Cybersecurity

Richard A. Kemmerer

Department of Computer Science
University of California Santa Barbara
kemm@cs.ucsb.edu

Abstract

As more business activities are being automated and an increasing number of computers are being used to store sensitive information, the need for secure computer systems becomes more apparent. This need is even more apparent as systems and applications are being distributed and accessed via an insecure network, such as the Internet. The Internet itself has become critical for governments, companies, financial institutions, and millions of everyday users. Networks of computers support a multitude of activities whose loss would all but cripple these organizations. As a consequence, cybersecurity issues have become national security issues. Protecting the Internet is a difficult task.

Cybersecurity can be obtained only through systematic development; it can not be achieved through haphazard seat-of-the-pants methods. Applying software engineering techniques to the problem is a step in the right direction. However, software engineers need to be aware of the risks and security issues associated with the design, development, and deployment of network-based software.

This paper introduces some known threats to cybersecurity, categorizes the threats, and analyzes protection mechanisms and techniques for countering the threats. Approaches to prevent, detect, and respond to cyber attacks are also discussed.

1. Introduction

In recent years, networks have evolved from a mere means of communication to a ubiquitous computational infrastructure. Networks have become larger, faster, and highly dynamic. The pervasive use of computer and network technologies in all walks of life has turned cybersecurity issues into national security issues.

1.1. How bad is it?

On September 11, 2001, the World Trade Center attack gave an example of how our lives can be affected by a single terrorist act. Within days of this physical attack, while the nation was still adjusting to the tragic loss of lives and property, an Internet worm cyber attack, called Nimda, spread nationwide in less than an hour and attacked 86,000 computers. This event raised our consciousness to how vulnerable to cyber attacks our systems are.

The latest Internet worm, called the Sapphire/Slammer SQL worm, is another example of how vulnerable systems on the Internet are and of how fast these worms can spread. According to preliminary analysis of the Sapphire worm performed by Silicon Defense and researchers at UC Berkeley [27] this was the fastest worm to date. It was able to spread worldwide in approximately 10 minutes, doubling in size every 8.5 seconds. According to their report, at its peak, which it reached just 3 minutes after its release, it "scanned the net at over 55 million IP addresses per second. It infected at least 75,000 victims and probably considerably more."

1.2. Why is it so bad?

Computers are everywhere. Almost everything that we do in our daily lives depends on computers and computer networks. The Internet has become a mission-critical infrastructure for governments, companies, and financial institutions. Computers and networks are used for controlling and managing manufacturing processes, water supplies, the electric power grid, air traffic control systems, and stock market systems, to mention a few. As a consequence, network attacks have started to impact the practical aspects of our lives.

Today's networks are very heterogeneous environments where highly critical software applications (e.g., the control system for a dam) run side-by-side with other non-critical network-based applications (e.g., a mail server). As a consequence, cyber attacks against apparently "non-critical" services may produce unforeseen side effects of devastating proportions.

The number, severity, sophistication, and cost of reported attacks is increasing [3]. Unfortunately, many attacks are not even detected; therefore, things are even worse than reported. That is, attackers are often able to hide their tracks by disabling logging facilities or modifying event logs, so their activity goes undetected.

The perpetrators of these attacks vary considerably. At the low end are *script kiddies*, who are usually unsophisticated users that download malicious software from hacker web sites and follow the posted instructions to execute an attack on some target. These attacks are often only annoyance attacks, but they can be more severe. At the next level are *hackers* who are trying to prove to their peers or to the world that they can compromise a specific system, such as a government web site. Next are *insiders*, who are legitimate users of a system that either access information that they should not have access to or damage the system or data because they are disgruntled. Insiders are often less knowledgeable then hackers, but they are often more dangerous because they have legal access to resources that the hackers need to access illegally. Next are organizational-level attacks. In this case, the organization's resources are used to get information illegally or to cause damage or deny access to other organizations to further the attacking organization's gain. These can be legitimate organizations, such as two companies bidding on the same contract where one wants to know the other's bid in order to make a better offer. They could also be criminal organizations that are committing fraud or some other illegal activity. At the highest level is the nation state that is trying to spy on or cause damage to another state. This level used to be called "national lab" attackers, because the attackers have a substantial amount of resources at their disposal, comparable to those that are available to researchers at a national lab, such as Los Alamos Laboratory or Lawrence Livermore Laboratory. After the September 11, 2001 terrorist attacks on the World Trade Center, the idea of nation state level cyber attacks being carried out by terrorists became a big concern.

In addition to the rising threat, as nation level attacks become more plausible, the vulnerabilities have also increased. One cause of this is that in addition to virtually every business being on the Internet, today most homes are also connected. This has become particularly true with the advent of relatively inexpensive DSL and cable connections. To make things worse, most home users do not know that their systems are vulnerable. They often say "I have nothing to hide," or "I don't care if someone else sees the information on my computer." Most of these users do not use a firewall, and they think that because their system came with an antivirus program they are protected. They often do not realize that virus programs need to be updated regularly. A big problem is that home users are not aware that their systems can serve as a jump off point for other attacks. In fact, many of the denial of service attacks use unsuspecting home computers as *zombies* to be awaken at a specific time to unwittingly accomplish a distributed denial of service.

Another reason that things are so bad is because, for the most part, computer security is reactive. That is, system administrators and security professionals are usually reacting to the latest attack. After they fix the vulnerability that allowed the attack, the attackers look for new vulnerabilities to exploit for new attacks. What is needed is a proactive approach.

Security is also expensive, both in dollars and in time taken from normal everyday tasks. As a result, security vulnerabilities often exist not because there is not a known fix, but because some administrator does not have the time to put in a patch to fix the problem.

Finally, there is not now, and never will be, a system with perfect security. That is, if functionality is a requirement, then there will always be security gaps. Security is a system attribute that must be traded off against user friendliness, efficiency, ease of use, and other desirable system properties.

The next section presents some basic security terminology and outlines some basic approaches to achieving secure systems. Section 3 presents eight principles to be followed when designing and implementing a secure system. Section 4 presents the twenty top vulnerabilities for Windows and Unix. Finally, Section 5 discusses how software engineering practice can help make systems more secure.

2. Background

In the early days of computing, when standalone systems were used by one user at a time, computer security consisted primarily of physical security. That is, the computer and its peripherals were locked in a secure area with a guard at the door that checked each user's identification before allowing them to enter the room.

As time sharing systems emerged in the mid to late 1960s and multiple jobs and users were able to run at the same time, controlling the access to the data on the system became a major point of concern. One solution that was used was to process classified data one level at a time and "sanitize" the system after the jobs from one level were run and before the jobs for the next level were run. This approach to computer security was known as *periods processing* because the jobs for each level were all run in their particular period of the day. This was an inefficient way to use the system, and an effort was made to find more efficient software solutions to the multilevel security problem.

Another effort that occurred in the mid to late 1970s was the use of *tiger teams* to test the security of a system. These teams attempted to obtain unauthorized access to the system by exploiting design and implementation errors [21, 10]. The tiger team studies demonstrated the difficulty of providing secure software; virtually every system that was attacked by a tiger team was penetrated.

Before proceeding with more background, it is necessary to introduce some basic terminology. In the context of this paper, the term *computer security* means the protection of resources (including data and programs) from unauthorized disclosure, modification or destruction. In addition, the system resources must also be protected (i.e., access to system services should not be denied). These properties are usually referred to as confidentiality, integrity, and availability. More precisely, *confidentiality* ensures that sensitive information is not disclosed to unauthorized recipients; *integrity* ensures that data and programs are modified or destroyed only in a specified and authorized manner, and *availability* ensures that the resources of the system will be usable whenever they are needed by an authorized user.

The degree to which each of these three properties is needed varies from one application to another. For instance, the military is primarily interested in confidentiality. In contrast, the banking industry is primarily interested in integrity, and for the telephone industry availability is most important. This is not to say that any of these applications do not care about the other properties. For instance, the military would not want missile targets to be changed in an unauthorized manner, and they would like their battle plans to be available when needed. Thus, they are interested in integrity and availability too. The exact requirements that are needed for a particular system or application are expressed in the security policy for that system or application. A *security policy* defines what is and what is not allowed.

Cybersecurity consists largely of defensive methods used to detect and thwart would-be intruders. The principles of computer security thus arise from the kinds of threats intruders can impose. For instance, one general security stance is that "everything that is not permitted is prohibited." If this principle is enforced, then an intruder can not get access to some object just because the security administrator did not consider whether it should be restricted or not. Most members of the security community believe that if software were designed with more of an emphasis on security and if systems were configured properly, then there would be fewer security problems[1].

There are four general approaches to achieving a secure computing environment. They are the use of special procedures for working with the system, the inclusion of additional functions or mechanisms in the system, the use of assurance techniques to increase one's confidence in the security of the system, and the use of intrusion detection systems. Each of these is discussed in the following subsections.

[1]Other security principles are discussed in Section 3.

Some security requirements can either be achieved by requiring procedures to be followed or by using system mechanisms or functions to enforce the requirement. However, in some cases system users need to follow specific procedures in order to make security mechanisms effective. It is also possible to trade off assurance techniques for less mechanism. For instance, one can use assurance techniques, like formal proofs, to assure that the system can not get into a particular state; thus, alleviating the need for the software mechanism that would deal with that state.

2.1. Procedural Approaches

Procedural approaches prescribe the appropriate behavior for a user to follow when using a system. The periods processing approach for processing jobs at different security levels, which was presented above, is an example of a procedural solution to satisfy a security requirement.

User guidelines for the appropriate choice of strong passwords is the most prevalent example of using procedural approaches to achieve a security requirement. For example, to deter password guessing by a potential intruder one should choose a long password (at least eight characters) that is not obvious and not easily guessable (e.g., not a spouse's first name, a middle name, a login name, or any of these spelled backwards). Passwords should also use both upper and lower case letters, numerics and possibly special symbols. In addition, a password should not be written down, or, if it is, it should not be written in an obvious place. Furthermore, users should be trained to change their passwords at appropriate intervals. Many of these restrictions can be enforced by the system when a user chooses a new password.

Another example of a procedural approach is a set of rules for the appropriate handling of removable storage devices. Oftentimes data that is perfectly safe while in a protected system is compromised by an attacker getting access to removable storage, such as a dump tape, and analyzing it on an unrestricted system. For this reason most security conscious organizations have explicit rules for handling removable media, such as requiring them to be stored in an approved vault.

2.2. Function and Mechanism

Including additional functions or mechanisms in a computer system is another way of enhancing computer security. The mechanisms presented in this section are grouped into authentication mechanisms, access control, and inference control.

2.2.1 Authentication Mechanisms

Authentication mechanisms are used to assure that a particular user is who he/she claims to be. The first mechanism discussed is the *secure attention key*. This key, when hit by a user at a terminal, kills any process running at the terminal except the true system listener and thus guarantees a trusted path to the system. This will foil attempts at "spoofing," which is the process of fooling a user into believing that he/she is talking to the system, while instead the user is interacting with a malicious application. For instance, the spoofer may display what looks like the system login prompt on a terminal to make the terminal appear to be idle. Then when an unsuspecting user begins to use the terminal, the spoofer retrieves the login name and displays a new message asking for the user's password. After obtaining this information, the spoofer displays a message to try again, and the spoofer returns ownership of the terminal to the system. If a secure attention key is used, it is important that users make a habit of always hitting the key to begin a dialogue with the system. One way of ensuring this is for the system to only display the login prompt after the key is depressed. This is an example of procedures and mechanism working together to achieve a security property.

Most of the password guidelines that were discussed above as procedural approaches can be enforced by the system. For instance, the password program can require long passwords and it can check the password chosen against an online dictionary or against a list of obvious passwords. The login program can also inform the user when it is time to change passwords and not allow further logins if the password is not changed in time. Finally, the system can generate secure passwords for the users using a secure password generator.

The most sophisticated authentication devices are those that depend on a unique physiological or behavioral characteristic that can be examined and quantified for each user. When a user wants to be authenticated his/her characteristics are sampled, converted into a digital form, and compared to the previously stored sample characteristics for that user. The most common biometric devices are based on fingerprints, handprints, retina patterns, or facial characteristics. The most common behavioral devices use voice, signature, or keystroke characteristics. Most of these devices are used in a two-factor authentication system, such as with passwords.

2.2.2 Access Control

Assuming that by using authentication mechanisms and good password practice the system can guarantee that users are who they claim to be, the next step is to provide a means of limiting a user's access to only those files that policy determines should be accessed. These controls are referred to as *access control*. Different applications and systems have different security requirements, and these requirements are expressed in the *access control policy* for the application or system. Access control policies are enforced by the access control mechanisms.

There are two types of access control: *discretionary access control* (DAC) and *mandatory access control* (MAC). More precisely, for discretionary access control the owner of an object specifies what type of access the other users can have to the object. Thus, access is at the discretion of the owner. In contrast, for mandatory access control the system determines whether a user can access a resource based on the security attributes (e.g., labels or markings) of both the user and the object. Mandatory access control is often called non-discretionary access control.

One of the earliest forms of discretionary access control was the use of passwords for file access. That is, the owner selects a password for each file and distributes the password to those users to whom the owner wants to give access to the file. A major problem with this approach is that one user may pass a password on to another user without the consent of the owner of the file. A second major problem is that the owner cannot revoke access from one user without revoking access from all users and then selecting a new password for the file. A third problem with this approach is that the passwords tend to get embedded in other files and are vulnerable to browsing. The UNIX *owner/group/other* approach is another example of discretionary access control, because it is the owner of the file that sets the access rights.

To assure that all access control policies are enforced a means of mediating each access of an object by a subject is needed. The *reference monitor* [1] provides this mediation. A reference monitor is defined by three basic properties.

1. It must be tamperproof. That is, it should be isolated from modification by system entities.
2. It must always be invoked. That is, it must mediate every access.
3. It must be small enough to be subjected to analysis and tests, the completeness of which can be assured.

A *security kernel* is defined as the hardware and software that realize the reference monitor. The idea is to keep the security kernel small and to have it contain the security relevant parts of the system.

2.2.3 Inference Controls

The last class of security mechanisms discussed in this section is inference controls. These controls attempt to restrict database access to sensitive information about individuals while providing access to statistics about groups of individuals. The ideal is to have a statistical database that discloses no sensitive data.

As an example of the type of threat that is addressed by inference control mechanisms consider a database that contains enrollment and grade statistics for a university class. If Morgan is the only economics major in a particular class one could deduce Morgan's grade by retrieving the average grade for the course, the average grade of all non-economics majors in the class, and the number of students in the class.

Two approaches to solving the inference problem are to restrict queries that reveal certain types of statistics and to add "noise" to the results returned. To foil small and large query set attacks, such as the Morgan example above, a technique called *query-set-size control* is introduced. This forbids the release of any statistics pertaining to a group less than some predetermined size n or greater than N-n, where N is the total number of records in the database. Other techniques, restrict queries with more than some predetermined number of records in common or with too many attributes specified. The techniques that add noise to the statistical results usually return a rounded result or a result based on random sampling. For more information on these techniques see [7].

2.3. Assurance Techniques

The third approach to enhancing the security of a system is to subject the system to rigorous assurance techniques that will raise one's confidence that the system will perform as desired. Among these techniques are penetration analysis, formal specification and verification, and covert channel analysis. None of these methods guarantee a secure system. They only increase one's confidence in the security of the system.

2.3.1 Penetration Analysis

One approach to locating security flaws in computer systems is *penetration analysis,* This approach uses a collection of known flaws, generalizes these flaws, and tries to apply them to the system being analyzed. The tiger teams, discussed in Section 2, are an example of this approach.

Today tiger teams are usually called *red teams,* and red team/blue team experiments pit the red team attackers against the blue team defenders. The problem with the tiger team approach is that, like testing, "penetration teams prove the presence, not the absence of protection failures."

This observation has led to the use of *formal specification* and *verification techniques* to increase ones confidence in the reliability and security of a computer system.

2.3.2 Formal Verification

Formal verification demonstrates that an implementation is consistent with its requirements. This task is approached by decomposing it into a number of easier problems. The critical requirements, which are usually a natural language statement of what is desired, are first stated in precise mathematical terms. This is known as the *formal model* or *critical criteria* for the system. For example, the formal model for a secure system could be that information at one security level does not flow to another security level. Next, a high level formal specification of the system is stated. This specification gives a precise mathematical description of the behavior of the system omitting all implementation details, such as resource limitations. This is followed by a series of less abstract specifications each of which implements the next higher level specification, but with more detail. Finally, the system is coded in a high order language. This high order language implementation must be shown to be consistent with the original requirements.

The advent of the security kernel as a means of encapsulating all security relevant aspects of the system made formal verification feasible. That is, by developing kernel architectures that minimize the amount and complexity of software involved in security decisions and enforcement, the chance of successfully verifying that the system meets its security requirements is greatly increased. Because the remainder of the system is written using the facilities provided by the kernel, only the kernel code must be verified. Examples of work in this area are [24, 36, 33, 2]

Formal verification techniques have also been used for analyzing encryption protocols [17, 5, 30]. In this case, a formal specification is written for the different steps of the protocol, and a model checker is used to find a counter example.

2.3.3 Covert Channel Analysis

Secure computer systems use both mandatory and discretionary access controls to restrict the flow of information through legitimate communication channels, such as files, shared memory, and process signals. Unfortunately, in practice one finds that computer systems are built such that users are not limited to communicating only through the intended communication channels. As a result, a well-founded concern of security-conscious system designers is the potential exploitation of system storage locations and timing facilities to provide unforeseen communication channels to users. These illegitimate channels are known as covert storage channels and covert timing channels [20, 22, 26, 15, 16].

Covert channels signal information through system facilities not intended for data transfer, and they support this communication using methods not detected or regulated by the access control mechanisms. *Storage channels* transfer information using the manipulation of storage locations for their coding scheme. That is, the sending process alters some system attribute (e.g., a file lock or a device busy flag) and the receiving process monitors the alteration. For example, if two processes have only write access to a file, then the sending process could lock the file, and the receiving process could detect this change by attempting to write to the file. In this way, the receiver could interpret the file being locked as a one and it not being locked as a zero. *Timing channels* transfer information using the passing of time for their coding scheme; the sending process modulates the receiver's response time to detect a change in some shared entity.

Although there is concern that a user at a high security level may use a covert channel to signal information to a user at a lower level, the major threat from a covert channel is its potential to be exploited by a Trojan horse. A *Trojan horse* is a program that gives the appearance of providing normal functionality, but whose execution results in undesired side effects.

The severity of a covert channel threat has been traditionally measured in terms of the channel's bandwidth (i.e., the number of bits signaled per second). The higher the bandwidth, the greater the potential for serious compromise. An alarming possibility regarding covert channels is that as operating systems are ported to faster hardware architectures, the bandwidths of their covert channels may increase significantly. In fact, timing channels with estimated bandwidths in the megabits per second range have been demonstrated on symmetric multi-processing architectures [4].

2.4. Intrusion Detection

Significant progress has been made toward the improvement of computer system security. Unfortunately, the undeniable reality remains that all computers are vulnerable to compromise. Even the most secure systems built today are vulnerable to authorized users (insiders) who abuse their privileges. Given this reality, the need for user accountability is very important, both as a deterrent and for terminating abusive computer usage once it is discovered. In addition, the need to maintain a record of the transactions that have occurred on a system is crucial for performing damage assessment. In recognition of these needs and in recognition that security violations are a fact of life for computers, many systems implement a form of transaction record keeping called *audit collection*. The data collected is called an *audit log*. Additional systems and/or software are often required to generate the necessary audit data. For example, in order to produce audit records for Sun Microsystem's Solaris one needs to use the Basic Security Module (BSM) [34].

Although the audit logs normally contain enough information to determine that a computer abuse has occurred, the amount of data collected is so voluminous that manual audit data analysis is impractical. By introducing automated audit data analysis tools, referred to as *intrusion detection systems* (IDSs), security violations that might have once gone unnoticed can be identified. In addition, when intrusion detection is performed in real-time, the audit data can be used to track a user's behavior and to determine if the user's current actions represent a threat to security. If they do, the system can then preempt the user's actions.

Intrusion detection is not limited to processing audit records. More generally, intrusion detection is the process of analyzing information about the activities performed in a computer system or network, looking for evidence of malicious behavior. In addition to audit records produced by the operating system auditing facilities, the information may come in the form of log messages produced by different types of sensors and network devices, or in the form of raw network traffic obtained by eavesdropping a network segment. These data sources are used by intrusion detection systems in different ways, according to different paradigms.

There are two basic categories of intrusion detection approaches: *anomaly detection* and *misuse detection*. Anomaly detection relies on models of the intended behavior of users and applications and interprets deviations from this "normal" behavior as an attack [14, 8, 18, 9]. Misuse detection takes a complementary approach. Misuse detection systems contain a number of attack descriptions (or "signatures") that are matched against the stream of audit data looking for evidence that the modeled attack is occurring [12, 29, 28, 13]. Misuse and anomaly detection both have advantages and disadvantages. Misuse detection systems can perform focused analysis of the audit data and usually produce only a few false positives, which are erroneous detections. In contrast, the main disadvantage of misuse detection systems is that they can detect only those attacks that are known. That is, they can detect only attacks for which they have a defined signature. In contrast, the main advantage of anomaly detection systems is that they are able to detect previously unknown attacks. By defining what's normal, they make it possible to identify any variation, no matter whether it is part of the threat model or not. Unfortunately, the advantage of being able to detect previously unknown attacks is paid for in terms of a large number of false positives. Anomaly detection systems are also difficult to train when used in a very dynamic environment.

3. Principles for Designing Secure Systems

In a landmark paper in 1975 Saltzer and Schroeder introduced eight principles to be followed when designing secure systems [31]. Three decades later, these principles still apply. In this section we will only list the principles. The interested reader is directed to the original paper to get more details.

The eight principles are:

1. **Least privilege**: A user should only have the rights necessary to complete his/her task.
2. **Economy of mechanism**: The system must be sufficiently small and simple as to be verified and implemented.
3. **Complete mediation**: Every access to every object must be checked.
4. **Open design**: The system design should be open for scrutiny by the community.
5. **Separation of privilege**: Access to objects should depend on more than one condition being satisfied.
6. **Least common mechanism**: The amount of mechanism common to more than one user should be minimized.
7. **Psychological acceptability**: The user interface must be easy to use.
8. **Fail-safe defaults**: The default is a lack of access.

The next section presents the vulnerabilities that are most often exploited. If any of these vulnerabilities are the result of one or more of the secure design principles being violated, then the violation is illuminated.

4. Top 20 Vulnerabilities for 2002

Cyber attacks usually conform to a general multi-step process. The first step of this process is surveillance, which is the collection of information about a specific target network. Examples are ping scans and port scans to find out what hosts exist and what services are provided by each host. The second phase is the remote exploitation of vulnerabilities associated with the services that were identified during the first phase. When successful, this phase gives the attacker access to the system. After having obtained an entry point in a target system the attacker is now an insider, who has access to much more information. At this point, other attacks are launched from the compromised system. The goal is to either escalate the attacker's privileges or to extend the compromise to neighboring parts of the network, possibly by exploiting internal trust relationships.

The time between each of the steps of a multi-step attack may be seconds, or it may be days, or even months. That is, compromising the initial system may just be laying the foundation for attacks that will not occur until much later. This is the case in many denial of service attacks, where the slave (or zombie) systems are compromised, and then they may not be used for months until it is time for them to collectively perform the necessary steps to cause a denial of service.

Because the evidence of multi-step cyber attacks is scattered on many systems and may be separated by appreciable amounts of time, it is often difficult to correlate the various pieces of evidence to determine the "big picture." That is, identifying who the attackers are and what their goal is from what may at first appear to be unrelated events can be a challenging task. Therefore, much of the current intrusion detection research is focusing on the correlation and fusion of cyber alerts.

Although multi-step attacks can differ in terms of structure and timing, most of these attacks are based on only a few vulnerabilities. This is because attackers usually go for the "low-hanging fruit." That is, they try the easiest attacks first and only progress to the more difficult attacks when the easier ones fail. The SANS Institute, along with the FBI post a list of the ten most exploited Windows vulnerabilities and the ten most exploited Unix vulnerabilities [32]. These are the most common exploits, and fixing them will not guarantee a secure system. It will, however, remove the low-hanging fruits and cause the hackers to work harder to compromise your system.

The next two subsections present the top ten vulnerabilities for Windows and Unix, respectively. The material in these sections is a brief summary of what appears at the SANS/FBI Top Twenty site [32].

4.1. Windows Vulnerabilities

This section presents the ten top vulnerabilities for Windows systems.

- Internet Information Services (IIS): There are three basic vulnerabilities in IIS. They are the failure to handle unanticipated requests, buffer overflows, and insecure sample applications. The fact that unanticipated requests are a source of vulnerability is not surprising, since one of the first places where hackers or tiger teams usually look for security holes in a system is where the response to some action is undefined. If the system designers had followed the fail-safe defaults principle these vulnerabilities might not exist. The sample applications problem is an example of the trade off between being user friendly and being secure. These applications are provided to help the user become familiar with how to use the system. Unfortunately, they were written as pedagogical examples, and security was not one of the concerns. As a result, the example applications cannot withstand attacks.
- Microsoft Data Access Components (MDAC) – Remote Data Services: This flaw occurs only in the Re-

mote Data Services (RDS) component of older versions of MDAC. These older versions have a program flaw that allows a remote user to run applications locally with administrative privileges. This flaw is indicative of a general problem with keeping cyberspace secure: even though the flaw has been identified and a solution has existed for more than two years, there are many outdated or misconfigured systems that are still vulnerable.

- Microsoft SQL Server (MSSQL): The attacks that can be launched against the MSSQL server vary in severity from allowing remote users to obtain sensitive information, to modifying information in the database, to compromising the SQL server, and in some cases to compromising the host itself. The most commonly exercised vulnerability is that the administrative account is installed with a blank password. This violates the principle of fail-safe defaults. It is also likely a violation of the economy of mechanism principle and of the least privilege principle.

- NETBIOS – Unprotected Windows Networking Shares: Windows network shares are used to allow hosts to share files with each other. The problem is that these network shares are often misconfigured, which may allow an attacker to access critical system files and completely compromise the remote host that is sharing the files. It was the use of misconfigured file shares that enabled the Nimda worm to spread so quickly. This is an example of another violation of the principles that states that you should only share what is absolutely needed (least common mechanism).

- Anonymous Logon – Null Sessions: In Windows the null session is used to allow remote users to obtain information such as shares and user name from a host anonymously. The vulnerability occurs because the SYSTEM account, which is used for many critical systems operations, uses null sessions to obtain system information from other hosts. The SYSTEM account needs to do this, since it has no password. However, hackers can also use the null session to obtain sensitive information.

- LAN Manager Authentication – Weak LM Hashing: This is another example of the problems caused by using older systems; that is, the current windows environments no longer use LAN Managers. The problem with the LAN Manager (LM) is that LM passwords are stored locally, and the LM encryption scheme is very weak. Therefore, the passwords can be broken in a very short amount of time. To be more specific, LM passwords are 14 characters long. If they are longer they are truncated to 14, and if they are shorter they are padded with blanks. In addition, they are converted to all upper case characters and then split into two seven character parts. Therefore, the challenge is to crack two seven character all upper case passwords. There are many tools available on the Internet that could brute force LM passwords in a few hours [19].

- General Windows Authentication – Accounts with No Passwords or Weak Passwords: This is the typical problem of using either a bad password or no password at all. By compromising a password, an outsider becomes an insider. The attacker now has all of the privileges of the user whose password was compromised. The attacker can also get information to support a privilege escalation attempt or to compromise other neighbor machines more easily.

- Internet Explorer: Vulnerabilities exist on all versions of Microsoft Explorer. These vary in severity from allowing the attacker to obtain cookies, local files, or input data, to executing local programs and completely compromising the host running the browser. The vulnerabilities are usually based on web page spoofing, ActiveX controls, MIME-type misinterpretation and buffer overflows. Often they are the combination of more than one of these [25, 6].

- Remote Registry Access: The Windows Registry is a centralized hierarchical database that manages software, device configurations, and user settings. The problem is that the permissions and security settings are often not set correctly. As a result, attackers can remotely compromise the system by exploiting the improper settings. The fixes for this problem are to review the configuration of critical registry keys and set them appropriately. Access to the system registry must also be restricted. The real problem is that the Registry is too complex, which is likely also the reason that many of the registry keys were set wrong in the first place. This is a violation of the economy of mechanism principle.

- Windows Scripting Host: The problem with the Windows Scripting Host is that it allows any text file with a `.vbs` extension to be executed as a Visual Basic script when the text is displayed. This is the vulnerability used by many of the worms; in particular, the "I Love You" worm exploited this vulnerability. This is an example of user friendliness versus security. The solution to this problem is to turn off the Windows Scripting Host.

4.2. Unix Vulnerabilities

This section presents the ten top vulnerabilities for Unix systems.

- Remote Procedure Calls (RPC): Remote procedure calls allow a remote user to call procedures on another host and pass input and get results, just as if the procedure were local. The main exploits against the RPC services are accomplished by using buffer overflows. Since many of the procedures that are called remotely execute with root privileges, this allows the attacker to gain root privilege remotely.
- Apache Web Server: The main Apache web server problems are the result of running CGI scripts. Because the CGI scripts execute with the same privilege as the server, the scripts can be very dangerous. If the server is being run as root, the problem is even worse.
- Secure Shell (SSH): Secure Shell services are intended to be secure replacements for ftp, telnet, and the R-command programs, which are remote shell (`rsh`), remote copy (`rcp`), remote login (`rlogin`), and remote execution (`rexec`). SSH is often the only service that is made accessible through firewalls and other protection mechanisms. For this reason its implementation is under continuous scrutiny by the hacker community. Therefore, a vulnerability in SSH gives immediate access to a very large number of systems.
- Simple Network Management Protocol (SNMP): SNMP is one of the most popular protocols for monitoring and configuring remote TCP/IP-enabled devices. Management is accomplished by exchanging messages between the manager and the agent software running on the remote device. The messages are used to request information or configuration changes, respond to requests, enumerate SNMP objects, and send unsolicited alerts, called traps. The vulnerabilities identified are in the trap messages and the request handling, and they can result in denial-of-service conditions, format string vulnerabilities, and buffer overflows. Older SNMP versions also have a problem with their authentication mechanism. The mechanism depends on a "community string", and this string is often left unchanged from its default value, which is "public". This is another example of violating the fail-safe defaults principle.
- File Transfer Protocol (FTP): Some ftp servers allow anonymous access, which does not require a unique password, but even those with authenticated access have vulnerabilities. Some of the exploits allow root access and others allow user level command execution. Another vulnerability of the ftp service is that names and passwords are sent in the clear, so they can be captured by a network sniffer.
- R-Services – Trust Relationships: Trust relationships exist whenever there are entries in the `/etc/hosts.equiv` file, or in the `/.rhosts` file of a user. The first allows any remote user to use any of the R-commands from a host listed in the file, without having to provide a password. The second entry allows the user whose .rhosts file contains the remote host name to use the R-commands without having to provide a password. from any host listed in the file. This weakness allows a user cracking an account on one machine to execute R-commands to all other machines that have a trust relationship with the compromised account or with the host where the account was compromised. This is a violation of the complete mediation and least privilege principles. Another vulnerability of the R-services is that all information passed between an R-command client and an R-service are in the clear.
- Line Printer Daemon (LPD): The line printer daemon (lpd) is used to connect to a printer on a local machine or remotely through port 515. There are multiple programming flaws in several implementations of the lpd that allow buffer overflows that can result in root access to the printer server.
- Sendmail: Sendmail is the program that is used on most Unix systems to send and receive mail. The two main vulnerabilities of the sendmail program are buffer overflows and using an improperly configured machine to relay mail for spamming purposes. No new high severity Sendmail vulnerabilities have been identified for more than two years. However, there are many outdated or misconfigured systems that are still vulnerable.
- BIND/DNS: A Domain Name Service (DNS) allows one to retrieve a machine's IP address by presenting the name of the machine. BIND is the Berkeley Internet Name Domain package, which is the most popular DNS package. The main vulnerabilities of the BIND package are caused by implementation problems and bad configurations. This results in denial of service, buffer overflows, and wrong id-to-name mappings.
- General Unix Authentication – Accounts with No Passwords or Weak Passwords: This is the typical problem of either using a weak password or no password at all. By compromising a password an outsider becomes an insider. The attacker now has all of the

privileges of the user whose password was compromised. The attacker can also get information to make a privilege escalation or to compromise other neighbor machines more easily.

4.3. Patches and Updates

The solution to fixing many of the vulnerabilities presented above is to apply patches provided by Microsoft, Sun Microsystems, Redhat, and others. In the case of Microsoft these patches come as Service Packs, Security Rollups, and Hotfixes. Automatic patches for Redhat Linux come from Redhat Update.

One of the problems with maintaining a secure system is that the administrator needs to be continuously applying patches to the system to keep up with the latest vulnerabilities. This takes time and energy. In addition, for large sites it may be necessary to continue using vulnerable software to meet operational or compatibility requirements [23]. The automatic patching facilities make this easier, but, unfortunately, they sometimes break the system.

5. How can Software Engineering Help?

Security needs to be proactive. That is, systems need to be designed and tested with security in mind right from the start. It can not be an add on or an afterthought. When designing a secure system, security is just one more desirable attribute for the system. It needs to be balanced with other desirable attributes, such as being user friendly and being efficient.

Secure systems should be built using good software engineering practices [35]. For instance, many of the known security breaches are the result of a vulnerability that was introduced when making an unplanned for change to the system. By planning for change and making secure systems extensible there are likely to be less problems introduced when changes need to be made to the system.

Testing is another area where more work is needed. Security testing is different from normal software testing. Normally, when testing a software system, the tester assumes that the user may only occasionally do something wrong. Therefore, some test cases with bad input values should be used. The majority of the test cases, however, will be for appropriate input values. In contrast, for security testing the tester assumes that the attacker is trying to do something wrong; therefore, the concentration should be on testing with unexpected input. For instance, buffer overflow attacks are the result of the user intentionally passing parameters that are larger than expected. As a result, the extra characters in the parameter overwrite the stack, often causing return addresses to be changed so that the program returns to the attacker's code rather than to where it was called. Security testing also takes a devious mind, for in order to do it right you need to anticipate what the attackers are likely to do.

Maintenance is also important. Most (or all) of the vulnerabilities presented in the previous section have patches. However, patching a system is often time-consuming and also may cause certain features to not work. Therefore, better tools for automatic configuration control and update are also needed.

There is also a need for software developers to be accountable for the systems they develop. That is, if a security breach occurs due to a programming flaw in a system implementation, then the software developer should be held liable for any losses that occur. One thing that may help in this area is the development of standards for developing secure software systems. The open source community has taken some initial steps in this direction with their Open-Source Security Testing Methodology Manual [11]. This is intended to set standards for performing Internet security testing.

Acknowledgments

Thanks to Giovanni Vigna for providing helpful comments on this paper.

This work was supported by the Army Research Office, under agreement DAAD19-01-1-0484. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Army Research Office, or the U.S. Government.

References

[1] J. Anderson, et al. Computer Security Technology Planning Study. Technical Report ESD-TR-73-51, Deputy for Command and Management Systems, HQ Electronic Systems Division (AFSC), 1972. Vol. 1.

[2] W. Bevier. Kit: A study in operating system verification. *IEEE Transactions on Software Engineerin*, 15(11), November 1989.

[3] Cert coordination center statistics. `http://www.cert.org/stats/`, 2003.

[4] Minutes of the First Workshop on Covert Channel Analysis. IEEE Cipher, July 1990. Los Angeles, CA.

[5] Z. Dang and R. Kemmerer. Using the astral model checker to analyze mobile i. In *Proceedings of the International Conference on Software Engineering*, pages 132–141, May 1999.

[6] F. De Paoli, A. dos Santos, and R. Kemmerer. *Mobile Agents and Security*, volume 1419, chapter Web Browsers and Security, pages 235–256. Springer-Verlag, 1998.

[7] D. Denning. *Cryptography and Data Security*. Addison Wesley, Reading, Massachusetts, 1982.

[8] S. Forrest. A Sense of Self for UNIX Processes. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 120–128, Oakland, CA, May 1996.

[9] A. Ghosh, J. Wanken, and F. Charron. Detecting Anomalous and Unknown Intrusions Against Programs. In *Proceedings of the Annual Computer Security Application Conference (ACSAC'98)*, pages 259–267, Scottsdale, AZ, December 1998.

[10] B. Hebbard, et al. A Penetration Analysis of the Michigan Terminal System. *ACM Operating Systems Review*, 14(1), January 1980.

[11] P. Herzog. Open source security testing methodology manual. `http://www.isecom.org/projects/osstmm.htm`, February 2002.

[12] K. Ilgun, R. Kemmerer, and P. Porras. State Transition Analysis: A Rule-Based Intrusion Detection System. *IEEE Transactions on Software Engineering*, 21(3):181–199, March 1995.

[13] Internet Security Systems. *Introduction to RealSecure Version 3.0*, January 1999.

[14] H. Javitz and A. Valdes. The NIDES Statistical Component Description and Justification. Technical report, SRI International, Menlo Park, CA, March 1994.

[15] R. Kemmerer. Shared Resource Matrix Methodology: An Approach to Identifying Storage and Timing Channels. *ACM Transactions on Computer Systems*, 1(3), August 1983.

[16] R. Kemmerer. A practical approach to identifying storage and timing channels: Twenty years later. In *Proceedings of the Eighteenth Annual Computer Security Applications Conference*, Las Vegas, Nevada, December 2002.

[17] R. Kemmerer, C. Meadows, and J. Millen. Three systems for cryptographic protocol analysis. *Journal of Cryptography*, 7(2):79–130, 1994.

[18] C. Ko, M. Ruschitzka, and K. Levitt. Execution Monitoring of Security-Critical Programs in Distributed Systems: A Specification-based Approach. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pages 175–187, May 1997.

[19] `www.l0pht.com/l0phcrack`.

[20] B. Lampson. A Note on the Confinement Problem. *Communications of the ACM*, 16:613–615, October 1973.

[21] R. Linde. Operating System Penetration. In *Proceedings of the National Computer Conference*, volume 44, Montvale, N.J., 1975. AFIPS Press.

[22] S. Lipner. A Comment on the Confinement Problem. In *Proceedings of the Fifth Symposium on Operating Systems Principles*, The University of Texas at Austin, November 1975.

[23] R. Lippmann, S. Webster, and D. Stetson. The effect of identifying vulnerabilities and patching software on the utility of network intrusion detection. In A. Wespi, G. Vigna, and L. Deri, editors, *Proceedings of Recent Advances in Intrusion Detection*, pages 307–326. Springer, 2002.

[24] E. McCauley and P. Drognowski. KSOS: The Design of a Secure Operating System. In *Proceedings of the National Computer Conference*. AFIPS Press, June 1979.

[25] G. McGraw and E. Felten. *Java Security: Hostile Applets, Holes and Antidotes*. John Wiley and Sons, 1996.

[26] J. Millen. Security Kernel Validation in Practice. *Communications of the ACM*, 19:243–250, May 1976.

[27] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. The spread of the sapphire/slammer worm. `http://www.silicondefense.com/research/sapphire/`, February 2003.

[28] NFR Security. *Overview of NFR Network Intrusion Detection System*, February 2001.

[29] M. Roesch. Snort - Lightweight Intrusion Detection for Networks. In *Proceedings of the USENIX LISA '99 Conference*, November 1999.

[30] P. Ryan, S. Schneider, M. Goldsmith, G. Lowe, and B. Roscoe. *Modelling and Analysis of Security Protocols*. Addison Wesley Professional, 2000.

[31] J. Saltzer and M. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, September 1975.

[32] Sans/fbi top 20 list. `http://www.sans.org/top20/`, 2002.

[33] J. Silverman. Reflections on the Verification of the Security of an Operating System Kernel. *Communications of the ACM*, 1983.

[34] Sun Microsystems, Inc. *Installing, Administering, and Using the Basic Security Module*. 2550 Garcia Ave., Mountain View, CA 94043, December 1991.

[35] J. Viega and G. McGraw. *Building Secure Software: How to Avoid Security Problems the Right Way*. Addison Wesley, October 2001.

[36] B. Walker, R. Kemmerer, and G. Popek. Specification and Verification of the UCLA Unix Security Kernel. *Communications of the ACM*, 23:118–131, 1980.