Chapter

8

Information Security

Change font sizeMain content

Chapter Contents

Chapter Introduction

r studying this chapter, you will be able to:

Explain the difference between authentication and authorization
Explain the use of a hash function to encrypt passwords on a computer system
Describe cyber-attacks including viruses, worms, Trojan horses, DoS attacks, and phishing, and explain how they differ from each other
Describe ways to increase the security of information on your computer and online

Encrypt and decrypt messages using simple Caesar ciphers and matrix-based block ciphers
Describe the overall process used by symmetric encryption algorithms such as DES
Describe the overall process used by RSA public-key encryption
Explain why web transmission protocols such as SSL and TLS use multiple forms of encryption to secure data transfer over the web

Explain the importance of computer security for networked embedded systems

8.1 Introduction

Information security means protecting data from those who do not have permission to access it. Information security could cover information locked in your filing cabinet, stuffed somewhere in your purse or wallet, or lying around on your desk. But today it usually means electronic information, such as data stored on your computer's hard disk, in the cloud, on your flash drive, or on your smartphone, or data transmitted across wired or wireless network connections.

In the early days of computing, large mainframe computers were considered showpieces and displayed prominently in a company for everyone to see. When management realized the folly of this situation, the mainframes were moved to secure rooms, frequently on secure floors or even in separate buildings. Only authorized persons had access. Now that there is a machine on virtually every desktop, a laptop or tablet in many briefcases, and a cell phone in every pocket, that kind of physical security is impossible to attain, but you can take some obvious precautionary steps: Don't leave your laptop or tablet lying around; lock or secure your workstation when you leave the room; don't share your password with anyone; and don't lose your cell phone!

However, the danger of someone swiping your laptop or smartphone pales in comparison with the security risks created by the Internet and the web. As our virtual environment expands, we celebrate the ability to reach out from our desktop, laptop, or handheld device to the rest of the world, receiving information, entertainment, and communications from thousands of sources. But we may be less aware of the potential for thousands of sources around the world to reach into our machines and our lives to do us harm or steal our information.

Security can be breached at many different points in the "virtual machine" we have presented in the last few chapters. Flaws in systems programs can be exploited, operating system protections can be circumvented, and computer networks (wired or wireless) present all kinds of opportunities for viewing, manipulating, or destroying data. Consequences can range from simple annoyance to serious identity theft to major economic losses or threats to national security. Because there are so many ways in which security can be compromised, and the consequences can be so serious, information security has become an extremely important topic in computer science research and practice.

8.2 Threats and Defenses

You no doubt are aware of the possible threats to the security of your personal property, such as your car or the contents of your home or apartment. That's why you probably have auto and home or renter's insurance. Aside from fire, flood, or other accidents, someone can steal your property, causing financial harm and emotional distress. Your best defenses against this kind of theft are to employ locks and possibly an alarm system. The alarm system only comes into play in an active manner when security has already been breached, but the mere fact that a property has an alarm system can be a deterrent to break-ins. Although it's true that an experienced thief can quickly pick a lock, it's certainly easier to break into an unlocked house. Be it thieves or computer hackers, criminals tend to attack the most vulnerable and poorly guarded sites.

This section discusses the threat of individual computers being accessed by the wrong people and the threats to which a computer is exposed through network connections; in addition, it describes various defenses against these threats.

Main content

Chapter Contents

8.2.1 Authentication and Authorization

The first line of defense against the illicit use of, or threats to, computer resources and sensitive information is a strong authentication and authorization process.

Authentication. You want to start up your computer, access your online bank account, or pay a bill online. What's the first thing you must do in all these cases? Generally, it is to log on to your machine or to the appropriate webpage by giving your user ID and password. The authentication process to log on to your computer is managed by your machine's operating system, as you learned in Chapter 6. On the webpage, it is managed by the operating system and perhaps special security software of the web server computer. **Authentication** is the process of verifying that you really are the person who has the right to access a particular device, whether it is your local machine or the remote server. The operating system maintains a file of user IDs and corresponding passwords. When a user attempts to log on to the machine, the operating system reads the user ID and checks that the password matches the password for that user in the password file. Hackers breaking into a computer system look for a file of user IDs and passwords as the "Open, Sesame" for all locked doors.

If the user ID/password list were in the form of a simple two-column list like this:

Tijara	popsicle
Murphy	badboy2
Jaylynn	mom
…	

then the entire system would be compromised if this password file were stolen or copied. To prevent this, the operating system **encrypts** the passwords—converts them into a representation that cannot be understood without the appropriate decryption algorithm. The encryption process that is used is called a **hash function**; this form of

encryption is easy to apply but hard to undo. The hash function takes the password the user originally chooses, chops it up, and stirs it around according to a given formula. As a very simple example, suppose the hash function process is the following:

The Metamorphosis of Hacking

The earliest use of the word *hack* appeared in English around 1200 and meant "to cut roughly, cut with chopping blows," and the term *hacker* meant "one who hacks." At MIT in 1955, the term was used at a meeting of the Tech Model Railroad Club to request that "anyone working or hacking on the electrical system turn the power off to avoid fuse blowing." From there the term *hacker* evolved to mean a "computer nerd" who seemed to know all the incomprehensible details about how computers worked and could fix anything that might go wrong. Over time, however, this term took on a much more negative connotation. Today, a **hacker** is a person who knows how to (and does) gain unauthorized access to a computer with malicious intent. Hacking is illegal and punishable under the law by fines or imprisonment, but as computers become ever more important in our society, hacking has become rampant.

Hacking is no longer the province solely of individuals. Just as there are organized auto theft rings that operate on a national or even international scale, there are organized hacking groups. Computer hacking on a large scale costs governments and businesses billions of dollars per year in terms of lost business, stolen information, and, more intangibly, loss of trust. Computer attacks from one country on the computing resources of another country with intent to damage or destroy computer systems or steal sensitive information are no longer mere "hacking" but cross over into the realm of **cyberwarfare**–politically motivated information theft or hardware damage for the purpose of sabotage, espionage, or even influence in democratic elections.

1. Take each letter in the password and replace it with a number representing its place in the alphabet (a → 1, b → 2, etc.). Leave each digit in the password alone. For example, Murphy's password, "badboy2", would become

   2 1 4 2 15 25 2
2. Add up these digits to get a single integer. In this example,

3. Divide the integer from Step 2 by 7 and determine the remainder. In this example, dividing 51 by 7 gives a remainder of 2 (51 equals 7 × 7 with 2 left over).
4. Add 1 to the remainder from Step 3, then multiply this value by 9. In this example, the result equals .
5. Reverse the digits in the integer from Step 4 and then replace each digit with the corresponding letter of the alphabet. The result for this example is 72, which becomes gb.
   The encrypted password file would contain an entry of

Murphy gb

and the original password "badboy2" has been discarded.

Now when Tom Murphy attempts to log on, he enters "Murphy" as his user ID and types in his password "badboy2". The operating system applies this exact same hash function to the password just entered and compares the encrypted result to the encrypted value "gb" stored in the password file for that user ID. If there is a match, Tom is recognized as an authorized user and allowed access to the computer. If the results do not match, access is denied.

You may have had the experience of forgetting your password to an online account at, let's say, Ninth Street Bank, and asked for help. The people in charge of Ninth Street Bank's web server will email you a temporary password that will allow you to log on and then require you to reset your password to something you choose (which will change your entry in the password file). You might find this annoying and wonder why they didn't just send you your original password. As we've just seen, the system at Ninth Street Bank doesn't actually know your original password, only its encrypted form and—as we are about to see—this isn't enough information to regenerate the original password.

The benefit of encryption is that, with a well-designed hashing function, possession of the encrypted password file does not make it possible to recover a user's original password (e.g., "badboy2") from the encrypted password ("gb"). This is true even if the details of the mathematical operations performed by the hashing function are known, as they often are for a particular operating system. In our simple hash function, you could reverse Steps 5 and 4, but not Step 3. For example, the following seven-character alphabetic string:

chjbup5

also hashes to the two-character string "gb", so clearly the process is not reversible, and the thief cannot determine whether the original password is "badboy2", "chjbup5", or any one of the hundreds or thousands of words that also produce the value *gb*.

But this appears to raise another problem. What if Fred and Alice each have different passwords that hash to the same encrypted value or they coincidentally chose the same original password? In general, this is not a problem—both Fred and Alice can log on to their own personal accounts using their respective passwords. But if Fred stole the password file and saw that his password and Alice's password hashed to the same value, he would have a better than random chance of guessing Alice's password; he would certainly try his own password with Alice's user ID, and he would be successful if indeed the passwords were the same.

To solve this problem, some operating systems keep a third entry for each user in the password file, namely, the exact time at which the user created the password. This *timestamp* gets appended to the original password, and the result is then run through the encryption algorithm and stored as the encrypted password in the password file. That way, two identical passwords do not hash to the same value because the probability that they were created at the exact same instant in time is infinitesimally small. This also solves the problem of two nonidentical passwords that otherwise would hash to the same value. When someone attempting to log on gives his or her password, the operating system consults the password file, appends the timestamp for that user ID to the password just entered, encrypts the result, and compares it with the encrypted password entry for that user ID in the password file.

Even with all of these safeguards in place, there are still ways in which the operating system's authentication process is vulnerable to hacking. Consider Ravi, who has not stolen the password file but nevertheless knows Alice's user ID and wants to hack into Alice's account. Because Ravi knows Alice personally, he might try to guess her password. He might try her nickname, the name of her pet poodle, or the title of her favorite band. Of course, he could also try "alice", "123456", or—a perennial favorite—

"password". Many systems set "password" as the default value, and if Alice hasn't changed (or been required to change) her password, this will get Ravi into Alice's account. Failing at these attempts, Ravi might use a brute force attack by trying all possible passwords. Suppose there are n possible characters that can be used in a password (at least uppercase and lowercase letters and the 10 digits are possibilities). To try all possible passwords of length k or less would require

attempts. On the average, Ravi might be successful after about

attempts, but this will be very time consuming (see Exercise 5 at the end of this chapter). In addition, most systems have a lockout after some number of failed tries, which would foil this brute force approach.

For someone who has stolen the password file, a better way to find a password is by using password-cracking software. For a given user ID (which our villain knows because the user ID is not encoded), **password-cracking software** first tries all words in its built-in dictionary, encrypting each with the well-known hash function and comparing the result with the password file. Such software is amazingly fast and can try millions of potential passwords per second. If this fails, it then goes on to more sophisticated techniques such as word list substitutions for common phrases. With knowledge of the hash algorithm, the software might use a *rainbow table*, a precomputed list of passwords and their hash values, so checking for a particular hash value just requires a search of the hash values in the table. A brute force attack is only used as a last resort.

Surprisingly, the easiest way to obtain someone's password is not to steal the password file (hard to do) or to guess that person's password (time consuming), but to use social engineering. **Social engineering** is the process of using people to get the information you want. If Ravi wants Alice's password, he might just ask her! In a business setting, he might get a chance to snoop around her office and find the yellow sticky note containing her password attached to her monitor or stuck beneath her keyboard. He might violate "password etiquette" and watch over her shoulder while she logs on. (This *shoulder surfing* method is often used to steal PIN numbers at ATM machines.) Or he could try an indirect approach; he could call Alice's (gullible) assistant and, posing as an IT technician, explain that Alice has called the IT service group about some computer problems she is experiencing and that to fix them, he needs Alice's password. Most companies try to educate their employees about the dangers of social engineering.

Your best defense against someone guessing your password is to be smart about how you choose and use your password. Most website operating systems impose rules about legal passwords (must be at least some minimum length, must include at least one digit and one special character, and so forth). Also, the user may be required to create a new password every so often. Managing passwords might be a bit of a hassle, but security measures are always a balancing act between user convenience and user protection. Consider using a **password manager** (password vault), a central site that securely stores all your passwords in encrypted form, leaving only the one vault password for you to remember.

User IDs and passwords are the most common authentication mechanism, but other security measures can be used in addition. Many systems ask you to preselect answers to a list of *security questions* about personal information that you will know but others might not (the first name of your maternal grandmother, the city in which you went to

high school, etc.). Then after you present your user ID and password, you must respond with the correct answer to one of these questions. Some laptops, tablets, and smartphones now use *biometric information*, for example, fingerprint scans or facial recognition. Some company networks use a **dual authentication** scheme that works as follows: The legitimate user enters his or her user ID and password. Each user has a small device (or perhaps a smartphone app) that then generates a random sequence of digits, which is good only for a few seconds, that the user must enter. Some websites present the user with an image of text written with distorted letters and the user then has to type the correct text into a textbox on the webpage. Such distorted letters are recognizable by humans but not by computers, so the idea here is to determine that the user is indeed human and thus prevent a machine from voting in a survey or filling out a form. A more elaborate version of visual recognition presents the user with a series of images and instructions such as "click on the images that show mountains."

Practice Problems

Using the hash function described in this section, verify that the password "chjbup5" also hashes to "gb".

Answer

Assume Judy uses the password "judy" for her account (in violation of the guidelines in the Special Interest Box, "Password Pointers"). Using the simple hashing function presented in this section, to what value does her password hash?

Answer

ed

If someone were to attempt to log on to Judy's account (from Practice Problem 2) using the password "mike", would he or she be granted access? Explain why or why not.

Answer

No; from Problem 2, the encrypted form of Judy's password "judy" is *ed*, but "mike" hashes to *fc*. The encrypted versions do not match.

A hacker believes he has obtained a copy of the password file for a system that uses the hash function described in this section. One of the encrypted password entries is "mt". What can the hacker conclude about this file?

Answer

The file is not legitimate. Using the hash function described, the final step is to change each digit back to a letter. The maximum digit is 9, which would correspond to *i*, so no letters beyond *i* can appear in an encrypted password.

Authorization. The intent of authentication is to allow the operating system to determine that you are the person you say you are. Once you are authenticated, there might still be operations that you are not permitted to carry out. **Authorization** governs what an authenticated user is allowed to do. Enforcing authorization rules is also one of the jobs of the operating system, as discussed in Chapter 6. On your own machine, you may have administrative privileges and be able to see everything on the machine. However, on your banking website, you can only see your own account information, and you are not allowed to change your balance or your interest rate.

The operating system maintains **access control lists** (also called permissions) that specify exactly what a user is allowed to do with each data file. Depending on who the users are, they have various levels of privilege, such as the following:

- Read access (can read a particular file)
- Write access (can modify a particular file)
- Execute access (can run a particular program file)
- Delete access (can delete a particular file)

A careful operating system will check every access every time by every user.

Main content

Chapter Contents

8.2.2 Threats from the Network

Once your handheld device, personal computer, business computer, or web server is connected to the Internet, there are many more possibilities for harm. Attacks can come from anonymous sources anywhere in the world via intermediate nodes that maintain varying levels of security. Recall from Chapter 7 that to send data across a network from the source at point A to the destination at point E, the packet may have to go through intermediate nodes B, C, and D. Also, if you remember that the Internet is not governed by one entity or one set of security guidelines, then it becomes rather clear what a difficult task maintaining "Internet security" can be.

Most of these security threats come in the form of **malware** (malicious software) that can attack an individual computer. The most common attacks to individual computers are by viruses, worms, Trojan horses, and denial of service.

Virus. A **virus** is a computer program that, like a biological virus, infects a host computer and then spreads. It embeds itself within another program or file. When that program or file is activated, the virus copies itself and attacks other files on the system. The results may be as simple as taunting pop-up messages, but could also include erratic behavior or drastic slowdown of the computer, corrupted or deleted files, loss of data, or system crashes. The virus is spread from one machine to another by passing on the infected file, perhaps on a flash drive. By far the most common mechanism for spreading a virus, however, is through email attachments. An infected file is attached to an email message and sent out to 100 people, for example. Anyone who downloads and opens the attachment causes the virus to replicate the infected file and perhaps send it out as an email attachment to the first 100 people in that person's personal address book. In this way, a virus can potentially spread like wildfire across the Internet.

Worm. A **worm** is very similar to a virus, but it can send copies of itself to other nodes on a computer network without having to be carried by an infected host file. It is a self-replicating piece of software that can travel from node to node without any human intervention. In its most benign form, a worm can clog the Internet so that legitimate traffic is slowed or shut out completely. In addition, the worm might also subvert the host systems it passes through so that, at a later time, those systems can be controlled by the worm's author and used to send **spam** (junk email), deface webpages, or perform other mischief.

Trojan Horse. A **Trojan horse** (in the software world, as opposed to Greek mythology) is a computer program that does some legitimate computational task but also, unbeknownst to the user, contains code to perform the same kinds of malicious attacks as viruses and worms—corrupt or delete files, slow down the computer, and the like. It might also upload or download files, capture the user's address book to send out spam, hide a **keystroke logger** that captures the user's passwords and credit card numbers (and sends them to someone else), or even put the computer under someone else's remote control at some point in the future. A computer can become infected by a Trojan horse when the user downloads infected software. In fact, even visiting an infected website can, behind the scenes, download a Trojan horse (an attack called a drive-by exploit or **drive-by download**).

Denial of Service. A **denial-of-service (DoS) attack** is typically directed at a business or government website. The attack automatically directs browsers, usually on many machines, to a single web address at roughly the same time. The result causes so much network traffic to the targeted site that it is effectively shut down to legitimate users. Although such an attack is seldom directed toward an individual's computer, the resulting effect can still inconvenience individual users of the targeted website. (Spam can accomplish a similar, but less targeted effect, by flooding the Internet with junk email messages that consume available bandwidth and clog mail servers.) If many machines are perpetrating this mischief, it's called a *distributed denial-of-service attack*, or *DDoS*. A DDoS may use thousands of machines, enabling much heavier attack traffic and at the same time making it harder to track down and disable all of the attacking machines. Many times, these machines are personal computers that were infected at some point by a Trojan horse. Then at a later time, the Trojan horse is activated in all these machines, putting them under the command of a single controller. This collection of machines is sometimes called a **zombie army** or **botnet** (short for "ro*bot net*work" because the machines act like robots under someone else's control).

Beware the Trojan Horse

Tiny Banker Trojan (also known as Tinba, short for Tiny Banker) is a relatively new example of Trojan horse malware that first appeared in 2012. As the name suggests, it targets online banking customers, both individuals and businesses. It is not the first such Trojan horse, but one of its most distinguishing features is that the Tinba package is indeed tiny in size, only 20 kilobytes, much smaller—and therefore harder to detect—than previous packages, some of which ran to as large as 20 megabytes, 1000 times larger. Tinba is downloaded when a user visits an infected website.

Once on the user's machine, Tinba can capture keystrokes for login data. More than that, it reads the network traffic to determine if the user is in the process of logging on to an online bank account. If so, it captures information from the bank website such as the bank title and logo, then pops up a window (supposedly from the bank) that asks the unwary user to confirm account information, such as account number, security codes, even credit card information, which it then sends off to one of its four controlling computers. If that domain is down, it switches to one of the remaining three, meaning that it has three backups for receiving the captured data. (Never reply to such a popup window; your bank will never reach out to you in this manner. Close the window and immediately run your antivirus software to scan your machine.) Finally, Tinba turns the user machine into part of a *botnet* (see for a definition of this term).

In 2014, the source code for Tinba was published, and since then more sophisticated capabilities and protections have been added to Tinba and its variants. These include measures designed to avoid

detection by antivirus software, such as increased encryption and the ability to update itself and its controlling servers. Tinba has "represented" major banking sites in the United States and around the world.

Your Money or Your Files

We mentioned earlier that cyberspace hacking is now frequently carried out by organized criminal groups, often operating on an international scale. And some of the schemes perpetrated, such as *ransomware*, are downright amazing.

Ransomware invades the user's computer, often by means of an infected email attachment. It then encrypts files on that computer, usually with a strong enough encryption algorithm that only the author of the malware software, who holds the decryption key, can unlock the files. The user is then presented with a "ransom note" demanding payment in order to release the files. Next, a screen appears with detailed instructions on how to pay the ransom and the timeframe within which it must be paid. Of course, even if the user decides to pay the ransom, the attacker might not release the files. Alternatively, the user might be warned that without payment, the files (which have by then also been sent to the server of the perpetrator) may be posted online and publicly released. The only credible defense against ransomware is a good, up-to-date backup system.

Ransomware has been aimed at individuals, corporations, federal agencies, and especially hospitals and other medical centers. Medical centers are particularly targeted because the threat to destroy or make public patient medical data often induces a prompt payment of the ransom. On May 12 2017, more than 75,000 ransomware attacks were carried out in more than 99 countries. In the U.K., many hospitals that are part of the National Health Service were attacked, forcing ambulances to go to alternate sites. Ransomware attacks grew from 4 million in 2015 to 638 million in 2016.* The typical "going rate" per infected computer is a few hundred dollars, often to be paid via *bitcoin* (we'll talk more about bitcoin in Chapter 14.) Nonetheless, some experts estimate that the annual profit in the ransomware business exceeds $500 million, and will soon reach $1 trillion per year. Some major corporations are purchasing bitcoin in order to be able to provide a rapid response to a possible ransomware attack.

In 2016, a new ransomware package named Locky made its appearance in a major way. On the first day it appeared, millions of emails were sent out with attachments carrying the Locky malware. By the following month, more than 50,000 infections had been detected. In addition to its aggressive spam campaign, Locky has several features that make it particularly dangerous:

The spam email is often configured to appear as if it comes from an address on the user's network.
As it busily works to encrypt data files, it postpones encrypting those most recently used so as to delay detection and allow the encryption process to be completed.
Locky uses two of the most sophisticated encryption algorithms, RSA and AES, discussed later in this chapter.

> Recently, however, DDoS attacks have relied less on massive botnets and instead have used a more sophisticated technique known as a **reflection and amplification** attack. This type of attack relies on integral features of the Internet, such as the Domain Name System (DNS; see Chapter 7). Queries to DNS servers are short, but these servers respond by sending much larger amounts of data back to the source of the query. The attacking computers alter the source address of the query so that it appears that the source is the DNS server itself. This results in the server receiving a torrent of information back from itself, amplifying the effect of the attacker machines by a factor of as much as 300 times the bandwidth (transmission rate) of the original query.

Phishing. Phishing is not a direct attack on a computer. Instead, **phishing** is a practice used to illegally obtain sensitive information such as credit card numbers, account numbers, and passwords. An email is sent claiming to be from a legitimate organization such as a bank, credit card company, online retailer, government agency, or social media site asking the recipient to click a link to update or verify his or her account information. Often the message contains a warning that your account will be suspended or canceled if you fail to provide this information (although this personal information is already on file with the legitimate organization if indeed you do have an account there). The link takes the unwary user to a fake website that appears to be legitimate and captures the information the user enters. Despite the fact that no legitimate bank or other financial organization ever operates this way, many people fall for this scheme and become victims of identity theft.

If you receive such a message, never follow the link or reply to the message; instead, delete the message immediately. If you want to check an account's status, open a separate browser window and access your account information as you normally would. Check that the web address area displays https:// and shows an icon of a green locked padlock, which indicates that the information will be securely transmitted, and that the site being visited is the one you intended to visit rather than a counterfeit site. If you click on the green locked padlock, you can probably see more information about the source/ownership of the webpage you are viewing and the security of the network connection.

The term *phishing* came about because perpetrators cast out bait, in the form of phony email messages, to a large number of potential victims in the hope that one or two will "bite" and fall for this scam. If the emails are targeted toward specific individuals or groups (*spear phishing*), the email content is "personalized" and the chances of success are higher. The cost to mount a phishing attack is minimal, so even a tiny number of "bites" brings a profit. The average phishing site is left online for less than three days, making it difficult to catch those responsible.

Defense against the Dark Arts

You may feel at this point that you should just unplug your computer from the Internet or disable your wireless capability. The good news is that by following a few simple guidelines you can protect yourself against almost all the attacks discussed so far.

Be sure your computer has up-to-date **antivirus software** from a reputable company. Such software can detect worms, viruses, and Trojan horses by the distinctive "signatures" those programs carry. It cleans your machine of infected files.

Be sure your computer has up-to-date firewall software from a reputable company. **Firewall** software guards the access points to your computer, blocking communications to or from sites you don't permit and preventing certain operations from being initiated across the Internet.

Be sure your computer has up-to-date **antispyware** from a reputable company. Antispyware routinely scans your computer for any "spyware" programs that may have infected your machine—programs that capture information on what websites you have visited and what passwords and credit card numbers you have used.

Set your browser to use a *pop-up* blocker. A pop-up is a small window that "pops up" on the webpage your browser is currently displaying. Most pop-ups are simply annoying advertising, but some contain links, perhaps disguised as "Close" buttons, that download a virus to your machine.

Always install the latest **security patches** or updates to your operating system; better yet, set your machine to install automatic updates.
Don't send personal or financial information in response to any email ("My wealthy Nigerian uncle left me a fortune that I am willing to share with you but I need your account number … ").

Practice Problems

Do some research on the Morris worm, the first major computer worm. Who wrote it, when was it released, how did it spread, how much damage was it estimated to have caused (in dollar amounts), and what happened to the originator of the worm?

The Anti-Phishing Working Group (APWG) is an industry and law enforcement association focusing on helping eliminate identity theft resulting from phishing (www.antiphishing.org). It provides discussion forums, maintains statistics, and tests promising technology solutions. According to APWG statistics, 1,220,523 phishing attacks occurred in 2016, representing a 65% increase over 2015.

Main content

Chapter Contents

8.2.3 White Hats vs. Black Hats

"White hats" are the security experts who try to keep us safe in cyberspace. "Black hats" are the bad guys (or gals!) who try to ferret out computer system weaknesses for financial gain, control, or outright destruction. These two groups are locked in constant battle. As we as a society grow more and more dependent on online services, the opportunities for reward increase, and therefore so do the number of attempts to breach system defenses. The technology of attacks (and defenses) changes rapidly and keeps gaining in sophistication. The defenders have the harder job; as is often pointed out with respect to security (or warfare in general), the attacker needs to find only one weakness, but the defender must defend all possible points of entry.

Asmall fontAmedium fontAlarge font

Add Bookmark to this Page

help? Main content

Chapter Contents

8.3 Encryption

Much of the focus of information security is to devise defenses so that the "bad guys" can't steal our information. If, despite all these precautions, personal files on a computer or sensitive data packets on a network are illegally accessed and fall into the wrong hands, we can still protect their contents through the process of encryption. Essentially, the purpose of encryption is to make information meaningless even if someone does manage to steal it. We've already discussed encryption of the password file by the operating system as a security measure, in case the password file is stolen. In this

section, we discuss various other encryption mechanisms, which apply to both stored and transmitted data.

You've Been Hacked

According to a report from a business advisory firm,* an estimated 15.4 million Americans were victims of identity theft in 2016, about 1 in every 16 adults. Identity theft occurs when personal information (name, address, telephone number, birth date, password, email address, credit and debit card numbers and expiration dates, and so forth) is stolen and used to commit fraud. Most of this stolen information was obtained not from computers owned by individual victims, but from corporate or government servers or point-of-sale computers. Here are some examples from 2016.

Who	When	What
U.S. Department of Justice	February 2016	Personal data on 10,000 Department of Homeland Security employees and 20,000 FBI employees
Omni Hotels and Resorts	January–June 2016	50,000 customers' credit and debit card data
Verizon Enterprise Solutions	March 2016	1,500,000 customers' contact information
Equifax W-2 Express	May 2016	431,000 Kroger employees' Social Security numbers and birth dates
Los Angeles County	May 2016	Personal data on 756,000 who had contact with branches of Los Angeles county government (through Auditor, Child Support Services, Mental Health, Probation, and many others)
Eddie Bauer	August 2016	Over 2 million customers' credit and debit card data
Michigan State University	November 2016	400,000 names, Social Security numbers, and MSU identification numbers of some current and former students and employees

In addition to potential financial or legal consequences, these organizations and others that have been similarly attacked have lost some level of trust from their customers, whether or not those customers ultimately suffered personal identity theft as a result. Trust is hard to regain, so this in turn represents a difficult-to-quantify but nonetheless real financial loss. Although institutions are generally loath to admit they have been hacked, they will suffer even greater backlash if they fail to promptly notify individuals that their personal information, account information, or credit card data may have been stolen.

8.3.1 Encryption Overview

Cryptography is the science of "secret writing." A message (*plaintext*) is encoded (encrypted) before it is sent, for the purpose of keeping its content secret if it is intercepted by the wrong parties. The encrypted message is called *ciphertext*. The ciphertext is decoded (decrypted) back to plaintext when it is received, in order to retrieve the original information. More formally, **encryption** is the process of using an algorithm to convert information into a representation that cannot be understood or utilized by anyone without the appropriate decryption algorithm; **decryption** is the reverse of encryption, using an algorithm that converts the ciphertext back into plaintext. Encryption and decryption date back thousands of years. The most famous instances of cryptography occur in military history, beginning with Julius Caesar of the Roman Empire, who developed the Caesar cipher, discussed in the next section. In more modern times, the military importance of cryptography was illustrated by the German Enigma code cracked by the Allies during World War II.*

Encryption and decryption are inverse operations because decryption must "undo" the encryption and reproduce the original text. (An exception is hash function encoding, used for password encryption, which is a oneway code and does not involve decryption.) There are many encryption/ decryption algorithms, and of course both the sender and the receiver must use the same system.

A **symmetric encryption algorithm** requires the use of a secret key known to both the sender and the receiver. The sender encrypts the plaintext using the key. The receiver, knowing the key, is easily able to reverse the process and decrypt the message. One of the difficulties with a symmetric encryption algorithm is how to securely transmit the secret key so that both the sender and the receiver know what it is; in fact, this approach seems to simply move the security problem to a slightly different level, from transmitting a message to transmitting a key.

In an **asymmetric encryption algorithm**, also called a **public key encryption algorithm**, the key for encryption and the key for decryption are quite different, although related. Person A can make an encryption key public, and anyone can encrypt a message using A's public key and send it to A. Only A has the decryption key, however, so only A can decrypt the message. This approach avoids the difficulty of secret key transmission, but it introduces a new problem: The relationship between the decryption

key and the encryption key must be sufficiently complex that it is not possible to derive the decryption key from knowledge of the public encryption key.

Main content

Chapter Contents

8.3.2 Simple Encryption Algorithms

Caesar Cipher. A **Caesar cipher**, also called a *shift cipher*, involves shifting each character in the message to another character some fixed distance farther along in the alphabet. Specifically, let *s* be some integer between 1 and 25 that represents the amount of shift. Each letter in the message is encoded as the letter that is *s* units farther along in the alphabet, with the last *s* letters of the alphabet shifted in a cycle to the first *s* letters. For example, if , then *A* is encoded as *D*, *B* is encoded as *E*, *X* is encoded as *A*, and *Z* is encoded as *C*. The integer *s* is the secret key. Decoding a message, given knowledge of *s*, simply means reversing the shift. For example, if , then the code word DUPB is decoded as ARMY.
The Caesar cipher is an example of a **stream cipher**; that is, it encodes one character at a time. This makes it easy to encode just by scanning the plaintext and doing the appropriate substitution at each character. On the other hand, there are only 25 possible keys, so a ciphertext message could easily be decoded by brute force, that is, by simply trying all possible keys.

In addition, the Caesar cipher is a *substitution cipher*, whereby a single letter of plaintext generates a single letter of ciphertext. We can replace the simple shift mechanism of the Caesar cipher with a more complex substitution mechanism, for example:

(Can you guess the substitution algorithm being used?) However, in any simple substitution cipher, the structure of the plaintext is maintained in the ciphertext—letter frequency, occurrence of double letters, frequently occurring letter combinations, and so forth. With a sufficiently long message, an experienced *cryptanalyst* (code breaker) can use these clues to recover the plaintext.

Block Cipher. In a **block cipher**, a group or block of plaintext letters gets encoded into a block of ciphertext, but not by substituting one character at a time for each letter. Each plaintext character in the block contributes to more than one ciphertext character, and each ciphertext character is the result of more than one plaintext letter. It is as if each plaintext character in a block gets chopped into little pieces, and these pieces are scattered among the ciphertext characters in the corresponding block. This tends to destroy the structure of the plaintext and make decryption more difficult.

As a simple example, we'll use a block size of 2 and an encoding key that is a 2 × 2 arrangement of numbers called a *matrix*. Here *A* and *B*

are matrices. We can define an operation of matrix multiplication as follows. The product *A* × *B* will also be a 2 × 2 matrix, where the element in row *i*, column *j* of *A* × *B* is obtained by multiplying each element in row *i* of *A* by its corresponding element in column *j* of *B* and adding the results. So to obtain the element in row 1, column 1 of the result, we multiply the row 1 elements of *A* by the corresponding column 1 elements of *B* and add the results:

To obtain the element in row 1, column 2 of the result, we multiply the row 1 elements of *A* by the corresponding column 2 elements of *B* and add the results:

The completed product *A* × *B* is .
However, for encryption purposes, we are going to modify this definition. When we add up the terms for each element, whenever we exceed 25, we will start over again, counting from 0. In this scheme, 26 → 0, 27 → 1, 28 → 2, …, 52 → 0, and so on.

Not every 2 × 2 matrix can serve as an encryption key; we need an *invertible matrix*. This is a matrix *M* for which there is another matrix *M*′ such that

For example, is invertible because for

This property is what allows *M*′ to reverse the effect of *M*. Also, part of our encryption algorithm is a simple substitution (mapping) *S* that maps letters into numbers; we'll let *S* be really simple here: , , …, . Obviously *S* is reversible, and we'll call the reverse mapping *S*′: , , …, .
To encode our message, we break it up into two-character blocks. Suppose the first two characters form the block (*D E*). We apply the *S* mapping to this block to get (4 5). Now we multiply (4 5) × *M* by treating (4 5) as the row of some matrix (and remember to wrap around if the result exceeds 25):

Practice Problems

Using a Caesar cipher with , encrypt the message NOW IS THE HOUR.
Answer

STB NX YMJ MTZW

A messenger tells you that the secret key for today for the Caesar cipher is . Should you trust the messenger? Why, or why not?
Answer

Because there are 26 letters in the alphabet, a shift of encrypts each character as itself, so you should not trust the messenger.

Using the matrix of this section, encrypt the two-character block (M Q).
Answer

Decrypt the results of Part a.

Answer

Finally, apply the S' mapping to get from digits back to characters: . This completes the encoding, and (V I) is the ciphertext for the message block (D E). Notice that the digit 4 (i.e., the plaintext letter D) contributed to both the 22 (V) and the 9 (I), as did the digit 5 (i.e., the plaintext letter E). This *diffusion* (scattering) of the plaintext within the ciphertext is the advantage of a block cipher.
For decoding, we reverse the previous steps. Starting with the cipher-text (V I), we first apply S to get (22 9). We then multiply (22 9) by M', the inverse of the encoding key (remembering to wrap around if the result exceeds 25):

Finally, we apply S' to get back—voilà!—the plaintext (D E).

Figure 8.1 summarizes the steps. Again, the matrix M is the secret encryption key, from which the decryption key M' can be derived.

Figure 8.1

Steps in encoding and decoding for a block cipher

Encoding

Apply S mapping to plaintext block.
Multiply result times M, applying wraparound.
Apply S' to the result.

Decoding

Apply S mapping to ciphertext block.
Multiply result times M', applying wraparound.
Apply S' to the result.

8.3.3 DES

Both of the previous encryption algorithms—the Caesar cipher and an elementary block cipher—are too simplistic to provide much real security. **DES (Data Encryption Standard)** is an encryption algorithm developed by IBM in the 1970s for the U.S. National Bureau of Standards (now called the U.S. National Institute of Standards and Technology, or NIST), and is certified as an international standard by the International

Organization for Standardization, or ISO (the same organization that certifies the MP3 digital audio format, as discussed in Chapter 4). One might expect this internationally standard algorithm to rest upon some extremely complex and obscure operations, but the DES algorithm actually uses very simple operations—however, it uses them over and over.

DES was designed to protect electronic information, so the plaintext is a binary string of 0s and 1s, just as it is stored in a computer. As we learned in Chapter 4, this means that ordinary text has already undergone an encoding using ASCII or Unicode to convert characters to bit strings. This encoding, however, is not for the purposes of secrecy and has nothing to do with the cryptographic encoding we are talking about in this chapter.

Hiding in Plain Sight

Steganography is the practice of hiding the very existence of a message. It's an old idea; an ancient Greek ruler was said to have sent a (not very urgent) message by tattooing the message on the shaved head of one of his slaves and then sending the slave off after his hair had grown back. The message was revealed on the other end by once more shaving the messenger's head.

These days, steganography has again come into favor in the form of hidden text within images posted on the web. As we learned in Chapter 4, a colored digital image is composed of individual pixels; in the usual RGB format, 8 bits are allocated for each of the red, green, and blue color components. This allows for variations of intensity for each color, from 0 to 255. Let's say the red component in a pixel has the following 8-bit value:
The least significant bit (the rightmost 0) contributes the least to the color intensity. If this bit is changed from 0 to 1, the red component becomes

Such a tiny change (1/256th of the original red color) would not be detectable to the human eye viewing the image.

A text file can be hidden in an image file by changing (if needed) the least significant bit of each byte of the image file to match the binary form of the characters in the text. For example, if the first letter of the text file to be hidden is "A", with ASCII code 01000001, then the first 8 bytes of the image file would be modified (if needed) so that the least significant bits are 0, 1, 0, 0, 0, 0, 0, and 1. To the naked eye, the image appears unaltered.

In the following set of images, the image on the left is the original. The image on the right uses steganography to hide 668 KB of text, over 140 doublespaced pages. Can you see any difference?

A standard Windows file compare sees a lot of difference:

In 2011, German Federal Criminal Police arrested a suspected al-Qaeda member who was found to be concealing a memory stick. It contained a pornographic video, which in turn contained, hidden via steganography, 141 separate text files reviewing the success of past al-Qaeda operations and laying out plans for future operations.

Photos courtesy of Judith Gersting

DES is a block cipher, and the blocks are 64 bits long, meaning that 64 plaintext bits at a time are processed into 64 ciphertext bits. The key is a 64-bit binary key, although only 56 bits are actually used. The algorithm begins by sending the plaintext 64-bit string through an initial *permutation* (rearrangement). The algorithm then cycles through 16 "rounds." Each round *i* performs the following steps:

1. The incoming 64-bit block is split into a left half Li and a right half . A copy of the right half gets passed through unchanged to become the left half of the next round, .
2. In addition, the 32 bits in the original right half get permuted according to a fixed formula and then expanded to 48 bits by duplicating some of the bits. Meanwhile, the 56-bit key is also permuted (the result is passed on as the key to the next round) and then reduced to 48 bits by omitting some of the bits. These two 48-bit strings are matched bit by bit, using an XOR (exclusive OR) gate for each bit. Figure 8.2 shows the standard symbol for an XOR gate, along with its truth table.

Figure 8.2The XOR gate

3. The resulting 48-bit string undergoes a substitution and reduction to emerge as a 32-bit string. This string is permuted one more time, and the resulting 32-bit string is matched bit by bit, using XOR gates, with the left half of the input. The result is passed to the next round as the new right half .
   After all 16 rounds are complete, the final left and right halves are recombined into a 64-bit string that is permuted one more time, and the resulting 64-bit string is the ciphertext. Figure 8.3 outlines the steps involved in the DES algorithm.

Figure 8.3The DES encryption algorithm

Two important points about the DES algorithm: The first is that every substitution, reduction, expansion, and permutation is determined by a well-known set of tables. So, given the same plaintext and the same key, everyone using DES ends up with the same ciphertext. The "secret" part is the 56-bit initial key. The second point is that the same algorithm serves as the decryption algorithm—just start with the ciphertext and apply the sequence of keys in reverse order, that is, the round-16 key first and the original secret key last.

With increased computing power now available to those trying to break a code, a 56-bit key does not seem as formidable as when DES was first introduced. It might even be feasible to use a brute force method and try all (72,057,594,037,927,936) possible keys. In 1998, the Electronic Frontier Foundation, a nonprofit civil-liberties organization, used a PC connected to a large array of custom chips to respond to a challenge in the form of a DES ciphertext message posed by RSA Laboratories, the research component of RSA Security, a leading electronic security company. Their system could test 88 billion keys per second and it found the correct 56-bit key in less than three days. If a system could evaluate 100 billion keys per second (using, for example, a massively parallel supercomputer like those described in Section 5.4), it would take only a little more than a week to try every possible secret key. *Triple DES* improves the security of DES; it

requires two 56-bit keys (which can be thought of as a 112-bit key length), and runs the DES algorithm three times: Encode using key 1, decode the result using key 2, encode the result using key 1 again.
Concerns about the eventual breakdown of DES in the face of ever-increasing computing power prompted NIST in 1997 to request proposals for a successor encryption scheme. The result was **AES (Advanced Encryption Standard)**, which was adopted for use by the U.S. government in 2001. AES is based on the Rijndael (pronounced Rindahl) algorithm, named for the two Belgian cryptographers who designed it, Vincent Rijmen and Joan Daemen. Like DES, AES uses successive rounds of computations that mix up the data and the key. The key length can be 128, 192, or even 256 bits, and the algorithm appears to be very efficient.

Main content

Chapter Contents

8.3.4 Public-Key Systems

The encryption algorithms we have discussed so far have all been symmetric encryption algorithms, requiring that both the sender and the receiver have knowledge of the secret key. Our final example is an asymmetric, or public-key, encryption algorithm. Remember that the main difficulty with a symmetric algorithm is how to securely transmit the secret key. In a publickey system, the encryption key for messages to go to a particular receiver is broadcast to everyone, but the decryption key cannot be derived from it and is known only by the receiver.

The most common public-key encryption algorithm is **RSA**, named for its developers, Ron Rivest, Adi Shamir, and Len Adleman at MIT (founders of RSA Security, mentioned above). This algorithm, developed in 1977, is based on results from the field of mathematics known as *number theory*.

A *prime number* is an integer greater than 1 that can only be written as the product of itself and 1. For example, 2, 3, 5, 7, 11, ... are prime numbers; you can only write 7, for example, as , the product of 1 and 7. The numbers 4, 6, 8, 10, and 12, for example, are not prime because they can be factored in a nontrivial way:

Any positive integer is either a prime number or a number that can be written in a unique way as a product of prime factors. For example, is the product of three prime factors. The success of RSA encryption is based on the fact that it is extremely difficult to find the prime factors for *n* if *n* is a large number. That is, it is easy to multiply prime factors together to get a result, such as . However, it is far more difficult to be given the value 34,371,246 and asked to find all its prime factors. So although information

encrypted using RSA is technically not secure, it is secure in practice because of the large amount of computation necessary to find the prime factors of the encoding key.
Here's how RSA works. Two large prime numbers *p* and *q* are chosen at random (where "large" means 200 or so digits). Then their product is computed. The product is also computed. Next, a large random number e is chosen in such a way that *e* and *m* have no common factors other than 1. This step guarantees the existence of a unique integer *d* between 0 and m, such that when we compute $e \times d$ using the same sort of wraparound arithmetic we used in the block encoding scheme—that is, whenever we reach *m*, we start over again counting from 0—the result is 1. There are computationally efficient ways to produce *p*, *q*, *e*, and *d*.
For example, suppose we pick and for our two prime numbers (a trivially small case). Then,

1.
2.
3. Choose (and have no common factors)
4. Then because , so when we compute $e \times d$ using wraparound arithmetic with respect to 12, we get 1.
   Now the number pair (*n*, *e*) becomes the public encryption key, and *d* is the decryption key. Let's suppose that the plaintext message has been converted into an integer *P*, using some sort of mapping from characters to numbers. The encryption process is to compute using wraparound arithmetic with respect to *n* (when you reach *n*, make that 0). Continuing with our example, suppose . Then the ciphertext is computed as

•5.

(Note that the sender uses both parts of the public key, *n* and *e*, to compute the ciphertext.) The receiver decodes the ciphertext *C* by computing using wraparound arithmetic with respect to *n*. In our example,

•6.

Of course, our example has a major problem in that *d* is the same as *e*. Obviously, in a real case, you want *e* and *d* to be different. The whole point is that even though *n* and *e* are known, the attacker must determine *d*, which involves finding the prime factors *p* and *q* of *n*. There is no known computationally efficient algorithm to carry out this task.

Quantum Computing vs. RSA

RSA encryption begins by multiplying two large prime numbers p and q to get the product *n*. The security of data encrypted via RSA depends on the computational difficulty (that is, time to perform this task) of reversing this process, that is, finding the two prime factors of *n*. The faster this can be done, the more feasible it becomes to "crack" an RSA-encrypted ciphertext.

Shor's algorithm, developed in 1994, uses quantum computing to solve the factorization problem (see the Special Interest Box "Quantum Computing" in Chapter 5). The difficulty is that this algorithm requires an "idealized" quantum computer, one with a sufficiently large number of qubits (the computational unit of a quantum computer, much as the bit is the computational unit of a conventional computer). It turns out that building a quantum computer with a large number of qubits is difficult because such systems rapidly "deteriorate."

In 2016, a team of researchers from MIT and the University of Innsbruck (in Austria) developed a new quantum computer architecture. As proof of concept, this team was able to factor 15 into 3×5 using Shor's algorithm with a machine using 5 qubits rather than the 15 qubits normally required. In addition

to reducing the number of qubits required, thus speeding up the computation, the architecture itself is scalable, meaning that, theoretically, the number of qubits can be arbitrarily large. The consequence is that in future, when such machines become available, RSA will no longer be a secure encryption algorithm.

Practice Problems

You receive a message "17" that was sent using the RSA public encryption key (21, 5) of the example in this section. Decode this to find the original numeric message. (You might want to use a spreadsheet to help with this computation.)

Answer

In the example, where and , then as well. To decode, compute
Therefore the original numeric message was 5.

Main content

Chapter Contents

8.4 Web Transmission Security

One area about which the public is very security conscious is in making online purchases, which require the purchaser to send his or her name, address, and—most worrisome of all—credit card number across a network. One method for achieving secure transfer of information on the web is **SSL (Secure Sockets Layer)**. This is a series of proprietary protocols developed by Netscape Communications (Netscape was an early web browser) in the mid-1990s. The **TLS (Transport Layer Security)** protocol, first defined in 1999, is based on SSL and is nearly identical to SSL but with a few technical improvements. The major difference is that TSL is nonproprietary and is a standard supported by the Internet Engineering Task Force (IETF), an open organization of individuals concerned with "the evolution of the Internet architecture and the smooth operation of the Internet."*

Both TLS and SSL are in use and are supported by all web browsers. Technically, TSL/SSL fits between the Transport layer, with its TCP protocols, and the Application layer, with its HTTP protocols (both discussed in the previous chapter). When you see a closed lock icon at the top or bottom of your web browser page, or when a web address begins with HTTPS, instead of HTTP, then you can be assured that the communication between your browser and the web server (the vendor's web computer) is secure and protected by TLS or SSL. TLS/SSL allows a client (the purchaser's web browser) and a web server to agree on the encryption methods to be used, exchange the necessary security keys, and authenticate the identity of each party to the other. (Here we are again with *encryption* and *authentication*, the two pillars of security we've seen before.)

Now that you know a bit more about encryption, you might ask what encryption algorithm TLS/SSL uses; is it DES encryption or the newer, stronger RSA encryption? Surprisingly, it is both. One of the problems with the RSA algorithm is the computational overload for encryption/decryption. What often happens is that RSA is used in the initial stage of communication between client and server. For example, in response to a client request, the server may send the client its public key along with a **public-key**

certificate (also known as a **digital certificate**). This is a certificate issued by a trusted third-party certificate authority; it's like a letter of introduction that attests that the server belongs to the organization the browser thinks it is talking to.

The client, using RSA and the public key of the server, encodes a short message containing the keys for a symmetric encryption algorithm. Because only keys are being encrypted, the message is short and the encryption can be done quickly with little RSA overload. The server receives and decodes this message and responds to the client with a message encoded using the symmetric key. The client and the server have now established a secure exchange of keys (one of the issues with a symmetric encryption algorithm) and can complete the transaction using, for example, DES. Figure 8.4 illustrates the major steps in this process, although the technical details may require a few additional transmissions between the client and the server. The exchange of setup information between the client and the server, preparatory to exchanging real data, is known as a **handshake**.

Figure 8.4 A typical TLS/SSL session

Another way to obtain network security is to transmit via a **Virtual Private Network (VPN)**. When you have an account with a VPN provider, then you can use the software provided by that VPN to connect to one of their servers (and later disconnect). This connection is made via an encrypted connection (sometimes called a *VPN tunnel*) so that all traffic back and forth between your device and that server is encrypted. The VPN server assigns a unique IP address to your device. For subsequent data traffic from your device, your ISP sends that traffic through the Internet to the VPN server for further processing, but that traffic is encrypted. Therefore, the ISP can neither read that data nor know what subsequent service you are asking the VPN server to perform, such as connecting you to some website. All traffic between the VPN server and the rest of the Internet sees only your assigned IP address.

One of the disadvantages of using a VPN is that your Internet traffic may be a bit slower than otherwise, because it goes through the VPN server. However, we have already mentioned two advantages of using a VPN:

1. Your device's IP address is "hidden" behind the IP address assigned by the VPN server, so web traffic cannot easily be connected directly to you, that is, to the device you are using.
2. All network traffic between you and the VPN server is encrypted, so at least that leg of the journey is secure.

In October 2016, The Federal Communications Commission adopted rules that prohibit ISPs from collecting, storing, and sharing/selling customer information (web browsing history, app usage history, geolocation, and much more that can be considered personal data) without customer consent. In April 2017, before these rules had even gone into effect, the U. S. President signed into law legislation that overturned those rules and prevented similar rules from being adopted in the future. Using a VPN means that much of such data is hidden from the ISP providers, who can only "see" that you are connected to a VPN server. Hence there will no doubt be more interest in using VPN services in the future, although it should be noted that VPN services are not regulated and the lengths to which providers go to protect your data vary greatly.

Although we have been talking here about commercial VPNs, the idea of a VPN originally was to allow employees of a business to securely connect to corporate networks while away from the office. This usage is still in effect, but often goes hand-in-hand with another software tool called a **Remote Desktop Protocol (RDP)**. RDP allows an employee at some other location to see and utilize the desktop of a specific computer, including using files and software applications available on that computer.

Main content

Chapter Contents

8.5 Embedded Computing

Up to now, we have primarily treated security as if it were always a matter between your personal computer and the rest of the world. But there are many other places where computing devices interact with the rest of the world.

Embedded computers are computational devices such as chips, processors, or computers that are embedded within another system. Unlike a general-purpose computer that can be programmed to do many different things, embedded computers usually perform just one or two tasks as part of the system in which they occur. Frequently this means they are small devices that you never see, and the ability to package more and more transistors on a single chip has contributed to the growth of embedded computers. Where are they found?

- In your car
- In your home thermostat
- In your home alarm system
- In your cell phone
- In your digital clock or watch
- In your iPod or MP3 player
- In your video game console
- In your TV remote controller
- In your digital camera
- In your credit card
- In your microwave
- In your car's navigation system
- In your car's emergency and vehicle diagnostics system (for example, General Motors OnStar)
- In a plane's flight control system
- In a bank ATM
- In a hospital patient monitoring system

Well, you get the idea—embedded computers are everywhere! It is estimated that 98% of all new CPUs (central processing units) are used in embedded systems.

Embedded computers differ from conventional computers in other ways as well. Input and output are usually not via keyboard and screen, respectively; instead, input may be data sent from sensors within the surrounding system, and output may be commands to control devices within the system. The embedded computer has very limited memory and processing capacity. And its response time is often in the "real-time" category (a

plane's flight control system has to respond instantly, so does the patient monitoring system, and you hope your antilock brakes do that as well). For these same safety-critical systems, reliability is a key requirement.

But reliability is not the same as security. Security comes into the picture when these embedded systems become networked or even Internet-enabled. Your car's navigation system gets updates from the web, the home alarm system connects over a network to the alarm company, the electric company may connect with your thermostat to raise the temperature slightly during peak air-conditioning load times, and sensors that monitor manufacturing processes communicate to streamline the production flow. The challenge is how to build security into network-enabled embedded computers that have little extra memory, often have energy constraints, and have no system administrator at hand to deal with security updates or breaches.

The term the *Internet of Things* originated in 1999. As mentioned in Chapter 7, the Internet of Things (IoT) refers to the idea that in the future, many of the objects we interact with daily will be connected (or at least connectable) to the Internet, often by means of RFID tags that can sense their environment and both send and receive data. The future is now here! In 2017, an estimated 20 billion devices were already part of the IoT.

This technology has many positive uses: medical monitoring, inventory control, home appliance and security management, traffic monitoring, industrial robot management, and so forth. An agricultural application we might not think of uses RFID ear tags for livestock to track and safeguard herds (see Figure 8.5). A home assistant can recognize voice commands and even hold a conversation ("What's the weather report for today?" "What is Tom Hanks' latest movie?").

Figure 8.5 Scottish Highland cow with RFID ear tag

But at the same time, there are many potential privacy risks: If all this data is floating around, then it can be captured, scanned, and perhaps manipulated by government agencies, criminal organizations, or others with less-than-honorable intent. Your home assistant can record everything it hears in your living room and store it in the cloud, which certainly raises questions about personal privacy.

On the other end of the size spectrum, computer systems control much of the world's vital infrastructure such as utility plants, electric grid systems, transportation, telecommunications, and financial markets, to say nothing of defense and military installations. It is not hard to imagine terrorist groups and perhaps hostile foreign powers contemplating attacks on one of these systems.

In February 2013, President Obama issued an executive order (Executive Order 13636: Cybersecurity Framework) for the development of a set of industry standards and best practices that could serve as a basis for organizations to measure and manage their cybersecurity risks. Exactly one year later, as a result of collaboration with representatives from industry, government, and academia, the National Institute of Standards and Technology issued the Framework for Improving Critical Infrastructure Cybersecurity, version 1.0. The framework guides organizations through tasks of

identifying and protecting resources critical to their business goals and of detecting, responding to, and recovering from attacks. Still, there is concern that such a voluntary policy does not go far enough to protect essential national infrastructure.

Mischief-Makers in the Internet of Things

In October 2016, the largest denial-of-service attack to date was perpetrated on a company called Dyn. Dyn controls much of the Internet domain-name structure, and bringing this company to its knees also caused Internet disruption for many U.S. and European companies. This DDoS attack was carried out by something called the Mirai botnet. But this botnet is not composed of compromised computers; instead it uses compromised devices from the Internet of Things. (That's right, your web-enabled home refrigerator might have been part of this attack!) Taking advantage of the multitude of devices on the IoT, this attack at its peak used an estimated 100,000 devices.

Mirai builds its list of insecure IoT devices to incorporate into the botnet by randomly scanning the Internet and trying various user ID/password combinations. IoT devices frequently have default (and hardwired, that is, can't be changed) combinations set by their manufacturers. In fact, Mirai uses a list of 60 such combinations that are tried on any device, combinations such as *admin/password* or *guest/12345*. How do we know this? Because shortly before this massive DDoS attack, the source code for the Mirai malware was publicly released, thus making additional attacks using Mirai or a variant even more likely.

> Safeguarding vital infrastructure is a complex but extremely important issue, and much work remains to be done. This is not just a U.S. issue, but a concern of every industrialized nation.

Main content

Chapter Contents

8.6 Conclusion

In this chapter, we've looked at components of information security, both on an isolated local computer and on a machine exposed to the network. Whether it's an individual's personal data, proprietary corporate information, sensitive government data, or military infrastructure, all are under threat. Still, a bit of caution and common sense can go a long way. As the well-worn watchword says, "Security: It's Everybody's Business."

Main content

Chapter Contents

8.7 Summary of Level 3

We have seen that the hardware described in Chapters 4 and 5 is, by itself, nearly unusable. Working directly with the hardware components of a Von Neumann machine—processors, memory, ALU—is impossible for any but the most technically knowledgeable users. To make the system accessible, the system software must create a people-oriented *virtual machine* or *virtual environment* that is easy to understand and use. In addition to ease of use, this virtual environment provides a number of other services, including resource management, security, access control, and efficient resource

use. A great deal of work has been done to try to identify and create an optimal virtual environment.

Operating systems—a critical component of the virtual environment—have evolved from early batch systems through multiprogramming and time-sharing to the current network and real-time operating systems. Most modern operating systems allow us to use a large collection of machines, called a computer network, almost as easily as if it were a single logical system. Network technology has expanded our virtual environment to encompass a worldwide grid of interconnected computers. More and more, the computer user on a networked system can deal only with *what* operations need to be done, not with where or how they can be done. Perhaps they are being done "in the cloud." The future of computer systems definitely lies in the direction of more abstract and more powerful virtual environments.

As our virtual environment has expanded, our most important asset—our data—is increasingly at risk for theft or destruction by clever intruders with malicious intent. Constant vigilance is required to maintain information security so that our virtual environment is not only user-friendly but also safe.

Now that we have created a vastly more usable environment in which to work, what do we want to do with it? Well, we probably want to write programs that solve important problems. In the next level of the text, we will begin our study of the software world.