintain email communications in the event of an outage.

An email encryption solution reduces the risks associated with regulatory violations, data loss and corporate policy violations while enabling essential business communications. The email security solution should work for any organization that needs to protect sensitive data, while still making it readily available to affiliates, business partners and users—on both desktops and mobile devices. An email encryption solution is especially important for organizations required to follow compliance regulations, like GDPR, HIPAA or SOX, or abide by security standards like PCI-DSS.

7.6.5 Importance of Email Security

It's important that users and organizations take measures to guarantee the security of their email accounts against known attacks, and it's especially important that a proper infrastructure is in place to stop any unauthorized attempts at accessing accounts or communications. Users are especially susceptible to phishing attacks against businesses, because they sidestep technical security protections, and instead lean into users themselves to expose weaknesses. This is why email security solutions should start with proper techniques like encryption, spyware detection, and login security. But it's equally important that employees are educated on the proper steps that should be taken to protect email.

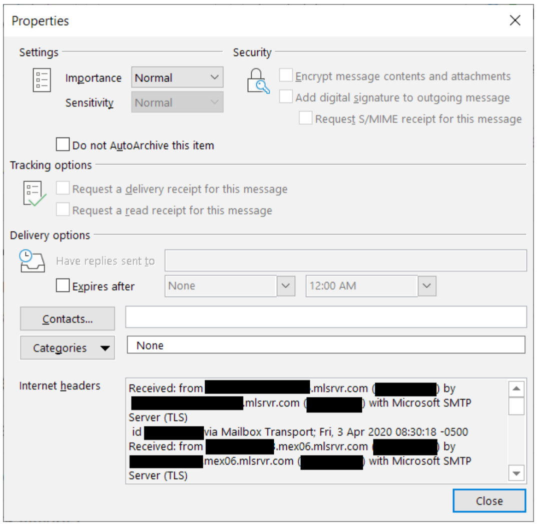
7.6.6 Email Security Tips to Secure Messages Sent via Mail Transfer Protocols

Below, we'll explore 10 practical checks you can use to achieve secure SMTP, IMAP, and POP3 communications for your email accounts:

Learn to Inspect Message Headers

Your email message headers are usually hidden by default, but you can Google ways to view the original message headers for your specific email client. For example, if you're using the Outlook 365 email client:

- Double-click on an email to open it in a new window.
- Go to the **File** menu and select **Properties**.
- In the Properties window, you'll see a field at the bottom that contains email header information.



Properties

Settings

Importance: Normal

Sensitivity: Normal

Security

☐ Encrypt message contents and attachments

☐ Add digital signature to outgoing message

☐ Request S/MIME receipt for this message

☐ Do not AutoArchive this item

Tracking options

☐ Request a delivery receipt for this message

☐ Request a read receipt for this message

Delivery options

Have replies sent to

☐ Expires after: None 12:00 AM

Contacts...

Categories: None

Internet headers

```
Received: from [redacted].mlsrvr.com ([redacted]) by
[redacted].mlsrvr.com ([redacted]) with Microsoft SMTP
Server (TLS)
 id [redacted]via Mailbox Transport; Fri, 3 Apr 2020 08:30:18 -0500
Received: from [redacted].mex06.mlsrvr.com ([redacted]) by
[redacted]mex06.mlsrvr.com ([redacted]) with Microsoft SMTP
Server (TLS)
```

Close

Once you can see the headers, look for the "Received From" field that tracks the route the message traveled across the net via servers to reach you. If you get a suspicious email, search for the sender's IP and do a reverse lookup to trace the message back to where it originated. You can also check if the message fails sender policy framework (SPF) and domain keys identification mail (DKIM) checks.

Though most mail programs have email security indications like a red question mark for unauthenticated emails in Gmail, knowing how to examine email headers is a useful skill to have.

Avoid Clicking on Links or Downloading Attachments

As most of us know, email security's biggest weakness often boils down to human error. This fact is continuously hammered into our brains by security experts and tech gurus. However, getting too curious to know what an attachment is, or being too absentminded to notice that we've accidentally clicked on a link are not impossible scenarios. Even the best of us can fall prey to phishing attacks — at least, the well-crafted ones. This is why, in addition to having spam filters and antimalware installed, we must be careful not to open any attachments or click on links from unknown senders (or attackers pretending to be Gary from the accounts department).

Update Your DMARC Records With the Domain Registrar

DMARC, aside from running checks on the messages using SPF and DKIM standards, is the only method that informs a receiving server of the action it should take in the event that a message fails these tests. If you're a domain owner, besides configuring SPF and DKIM, consider setting up DMARC records with your domain registrar. iIn case you're unsure about the process, they should be able to help you with it.

Neither SPF nor DKIM can prevent attackers from forging the "From" address that you see displayed in your inbox. However, DMARC verifies that the "from" matches the return-path checked by SPF and the domain name in the DKIM signature.

Test Your SMTP Server

To do this, try sending test emails to see how it responds to genuine and spam messages alike by monitoring the SPF, DMARC records. If it's possible to tweak the SMTP configurations, change the default settings and update them with more secure alternatives (starting with changing default admin usernames and passwords).

Make Use of SMTP SSL/TLS Ports

SMTPS traditionally has used port 465 as a way to secure SMTP at the transport layer by running it over a TLS connection. When we refer to an SMTP SSL port (or, more accurately, SMTP TLS port), that's exactly what we mean — it's a way to have a secure exchange of messages between the email client and the email server over SSL/TLS channels.

TLS implementation can be done using two approaches – opportunistic TLS or forced TLS. With opportunistic (explicit) TLS, we try to shift from the use of unencrypted SMTP to a secure TLS encrypted channel utilizing the STARTTLS SMTP command. If the attempt fails, the transmission resumes in plain text, meaning without the use of any encryption. However, with forced (implicit) TLS, the email client and server are either able to negotiate an encryption version they can both support, or the transmission stops and the email communication doesn't progress. You can make your choice depending on whether you want maximum deliverability or maximum privacy.

The Internet Assigned Numbers Authority (IANA) had registered port 465 for SMTPS, though it was never published as an official SMTP channel by the Internet Engineering Task Force (IETF). A new service had been assigned to port 465 by the end of 1998. while 465 functioned as a secure SMTP port, port 25 continues to be used as the default port for SMTP relaying. ISPs and hosting providers have restricted the use of port 25 for SMTP connections (to send mails across the net), and most modern email clients don't use this port at all. Unless you're managing a mail server (a message transfer agent or an MTA), typically, you should see no traffic over this port.

Port 587, along with TLS encryption, should be used as the default secure SMTP port for message submission as recommended by IETF in accordance with RFC 6409 that separates message submission (port 587) from message relay (port 25). Because many legacy systems continue using port 465 for SMTPS, you may still be able to find support for it from your ISP or hosting provider, but it is not recommended to use this port. Lastly, if port 587 is blocked, port 2525 though not officially recognized, is a commonly used alternative supported by most email service providers.

Deploy End-to-End Encryption for Maximum Email Security

With the note from the authors of RFC 5321 in mind, a note that indicates that SMTP mail is inherently insecure, consider using end-to-end encryption standards like S/MIME or PGP to encrypt messages on the sender's device, as well as during transmission. This ensures that even if the message falls into the hands of an attacker, all they see is garbled data that makes no sense.

An additional benefit of using an S/MIME certificate (or email signing certificate, as it's also known) is that it enables you to add a digital signature. This verifies the authenticity of the sender and validates message integrity.

Use TLS With IMAP and POP3

So, what's POP3 and IMAP? The internet access message protocol (IMAP) and post office protocol (POP3, indicating version 3) deal with retrieving the messages from the receiving server. These are the protocols used by email clients like Outlook when getting your emails from mail servers. While IMAP syncs messages across all of your devices, POP3 downloads the message onto a single machine so that it's available offline before deleting it from the server. Encrypted POP3 connections use port 995 (also known as POP3S), and IMAPS uses port 993.

Maintain IP Blacklists to Block Targeted Spams

If you're frequently the target of junk and spam messages from IP addresses that share unsolicited marketing and sales pitches, it makes sense to block them on your email server.

To do this, you can use DNS blacklists (e.g., DNSBL, Spamhaus, etc.) or spam URI real-time block lists (e.g., SURBL, URIBL, etc.). A quick Google search will show you a bunch of available options, but be careful utilizing these kinds of tools — they're not free of controversies and may inadvertently block some legitimate emails.

Use Restrictive Mail Relay Options

You don't want to be an open relay because any spammer from anywhere in the world can use your server and resources for spamming others. The mail relay parameter specifies for which domains or IPs your server can forward mail. Configure these options with the utmost care if you wish to avoid getting on a blacklist.

Other Considerations to Improve Email Security

Some additional email security considerations that may come in handy include but are not limited to the following:

- **Limit the number of connections to your SMTP server.** You can do this based upon usage and server hardware specifications as these checks can prevent denial of service stacks.
- **Define a failover configuration for MX records**. Whenever possible, have a failover configuration when listing MX records to improve availability.
- **Set up reverse DNS lookup to block IPs when authentication fails**. Activate reverse DNS lookup that blocks emails if an IP mismatch occurs between the hostname and domain name of the sender.

REFERENCES

1. Anderson, R. (2001) *Security Engineering: A Guide to Building Dependable Distributed Systems* , Wiley.
2. BS 7799-2 (2002) *Information Security Management Systems – Specification with Guidance for Use* , British Standards Institution.
3. Dieter Gollmann, *Computer Security, 3/e* (2011, Wiley).
4. Ellis, J. and Speed, T. (2001) *The Internet Security Guidebook*, Academic Press.
5. Goodrich and Tamassia, *Introduction to Computer Security* (2010, Addison-Wesley).
6. Halsall, F. (2001) *Multimedia Communications*, Addison Wesley.
7. ISO/IEC 17799 (2000) *Information Technology – Code of Practice for Information Security Management* , International Organization for Standardization.
8. ITU-T X.509 (2000) *Information Technology – Open Systems Interconnection – The Directory: Public-Key and Attribute Certificate Frameworks* , International Telecommunication Union.
9. Mark Stamp, *Information Security: Principles and Practice, 2/e* (2011, Wiley).
10. Matt Bishop, *Computer Security: Art and Science* (2002, Addison-Wesley). Shorter version which «omits much of the mathematical formalism»: *Introduction to Computer Security* (2005, Addison-Wesley).
11. Paul van Oorschot, Computer Security and the Internet: Tools and Jewels (2020, Springer). Personal use copy freely available on author's web site.
12. Pfleeger and Pfleeger, *Security in Computing, 4/e* (2007, Prentice Hall).
13. Smith and Marchesini, *The Craft of System Security* (2007, Addison-Wesley).
14. Smith, *Elementary Information Security* (2011, Jones & Bartlett Learning).

15. Stallings and Brown, *Computer Security: Principles and Practice, 3/e* (2014, Prentice Hall).

16. Wenliang Du, *Computer Security: A Hands-on Approach* (2017, self-published). Updated May 2019.

INDEX

Secure Computing

Data security has consistently been a major issue in information technology. In the cloud computing environment, it becomes particularly serious because the data is located in different places even in all the globe. The security of computer networks plays a strategic role in modern computer systems. In order to enforce high protection levels against malicious attack, a number of software tools have been currently developed. Intrusion Detection System has recently become a heated research topic due to its capability of detecting and preventing the attacks from malicious network users. Data security and privacy protection are the two main factors of user's concerns about the cloud technology. Though many techniques on the topics in cloud computing have been investigated in both academics and industries, data security and privacy protection are becoming more important for the future development of cloud computing technology in government, industry, and business. Data security and privacy protection issues are relevant to both hardware and software in the cloud architecture.
This book is aimed to cover different security techniques and challenges from both software and hardware aspects for protecting data in the cloud and aims at enhancing the data security and privacy protection for the trustworthy cloud environment. Network and computer security is critical to the financial health of every organization. Over the past few years, Internet-enabled business, or e-business, has drastically improved efficiency and revenue growth. E-business applications such as e-commerce, supply-chain management, and remote access allow companies to streamline processes, lower operating costs, and increase customer satisfaction. Such applications require mission-critical networks that accommodate voice, video, and data traffic, and these networks must be scalable to support increasing numbers of users and the need for greater capacity and performance. However, as networks enable more and more applications and are available to more and more users, they become ever more vulnerable to a wider range of security threats. To combat those threats and ensure that e-business transactions are not compromised, security technology must play a major role in today's networks. As time goes on, more and more new technology will be developed to further improve the efficiency of business and communications. At the same time, breakthroughs in technology will provide even greater network security, therefore, greater piece of mind to operate in cutting edge business environments. Provided that enterprises stay on top of this emerging technology, as well as the latest security threats and dangers, the benefits of networks will most certainly outweigh the risks.

Cate Thomas, PhD, is currently working as Associate Professor. Her research interest areas include wired and wireless networks, intrusion detection systems, secure routing protocols in wireless ad hoc and sensor networks. She has published in more than 60 international journals and international conference proceedings, as well as 50 national conference publications.

ISBN 978-1-98467-777-8
00000
9 781984 677778

BIBLIOTEX
Digital Library