UNIT II
DATA SECURITY AND PRIVACY

Topics Covered :
Program Security: Secure Programs, Nonmalicious Program Errors, viruses and other malicious code, Targeted Malicious code, controls Against Program Threats,
Protection in General- Purpose operating system protected objects and methods of protection memory and addmens protection, File protection Mechanisms, User Authentication
Security in Network: Threats in Network, Network Security Controls, Firewalls, Intrusion Detection Systems,Secure E-Mail.
Privacy in Computing -Privacy Concepts , Privacy Principles and Policies Section , Authentication and Privacy Section, Data Mining Section, Privacy on the Web Section, E-Mail Security Section, Impacts on Emerging Technologies

PROGRAM SECURITY

SECURE PROGRAM

Consider what we mean when we say that a program is "secure." We know thatsecurity 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 analysing 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.

Early work in computer security was based on the paradigm of "penetrate and patch," inwhich 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.

1.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.

2.The fault often had nonobvious side effects in places other than the immediate area of the fault.

3. Fixing one problem often caused a failure somewhere else, or the patch addressed the problem in only one place, not in other related places.

4.The fault could not be fixed properly because system functionality or performance would suffer as a consequence.

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 behaviour caused by a program vulnerability.

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 fromthe 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.

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, Landwehr et al. present 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

NON MALICIOUS PROGRAM ERRORS

Being human, programmers and other developers make many mistakes, most of which are unintentional andnonmalicious. 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.

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 ten elements of the array, sample[0] through sample[9]. Now we execute the statement:

sample[10] = 'A';

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] = 'A';

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. 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.

VIRUS 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, so they 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? Most users cannot answer these questions. However, since computer data are not usually seen directly by users, 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.

Why Worry About Malicious Code?

None of us likes 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.

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 modem), 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.

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. In that lecture, he refers to an earlier Air Force document, the Multics security evaluation [KAR74, KAR02]. In fact,

references to virus behavior go back at least to 1970. Ware's 1970 study (publicly released in 1979 [WAR79]) and Anderson's planning study for the U.S. Air Force [AND72] (to which Schell also refers) *still* accurately describe threats, 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.

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? We address these questions in the following sections.

Kinds of Malicious Code

Malicious code or a **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 eliminates 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 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 have 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.

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,

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.

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.

A **worm** is a program that spreads copies of itself through a network. 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.

White et al. also define 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 3-1.

TABLE 3-1 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 resource

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 will not be 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, and so in the following sections we use the term "virus" almost exclusively. The points made apply also to other forms of malicious code.

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 an 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), will 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.

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.

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.

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.

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.. Clearly, the virus writer has to know the exact structure of the original program to know where to insert which pieces of the virus.

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.

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, 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.

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" (like some rock stars, where the target is the artiste formerly known as 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.

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.

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, non-specific 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.

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.

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.

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 knows to look for it. The virus's signature is important for creating a program, called a **virus scanner**, that can automatically 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?NNNNNNNNNNNNNNNNNNNNNNNNNNNNNN

NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN

NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN

NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN

NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN

%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. Side-bar 3-4 describes how viruses were the primary security breach among companies surveyed in 2001.

Storage Patterns

Most viruses attach to programs that are stored on media such as disks. The attached virus piece is invariant, so that 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.

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 wants to be executed first).

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 3-2, 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, there are no key signals that point to the presence of a virus.

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.

TABLE 3-2 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 handler address table	· Intercept interrupt by modifying interrupt · 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 falsify result	· Intercept system calls that would reveal self and · 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 deactivation	· Activate before deactivating program and block · Store copy to reinfect after deactivation

Section 3.3 Viruses and Other Malicious Code

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 diskette 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 diskettes as they are accessed.

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. 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 more sophisticated polymorphic virus randomly intersperses harmless instructions throughout its code. Examples of harmless instructions include addition of zero to a number,

movement of a data value to its own location, or a jump to the next instruction. These "extra" instructions make it more difficult to locate an invariant signature.

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, and 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.

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 may—or may not—be security relevant. One may run faster, or the first may use a temporary file for work space 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.

Prevention of Virus Infection

The only way to prevent the infection of a virus is not to share executable code with 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, and other office- or business-related files can contain code or scripts that can be executed in various ways—and 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.

Since 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, Kephart et al. observe that 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 with no hard disk, not connected to a network, and with the boot disk removed. 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 running the

suspect program. Only if the program passes these tests should it be installed 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).

- *Use virus detectors (often called virus scanners) regularly and update them daily.* Many of the virus detectors available 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.

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 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 workstations, and mainframe computers. In fact, no writeable stored-program computer is immune to possible virus attack. As we noted in Chapter 1, this situation means that *all* devices containing computer code, including automobiles, airplanes, microwave ovens, radios, televisions, 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 upon 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 in 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 electronic mail message or made available for download from a web site. Any mechanism for sharing files—of programs, data, documents, and so forth—can be used to transfer a virus.

- *Viruses cannot remain in memory after a complete power off/power on reboot*. ***True***. If a virus is resident in memory, the virus is lost when the memory loses power. That is, computer memory (RAM) is volatile, so that all contents are deleted when power is lost.[2] However, viruses written to disk certainly can remain through a reboot cycle and reappear after the reboot. 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 (so-called firmware) that can be accessed under program control, that storage *is* subject to virus attack. There have been a few

- *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.

TARGETED MALICIOUS PROGRAM

So far, we have looked at anonymous code written to affect users and machines indiscriminately. Another class of malicious code is written for a particular system, for a particular application, and for a particular purpose. Many of the virus writers' techniques apply, but there are also some new ones.

Trapdoors

A **trapdoor** is an undocumented entry point to a module. The trapdoor is inserted during code development, perhaps to test the module, to provide "hooks" by which to connect future modifications or enhancements or to allow access if the module should fail in the future. In addition to these legitimate uses, trapdoors can allow a programmer access to a program once it is placed in production.

Salami Attack

An attack known as a **salami attack**. This approach gets its name from the way odd bits of meat and fat are fused together in a sausage or salami. In the same way, a salami attack merges bits of seemingly inconsequential data to yield powerful results. For example,

programs often disregard small amounts of money in their computations, as when there are fractional pennies as interest or tax is calculated.

Such programs may be subject to a salami attack, because the small amounts are shaved from each computation and accumulated elsewhere—such as the programmer's bank account! The shaved amount is so small that an individual case is unlikely to be noticed, and the accumulation can be done so that the books still balance overall. However, accumulated amounts can add up to a tidy sum, supporting a programmer's early retirement or new car. It is often the resulting expenditure, not the shaved amounts, that gets the attention of the authorities.

Covert Channels: Programs That Leak Information

So far, we have looked at malicious code that performs unwelcome actions. Next, we turn to programs that communicate information to people who should not receive it. The communication travels unnoticed, accompanying other, perfectly proper, communications. The general name for these extraordinary paths of communication is **covert channels**.

Suppose a group of students is preparing for an exam for which each question has four choices (a, b, c, d); one student in the group, Sophie, understands the material perfectly and she agrees to help the others. She says she will reveal the answers to the questions, in order, by coughing once for answer "a," sighing for answer "b," and so forth. Sophie uses a communications channel that outsiders may not notice; her communications are hidden in an open channel. This communication is a human example of a covert channel.

Timing Channels

Other covert channels, called **timing channels**, pass information by using the speed at which things happen. Actually, timing channels are shared resource channels in which the shared resource is time.

A service program uses a timing channel to communicate by using or not using an assigned amount of computing time. In the simple case, a multiprogrammed system with two user

processes divides time into blocks and allocates blocks of processing alternately to one process and the other. A process is offered processing time, but if the process is waiting for another event to occur and has no processing to do, it rejects the offer. The service process either uses its block (to signal a 1) or rejects its block (to signal a 0).

CONTROL AGAINST PROGRAM THREAT

There are many ways a program can fail and many ways to turn the underlying faults into security failures. It is of course better to focus on prevention than cure; how do we use controls during software development—the specifying, designing, writing, and testing of the program—to find and eliminate the sorts of exposures we have discussed? The discipline of software engineering addresses this question more globally, devising approaches to ensure the quality of software. In this book, we provide an overview of several techniques that can prove useful in finding and fixing security flaws .

In this section we look at three types of controls: developmental, operating system, and administrative. We discuss each in turn.

Developmental Controls

Many controls can be applied during software development to ferret out and fix problems. So let us begin by looking at the nature of development itself, to see what tasks are involved in specifying, designing, building, and testing software. The Nature of Software Development Software development is often considered a solitary effort; a programmer sits with a specification or design and grinds out line after line of code. But in fact, software development is a collaborative effort, involving people with different skill sets who combine their expertise to produce a working product. Development requires people who can

- specify the system, by capturing the requirements and building a model of how the system should work from the users' point of view
- design the system, by proposing a solution to the problem described by the requirements and building a model of the solution
- implement the system, by using the design as a blueprint for building a working solution
- test the system, to ensure that it meets the requirements and implements the solution as called for in the design
- review the system at various stages, to make sure that the end products are consistent with the specification and design models
- document the system, so that users can be trained and supported
- manage the system, to estimate what resources will be needed for development and to track when the system will be done
- maintain the system, tracking problems found, changes needed, and changes made, and evaluating their effects on overall quality and functionality

One person could do all these things. But more often than not, a team of developers works together to perform these tasks. Sometimes a team member does more than one activity; a tester can take part in a requirements review, for example, or an implementer can write documentation. Each team is different, and team dynamics play a large role in the team's success.

We can examine both product and process to see how each contributes to quality and in particular to security as an aspect of quality. Let us begin with the product, to get a sense of how we recognize highquality secure software.

Modularity, Encapsulation, and Information Hiding

Code usually has a long shelf-life, and it is enhanced over time as needs change and faults are found and fixed. For this reason, a key principle of software engineering is to create a design or code in small, self-contained units, called components or modules; when a system is written this way, we say that it is modular. Modularity offers advantages for program development in general and security in particular.

If a component is isolated from the effects of other components, then it is easier to trace a problem to the fault that caused it and to limit the damage the fault causes. It is also easier to maintain the system, since changes to an isolated component do not affect other components. And it is easier to see where vulnerabilities may lie if the component is isolated. We call this isolation encapsulation.

Information hiding is another characteristic of modular software. When information is hidden, each component hides its precise implementation or some other design decision from the others. Thus, when a change is needed, the overall design can remain intact while only the necessary changes are made to particular components

2.6 PROTECTION IN GENERAL PURPOSE OPERATING SYSTEM PROTECTED OBJECT AND METHOD OF PROTECTION MEMORY AND ADDRESS PROTECTION

Protected objects

The rise of multiprogramming meant that several aspects of a computing system required protection:

- memory
- sharable I/O devices, such as disks
- serially reusable I/O devices, such as printers and tape drives
- sharable programs and subprocedures
- networks
- sharable data

As it assumed responsibility for controlled sharing, the operating system had to protect theseobjects.

Security in operating system

The basis of protection is separation: keeping one user's objects separate from other users.Rushby and Randell noted that separation in an operating system can occur in several ways:

- *physical separation*, in which different processes use different physical objects, such as separate printers for output requiring different levels of security

- *temporal separation*, in which processes having different security requirements are executed at different times
- *logical separation*, in which users operate under the illusion that no other processes exist, as when an operating system constrains a program's accesses so that the program cannot access objects outside its permitted domain
- *cryptographic separation*, in which processes conceal their data and computations in such a way that they are unintelligible to outside processes

Of course, combinations of two or more of these forms of separation are also possible.
The categories of separation are listed roughly in increasing order of complexity to implement, and, for the first three, in decreasing order of the security provided. However, the first two approaches are very stringent and can lead to poor resource utilization. Therefore, we would like to shift the burden of protection to the operating system to allow concurrent execution of processes having different security needs.
But separation is only half the answer. We want to separate users and their objects, but we also want to be able to provide sharing for some of those objects. For example, two users with different security levels may want to invoke the same search algorithm or function call.

We would like the users to be able to share the algorithms and functions without compromisingtheir individual security needs. An operating system can support separation and sharing in several ways, offering protection at any of several levels.

- *Do not protect.* Operating systems with no protection are appropriate when sensitive procedures are being run at separate times.
- *Isolate*. When an operating system provides isolation, different processes running concurrently are unaware of the presence of each other. Each process has its own address space, files, and other objects. The operating system must confine each process somehow so that the objects of the other processes are completely concealed.
- *Share all or share nothing.* With this form of protection, the owner of an objectdeclares it to be public or private. A public object is available to all users, whereas a

private object is available only to its owner.

- *Share via access limitation.* With protection by access limitation, the operating system checks the allowability of each user's potential access to an object. That is, access control is implemented for a specific user and a specific object. Lists of acceptable actions guide the operating system in determining whether a particular user should have access to a particular object. In some sense, the operating system acts as a guard between users and objects, ensuring that only authorized accesses occur.
- *Share by capabilities.* An extension of limited access sharing, this form of protection allows dynamic creation of sharing rights for objects. The degree of sharing can depend on the owner or the subject, on the context of the computation, or on the object itself.
- *Limit use of an object.* This form of protection limits not just the access to an object but the use made of that object after it has been accessed. For example, a user may be allowed to view a sensitive document, but not to print a copy of it. More powerfully, a user may be allowed access to data in a database to derive statistical summaries (such as average salary at a particular grade level), but not to determine specific data values (salaries of individuals).

Methods of memory protection

Memory protection is a way to control memory access rights on a computer, and is a part of most modern operating systems. The main purpose of memory protection is to prevent a process from accessing memory that has not been allocated to it. This prevents a bug within a process from affecting other processes, or the operating system itself, and instead results in

a segmentation fault or storage violation exception being sent to the offending process, generally causing abnormal termination (killing the process). Memory protection for computer security includes additional techniques such as address space layout randomization and executable space protection.

Segmentation

Segmentation refers to dividing a computer's memory into segments. A reference to a memory location includes a value that identifies a segment and an offset within that segment.The x86 architecture has multiple segmentation features, which are helpful for using protected memory on this architecture. On the x86 processor architecture, the Global Descriptor Table and Local Descriptor Tables can be used to reference segments in the computer's memory. Pointers to memory segments on x86 processors can also be stored in the processor's segment registers. Initially x86 processors had 4 segment registers, CS (code segment), SS (stack segment), DS (data segment) and ES (extra segment); later another two segment registers were added – FS and GS.

Paged virtual memory

In paging the memory address space is divided into equal-sized blocks called pages. Using virtual memory hardware, each page can reside in any location of the computer's physical memory, or be flagged as being protected. Virtual memory makes it possible to have a linear virtual memory address space and to use it to access blocks fragmented over physical memory address space.Most computer architectures which support paging also use pages as the basis for memory protection.

A *page table* maps virtual memory to physical memory. The page table is usually invisible to the process. Page tables make it easier to allocate additional memory, as each new page can be allocated from anywhere in physical memory.

It is impossible for an application to access a page that has not been explicitly allocated to it, because every memory address either points to a page allocated to that application, or generates an interrupt called a *page fault*. Unallocated pages, and pages allocated to any other application, do not have any addresses from the application point of view.

A page fault may not necessarily indicate an error. Page faults are not only used for memory protection. The operating system may manage the page table in such a way that a reference to a page that has been previously swapped out to disk causes a page fault. The operating system intercepts the page fault and, loads the required memory page, and the application continues as if no fault had occurred. This scheme, known as virtual memory, allows in-memory data not currently in use to be moved to disk storage and back in a way which is transparent to applications, to increase overall memory capacity.

software fault handler can, if desired, check the missing key against a larger list of keys maintained by software; thus, the protection key registers inside the processor may be treated as a software-managed cache of a larger list of keys associated with a process.

Simulated segmentation

Simulation is use of a monitoring program to interpret the machine code instructions of some computer architectures. Such an Instruction Set Simulator can provide memory protection by using a segmentation-like scheme and validating the target address and length of each instruction in real time before actually executing them. The simulator must calculate the target address and length and compare this against a list of valid address ranges that it holds concerning the thread's environment, such as any dynamic memoryblocks acquired since the thread's inception, plus any valid shared static memory slots. The meaning of "valid" may change throughout the thread's life depending upon context. It may sometimes be allowed to alter a static block of storage, and sometimes not, depending upon the current mode of execution, which may or may not depend on a storage key or supervisor state.

It is generally not advisable to use this method of memory protection where adequate facilities exist on a CPU, as this takes valuable processing power from the computer. However, it is generally used for debugging and testing purposes to provide an extra fine level of granularity to otherwise generic storage violations and can indicate precisely which

instruction is attempting to overwrite the particular section of storage which may have the same storage key as unprotected storage.

FILE PROTECTION MECHANISM

Until now, we have examined approaches to protecting a general object, no matter the object's nature or type. But some protection schemes are particular to the type. To see how they work, we focus in this section on file protection. The examples we present are only representative; they do not cover all possible means of file protection on the market.

Basic Forms of Protection

We noted earlier that all multiuser operating systems must provide some minimal protection to keep one user from maliciously or inadvertently accessing or modifying the files of another. As the number of users has grown, so also has the complexity of these protection schemes.

All “None Protection

In the original IBM OS operating systems, files were by default public. Any user could read, modify, or delete a file belonging to any other user. Instead of software- or hardware-based protection, the principal protection involved trust combined with ignorance. System designers supposed that users could be trusted not to read or modify others' files, because the users would expect the same respect from others. Ignorance helped this situation, because a user could access a file only by name ; presumably users knew the names only of those files to which they had legitimate access.

However, it was acknowledged that certain system files were sensitive and that the system administrator could protect them with a password. A normal user could exercise this feature, but passwords were viewed as most valuable for protecting operating system files. Two philosophies guided password use. Sometimes, passwords were used to control all accesses (read, write, or delete), giving the system administrator complete control over all files. But at other times passwords would control only write and delete accesses , because only these two actions affected other users. In either case, the password mechanism required a system operator's intervention each time access to the filebegan .

However, this all-or-none protection is unacceptable for several reasons.

- Lack of trust . The assumption of trustworthy users is not necessarily justified. For systems with few users who all know each other, mutual respect might suffice; but in large systems where not every user knows every other user, there is no basis for trust.
- All or nothing . Even if a user identifies a set of trustworthy users, there is no convenient way to allow access only to them.
- Rise of timesharing . This protection scheme is more appropriate for a batch environment, in which users have little chance to interact with other users and in which users do their thinking and exploring when not interacting with the system. However, on timesharing systems, users interact with other users. Because users choose when to execute programs, they are more likely in a timesharing environment to arrange computing tasks to be able to pass results from one program or one user to another.
- Complexity . Because (human) operator intervention is required for this file protection, operating system performance is degraded. For this reason, this type of file protection is discouraged by computing centers for all but the most sensitive data sets.
- File listings . For accounting purposes and to help users remember for what files they are responsible, various system utilities can produce a list of all files. Thus, users are not

necessarily ignorant of what files reside on the system. Interactive users may try to browse through anyunprotected files.

Group Protection

Because the all-or-nothing approach has so many drawbacks, researchers sought an improved way to protect files. They focused on identifying groups of users who had some common relationship. In a typical implementation, the world is divided into three classes: the user, a trusted working group associated with the user, and the rest of the users. For simplicity we can call these classes user, group, and world . This form of protection is used on some network systems and the Unix system.

All authorized users are separated into groups. A group may consist of several members working on a common project, a department, a class, or a single user. The basis for group membership is need to share . The group members have some common interest and therefore are assumed to have files to share with the other group members. In this approach, no user belongs to more than one group. (Otherwise, a member belonging to groups A and B could pass along an A file to another B group member.)

When creating a file, a user defines access rights to the file for the user, for other members of the same group, and for all other users in general. Typically, the choices for access rights are a limited set, such as {read, write, execute, delete}. For a particular file, a user might declare read-only access to the general world, read and write access to the group, and all rights to the user. This approach would be suitable for a paper being developed by a group, whereby the different members of the group might modify sections being written within the group. The paper itself should be available for people outside the group to review but not change.

A key advantage of the group protection approach is its ease of implementation. A user is recognized by two identifiers (usually numbers): a user ID and a group ID. These identifiers are stored in the file directory entry for each file and are obtained by the operating system when a user logs in. Therefore, the operating system can easily check whether a proposed access to a file is requested from someone whose group ID matches the group ID for the file to be accessed.

Although this protection scheme overcomes some of the shortcomings of the all-or-nothing scheme, it introduces some new difficulties of its own.

- Group affiliation . A single user cannot belong to two groups. Suppose Tom belongs to one group with Ann and to a second group with Bill. If Tom indicates that a file is to be readable by the group, to which group(s) does this permission refer? Suppose a file of Ann's is readable by the group; does Bill have access to it? These ambiguities are most simply resolved by declaring that every user belongs to exactly one group. (This restriction does not mean that all users belong to the same group.)
- Multiple personalities . To overcome the one-person one-group restriction, certain people might obtain multiple accounts, permitting them, in effect, to be multiple users. This hole in the protection approach leads to new problems, because a single person can be only one user

at a time. To see how problems arise, suppose Tom obtains two accounts, thereby becoming Tom1 in a group with Ann and Tom2 in a group with Bill. Tom1 is not in the same group as Tom2, so any files, programs, or aids developed under the Tom1 account can be available to Tom2 only if they are available to the entire world. Multiple personalities lead to a proliferation of accounts, redundant files, limited protection for files of general interest, and inconvenience to users.
- All groups . To avoid multiple personalities, the system administrator may decide that Tom should have access to all his files any time he is active. This solution puts the responsibility

on Tom to control with whom he shares what things. For example, he may be in Group1 with Ann and Group2 with Bill. He creates a Group1 file to share with Ann. But if he is active in Group2 the nexttime he is logged in, he still sees the Group1 file and may not realize that it is not accessible to Bill, too.
- Limited sharing . Files can be shared only within groups or with the world. Users want to be able to identify sharing partners for a file on a per-file basis, for example, sharing one file

with ten people and another file with twenty others.

Single Permissions

In spite of their drawbacks, the file protection schemes we have described are relatively simple and straightforward. The simplicity of implementing them suggests other easy-to-manage methods that provide finer degrees of security while associating permission with a single file.

Password or Other Token

We can apply a simplified form of password protection to file protection by allowing a user to assign a password to a file. User accesses are limited to those who can supply the correct password at the time the file is opened. The password can be required for any access or only for modifications (write access).

Password access creates for a user the effect of having a different "group" for every file. However, file passwords suffer from difficulties similar to those of authentication passwords:

- Loss . Depending on how the passwords are implemented, it is possible that no one will be

able to replace a lost or forgotten password. The operators or system administrators can certainlyintervene and unprotect or assign a particular password, but often they cannot

determine what password a user has assigned; if the user loses the password, a new one must be assigned.

- Use . Supplying a password for each access to a file can be inconvenient and time consuming.
- Disclosure . If a password is disclosed to an unauthorized individual, the file becomes immediately accessible. If the user then changes the password to reprotect the file, all the other legitimate users must be informed of the new password because their old password will fail.
- Revocation . To revoke one user's access right to a file, someone must change the password, thereby causing the same problems as disclosure.

Temporary Acquired Permission

The Unix operating system provides an interesting permission scheme based on a three-level user “group “world hierarchy. The Unix designers added a permission called set userid (suid) . If this protection is set for a file to be executed, the protection level is that of the file's owner , not theexecutor . To see how it works, suppose Tom owns a file and allows Ann to execute it with suid . When Ann executes the file, she has the protection rights of Tom, not of herself.

This peculiar-sounding permission has a useful application. It permits a user to establish data files to which access is allowed only through specified procedures.

For example, suppose you want to establish a computerized dating service that manipulates a database of people available on particular nights. Sue might be interested in a date for Saturday, but she might have already refused a request from Jeff, saying she had other plans. Sue instructs the service not to reveal to Jeff that she is available. To use the service, Sue, Jeff, and others must be able to read and write (at least indirectly) the file to determine who is available or to post their availability. But if Jeff can read the file directly, he would find that

Sue has lied. Therefore, your dating service must force Sue and Jeff (and all others) to access this file only through an access program that would screen the data Jeff obtains. But if the file access is limited to read and write by you as its owner, Sue and Jeff will never be able to enter data into it.

The solution is the Unix SUID protection. You create the database file, giving only you access permission. You also write the program that is to access the database, and save it with the SUID protection. Then, when Jeff executes your program, he temporarily acquires your access permission, but only during execution of the program. Jeff never has direct access to the file because your program will do the actual file access. When Jeff exits from your program, he regains his own access rights and loses yours. Thus, your program can access the file, but the program must display to Jeff only the data Jeff is allowed to see.

This mechanism is convenient for system functions that general users should be able to perform only in a prescribed way. For example, only the system should be able to modify the file of users' passwords, but individual users should be able to change their own passwords any time they wish. With the SUID feature, a password change program can be owned by the system, which will therefore have full access to the system password table. The program to change passwords also has SUID protection, so that when a normal user executes it, the program can modify the password file in acarefully constrained way on behalf of the user.

USER AUTHENTICATION

An operating system bases much of its protection on knowing who a user of the system is. In real-life situations, people commonly ask for identification from people they do not know: A bank employee may ask for a driver's license before cashing a check, library employees may require some identification before charging out books, and immigration officials ask for passports as proof of identity. In-person identification is usually easier than remote identification. For instance, some universities do not report grades over the telephone because the office workers do not necessarily know the students calling. However, a professor who recognizes the voice of a certain student can release that student's grades. Over time, organizations and systems have developed means of authentication, using documents, voice recognition, fingerprint and retina matching, and other trusted means of identification.

In computing, the choices are more limited and the possibilities less secure. Anyone can attempt to log in to a computing system. Unlike the professor who recognizes a student's voice, the computer cannot recognize electrical signals from one person as being any different from those of anyone else. Thus, most computing authentication systems must be based on some knowledge shared only by the computing system and the user. Authentication mechanisms use any of three qualities to confirm a user's identity.

1. Something the user *knows*a Passwords, PIN numbers, passphrases, a secret handshake, and mother's maiden name are examples of what a user may know.
2. Something the user *has*a Identity badges, physical keys, a driver's license, or a uniform are common examples of things people have that make them recognizable.
3. Something the user *is*a These authenticators, called biometrics, are based on a physical characteristic of the user, such as a fingerprint, the pattern of a person's voice, or a face (picture). These authentication methods are old (we recognize friends in person by their faces or on a telephone by their voices) but are just starting to be used in computer authentication.

Passwords as Authenticators

The most common authentication mechanism for user to operating system is a password, a "word" known to computer and user. Although password protection seems to offer a relatively secure system, human practice sometimes degrades its quality. In this section we consider passwords, criteria for selecting them, and ways of using them for authentication. We conclude by noting other authentication techniques and by studying problems in the authentication process, notably Trojan horses masquerading as the computer authentication process.

Use of Passwords

Passwords are mutually agreed-upon code words, assumed to be known only to the user and the system. In some cases a user chooses passwords; in other cases the system assigns them. The length and format of the password also vary from one system to another.

Even though they are widely used, passwords suffer from some difficulties of use:

- Loss. Depending on how the passwords are implemented, it is possible that no one will be able to replace a lost or forgotten password. The operators or system administrators can certainly intervene and unprotect or assign a particular password, but often they cannot determine what password a user has chosen; if the user loses the password, a new one must be assigned.
- Use. Supplying a password for each access to a file can be inconvenient and time consuming.
- Disclosure. If a password is disclosed to an unauthorized individual, the file becomes immediately accessible. If the user then changes the password to reprotect the file, all other legitimate users must be informed of the new password because their old password will fail.

· Revocation. To revoke one user's access right to a file, someone must change the password, thereby causing the same problems as disclosure.

The use of passwords is fairly straightforward. A user enters some piece of identification, such as a name or an assigned user ID; this identification can be available to the public or easy to guess because it does not provide the real security of the system. The system then requestsa password from the user. If the password matches that on file for the user, the user is authenticated and allowed access to the system. If the password match fails, the system requests the password again, in case the user mistyped.

Security in Network

THREATS IN NETWORK

Main aims of threats are to compromise confidentiality, integrity applied against data, software, hardware by nature accidents, non-malicious humans and malicious attackers.

What Makes A Network Vulnerable?

1. *Anonymity*
2. *Many Points Of Attack*
3. *Sharing*
4. *Complexity Of System*

Threat Precursors:

1. Port scan
2. Social Engineering
3. Reconnaissance
4. Operating System and Application fingerprinting
5. Bulletin Boards and chats
6. Availability of Documentation

Threats In Transit: Eavesdropping and Wiretapping

The term **eavesdrop** implies overhearing without expanding any extra effort. For example we can say that an attacker is eavesdropping by monitoring all traffic passing through a node.

The more hostile term is **wiretap**, which means intercepting communication through some effort.

Choices of wiretapping are:

1. Cable
2. Microwave
3. Satellite Communication
4. Optical Fiber
5. Wireless

From, a security stand point we should assume all communication links between network nodes that can broken. For this reason commercial network users employ encryption to protect the confidentiality of their communication.

Protocol Flaws:

Each protocol is identified by its Request For Comment (RFC) number. In TCP, the sequence number of the client increments regularly which can be easily guessed and also which will be the next number.

Impersonation:

In many instances, there is an easier way than wiretapping for obtaining information on a network: impersonate another person or process.

In impersonation, an attacker has several choices:

- Guess the identity and authentication details of the target
- Disable authentication mechanism at the target computer
- Use a target that will not be authenticated
- Use a target whose authentication data are known

Spoofing:

Obtaining the network authentication credentials of an entity(a user, an account, a process, a node, a device) permits an attacker to create a full communication under the entity's identity. Examples of spoofing are masquerading, session hijacking, and man-in-the-middle attacks.

- In a masquerade one host pretends to be another.
- Session hijacking is intercepting and carrying on a session begun by another entity.
- Man-in-the-middle attack is a similar form of attack, in which one entity intrudes between two others.

Message Confidentiality Threats:

An attacker can easily violate message confidentiality (and perhaps integrity) because of the public nature of networks. Eavesdropping and impersonation attacks can lead to a confidentiality or integrity failure. Here we consider several other vulnerabilities that can affect confidentiality.

1. Misdelivery
2. Exposure
3. Traffic Flow Analysis

Message Integrity Threats:

In many cases, the *integrity* or correctness of a communication is at least as important as its confidentiality. In fact for some situations, such as passing authentication data, the integrity of the communication is paramount. Threats based upon failures of integrity in communication

- Falsification of messages
- Noise

Web Site Defacement:

One of the most widely known attacks is the web site defacement attack. Because of the large number of sites that have been defaced and the visibility of the result, the attacks are often reported in the popular press. A defacement is common not only because of its visibility but also because of the ease with which one can be done.

The website vulnerabilities enable attacks known as buffer overflows, dot-dot problems, application code errors, and server side include problems.

Denial of Service:

Availability attacks, sometimes called denial-of-service or DOS attacks, are much more significant in networks than in other contexts. There are many accidental and malicious threats to availability or continued service. There are many accidental and malicious threats to availability or continued service.

1) Transmission Failure
2) Connection Flooding
3) Echo-Chargen
4) Ping of Death
5) Smurf
6) Syn Flood
7) Teardrop
8) Traffic Redirection
9) DNS Attacks

Threats in Active or Mobile Code:

Active code or mobile code is a general name for code that is pushed to the client for execution. Why should the web server waste its precious cycles and bandwidth doing simple work that the client's workstation can do? For example, suppose you want your web site to have bears dancing across the top of the page. To download the dancing bears, you could download a new image for each movement the bears take: one bit forward, two bits forward, and so forth. However, this approach uses far too much server time and bandwidth to compute the positions and download new images. A more efficient use of (server) resources is to download a program that runs on the client's machine and implements the movement of the bears.

Network Security Controls

The list of security attacks is long, and the news media carry frequent accounts of serious security incidents.

Security Threat Analysis:

The three steps of a security threat analysis in other situations are described here. First, we scrutinize all the parts of a system so that we know what each part does and how it interacts with other parts. Next, we consider possible damage to confidentiality, integrity, and availability. Finally, we hypothesize the kinds of attacks that could cause this damage. We can take the same steps with a network. We begin by looking at the individual parts of a network:

All the threats are summarized with a list as

- Intercepting data in traffic
- Accessing programs or data at remote hosts
- Modifying programs or data at remote hosts
- Modifying data in transit
- Inserting communications
- Impersonating a user
- Inserting a repeat of a previous communication
- Blocking selected traffic
- Blocking all traffic
- Running a program at a remote host

Design and Implementation:

Architecture:

As with so many of the areas we have studied, planning can be the strongest control. In particular, when we build or modify computer-based systems, we can give some thought to their overall architecture and plan to "build in" security as one of the key constructs. Similarly, the architecture or design of a network can have a significant effect on its security. The main areas to cover are

- Segmentation
- Redundancy
- Single point of failure
- Mobile agents

Encryption:

Encryption is powerful for providing privacy, authenticity, integrity, and limited access to data. Because networks often involve even greater risks, they often secure data with encryption, perhaps in combination with other controls. There are 2 types of encryption scheme exists:

- Link encryption (data are encrypted just before the system places them on the physical communications link)
- End-to-end encryption (provides security from one end of a transmission to the other)

Content Integrity:

Content integrity comes as a bonus with cryptography. No one can change encrypted data in a meaningful way without breaking the encryption. This does not say, however, that encrypted data cannot be modified. Changing even one bit of an encrypted data stream affects the result after decryption, often in a way that seriously alters the resulting plaintext. We need to consider three potential threats:

Malicious modification that changes content in a meaningful way

Malicious or non-malicious modification that changes content in a way that is not necessarily meaningful

non-malicious modification that changes content in a way that will not be detected

Encryption addresses the first of these threats very effectively. To address the others, we can use other controls.

Strong Authentication:

In the network case, however, authentication may be more difficult to achieve securely because of the possibility of eavesdropping and wiretapping, which are less common in non-networked environments. Also, both ends of a communication may need to be authenticated to each other.

Here the main issues are

- One time password
- Challenge response systems
- Digital distributed authentication

Access Controls:

Authentication deals with the *who* of security policy enforcement; access controls enforce the *what* and *how*.

ACLs on Routers

Routers perform the major task of directing network traffic either to sub-networks they control or to other routers for subsequent delivery to other sub-networks. Routers convert external IP addresses into internal MAC addresses of hosts on a local sub-network. Suppose a host is being spammed (flooded) with packets from a malicious rogue host. Routers can be configured with access control lists to deny access to particular hosts from particular hosts. So, a router could delete all packets with a source address of the rogue host and a destination address of the target host.

Alarms and Alerts:

The logical view of network protection looks like the figure below, in which both a router and a firewall provide layers of protection for the internal network. Now let us add one more layer to this defense.

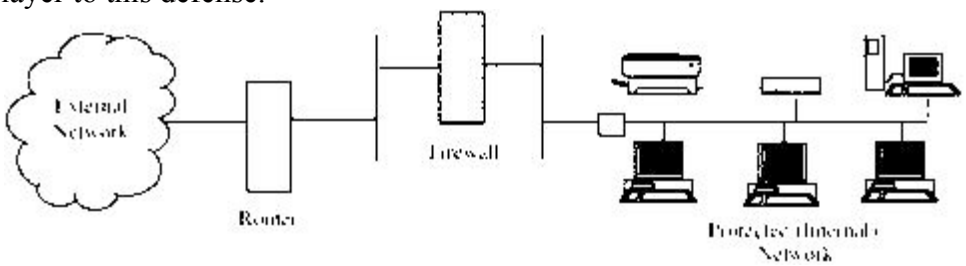

Fig. Layered network protection

Honey Pot: (A computer system open for attackers**)**

A honey pot has no special features. It is just a computer system or a network segment, loaded with servers and devices and data. It may be protected with a firewall, although you want the attackers to have some access. There may be some monitoring capability, done carefully so that the monitoring is not evident to the attacker.

We put up a honey pot for several reasons:

To watch what attackers do, in order to learn about new attacks (so that you can strengthen your defenses against these new attacks)
To lure an attacker to a place in which you may be able to learn enough to identify and stop the attacker
To provide an attractive but diversionary playground, hoping that the attacker will leave your real system alone

Firewalls

Firewalls were officially invented in the early 1990s, but the concept really reflects the reference monitor from two decades earlier.

What is a Firewall?

A firewall is a device that filters all traffic between a protected or "inside" network and a less trustworthy or "outside" network. Usually a firewall runs on a dedicated device; because it is a single point through which traffic is channeled, performance is important, which means non-firewall functions should not be done on the same machine. Because a firewall is executable code, an attacker could compromise that code and execute from the firewall's device. Thus, the fewer pieces of code on the device, the fewer tools the attacker would have by compromising the firewall. Firewall code usually runs on a proprietary or carefully minimized operating system. The purpose of a firewall is to keep "bad" things outside a protected environment. To accomplish that, firewalls implement a security policy that is specifically designed to address what bad things might happen. For example, the policy might be to prevent any access from outside (while still allowing traffic to pass *from* the inside *to* the outside). Alternatively, the policy might permit accesses only from certain places, from certain users, or for certain activities. Part of the challenge of protecting a network with a firewall is determining which security policy meets the needs of the installation.

Design of Firewalls:

A reference monitor must be

Always invoked
Tamperproof
Small and simple enough for rigorous analysis

A firewall is a special form of reference monitor. By carefully positioning a firewall within a network, we can ensure that all network accesses that we want to control must pass through it. This restriction meets the "always invoked" condition. A firewall is typically well isolated, making it highly immune to modification. Usually a firewall is implemented on a separate computer, with direct connections only to the outside and inside networks. This isolation is

expected to meet the "tamperproof" requirement. And firewall designers strongly recommend keeping the functionality of the firewall simple.

Types of Firewalls:

Firewalls have a wide range of capabilities. Types of firewalls include

- Packet filtering gateways or screening routers
- Stateful inspection firewalls
- Application proxie
- Guards
- Personal firewalls

Packet Filtering Gateway:

A packet filtering gateway or screening router is the simplest, and in some situations, the most effective type of firewall. A packet filtering gateway controls access to packets on the basis of packet address (source or destination) or specific transport protocol type (such as HTTP web traffic). As described earlier in this chapter, putting ACLs on routers may severely impede their performance. But a separate firewall behind (on the local side) of the router can screen traffic before it gets to the protected network. Figure 7-34 shows a packet filter that blocks access from (or to) addresses in one network; the filter allows HTTP traffic but blocks traffic using the Telnet protocol.

Stateful Inspection Firewall:

Filtering firewalls work on packets one at a time, accepting or rejecting each packet and moving on to the next. They have no concept of "state" or "context" from one packet to the next. A stateful inspection firewall maintains state information from one packet to another in the input stream.

One classic approach used by attackers is to break an attack into multiple packets by forcing some packets to have very short lengths so that a firewall cannot detect the signature of an attack split across two or more packets. (Remember that with the TCP protocols, packets can arrive in any order, and the protocol suite is responsible for reassembling the packet stream in proper order before passing it along to the application.) A stateful inspection firewall would track the sequence of packets and conditions from one packet to another to thwart such an attack.

Application Proxy

Packet filters look only at the headers of packets, not at the data *inside* the packets. Therefore, a packet filter would pass anything to port 25, assuming its screening rules allow inbound

connections to that port. But applications are complex and sometimes contain errors. Worse, applications (such as the e-mail delivery agent) often act on behalf of all users, so they require privileges of all users (for example, to store incoming mail messages so that inside users can read them). A flawed application, running with all users' privileges, can cause much damage. An application proxy gateway, also called a bastion host, is a firewall that simulates the (proper) effects of an application so that the application receives only requests to act properly. A proxy gateway is a two-headed device: It looks to the inside as if it is the outside (destination) connection, while to the outside it responds just as the insider would.

An application proxy runs pseudo-applications. For instance, when electronic mail is transferred to a location, a sending process at one site and a receiving process at the destination communicate by a protocol that establishes the legitimacy of a mail transfer and then actually transfers the mail message. The protocol between sender and destination is carefully defined. A proxy gateway essentially intrudes in the middle of this protocol exchange, seeming like a destination in communication with the sender that is outside the firewall, and seeming like the sender in communication with the real destination on the inside. The proxy in the middle has the opportunity to screen the mail transfer, ensuring that only acceptable e-mail protocol commands are sent to the destination.

Guard:

A guard is a sophisticated firewall. Like a proxy firewall, it receives protocol data units, interprets them, and passes through the same or different protocol data units that achieve either the same result or a modified result. The guard decides what services to perform on the user's behalf in accordance with its available knowledge, such as whatever it can reliably know of the (outside) user's identity, previous interactions, and so forth. The degree of control a guard can provide is limited only by what is computable. But guards and proxy firewalls are similar enough that the distinction between them is sometimes fuzzy. That is, we can add functionality to a proxy firewall until it starts to look a lot like a guard.

Personal Firewalls:

A personal firewall is an application program that runs on a workstation to block unwanted traffic, usually from the network. A personal firewall can complement the work of a conventional firewall by screening the kind of data a single host will accept, or it can compensate for the lack of a regular firewall, as in a private DSL or cable modem connection.

The personal firewall is configured to enforce some policy. For example, the user may decide that certain sites, such as computers on the company network, are highly trustworthy, but most other sites are not. The user defines a policy permitting download of code, unrestricted data sharing, and management access from the corporate segment, but not from other sites. Personal firewalls can also generate logs of accesses, which can be useful to examine in case something harmful does slip through the firewall.

A personal firewall runs on the very computer it is trying to protect. Thus, a clever attacker is likely to attempt an undetected attack that would disable or reconfigure the firewall for the future. Still, especially for cable modem, DSL, and other "always on" connections, the static workstation is a visible and vulnerable target for an ever-present attack community. A personal firewall can provide reasonable protection to clients that are not behind a network firewall.

Comparison of Firewall types:

Packet Filtering	Sateful Inspection	Application Proxy	Guard	Personal firewall
Simple	More complex	Even complex	Most complex	Similar to packet filtering
Sees only addresses and service protocol type	Can see either addresses or data	Sees full data portion of packet	Sees full text of communication	Can see full data portion of packet
Auditing difficult	Auditing possible	Can audit activity	Can audit activity	Can and usually does audit activity
Screensbased onconnection rules	Screensbased on information across packetsin either header or data field	Screens based on behavior of proxies	Screens based on interpretation of message contents	Typically, screens based on information in a single packet, using header or data
Complex addressing rules can make configuration tricky	Usually preconfigured to detect certain attack signatures	Simple proxies can substitute for complex addressing rules	Complex guard functionality can limit assurance	Usually starts in "deny all inbound" mode, to which user adds trusted addresses as they appear

Intrusion Detection System:

An intrusion detection system (IDS) is a device, typically another separate computer, that monitors activity to identify malicious or suspicious events. An IDS is a sensor, like a smoke detector, that raises an alarm if specific things occur. A model of an IDS is shown in below figure. The components in the figure are the four basic elements of an intrusion detection system, based on the Common Intrusion Detection Framework of [STA96]. An IDS receives raw inputs from sensors. It saves those inputs, analyzes them, and takes some controlling action.

Types of IDSs

The two general types of intrusion detection systems are signature based and heuristic. Signature-based intrusion detection systems perform simple pattern-matching and report situations that match a pattern corresponding to a known attack type. Heuristic intrusion detection systems, also known as anomaly based, build a model of acceptable behavior and flag exceptions to that model; for the future, the administrator can mark a flagged behavior as acceptable so that the heuristic IDS will now treat that previously unclassified behavior as acceptable.

Intrusion detection devices can be network based or host based. A network-based IDS is a stand-alone device attached to the network to monitor traffic throughout that network; a host-based IDS runs on a single workstation or client or host, to protect that one host.

Signature-Based Intrusion Detection:

A simple signature for a known attack type might describe a series of TCP SYN packets sent to many different ports in succession and at times close to one another, as would be the case for a port scan. An intrusion detection system would probably find nothing unusual in the first SYN, say, to port 80, and then another (from the same source address) to port 25. But as more and more ports receive SYN packets, especially ports that are not open, this pattern reflects a possible port scan. Similarly, some implementations of the protocol stack fail if they receive an ICMP packet with a data length of 65535 bytes, so such a packet would be a pattern for which to watch.

Heuristic Intrusion Detection:

Because signatures are limited to specific, known attack patterns, another form of intrusion detection becomes useful. Instead of looking for matches, heuristic intrusion detection looks for behavior that is out of the ordinary. The original work in this area focused on the individual, trying to find characteristics of that person that might be helpful in understanding normal and abnormal behavior. For example, one user might always start the day by reading e-mail, write many documents using a word processor, and occasionally back up files. These actions would be normal. This user does not seem to use many administrator utilities. If that person tried to access sensitive system management utilities, this new behavior might be a clue that someone else was acting under the user's identity.

Inference engines work in two ways. Some, called state-based intrusion detection systems, see the system going through changes of overall state or configuration. They try to detect when the system has veered into unsafe modes. Others try to map current activity onto a model of unacceptable activity and raise an alarm when the activity resembles the model. These are called model-based intrusion detection systems. This approach has been extended to networks in [MUK94]. Later work sought to build a dynamic model of behavior, to accommodate variation and evolution in a person's actions over time. The technique compares real activity with a known representation of normality.

Alternatively, intrusion detection can work from a model of known bad activity. For example, except for a few utilities (login, change password, create user), any other attempt to access a password file is suspect. This form of intrusion detection is known as misuse intrusion detection. In this work, the real activity is compared against a known suspicious area.

Stealth Mode:

An IDS is a network device (or, in the case of a host-based IDS, a program running on a network device). Any network device is potentially vulnerable to network attacks. How useful would an IDS be if it itself were deluged with a denial-of-service attack? If an attacker succeeded in logging in to a system within the protected network, wouldn't trying to disable the IDS be the next step?

To counter those problems, most IDSs run in stealth mode, whereby an IDS has two network interfaces: one for the network (or network segment) being monitored and the other

to generate alerts and perhaps other administrative needs. The IDS uses the monitored interface as input only; it *never* sends packets out through that interface. Often, the interface is configured so that the device has no published address through the monitored interface; that is, a router cannot route anything to that address directly, because the router does not know such a device exists. It is the perfect passive wiretap. If the IDS needs to generate an alert, it uses only the alarm interface on a completely separate control network.

Goals for Intrusion Detection Systems:

1. *Responding to alarms:*

Whatever the type, an intrusion detection system raises an alarm when it finds a match. The alarm can range from something modest, such as writing a note in an audit log, to something significant, such as paging the system security administrator. Particular implementations allow the user to determine what action the system should take on what events.

In general, responses fall into three major categories (any or all of which can be used in a single response):

Monitor, collect data, perhaps increase amount of data collected Protect, act to reduce exposure

Call a human

2. *False Results:*

Intrusion detection systems are not perfect, and mistakes are their biggest problem. Although an IDS might detect an intruder correctly most of the time, it may stumble in two different ways: by raising an alarm for something that is not really an attack (called a false positive, or type I error in the statistical community) or not raising an alarm for a real attack (a false negative, or type II error). Too many false positives means the administrator will be less confident of the IDS's warnings, perhaps leading to a real alarm's being ignored. But false negatives mean that real attacks are passing the IDS without action. We say that the degree of false positives and false negatives represents the sensitivity of the system. Most IDS implementations allow the administrator to tune the system's sensitivity, to strike an acceptable balance between false positives and negatives.

IDS strength and limitations:

On the upside, IDSs detect an ever-growing number of serious problems. And as we learn more about problems, we can add their signatures to the IDS model. Thus, over time, IDSs continue to improve. At the same time, they are becoming cheaper and easier to administer. On the downside, avoiding an IDS is a first priority for successful attackers. An IDS that is not well defended is useless. Fortunately, stealth mode IDSs are difficult even to find on an internal network, let alone to compromise. IDSs look for known weaknesses, whether through patterns of known attacks or models of normal behavior. Similar IDSs may have identical vulnerabilities, and their selection criteria may miss similar attacks. Knowing how to evade a particular model of IDS is an important piece of intelligence passed within the attacker community. Of course, once manufacturers become aware of a shortcoming in their products, they try to fix it. Fortunately, commercial IDSs are pretty good at identifying attacks. Another IDS limitation is its sensitivity, which is difficult to measure and adjust. IDSs will never be perfect, so finding the proper balance is critical.

In general, IDSs are excellent additions to a network's security. Firewalls block traffic to particular ports or addresses; they also constrain certain protocols to limit their impact. But by definition, firewalls have to allow some traffic to enter a protected area.

Watching what that traffic actually does inside the protected area is an IDS's job, which it does quite well.

Secure Email:

We rely on e-mail's confidentiality and integrity for sensitive and important communications, even though ordinary e-mail has almost no confidentiality or integrity. Here we investigate how to add confidentiality and integrity protection to ordinary e-mail.

Security of email:

Sometimes we would like e-mail to be more secure. To define and implement a more secure form, we begin by examining the exposures of ordinary e-mail.

Threats to E-mail

Message interception (confidentiality)
Message interception (blocked delivery)
Message interception and subsequent replay
Message content modification
Message origin modification
Message content forgery by outsider
Message origin forgery by outsider
Message content forgery by recipient
Message origin forgery by recipient
Denial of message transmission

Requirements and solutions:

Following protections must be taken for protection in emails

Message confidentiality (the message is not exposed en route to the receiver)
Message integrity (what the receiver sees is what was sent)
Sender authenticity (the receiver is confident who the sender was)
Non repudiation (the sender cannot deny having sent the message)

Designs:

One of the design goals for encrypted e-mail was allowing security-enhanced messages to travel as ordinary messages through the existing Internet e-mail system. This requirement ensures that the large existing e-mail network would not require change to accommodate security. Thus, all protection occurs within the body of a message.

Confidentiality:

The encrypted e-mail standard works most easily as just described, using both symmetric and asymmetric encryption. The standard is also defined for symmetric encryption only: To use symmetric encryption, the sender and receiver must have previously established a shared

secret encryption key. The processing type ("Proc-Type") field tells what privacy enhancement services have been applied. In the data exchange key field ("DEK-Info"), the kind of key exchange (symmetric or asymmetric) is shown. The key exchange ("Key-Info") field contains the message encryption key, encrypted under this shared encryption key. The field also identifies the originator (sender) so that the receiver can determine which shared symmetric key was used. If the key exchange technique were to use asymmetric encryption, the key exchange field would contain the message encryption field, encrypted under the recipient's public key. Also included could be the sender's certificate (used for determining authenticity and for generating replies). The encrypted e-mail standard supports multiple encryption algorithms, using popular algorithms such as DES, triple DES, and AES for message confidentiality, and RSA and Diffie-Hellman for key exchange.

Encryption of secure e-mail:

Encrypted e-mail provides strong end-to-end security for electronic mail. Triple DES, AES, and RSA cryptography are quite strong, especially if RSA is used with a long bit key (1024 bits or more). The vulnerabilities remaining with encrypted e-mail come from the points not covered: the endpoints. An attacker with access could subvert a sender's or receiver's machine, modifying the code that does the privacy enhancements or arranging to leak a cryptographic key.

Examples of Secure E-mail:

- PGP (Pretty Good Privacy)
- S/MIME (Secure Multipurpose Internet Mail Extensions)

PRIVACY IN COMPUTING

Privacy

Privacy is a human right, although people can legitimately disagree over when or to what extent privacy is deserved; this disagreement may have cultural, historical, or personal roots

Aspects of Information Privacy

Information privacy has three aspects: sensitive data, affected parties, and controlled disclosure

Controlled Disclosure

What is privacy?

A good working definition is that privacy is the right to control who knows certain aspects about you, your communications, and your activities. In other words, you voluntarily choose who can know things about you and what those things are.

People ask you for your telephone number: your auto mechanic, a clerk in a store, your tax authority, a new business contact, or a cute person in a bar. You consider why the person wants the number and decide whether to give it out. But the key point is you decide. So privacy is something over which you have considerable influence.

Sensitive Data

Someone asks you for your shoe size; you might answer, "I'm a very private person and cannot imagine why you would want to know such an intimate detail" or you could say "10C"; some people find that data more sensitive than others. We know things people usually consider sensitive, such as financial status, certain health data, unsavory events in their past, and the like, so if you learn something you consider sensitive about someone, you will keep it quiet. But most of us are not too sensitive about our shoe size, so we don't normally protect that if we learn it about someone else. Of course, if a friend told me not to pass that along, I wouldn't. It is not up to me to question why someone else considers something private.

Here are examples (in no particular order) of data many people consider private.

- identity, the ownership of private data and the ability to control its disclosure
- finances, credit, bank details
- legal matters
- medical conditions, drug use, DNA, genetic predisposition to illnesses

Affected Subject

This brings us to another point about privacy: Individuals, groups, companies, organizations, and governments all have data they consider sensitive. So far we have described privacy from the standpoint of a person. Companies may have data they consider private or sensitive: product plans, key customers, profit margins, and newly discovered technologies. For organizations such as companies, privacy usually relates to gaining and maintaining an edge over the competition. Other organizations, for example, schools, hospitals, or charities, may need to protect personal data on their students, patients, or donors, or they may want to control negative news, and so forth. Governments consider military and diplomatic matters sensitive, but they also recognize a responsibility to keep confidential data they collect from citizens, such as tax information. We may use terms like subject or owner to cover privacy issues affecting people, groups, and the like.

Computer-Related Privacy Problems

Rezgui et al. [REZ03] list eight dimensions of privacy (specifically as it relates to the web, although the definitions carry over naturally to other types of computing).

- Information collection: Data are collected only with knowledge and explicit consent.
- Information *usage*: Data are used only for certain specified purposes.
- *Information retention*: Data are retained for only a set period of time.
- Information disclosure: Data are disclosed to only an authorized set of people.
- Information security: Appropriate mechanisms are used to ensure the protection of the data.
- *Access control*: All modes of access to all forms of collected data are controlled.
- *Monitoring*: Logs are maintained showing all accesses to data.
- *Policy changes*: Less restrictive policies are never applied after-the-fact to already obtained data.

Here are the privacy issues that have come about through use of computers.

Privacy Principles and policies

Fair Information Policies

In 1973 Willis Ware of the RAND Corporation chaired a committee to advise the Secretary of the U.S. Department of Human Services on privacy issues. The report (summarized in [WAR73a]) proposes a set of principles of fair information practice.

- Collection limitation. Data should be obtained lawfully and fairly.
- *Data quality*. Data should be relevant to their purposes, accurate, complete, and up-to-date.
- Purpose specification. The purposes for which data will be used should be identified and the data destroyed if no longer necessary to serve that purpose.
- Use limitation. Use for purposes other than those specified is authorized only with consent of the data subject or by authority of law.
- Security safeguards. Procedures to guard against loss, corruption, destruction, or misuse of data should be established.
- Openness. It should be possible to acquire information about the collection, storage, and use of personal data systems.
- Individual participation. The data subject normally has a right to access and to challenge data relating to her.
- *Accountability*. A data controller should be designated and accountable for complying with the measures to give effect to the principles.

These principles describe the rights of individuals, not requirements on collectors; that is, the principles do not require protection of the data collected.

Ware [WAR73b] raises the problem of linking data in multiple files and of overusing keys, such as social security numbers, that were never intended to be used to link records. And although he saw that society was moving toward a universal identity number, he feared that movement would be without plan (and hence without control). He was right, even though he could not have foreseen the amount of data exchanged 30 years later.

Turn and Ware [TUR75] consider protecting the data themselves, recognizing that collections of data will be attractive targets for unauthorized access attacks. They suggest four ways to protect stored data:

- Reduce exposure by limiting the amount of data maintained, asking for only what is necessary and using random samples instead of complete surveys.
- Reduce data sensitivity by interchanging data items or adding subtle errors to the data (and warning recipients that the data have been altered).
- Anonymize the data by removing or modifying identifying data items.
- Encrypt the data.

U.S. Privacy Laws

Ware and his committee expected these principles to apply to all collections of personal data on individuals. Unfortunately, that is not the way the legislation developed.

The Ware committee report led to the 1974 Privacy Act (5 USC 552a), which embodies most of these principles, although that law applies only to data

maintained by the U.S. government. The Privacy Act is a broad law, covering all data collected by the government. It is the strongest U.S. privacy law because of its breadth: It applies to all personal data held anywhere in the government.
The United States subsequently passed laws protecting data collected and held by other organizations, but these laws apply piecemeal, by individual data type. Consumer credit is addressed in the Fair Credit Reporting Act, healthcare information in the Health Insurance,Portability and Accountability Act (HIPAA), financial service organizations in the GrammLeachBliley Act (GLBA), children's web access in the Children's Online Privacy Protection Act (COPPA), and student records in the Federal Educational Rights and Privacy Act. Not
surprisingly these separate laws are inconsistent in protecting privacy.

Laws and regulations do help in some aspects of privacy protection. Antón et al. investigated the impact of the HIPAA law by analysing companies' posted privacy policies before and after the privacy provisions of the law became effective [ANT06]. They found the following
in policies posted after HIPAA:
 Statements on data transfer (to other organizations) were more explicit than before HIPAA.
 Consumers still had little control over the disclosure or dissemination of their data.
 Statements were longer and more complex, making them harder for consumers to understand.
 Even within the same industry branch (such as drug companies), statements varied substantially, making it hard for consumers to
compare policies.
 Statements were unique to specific web pages, meaning they covered more precisely the content and function of a particular page.

Controls on U.S. Government Web Sites

Because privacy is ambiguous, privacy policies are an important way to both define the concept in a particular setting and specify what should or will be done about it.
The Federal Trade Commission (FTC) has jurisdiction over web sites, including those of the federal government, that solicit potentially private data. In 2000 [FTC00], the FTC set requirements for privacy policy for government web sites. Because government web sites are covered by the Privacy Act, it was easy for the FTC to require privacy protection. The FTC determined that in order to obey the Privacy Act, government web sites would have to address five privacy factors.
 Notice. Data collectors must disclose their information practices before collecting personal information from consumers.
 Choice. Consumers must be given a choice as to whether and how personal information collected from them may be used.
 Access. Consumers should be able to view and contest the accuracy and completeness of data collected about them.
 Security. Data collectors must take reasonable steps to ensure that information collected from consumers is accurate and secure
from unauthorized use.
 Enforcement. A reliable mechanism must be in place to impose sanctions for noncompliance with these fair information practices.
In 2002, the U.S. Congress enacted the e-Government Act of 2002 requiring that federal government agencies post privacy policies on their
web sites. Those policies must disclose

 the information that is to be collected
 the reason the information is being collected
 the intended use by the agency of the information
 the entities with whom the information will be shared
 the notice or opportunities for consent that would be provided to individuals regarding what information is collected and how that
information is shared
 the way in which the information will be secured
 the rights of the individual under the Privacy Act and other laws relevant to the protection of the privacy of an individual

These two acts apply only to web sites; data collected by other means (for example, by filing forms) are handled differently, usually on a case-by-case or agency-by-agency basis. The requirements reflected in the e-Government Act focus on the type of data (data supplied to the government through a web site) and not on the general notion of privacy.

Controls on Commercial Web Sites

The e-Government Act places strong controls on government data collection through web sites. As we described, privacy outside the government is protected by law in some areas, such as credit, banking, education, and healthcare. But there is no counterpart to the e- Government Act for private companies.

No Deceptive Practices

The Federal Trade Commission has the authority to prosecute companies that engage in deceptive trade or unfair business practices. If a company advertises in a false or misleading way, the FTC can sue. The FTC has used that approach on web privacy: If a company advertises a false privacy protectionthat is, if the company says it will protect privacy in some way but does not do sothe FTC considers that false advertising and can take legal action. Because of the FTC, privacy notices at the bottom of web sites do have meaning. This practice leads to a bizarre situation, however. A company is allowed to collect personal information and pass it in any form to anyone, as long as the company's privacy policy said it would do so, or at least the policy did not say it would not do so. Vowing to maintain privacy and intentionally not doing so is an illegal deceptive practice. Stating an intention to share data with marketing firms or "other third parties" makes such sharing acceptable, even though the third parties could be anyone.

Examples of Deceptive Practices

The FTC settled a prosecution in 2005 against CartManager International, a firm that runs familiar web shopping cart software to collect items of an order, obtain the purchaser's name and address, and determine shipping and payment details. This software runs as an application under other well-known retail merchants' web sites to handle order processing. Some of these other retailers had privacy
statements on their web sites saying, in effect, that they would not sell or distribute customers' data, but CartManager did sell the data it collected. The FTC held that the relationship to CartManager was invisible to users, and so the policy from the online merchants applied also to CartManager.

In another case, Antón [ANT04] analyzed the privacy policy posted on the web site of Jet Blue airlines and found it misleading. Jet Blue
stated that it would not disclose passenger data to third parties. It then released passenger data, "in response to a special request from the Department of Defense" to Torch Concepts, which in turn passed it to the Defense Department to use to test

passenger screening algorithms for airline security. The data in question involved credit card information: Clearly the only reason for Jet Blue to have collected those data from passengers was to process charges for airline tickets.

The analysis by Antón is interesting for two reasons: First, Jet Blue violated its own policy. Second, the Department of Defense may have circumvented the e-Government Act by acquiring from a private company data it would not have been able to collect as a government entity. The purpose for which the data were originally collected was ordinary business and accounting activities of Jet Blue. Using those same records to screen for terrorists was outside the scope of the original data collection.

Commercial sites have no standard of content comparable to the FTC recommendation from the e-Government Act. Some companies display solid and detailed privacy statements that they must obey. On the other hand, you may find no statement at all, which gives the company the greatest flexibility because it is impossible to lie when saying nothing. Cranor [CRA03] makes some recommendations for useful web privacy policies.

Non-U.S. Privacy Principles

In 1981, the Council of Europe (an international body of 46 European countries, founded in 1949) adopted Convention 108 for the protection of individuals with regard to the automatic processing of personal data, and in 1995, the European Union (E.U.) adopted Directive 95/46/EC on the processing of personal data. Directive 95/46/EC, often called the European Privacy Directive, requires that rights of privacy of individuals be maintained and that data about them be

- processed fairly and lawfully
- collected for specified, explicit and legitimate purposes and not further processed in a way incompatible with those purposes

(unless appropriate safeguards protect privacy)

- adequate, relevant, and not excessive in relation to the purposes for which they are collected and/or further processed
- accurate and, where necessary, kept up to date; every reasonable step must be taken to ensure that inaccurate or incomplete data having regard for the purposes for which they were collected or for which they are further processed, are erased or rectified
- kept in a form that permits identification of data subjects for no longer than is necessary for the purposes for which the data were collected or for which they are further processed

In addition, individuals have the right to access data collected about them, to correct inaccurate or incomplete data, and to have those corrections sent to those who have received the data. The report adds three more principles to the Fair Information Policies.

- Special protection for sensitive data. There should be greater restrictions on data collection and processing that involves "sensitive data." Under the E.U. data protection directive, information is sensitive if it involves "racial or ethnic origin, political opinions, religious beliefs, philosophical or ethical persuasion . . . [or] health or sexual life."
- Data transfer. This principle explicitly restricts authorized users of personal information from transferring that information to third parties without the permission of the data subject.
- Independent oversight. Entities that process personal data should not only be accountable but should also be subject to independent oversight. In the case of the government, this requires oversight by an office or department that is separate and

independent from the unit engaged in the data processing. Under the data protection directive, the independent overseer must have the authority to audit data processing systems, investigate complaints brought by individuals, and enforce sanctions for noncompliance.

Anonymity, Multiple Identities

One way to preserve privacy is to guard our identity. Not every context requires us to reveal our identity, so some people wear a form of electronic mask.

Anonymity

A person may want to do some things anonymously. For example, a rock star buying a beach house might want to avoid unwanted attention from neighbors, or someone posting to a dating list might want to view replies before making a date.

Mulligan [MUL99] lists several reasons people prefer anonymous activity on the web. Some people like the anonymity of the web because it reduces fears of discrimination. Fairness in housing, employment, and association are easier to ensure when the basis for potential discrimination is hidden. Also, people researching what they consider a private matter, such as a health issue or sexual orientation, are more likely to seek first information from what they consider an anonymous source, turning to a human when they have found out more about their situation.

Multiple IdentitiesLinked or Not

Most people already have multiple identities. To your bank you might be the holder of account 123456, to your motor vehicles bureau you might be the holder of driver's license number 234567, and to your credit card company you might be the holder of card 345678. For their purposes, these numbers are your identity; the fact that each may (or may not) be held in your name is irrelevant. The name does become important if it is used as a way to link these records. How many people share your name? Can (or should) it serve as a key value to link these separate databases? We ignore the complication of misspellings and multiple valid forms (with and without middle initials, with full middle name, with one of two middle names if you have them, and so forth).

Government and Privacy

The government gathers and stores data on citizens, residents, and visitors. Government facilitates and regulates commerce and other kinds of personal activities such as healthcare, employment, education, and banking. In those roles the government is both an enabler or regulator of privacy and a user of private data. Government use of private data should be controlled. In this section we consider some of the implications of government access to private data.

Authentication

Government plays a complex role in personal authentication. Many government agencies (such as the motor vehicles bureau) use identifiers to perform their work. Authentication documents (such as passports and insurance cards) often come from the government. The government may also regulate the businesses that use identification and authentication keys. And sometimes the government obtains data based on those keys from others (for example, the U.S. government planned to buy credit reports from private companies to help with screening airline passenger lists for terrorists). In these multiple roles, the government may misuse data and violate privacy rights.

Data Access Risks

Recognizing that there were risks in government access to personal data, the Secretary of Defense appointed a committee to investigate private data collection.

The Technology and Privacy Advisory Committee, chaired by Newton Minow, former chair of the Federal
Communications Commission, produced its report in 2004 [TAP04]. Although their charge had been to review privacy and data collection within the Department of Defense, they found it impossible to separate the DoD from the rest of government, so they made recommendations for both the Department of Defense and the federal government as a whole.

They recognized risks when the government started to acquire data from other parties:

- *data errors:* ranging from transcription errors to incorrect analysis
- *inaccurate linking:* two or more correct data items but incorrectly linked on a presumed common element
- difference of form and content: precision, accuracy, format, and semantic errors
- purposely wrong: collected from a source that intentionally gives incorrect data, such as a forged identity card or a false address given to mislead
- *false positive:* an incorrect or out-of-date conclusion that the government does not have data to verify or reject, for example, delinquency in paying state taxes
- *mission creep:* data acquired for one purpose leading to a broader use because the data will support that mission
- *poorly protected:* data of questionable integrity because of the way it has been managed and handled

These risks apply to all branches of government, and most of them apply to private collection and use of data.

Steps to Protect Against Privacy Loss

The committee recommended several steps the government can take to help safeguard private data.

- *Data minimization.* Obtain the least data necessary for the task. For example, if the goal is to study the spread of a disease, only the
condition, date, and vague location (city or county) may suffice; the name or contact information of the patient may be
unnecessary.
- *Data anonymization*. Where possible, replace identifying information with untraceable codes (such as a record number); but make
sure those codes cannot be linked to another database that reveals sensitive data.
- *Audit trail.* Record who has accessed data and when, both to help identify responsible parties in the event of a breach and to
document the extent of damage.
- *Security and controlled access.* Adequately protect and control access to sensitive data.
- *Training*. Ensure people accessing data understand what to protect and how to do so.
- *Quality.* Take into account the purpose for which data were collected, how they were stored, their age, and similar factors to
determine the usefulness of the data.
- *Restricted usage.* Different from controlling access, review all proposed uses of the data to determine if those uses are consistent with the purpose for which the data were collected and the manner in which they were handled (validated, stored, controlled).
- *Data left in place.* If possible, leave data in place with the original owner. This step helps guard against possible misuses of the data from expanded mission just because the data are available.

 Policy. Establish a clear policy for data privacy. Do not encourage violation of privacy policies.

These steps would help significantly to ensure protection of privacy.

Identity Theft

As the name implies, identity theft is taking another person's identity. Use of another person's credit card is fraud; taking out a new credit card in that person's name is identity theft. Identity theft has risen as a problem from a relatively rare issue in the 1970s. In 2005, the U.S.

Federal Trade Commission received over 250,000 complaints of identity theft [FTC06]. Most cases of identity theft become apparent in a month or two when fraudulent bills start coming in. By that time the thief has made a profit and has dropped this identity, moving on to a new victim.

Having relatively few unique keys facilitates identity theft: A thief who gets one key can use that to get a second, and those two to get a third. Each key gives access to more data and resources. Few companies or agencies are set up to ask truly discriminating authentication

questions (such as the grocery store at which you frequently shop or the city to which you recently bought an airplane ticket or third digit

on line four of your last tax return). Because there are few authentication keys, we are often asked to give the same key (such as mother's

maiden name) out to many people, some of whom might be part-time accomplices in identity theft.

10.3. Authentication and Privacy

What Authentication Means

We use the term authentication to mean three different things [KEN03]: We authenticate an individual, identity, or attribute. An individual is a unique person. Authenticating an individual is what we do when we allow a person to enter a controlled room: We want only that human being to be allowed to enter. An identity is a character string or similar descriptor, but it does not necessarily correspond to a single person, nor does each person have only one name. We authenticate an identity when we acknowledge that whoever (or whatever) is trying to log in as admin has presented an authenticator valid for that account. Similarly, authenticating an identity in a chat room as SuzyQ does not say anything about the person using that identifier: It might be a 16-year-old girl or a pair of middle-aged male police detectives, who at other times use the identity FrereJacques.

Finally, we authenticate an attribute if we verify that a person has that attribute. An attribute is a characteristic. Here's an example of

authenticating an attribute. Some places require one to be 21 or older in order to drink alcohol. A club's doorkeeper verifies a person's age and stamps the person's hand to show that the patron is over 21. Note that to decide, the doorkeeper may have looked at an identity card listing the person's birth date, so the doorkeeper knew the person's exact age to be 24 years, 6 months, 3 days, or the doorkeeper might be authorized to look at someone's face and decide if the person looks so far beyond 21 that there is no need to verify. The stamp authenticator signifies only that the person possesses the attribute of being 21 or over.

In computing applications we frequently authenticate individuals, identities, and attributes. Privacy issues arise when we confuse these different authentications and what they mean. For example, the U.S. social security number was never intended to be an identifier, but now it often serves as an identifier, an authenticator, a database key, or all of these. When one data value serves two or more uses, a person acquiring it for one purpose can use it for another.

Individual Authentication

There are relatively few ways of identifying an individual. When we are born, for most of us our birth is registered at a government records office, and we (probably our parents) receive a birth certificate. A few years later our parents enroll us in school, and they have to present the birth certificate, which then may lead to receiving a school identity card. We submit the birth certificate and a photo to get a passport or a national identity card. We receive many other authentication numbers and cards throughout life.

The whole process starts with a birth certificate issued to (the parents of) a baby, whose physical description (height, weight, even hair color) will change significantly in just months. Birth certificates may contain the baby's fingerprints, but matching a poorly taken fingerprint of a newborn baby to that of an adult is challenging at best. (For additional identity authentication problems, Fortunately, in most settings it is acceptable to settle for weak authentication for individuals: A friend who has known you since childhood, a schoolteacher, neighbors, and coworkers can support a claim of identity.

Identity Authentication

We all use many different identities. When you buy something with a credit card, you do so under the identity of the credit card holder. In some places you can pay road tolls with a radio frequency device in your car, so the sensor authenticates you as the holder of a particular toll device. You may have a meal plan that you can access by means of a card, so the cashier authenticates you as the owner of that card.

You check into a hotel and get a magnetic stripe card instead of a key, and the door to your room authenticates you as a valid resident for the next three nights. If you think about your day, you will probably find 10 to 20 different ways some identity of you has been authenticated.

From a privacy standpoint, there may or may not be ways to connect all these different identities. A credit card links to the name and address of the card payer, who may be you, your spouse, or anyone else willing to pay your expenses. Your auto toll device links to the name and perhaps address of whoever is paying the tolls: you, the car's owner, or an employer. When you make a telephone call, there is an
authentication to the account holder of the telephone, and so forth.

Data Mining

Here we consider how to maintain privacy in the context of data mining.

Private sector data mining is a lucrative and rapidly growing industry. The more data collected, the more opportunities for learning from various aggregations. Determining trends, market preferences, and characteristics may be good because they lead to an efficient and effective market. But people become sensitive if the private information becomes known without permission.

Government Data Mining

Especially troubling to some people is the prospect of government data mining. We believe we can stop excesses and intrusive behavior of private companies by the courts, unwanted publicity, or other forms of pressure. It is much more difficult to stop the government. In many examples governments or rulers have taken retribution against citizens deemed to be enemies, and some of those examples come from presumably responsible democracies. Much government data collection and analysis occurs without publicity; some programs are just not announced and others are intentionally kept secret. Thus, citizens have a fear of what unchecked

government can do. Citizens' fears are increased because data mining is not perfect or exact, and as many people know, correcting erroneous data held by the government is next to impossible.

Privacy-Preserving Data Mining

Because data mining does threaten privacy, researchers have looked into ways to protect privacy during data mining operations. A naïve and ineffective approach is trying to remove all identifying information from databases being mined. Sometimes, however, the identifying
information is precisely the goal of data mining. More importantly, as the preceding example from Sweeney showed, identification may be possible even when the overt identifying information is removed from a database.

Data mining has two approachescorrelation and aggregation. We examine techniques to preserve privacy with each of those approaches.

Privacy for Correlation

Correlation involves joining databases on common fields. As in a previous example, the facts that someone is named Erin and someone has diabetes have privacy significance only if the link between Erin and diabetes exists. Privacy preservation for correlation attempts to control that linkage.

Vaidya and Clifton [VAI04] discuss data perturbation as a way to prevent privacy-endangering correlation. As a simplistic example, assume two databases contain only three records, as shown in Table . The ID field linking these databases makes it easy to see that Erin has diabetes.

One form of data perturbation involves swapping data fields to prevent linking of records. Swapping the values Erin and Geoff (but not the ID values) breaks the linkage of Erin to diabetes. Other properties of the databases are preserved: Three patients have actual names and three conditions accurately describe the patients. Swapping all data values can prevent useful analysis, but limited swapping balances privacy and accuracy. With our example of swapping just Erin and Geoff, you still know that one of the participants has diabetes, but you cannot know if Geoff (who now has ID=1) has been swapped or not. Because you cannot know if a value has been swapped, you cannot assume any such correlation you derive is true.

Our example of three data points is, of course, too small for a realistic data mining application, but we constructed it just to show how value swapping would be done. Consider a more realistic example on larger databases. Instead of names we might have addresses, and the
purpose of the data mining would be to determine if there is a correlation between a neighborhood and an illness, such as measles.

Swapping all addresses would defeat the ability to draw any correct conclusions regarding neighborhood. Swapping a small but significant
number of addresses would introduce uncertainty to preserve privacy. Some measles patients might be swapped out of the high-incidence neighborhoods, but other measles patients would also be swapped in. If the neighborhood has a higher incidence than the general population, random swapping would cause more losses than gains, thereby reducing the strength of the correlation. After value swapping an already weak correlation might become so weak as to be statistically insignificant. But a previously strong correlation would still be significant, just not as strong.

Thus value-swapping is a technique that can help to achieve some degrees of privacy and accuracy under data mining.

Privacy for Aggregation

Aggregation need not directly threaten privacy, an aggregate (such as sum, median, or count) often depends on so many data items that the sensitivity of any single contributing item is hidden. Government statistics show this well: Census data, labor statistics, and school results show trends and patterns for groups (such as a neighborhood or school district) but do not violate the privacy of any single person.

Privacy on the Web

The Internet is perhaps the greatest threat to privacy. As an advantage of the Internet, which is also a disadvantage, is anonymity. A user can visit web sites, send messages, and interact with applications without revealing an identity. At least that is what we would like to think. Unfortunately, because of things like cookies, ad-ware, spybots, and malicious code, the anonymity is superficial and largely one-sided. Sophisticated web applications can know a lot about a user, but the user knows relatively little about the application.

The topic is clearly of great interest: a recent Google search returned 7 billion hits for the phrase "web privacy."

In this section we investigate some of the ways a user's privacy is lost on the Internet.

Understanding the Online Environment

The Internet is like a nightmare of a big, unregulated bazaar. Every word you speak can be heard by many others. And the merchants' tents are not what they seem: the spice merchant actually runs a gambling den, and the kind woman selling scarves is really three pirate brothers and a tiger. You reach into your pocket for money only to find that your wallet has been emptied. Then the police tell you that they would love to help but, sadly, no laws apply. Caveat emptor in excelsis.

We have previously described the anonymity of the web. It is difficult for two unrelated parties to authenticate each other. Internet authentication most often confirms the user's identity, not the server's, so the user is unsure that the web site is legitimate. This uncertainty makes it difficult to give informed consent to release of private data: How can consent be informed if you don't know to whom you are giving consent?

Payments on the Web

Customers of online merchants have to be able to pay for purchases. Basically, there are two approaches: the customer presents a credit card to the merchant or the customer arranges payment through an online payment system such as PayPal.

Credit Card Payments

With a credit card, the user enters the credit card number, a special number printed on the card (presumably to demonstrate that the user actually possesses the card), the expiration date of the card (to ensure that the card is currently active), and the billing address of the credit card (presumably to protect against theft of credit card). These protections are all on the side of the merchant: They demonstrate that the merchant made a best effort to determine that the credit card use was legitimate. There is no protection to the customer that the merchant will secure these data. Once the customer has given this information to one merchant, that same information is all that would be required for another merchant to accept a sale charged to the same card.

Furthermore, these pieces of information provide numerous static keys by which to correlate databases. As we have seen, names can be difficult to work with because of the risk of misspelling, variation in presentation, truncation, and the like. Credit

card numbers make excellent keys because they can be presented in only one way and there is even a trivial check digit to ensure that the card number is a valid sequence.

Because of problems with stolen credit card numbers, there has been some consideration of disposable credit cards: cards you could use for one transaction or for a fixed short period of time. That way, if a card number is stolen or intercepted, it could not be reused. Furthermore, having multiple card numbers limits the ability to use a credit card number as a key to compromise privacy.

Payment Schemes

The other way to make web payments is with an online payment scheme, such as PayPal (which is now a subsidiary of the eBay auction site). You pay PayPal a sum of money and you receive an account number and a PIN. You can then log in to the PayPal central site, give an e-mail address and amount to be paid, and PayPal transfers that amount. Because it is not regulated under the same banking laws as credit cards, PayPal offers less consumer protection than does a credit card. However, the privacy advantage is that the user's credit card or financial details are known only to PayPal, thus reducing the risk of their being stolen. Similar schemes use cell phones.

Site and Portal Registrations

Registering to use a site is now common. Often the registration is free; you just choose a user ID and password. Newspapers and web portals (such as Yahoo or MSN) are especially fond of this technique. The explanation they give sounds soothing: They will enhance your browsing experience (whatever that means) and be able to offer content to people throughout the world. In reality, the sites want to obtain customer demographics that they can then sell to marketers or show to advertisers to warrant their advertising.

People have trouble remembering numerous IDs so they tend to default to simple ones, often variations on their names. And because people have trouble remembering IDs, the sites are making it easier: Many now ask you to use your e-mail address as your ID. The problem with using the same ID at many sites is that it now becomes a database key on which previously separate databases from different sites can be merged. Even worse, because the ID or e-mail address is often closely related to the individual's real name, this link also connects a person's identity with the other collected data. So now, a data aggregator can infer that V. Putin browsed the New York Times looking for articles on vodka and longevity and then bought 200 shares of stock in a Russian distillery.

You can, of course, try to remember many different IDs. Or you can choose a disposable persona, register for a free e-mail account under a name like xxxyyy, and never use the account for anything except these mandatory free registrations.

Whose Page Is This?

The reason for registrations has little to do with the newspaper or the portal; it has to do with advertisers, the people who pay so the web content can be provided. The web offers much more detailed tracking possibilities than other media. If you see a billboard for a candy bar in the morning and that same advertisement remains in your mind until lunch time and you buy that same candy bar at lunch, the advertiser is very happy: The advertising money has paid off. But the advertiser has no way to know whether you saw an ad (and if so which one).

There are some coarse measures: After an ad campaign if sales go up, the campaign probably had some effect. But advertisers would really like a closer cause-and-effect relationship. Then the web arrived.

Third-Party Ads

You log in to Yahoo Sports and you might see advertisements for mortgages, banking, auto loans, maybe some sports magazines or a cable television offer, and a fast food chain. You click one of the links and you either go directly to a "buy here now" form or you get to print a special coupon worth something on your purchase in person. Web advertising is much more connected to the purchaser: You see the ad, you click on it, and they know the ad did its job by attracting your attention. (With a highway billboard they never know if you watch it or
traffic.) When you click through and buy, the ad has really paid off. When you ,mclick through and print a coupon that you later present, a tracking number on the coupon lets them connect to advertising on a particular web site. From the advertiser's point of view, the immediate feedback is great.

Contests and Offers

We cannot resist anything free. We will sign up for a chance to win a large prize, even if we have only a minuscule chance of succeeding.

Advertisers know that. So contests and special offers are a good chance to get people to divulge private details. Another thing advertisers know is that people are enthusiastic at the moment but enthusiasm and attention wane quickly.

A typical promotion offers you a free month of a service. You just sign up, give a credit card number, which won't be charged until next ,month, and you get a month's use of the service for free. As soon as you sign up, the credit card number and your name become keys by which to link other data. You came via a web access, so there may be a link history from the forwarding site.

Precautions for Web Surfing

In this section we discuss cookies and web bugs, two technologies that are frequently used to monitor a user's activities without the user's knowledge.

Cookies

Cookies are files of data set by a web site. They are really a cheap way to transfer a storage need from a web site to a user.

A portal such as Yahoo allows a user to customize the look of the web page. Sadie wants the news headlines, the weather, and her e-mail, with a bright background; Norman wants stock market results, news about current movies playing in his area, and interesting things that happened on this day in history, displayed on a gentle pastel background. Yahoo could keep all this preference information in its database so that it could easily customize pages it sends to these two users. But Netscape realized that the burden could be shifted to the user. The web protocol is basically stateless, meaning that the browser displays whatever it is given, regardless of anything that has happened previously.

A cookie is a text file stored on the user's computer and passed by the user's browser to the web site when the user goes to that site. Thus, preferences for Sadie or Norman are stored on their own computers and passed back to Yahoo to help Yahoo form and deliver a web page according to Sadie's or Norman's preferences.

A cookie contains six fields: name, value, expiration date, path on the server to which it is to be delivered, domain of the server to which it is to be delivered, and whether a secure connection (SSL) is required in order for the cookie to be delivered. A site can set as many cookies as it wants and can store any value (up to 4,096 bytes) it wants. Some sites use cookies to avoid a customer's having to log in on each visit to a site; these cookies contain the user's ID and password. A cookie could contain a credit card number, the customer name and shipping address, the date of the last visit to the site, the number of items purchased or the dollar
volume of purchases.

Obviously for sensitive information, such as credit card number or even name and address, the site should encrypt or otherwise protect the data in the cookie. It is up to the site what kind of protection it wants to apply to its cookies. The user never knows if or how data are protected.

The path and domain fields protect against one site's being able to access another's cookies. Almost. As we show in the next section, one company can cooperate with another to share the data in its cookies.

Third-Party Cookies

When you visit a site, its server asks your browser to save a cookie. When you visit that site again your browser passes that cookie back.

The general flow is from a server to your browser and later back to the place from which the cookie came. A web page can also contain cookies for another organization. Because these cookies are for organizations other than the web page's owner, they are called **third-party cookies**.

Here are some examples of things a third-party cookie can do.

- Count the number of times this browser has viewed a particular web page.
- Track the pages a visitor views, within a site or across different sites.
- Count the number of times a particular ad has appeared.
- Match visits to a site with displays of an ad for that site.
- Match a purchase to an ad a person viewed before making the purchase.
- Record and report search strings from a search engine.

Web Bugs: Is There an Exterminator?

The preceding discussion of DoubleClick had a passing reference to an invisible image. Such an image is called a **clear GIF**, **1 x 1 GIF**, or **web bug**. It is an image file 1 pixel by 1 pixel, so it is far too small to detect by normal sight. To the web browser, an image is an image, regardless of size; the browser will ask for a file from the given address. The distinction between a cookie and a bug is enormous. A cookie is a tracking device, transferred between the user's machine and the server. A web bug is an invisible image that invites or invokes a process. That process can come from any location. A typical advertising web page might have 20 web bugs, inviting 20 other sites to drop images, code, or other bugs onto the user's machine. All this occurs without the user's direct knowledge or certainly control.

Unfortunately, extermination is not so simple as prohibiting images smaller than the eye can see, because many web pages use such images innocently to help align content. Or some specialized visual applications may actually use collections of minute images for a valid purpose. The answer is not to restrict the image but to restrict the collection and dissemination of data.

Spyware

Cookies are tracking objects, little notes that show where a person has been or what a person has done. The only information they can gather is what you give them by entering data or selecting an object on a web page. As we see in the next section, spyware is far more powerfuland potentially dangerous.

Cookies are passive files and, as we have seen, the data they can capture is limited. They cannot, for example, read a computer's registry, peruse an e-mail outbox, or capture the file directory structure. Spyware is active code that can do all these things that cookies cannot, generally anything a program can do because that is what they are: programs.

Spyware is code designed to spy on a user, collecting data (including anything the user types). In this section we describe different types of
spyware.

Keystroke Loggers and Spyware

We have previously referred to keystroke loggers, programs that reside in a computer and record every key pressed. Sophisticated loggers discriminate, recording only web sites visited or, even more serious, only the keystrokes entered at a particular web site (for example, the login ID and password to a banking site.)

A **keystroke logger** is the computer equivalent of a telephone wiretap. It is a program that records every key typed. As you can well imagine, keystroke loggers can seriously compromise privacy by obtaining passwords, bank account numbers, contact names, and web msearch arguments.

Spyware is the more general term that includes keystroke loggers and also programs that surreptitiously record user activity and system data, although not necessarily at the level of each individual keystroke. A form of spyware, known as adware (to be described shortly) records these data and transmits them to an analysis center to present ads that will be interesting to the user. The objectives of general ,spyware can extend to identity theft and other criminal activity.

In addition to the privacy impact, keystroke loggers and spyware sometimes adversely affect a computing system. Not always written or tested carefully, spyware can interfere with other legitimate programs. Also, machines infected with spyware often have several different pieces of spyware, which can conflict with each other, causing a serious impact on performance.

Another common characteristic of many kinds of spyware is the difficulty of removing it. For one spyware product, Altnet, removal involves at least twelve steps, including locating files in numerous system folders .

Hijackers

Another category of spyware is software that hijacks a program installed for a different purpose. For example, file-sharing software is typically used to share copies of music or movie files. Services such as KaZaa and Morpheus allow users to offer part of their stored files to other users

Adware

Adware displays selected ads in pop-up windows or in the main browser window. The ads are selected according to the user's characteristics, which the browser or an added program gathers by monitoring the user's computing use and reporting the information to a home base.

Adware is usually installed as part of another piece of software without notice. Buried in the lengthy user's license of the other software is reference to "software x and its extension," so the user arguably gives permission for the installation of the adware. File-sharing software is a common target of adware, but so too are download managers that retrieve large files in several streams at once for faster downloads. And products purporting to be security tools, such as antivirus agents, have been known to harbor adware.

Writers of adware software are paid to get their clients' ads in front of users, which they do with pop-up windows, ads that cover a legitimate ad, or ads that occupy the entire screen surface. More subtly, adware can reorder search engine results so that clients' products get higher placement or replace others' products entirely.

E-Mail Security

E-mail is exposed as it travels through the web. Furthermore, the privacy of an e-mail message can be compromised on the sender's or
receiver's side, without warning.

Consider the differences between e-mail and regular letters. Regular mail is handled by a postal system that by law is forbidden to look inside letters. A letter is sealed inside an opaque envelope, making it almost impossible for an outsider to

see the contents. The physical envelope is tamper-evident, meaning it shows if someone opens it. A sender can drop a letter in any mailbox, making the sending of a letter anonymous. For these reasons, we have a high expectation of privacy with regular mail.

In this section we look at the reality of privacy for e-mail.

Where Does E-Mail Go, and Who Can Access It?

E-mail is conceptually a point-to-point communication. If Janet sends e-mail to Scott, Janet's computer establishes a virtual connection with Scott, the computers synchronize, and the message is transferred by SMTP (simple mail transfer protocol). However, Scott may not be online at the moment Janet wants to send her message, so the message to Scott is stored for him on a server (called a POP or post office protocol server). The next time Scott is online, he downloads that message from the server. In the point-to-point communication, Janet's message is private; in the server version, it is potentially exposed while sitting on the server.

Janet may be part of a large organization (such as a company or university), so she may not have a direct outbound connection herself; instead, her mail is routed through a server, too, where the message's privacy is in jeopardy. A further complication is aliases and forwarding agents that add more midpoints to this description. Also, Internet routing can make many hops out of a conceptual point-topoint model.

What started as a simple case can easily have at least five parties: (a) Janet and her computer, (b) Janet's organization's SMTP server, (c)

Janet's organization's ISP, (d) Scott's POP server, and (e) Scott and his computer. For now, we are most interested in the three middle parties: (b), (c), and (d). Any of them can log the fact that it was sent or can even keep a copy of the message.

Interception of E-mail

E-mail is subject to the same interception risks as other web traffic: While in transit on the Internet, e-mail is open for any interceptor to read.

we described techniques for encrypting e-mail. In particular, S/MIME and PGP are two widely used e-mail protection programs. S/MIME and PGP are available for popular mail handlers such as Outlook, Outlook Express, Eudora, Apple Mail, Netscape Communicator, and others. These products protect e-mail from the client's workstation through mail agents, across the Internet, and to the

recipient's workstation. That protection is considered end-to-end, meaning from the sender to the recipient. Encrypted e-mail protection is subject to the strength of the encryption and the security of the encryption protocol.

A virtual private network, also described ,it can protect data on the connection between a client's workstation and some edge point, usually a router or firewall, at the organization to which the client belongs. For a corporate or government employee or a university student, communication is protected just up to the edge of the corporate, government, or university network. Thus, with a virtual private network, e-mail is protected only from the sender to the sender's office, not even up to the sender's mail agent, and certainly not to the recipient.

Some organizations routinely copy all e-mail sent from their computers. Purposes for these copies include using the e-mail as evidence in legal affairs and monitoring the e-mail for inappropriate content.

Monitoring E-Mail

Companies and government agencies can legitimately monitor their employees' e-mail use. Schools and libraries can monitor the computer use of patrons. Network administrators and ISPs can monitor traffic for normal business purposes, such as to measure traffic patterns or to detect spam. Organizations must advise users of

this monitoring, but the notice can be a small notice in a personnel handbook or in the fine print of a service contract. Organizations can use the monitoring data for any legal purpose, for example, to investigate leaks, to manage resources, or to track user behavior.

Network users should have no expectation of privacy in their e-mail or general computer use.

Anonymous E-mail and Remailers

We have described anonymity in other settings; there are reasons for anonymous e-mail, as well.

As with telephone calls, employees sending tips or complaining to management may want to do so anonymously. For example, consumers may want to contact commercial establishmentsto register a complaint, inquire about products, or request informationwithout getting on a mailing list or becoming a target for spam. Or people beginning a personal relationship may want to pass along some information without giving away their identities. These are some of the reasons people want to be able to send anonymous e-mail.

Free e-mail addresses are readily available from Yahoo, Microsoft Hotmail, and many other places. People can treat these addresses as disposable: Obtain one, use it for a while, and discard it (by ceasing to use it).

Simple Remailers

Another solution is a remailer. A **remailer** is a trusted third party to whom you send an e-mail message and indicate to whom you want it sent. The remailer strips off the sender's name and address, assigns an anonymous pseudonym as the sender, and forwards the message to the designated recipient. The third party keeps a record of the correspondence between pseudonyms and real names and addresses. If the recipient replies, the remailer removes the recipient's name and address, applies a different anonymous pseudonym, and forwards the message to the original sender. Such a remailer knows both sender and receiver, so it provides pseudonymity, not anonymity.

Mixmaster Remailers

A more complicated design is needed to overcome the problem that the remailer knows who are the real sender and receiver. This approach is similar to the concept of onion routing . The basic tool is a set of cooperating hosts that agree to forward mail. Each host publishes its own public encryption key.

The sender creates a message and selects several of the cooperating hosts. The sender designates the ultimate recipient (call it node n) and places a destination note with the content. The sender then chooses one of the cooperating hosts (call it node n-1), encrypts the package with the public key of node (n-1) and places a destination note showing node (n) with the encrypted package. The sender chooses another node (n-2), encrypts, and adds a destination note for (n-1). The sender thus builds a multilayered package, with the message inside; each layer adds another layer of encryption and another destination.

Each remailer node knows only from where it received the package and to whom to send it next. Only the first remailer knows the true

recipient, and only the last remailer knows the final recipient. Therefore, no remailer can compromise the relationship between sender and receiver. Although this strategy is sound, the overhead involved indicates that this approach should be used only when anonymity is very important.

Spoofing and Spamming

E-mail has very little authenticity protection. Nothing in the SMTP protocol checks to verify that the listed sender (the From: address) is accurate or even legitimate.

Spoofing the source address of an e-mail message is not difficult. This limitation facilitates the sending of spam because it is impossible to trace the real sender of a spam message. Sometimes the apparent sender will be someone who knows the recipient or someone on a common mailing list with the recipient. Spoofing such an apparent sender is intended to lend credibility to the spam message.

Phishing is a form of spam in which the sender attempts to convince the sender to reveal personal data, such as banking details. The sender enhances the credibility of a phishing message by spoofing a convincing source address, or using a deceptive domain name

These kinds of e-mail messages entice gullible users to reveal sensitive personal data. Because of limited regulation of the Internet, very little can be done to control these threats. User awareness is the best defense.

Impacts on Emerging Technologies

.

RFID

Radio frequency identification or **RFID** is a technology that uses small, low-power wireless radio transmitters called RFID **tags**. The devices can be as small as a grain of sand and they cost just pennies apiece. Tags are tuned to a particular frequency and each has a unique ID number. When a tag receives its signal, it sends its ID number signal in response. Many tags have no power supply of their own and receive their power to send a signal from the very act of receiving a signal. Thus, these devices are passive until they receive a signal from an interrogating reader.

The distance at which they can receive and broadcast a receivable signal varies from roughly five centimeters at the least powerful end to several meters at the most powerful end. Some transmitters have their own power supply (battery) and can transmit over an even greater distance. Probably as receivers get better, the reception distance will increase.

Current uses of RFID tags include

- toll plaza payments
- transit system fare cards
- stock or inventory labels
- passports and identity cards

Two applications of RFID tags are of special interest from a privacy standpoint, as we show in the next sections.

Consumer Products

Assume you have bought a new shirt. If the manufacturer has embedded an RFID tag in the shirt, the tag will assist the merchant in processing your sale, just as barcodes do today. But barcodes on merchandise identify only a manufacturer's product, such as an L.L Bean green plaid flannel shirt, size M. The RFID tag can identify not only the product but also the batch and shipment; that is, the tag's value designates a specific shirt. The unique ID in the shirt helps the merchant keep track of stock, knowing that this shirt was from a shipment that has been on the sales display for 90 days. The tag also lets the manufacturer determine precisely when and where it was produced, which could be important if you returned the shirt because of a defect.

As you walk down the street, your shirt will respond to any receiver within range that broadcasts its signal. With low-power tags using today's technology, you would have to pass quite close to the receiver for it to obtain your signal, a few centimeters at most. Some scientists think this reception will be extended in the future, and others think the technology exists today for high-power readers to pick

up the signal a meter away. If the distance is a few centimeters, you would almost have to brush up against the receiver in order for it to track the tag in your shirt; at a meter, someone could have a reader at the edge of the sidewalk as you walk past.
Your shirt, shoes, pen, wallet, credit card, mobile phone, media player, and candy bar wrapper might each have an RFID tag. Any one of these would allow surreptitious tracking; the others provide redundancy. Tracking scenarios once found only in science fiction are now close to reality.
One privacy interest is the accumulation of readings as you go about your business. If a city were fitted with readers on every street corner,
it would be possible to assemble a complete profile of your meanderings; timestamps would show when you stopped for a while between two receivers. Thus, it is imaginable and probably feasible to develop a system that could track all your movements.
The other privacy concern is what these tags say about you: One tag from an employee ID might reveal for whom you work, another from a medicine bottle might disclose a medical condition, and still another from an expensive key fob might suggest your finances. Currently you mcan conceal objects like your employee ID in your pocket; with RFID technology you may have to be more careful to block invisible radio signals.

RFID Tags for Individuals

Tagging a shirt is a matter of chance. If you buy the right kind of shirt you will have a tag that lets you be monitored. But if you buy an untagged shirt, or find and cut out the tag, or disable the tag, or decide not to wear a shirt, you cannot be tracked.
Some people choose to be identifiable, regardless of what they wear. Some people with an unusual medical condition have already had an RFID tag permanently implanted in their arm. This way, even if a patient is brought unconscious to a hospital, the doctors can scan for a tag, receive the person's unique number, and look up the person's medical record by that number. A similar approach is being used to permit animals to cross quarantine borders or to uniquely identify animals such as valuable racehorses.
In these examples, individuals voluntarily allow the tags to be implanted. But remember that once the tags are implanted, they will respond to any appropriate receiver, so our example of walking down the street still holds.
RFID advocates hasten to point out that the technology does not currently permit reading the simplest tags at a distance and that receivers are so expensive that it would be prohibitive to build a network capable of tracking someone's every movement. As we point out in cryptography and reiterate in software, you should not base your security just on what is technically possible or economiclly feasible today.

Security and Privacy Issues

We have already described two of RFID's major privacy issues: the ability to track individuals wherever they go and the ability to discern sensitive data about people. The related issue is one of correctness. The reading sensor may malfunction or the software processing IDs may fail; both cases lead to mistaken identity. How do you challenge that you were not someplace when the receiver shows you were?
Another possible failure is forgery of an RFID tag. Here again the sensor would pick up a reading of a tag associated with you. The only
way you could prove you were not near the sensor is to have an alibi, supporting where you actually were.

Juels [JUE05] presents several privacy-restoring approaches to RFID use. Among the ideas he proposes are blasting (disabling a tag), blocking (shielding a tag to block its access by a reader), reprogramming (so a tag emits a different number), and encrypting (so the output,is selectively available).
RFID technology is still very young, but its use is growing rapidly. As with similarly sensitive technologies, protecting privacy will be easier before the uses proliferate.

Electronic Voting

Voting is another area in which privacy is important. We want votes to be private, but at the same time we want a way to demonstrate that all collected votes are authentic. With careful control of paper ballots, we can largely satisfy both those requirements, but the efficiency of such systems is poor. We would like to use computerized voting systems to improve efficiency without sacrificing privacy or accuracy.
In this section we consider the privacy aspects of computerized voting.

Computer Voting

Citizens want to vote anonymously. Although anonymity is easy to achieve with paper ballots (ignoring the possibility of fingerprint tracing or secretly marked ballots) and fairly easy to achieve with machines (assuming usage protocols preclude associating the order in which people voted with a voting log from the machine), it is more difficult with computers. Properties essential to a fair election were enumerated by Shamos [SHA93].

- Each voter's choices must be kept secret.
- Each voter may vote only once and only for allowed offices.
- The voting system must be tamperproof, and the election officials must be prevented from allowing it to be tampered with.
- All votes must be reported accurately.
- The voting system must be available for use throughout the election period.
- An audit trail must be kept to detect irregularities in voting, but without disclosing how any individual voted.

These conditions are challenging in ordinary paper- and machine-based elections; they are even harder to meet in computer-based elections. Privacy of a vote is essential; in some repressive countries, voting for the wrong candidate can be fatal.

Privacy and the Process

Counting ballots is only one step in the election process; building and maintaining the list of eligible voters, recording who has voted (and keeping one person from voting twice), supporting absentee ballots, assisting voters at the wrong polling place, and transmitting election results to election headquarters are other important steps. Each of these has obvious privacy implications. For example, in some political cultures, it may be desirable to maintain privacy of who has voted (to prevent retaliation against people who did not vote for a powerful candidate). Similarly, as we know from other security studies, it is important to protect the privacy of votes in transmission to election
headquarters.
The Computer Science and Telecommunications Board of the National Academy of Science [NRC05] studied electronic voting. Its purpose was to raise questions to ensure they are considered in the debate about electronic voting. The privacy questions they asked concerned individual privacy in voter registration, the privacy of individual voters, and public confidence in the process.

Rubin [RUB02], Schneier [SCH04b], and Bennet [BEN04], among others, have studied electronic voting. Rubin raises the question of Internet voting, which has an obvious benefit of easy access for a segment of the population (and a corresponding weakness of more difficult access for people who do not have Internet access or who are not comfortable with computing technology). But given the very weak privacy protections we have already seen for the Internet, the privacy aspects of such a proposal require a careful look.

VoIP and Skype

Privacy aspects of traditional telephony were fairly well understood: Telephone companies were regulated monopolies that needed to preserve the confidentiality of their clients' communications. Exceptions occur under statutorially defined circumstances for law
enforcement purposes and in emergencies. Furthermore, the technology was relatively resistant to eavesdropping, with the greatest exposure at the endpoints.
Cellular telephony and Internet-based phone service have significantly changed that situation.

Voice over IP (**VoIP**) is a protocol for transmission of voice-grade telephone traffic over the Internet. The major VoIP carrier is Skype. (VoIP rhymes with "boy" plus P, and Skype rhymes with "hype.") You use a telephone handset or microphone and speaker connected to your computer. To call from London to
Rio, for example, you would invoke the VoIP application, giving it the telephone number in Rio. A local office in Rio would call the number in Rio and patch that call to its Internet servers. (The process is even easier if both endpoints use VoIP.)
The advantage of VoIP is cost: For people who already have a fixed-price broadband Internet connection, adding VoIP need only cover the costs of the local connection on the remote end and a fee for software. But as we have seen in other Internet applications, privacy is sacrificed. Even if the voice traffic is solidly encrypted, the source and destination of the phone call will be somewhat exposed through packet headers.