CHAPTER 6

Computer Security

Computer security is an extremely broad field spanning across multiple domains, ranging from the security of user data to the physical safety of the user. On a commercial robot, for instance, computer security is used to protect sensors and actuators whose malicious use can have a devastating impact on human life. Some examples of critical computer systems that could be compromised, if they have security issues, are healthcare systems, missile defense systems, and aviation systems, to name a few. Security spans from hardware and software to the social behavior of the users of the computer. In computer security, most things are not unconditionally secure; in general, they are only computationally secure. In other words, most security primitives are secure only under a given set of assumptions about the adversary. As computers are evolving, the amount of computation resources available to an adversary is increasing exponentially, year over year. As a result, security mechanisms must be upgraded to ensure the same level of security over time. Colloquially, it is a cat-and-mouse game where the defender must stay one step ahead of the adversary. The complex topic of physical safety is out of scope for this book. In this chapter, we will focus on data security, including types of security, adversary models, and mechanisms to secure data at rest (in storage) and data in transit between computer systems.

Privacy is another field that is always strongly associated with security. The fundamental security mechanisms used to protect data are the same mechanisms used to protect privacy. Privacy is associated with one or

P. D. Crutcher et al., *Essential Computer Science*,
https://doi.org/10.1007/978-1-4842-7107-0_6

more users and pertains to the confidentiality of Personally Identifiable Information (PII). With the advent of targeted online advertisements, privacy of users comes under scrutiny as both users and governments are worried about the data collection companies knowing too much about users and being able to predict their next actions. Sometimes, users voluntarily share their personal data in exchange for a free service; and in other cases, hidden software collects data from cameras, computers, and other devices used by the consumer without the explicit approval of the user. This is an evolving field with new primitives like differential privacy being developed in order to balance economic and human needs.

Access Control

Like any other advanced field, computer security has its own jargon. In this section, we demystify common terms that are used by media, industry, and security experts to express ideas around security. As security is all about protecting data, there are certain fundamental security properties of data that need exposition. Data can have the following security properties: Read, Write, and Execute. Data is readable for an actor if the actor can read the data without being blocked by any agent in the system. The data is writable if the actor can write the data in the system without being blocked by any other agent. Finally, the data is executable if an actor can point an agent to execute the commands. An astute reader will notice that the properties of data are from the perspective of an actor. In other words, same piece of data that is readable for one actor may be only writable for another or executable for yet another actor, or a combination of these properties for the same piece of data may be valid for another actor. In a computer system, these properties are specified by software and enforced by hardware. This is also called as *access control* of data and is enforced by a *trusted* agent in the system. The trusted agent will have all access to the

data and will grant selective access to other agents and actors. The agents and actors in a computer system span both hardware and software.

In many cases, it is not possible to use access control for enforcing the properties of data. In those scenarios, we need to use cryptography. Cryptography is the art and science of protection of data in the presence of adversaries. This is a vast field of study, and in this chapter, we will talk about the fundamental properties of data that can be enforced using cryptography. The cryptographic properties are similar to the access control properties that can be enforced, but the mechanisms of the enforcement are vastly different. In addition, in certain scenarios, access control can be enforced, while in other scenarios, like sending data over an untrusted channel, cryptographic mechanisms must be used. It is important to point out here that most of the modern cryptographic algorithms and protocols are only secure under assumptions of compute limitations of an adversary. An adversary with unlimited compute capability can bypass most of the cryptographic mechanisms being used today. We introduce common cryptographic and security properties in the rest of this section.

Confidentiality

Confidentiality of data covers if the data is secret or not. It is clearly a corollary to the Read property explained in the preceding under access control. If an actor can read a data, it is not confidential to the actor. If the actor cannot read it, it is confidential to the actor. There is a long list of encryption algorithms that are used to encrypt data to ensure its confidentiality. Unencrypted data, called *plaintext*, is sent through an encryption algorithm to generate a ciphertext. A key is used for encryption. As shown in Figure 6-1, in a symmetric encryption algorithm, the same key is also used for decryption, the process of generating the plaintext from ciphertext. Any actor that has the key has the read access to this data since it can *decrypt* the ciphertext and read the plaintext. Since the

same key is used for encryption and decryption, this mechanism is called Symmetric Encryption. In the United States, there is a body called the National Institute of Standards and Technology (NIST) that standardizes encryption algorithms that are used by most of the industry. Currently, the strongest encryption algorithm standardized and recommended by NIST is Advanced Encryption Standard (AES).

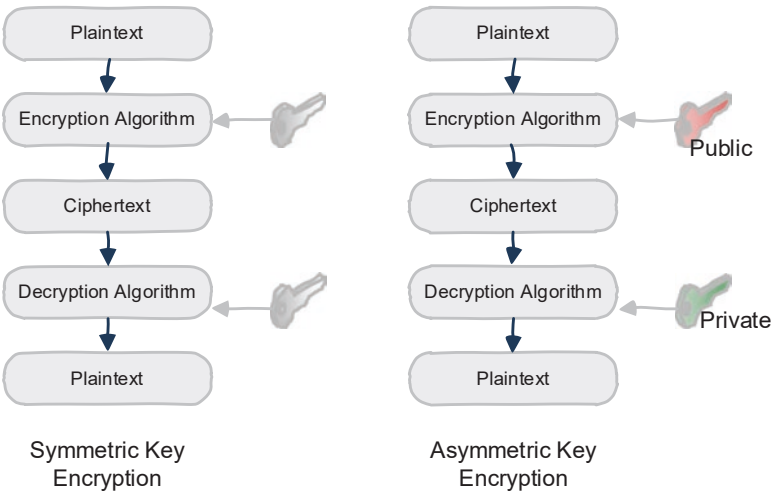

Figure 6-1. *Symmetric and Asymmetric Encryption*

AES succeeds DES (Data Encryption Standard) and 3DES and is considered much more secure than its predecessors. Most cryptographic libraries provide APIs (application programming interfaces) for AES encryption; and most general-purpose processors from Intel, Apple, AMD (Advanced Micro Devices), and ARM support instructions for acceleration of AES encryption and decryption.

Integrity

Integrity of data implies whether the data has been modified or not. It is a corollary to the Write property explained in the preceding under access control. If an actor can write a data, the integrity of the data is controlled by the actor. To ensure the integrity of data, data may be encrypted using a symmetric algorithm, and the same ciphertext will give the same plaintext using the same key. The problem we have is what if the adversary changes the ciphertext; on subsequent decryption of the modified ciphertext, the plaintext will be different from the original plaintext, which is clearly not desirable. The concept of hashing was introduced to solve this problem. Hashing is the process of mapping an arbitrary-length data blob to a fixed-size data blob called hash. The hashing algorithms are one-way functions such that given a hash value, it is computationally infeasible (or extremely hard) to find another data blob that would compress to the same hash value. In addition, a given data blob will always hash to the same hash value as long as the same algorithm is used. In order to ensure the integrity of the data blob, the hash value is protected from the adversary by either storing it separately from the data blob or encrypting it. If the adversary modifies the data blob, the hash value will change, and the hash of the modified data blob will not match the original hash value. Also, as mentioned in the preceding, it is computationally infeasible to find two blobs that will map to the same hash.

SHA-3 and SHA-512 are common hash functions used in cryptography. SHA-512 and SHA-3 can support a maximum of 512 bits of hash. In other words, any large blob can be hashed to 512 bits of hash. An astute reader will note that multiple large blobs can be hashed to a set of 512 bits. As a result, the data-to-hash relationship is two way; however, it is considered extremely hard (computationally intensive) to find a *new plaintext* that will hash to the same hash. In other words, if H(x) =Y, it is highly computationally intensive to find x' such that H(x') =Y.

Availability

Availability of data points to the physical presence of the data. It implies if the data is available to the actor to read, write, or execute. In other words, if an actor has access to the data but the data is deleted from its location in storage or memory or an adversary prevents a legitimate actor from accessing the data to which the actor has permissions, the data is considered to be no longer *available*. Cryptography generally does not help with availability as the operating system controls deletion of data. Availability is ensured by restricting access of an adversary to the data.

Symmetric Key Cryptography

The preceding cryptographic and access control mechanisms work very well within a single system. However, the security problems become much more complex when multiple systems on the network are involved and the network channel is untrusted. If we have multiple systems on the network, we need protocols to ensure that a network adversary cannot interfere with the integrity of the data being transferred on the network. If Alice's computer wants to send a confidential letter to Bob's computer, Mallory, who has access to the network channel, should not be able read or write the messages between Bob and Alice. In addition, we want to ensure that Mallory should not be able to *replay* the messages from Alice to Bob. For example, if Alice wants to ask Bob to withdraw 20 dollars from the bank, Mallory should not be able to capture the message and replay it to Bob and make Bob withdraw 20 dollars multiple times, something which Alice never intended to do. The same mechanisms for encryption and hashing work in this scenario, but we get into a problem of sharing the keys between Alice and Bob so that Bob could decrypt the message.

Symmetric encryption algorithms become problematic in network communication because there needs to be a secure way for the two sides to have the same key. In the preceding example, if Alice needs to encrypt and Bob needs to decrypt, they both must have the same key. In the absence of fast and secure communication channels, there is no way to send the key such that Mallory cannot get it. This is where *asymmetric key* cryptography comes in.

Asymmetric Key Cryptography

Asymmetric key cryptography, as shown in Figure 6-1, is a set of algorithms enabling cryptographic operation with one key and its reversal with another key. For example, a data blob can be encrypted with a public key PuKey1 and decrypted with another private key PrKey1. As a result, Alice and Bob can agree on a PuKey1 and a PrKey1 allowing Alice to send messages encrypted with PrKey1 and letting Bob decrypt the messages with PuKey1. Alice's key is called the private key, and Bob's key is called the public key. The mechanism used by Alice and Bob is called the key establishment protocol. There are multiple key establishment protocols being used in the industry including RSA based on its authors Rivest-Shamir-Adelman, DHKP (Diffie-Hellman Key Establishment Protocol), and ECC (Elliptic Curve Cryptography). All the protocols are based on a mathematical algorithm like discrete logarithms, factorization of prime numbers, and so on with a common theme that it is easy to compute the results in one way and almost impossible (without the knowledge of the keys) to reverse this computation. These key establishment protocols are computationally expensive; hence, they are not used to encrypt a lot of data. Instead, they are used to establish a shared symmetric key between the two parties, and the symmetric key is then used to encrypt/decrypt the data on the channel. This provides the best security while minimizing the overhead that comes with such security solutions.

Digital Signatures

Cryptography provides us a way of doing digital signatures, exactly like we sign a checkbook from our bank. When we sign a checkbook, the bank can verify our signature, we cannot deny signing it, and we cannot reuse a check once the money has been withdrawn and cannot repudiate that we signed the check. Digital signatures provide all these properties for digital documents. A digital signature is done by hashing a document and then encrypting the hash with a private key. Any entity (like a bank) that has the public key can verify that the document is signed by the owner of the private key. The bank saves the signed copy with a check number such that the owner or some intermediary cannot reuse the signed document again. One important thing to note is that digital signatures do not provide confidentiality but only provide nonrepudiation and integrity. The protocols built on digital signatures provide protection against replay attacks. In many countries including the United States, European Union countries, and India, digital signatures are also legally admissible in court.

Digital Certificates

Any entity with a computer can generate a public/private key pair. How do we know that this public/private key belongs to Alice or Bob? Well, we need someone to tell us that the public/private key belongs to Alice or Bob. This is where the certificate authority comes into play. A certificate authority is a well-known agency like the driver license office or the government that issues passports. Everybody knows the driver license office and trusts them. This concept is taken to the digital world where well-known companies sign up to become certificate authorities (CAs). They are trusted by the rest of us because these companies are in the business of issuing certificates and any wrong endorsement from them significantly hurts their credibility and their business. In this scenario, the CA issues a certificate that tells everybody the hash of the public key

of Alice or Bob. This way whenever Alice or Bob signs anything with their private key, any verifier can verify the signature and also check whether the key belonged to Alice or Bob.

The most common certificates used in the industry are X.509 certificates. These certificates have the details of the subject and the issuer and the public key of the subject. They are signed with the private key of the issuer (CA).

Certificate Chains

Since one CA cannot sign certificates for everyone, the certificate signing architectures are generally distributed, where one or more CAs form a central ring and they issue certificates to the large corporations in the outer ring. Large corporations issue certificates to their people, products, and devices, essentially forming a certificate chain rooted in the innermost circle, the CA itself. These certificate chains are also called chains of trust where the root certificate is the trusted certificate and all the child certificates derive trust from the root certificate.

Salts and Nonces

Salts are random bits of data generated using a random number generator. Salts are commonly used as an input to a hashing or an encryption algorithm such that the output of the algorithm is randomized. They are commonly used in password systems where passwords are stored to protect them from pre-computation or dictionary attacks. For example, a password X= “password” can be hashed using a function F(X)=Y. If an adversary knows the length of the password and that it is made from the English alphabet, the adversary can pre-compute a dictionary of all permutations of eight English alphabet letters. As a result, when it sees the hash, all it needs is to look for the corresponding hash in its dictionary and it can find the password string. However, if we generate a 64-bit random

number R and concatenate it with the password such that F(X||R) = Y', the adversary will have to generate 2^{64} = 18,446,744,073,709,551,616 (20 digits) dictionaries in order to find the password.

Nonces are also random bits of data generated using a random number generator and used as an input to various cryptographic algorithms. Nonces are not a secret from an adversary and commonly not repeated. In network protocols, nonces are used for ensuring the order of the packets and protecting from an adversary that tries to benefit by reordering the packets.

Random Numbers

Random numbers are a foundational element of cryptography and computer security. They are used for generating keys, nonces, and salts. A salt is a random bit. Sometimes, they are used to seed counters used in symmetric encryption algorithms. Intuitively simple, true random numbers are extremely hard to generate in a computer system because of lack of entropy (randomness) in computer hardware and software algorithms. As a result, special-purpose primitives are built in the computer systems to provide this entropy. There are really three kinds of random numbers. First, *True Random Numbers (TRNs)* are numbers generated from a physical phenomenon like a flip of a coin. They are exceedingly difficult to emulate with deterministic algorithms on computers. The second kind of random numbers are *pseudorandom numbers*. Here, a seed is created from randomness of the computer, using an entropy source like user inputs, heat of the system, speed of the fan, and so on; and this random value is used as a seed to generate pseudorandom numbers. Given the seed and the algorithm, the next number can be predicted, hence the name pseudorandom. *Cryptographically secure random numbers* are the third class of random numbers commonly used in cryptography. These provide forward secrecy (knowing a number from the series will not divulge any previous numbers in the series) and break-in recovery (knowing a number from the series will not divulge future numbers).

Security in Client Computing Systems

In the previous section, we read about the fundamental primitives for security of any system. The two fundamental primitives we read about are cryptographic mechanisms and access control. In most security solutions, one of these two mechanisms is used for protecting any asset. In the next section, we look at some of the contemporary technologies that the industry has developed in order to provide secure experiences in client computing. We have talked about the fundamental principles and primitives used for security and cryptography in the previous section. In the next few sections, let us discuss how these primitives are used in modern-day compute clients, servers, and the network. Modern-day clients (including desktops, laptops, phones), networks (including the Internet), and servers (IT [information technology] servers, external servers on the Internet, cloud servers) all attempt to work together to provide seamless security to the user. Client systems not only depend on the local platform mechanisms for security, but they also depend on servers in the cloud to configure and manage security locally. The client security comprises primitives for protecting data at rest, data in motion, and data in use and intersects with network security wherever data in motion must be protected.

Malware, the Bad Apples of Software

In an industry where millions of lines of code are written per day, there are thousands of hidden defects in said code. In the software parlance, these defects are called bugs. These bugs can be further classified into two main categories. The first category consists of nonsecurity bugs where the code is doing something other than what the programmer intended it to do. These bugs may impact the user experience, functionality, safety, and/or the performance of the system. The second kind of bugs are more interesting from a security perspective. These bugs, named *vulnerabilities*,

are opportunities for an adversary to exploit the system to steal and abuse user data and/or illegitimately change the behavior and/or characteristics of the system. Malicious software that exploits these vulnerabilities is called malware. The malware that exploits these vulnerabilities is further classified into virus, worm, trojan, and so on based on the mechanisms used by the malware, its goals, and the impact it has on the user's system. Skoudis et al. supply a good overview of the classification of all kinds of malware found in the wild in their book. In this chapter, we will abstract out the types of malware and focus on malware in general.

Malware is written by a myriad of actors, from so-called script kiddies who cobble together scripts to exploit a vulnerability to organized crime houses, sometimes funded by state agencies to indulge in cyber warfare. There is also an open market for malware called Darknet. Most malware will use multiple vulnerabilities to attack the system and follows the BORE (Break Once, Run Everywhere) model. This provides the malware writer motivation to devote resources to write the malware and then be able to use it repeatedly on a large number of machines till the vulnerability is fixed. Even when the vulnerability patches (software updates) are released by the original software vendor, it can take a long time (sometimes years) for these patches to reach all the end systems. Although the delivery mechanisms have become more efficient in the vertically integrated ecosystems like some phones, they are far from perfect.

Malware is extremely hard to detect because it looks like benign software to the untrained eye. However, the anti-malware industry has figured out a way to detect *known* malware with the help of antivirus (AV) scanners. The anti-malware industry employs security researchers to characterize a malware and generate a fingerprint for it. This fingerprint, called the *signature* (not to be confused with a cryptographic signature), is then fed into the antivirus scanners running on the computers. The antivirus scanner then searches for the known signatures in the software stored and executing on the platform. If a signature matches, it alerts the user and/or deletes the malware from the system. This search-based

mechanism has served the industry well since 1988 when the *Morris Worm* was found in the field. However, these signatures are very fragile such that changing one bit in the malware code can change this signature and provide a way for the malware to bypass the scrutiny of the antivirus running on the system. As the number of viruses is increasing and the number of corresponding signatures is rising to the order of millions, the antivirus companies are struggling in this battle with malware writers. Malware authors can now write self-modifying malware, also called homomorphic malware. The enormous number of signatures does not only consume heavy compute resources, but they are also easy to circumvent due to the ability of the malware to self-modify. Fortunately, the advent of artificial intelligence (AI) and neural networks has given us a new set of tools against malware. In the AI-based approach, the antivirus (AV) companies extract attributes of the malware and create a deep learning model from those attributes. Some examples of these attributes include function names in the malware, IP (Internet Protocol) addresses used, variable names, source of malware used, and so on. Since malware writers tend to reuse code, even modified malware has remnants of its parents. A new malware when passed through the inference engine is likely to get detected as malware even if some bits have been modified from the parent.

Security of Data at Rest

Most user data on clients is stored on either a flash-based SSD (solid-state drive) or a magnetic disk. This data is the easiest to steal for an adversary. The adversary can steal the device, pull out the hard disk or SSD, connect it to another system, and read all the data. The industry has been worried about this physical attack for a long time; as a result, full-disk encryption solutions have been developed to fend off such attacks. All user data stored on the disk is encrypted, and the key is bound to the user and the device such that the data can only be decrypted when the user logs into the same

device. This prevents an adversary from using another device or another user login to illegitimately access the data. Most modern operating systems have disk encryption built in them including Windows, Chrome OS (operating system), iOS, Android, and macOS. Disk encryption provides the user an assurance that their data is secured even if the device is lost or stolen.

Security of Data in Use

Protecting data at runtime is harder due to the fast-evolving nature of malware that tries to steal data and/or alter execution paths at runtime. Most general-purpose compute devices provide hardware mechanisms for software isolation like

- Process isolation
- Separation of privileged code from nonprivileged code
- Execute-disable bits – make modifiable memory as non-executable
- Mechanisms to protect the stacks – protection against stack overflows
- Protections against Return-Oriented Programming attacks (ROP attacks)

Client systems even go further to provide trusted execution environments (TEEs) to run algorithms at higher-privilege levels. Some examples of these TEEs are the secure virtual machines running on top of VMMs (Virtual Machine Monitors) and security controllers in the platform. They all run code at high-privilege levels where most malware finds it hard to attack them. Although the industry has been churning out increasingly capable defense mechanisms, the bad guys have not stopped. As a result, we are likely to see increased progress in mechanisms for protecting runtime environments on client platforms in the coming years.

In 2018, some researchers from the Google Zero project found a way to exploit branch prediction in CPUs (central processing units) to do a privilege escalation attack. The most prominent attacks on branch prediction have been Spectre and Meltdown. This led to a flurry of security fix patches from silicon and operating system vendors, impacting millions of systems. This was an attack that was thought to be too computationally intensive to run, but with improved CPU performance on modern systems, it is now extremely feasible. This was a stark reminder for the industry that nothing is really absolutely secure. Even if something has stood up to the test of time for decades, it does not mean it is completely secure.

Application vs. Kernel vs. Drivers

Most general-purpose operating systems are structured in a similar way such that the operating system (or the kernel) manages the hardware and runs at a higher privilege than the applications. General-purpose computing processors provide hardware mechanisms for the operating systems to protect their own execution and I/O (input/output) from applications. Applications run at user privilege, a lower privilege level than the OS itself. In addition, we try to make sure that for any code that runs on the platform, its provenance or origin is known before it executes. As a result, most applications are signed by their owners and verified by the operating system on which they execute. These signatures are cryptographic signatures that have a certificate chain rooted in a well-known CA that is used to identify their owners. This provides multiple security benefits: (a) It makes sure that the owner of the application does not introduce a malware in the application, since it can be traced back to them. (b) It deters malware writers since they must get a certificate in order to sign the malicious application.

I/O hardware generally has an associated piece of code that is used to manage the hardware. This piece of code, called the driver, typically runs in a privileged mode under the OS. Drivers decouple the I/O from

the rest of the operating system, provide a granular way to manage the I/O including updates, and are isolated from the applications. However, since these drivers live in the privileged domain, they must be protected, and the kernel must be protected from them. To harden them, these drivers are signed and verified by the OS like other applications. Every general-purpose operating system provides a mechanism of signing and verifying drivers.

User Authentication and Authorization

Another way of protecting user data is to provide strong authentication for the user and the actors trying to access the data. Identifying and verifying the identity of the user is named as user authentication. Once the user is authenticated, it is granted access to certain resources. This grant is called authorization. User authentication has significantly evolved in the industry, from user passwords that by themselves are inherently unsecure to biometric authentication that may use face and/or fingerprinting to other multifactor techniques, such as texting passcodes. Biometrics and multifactor techniques have significantly enhanced the security and experience of authentication. Multifactor authentication requires the user to prove who they are via two or more of the following criteria: something they have (e.g., a phone that can be texted a passcode), something they know (e.g., a password), and/or something they are (e.g., a fingerprint or their face). The fundamental problems with passwords are that as the length and complexity requirements of passwords increased, it became harder for the users to remember them. As a result, users started using the same passwords for different systems, like websites. To address such password reuse, these sites add “salt” and then hash these passwords and save the resulting hash to disk. With this scheme, if an adversary compromises a server, it would be able to see only the hashed passwords. It can still do an offline dictionary attack (copy the file to its local storage and try to crack it) on the salted password, but this is a much harder

problem than cracking unsalted passwords. If the hacker can discover the password, they can potentially compromise many of that user's accounts over many websites where the user was using the same password. This was clearly an undesirable situation.

It used to be that industry used to shy away from biometric authentication because of the fundamental concerns around non-replaceability of biometric data for a given user. That has changed now, and the industry is rallying behind biometric authentication although sending biometric data over the network is still frowned upon. User authentication is done at multiple levels, from a user login into the OS to a user login into a website. The current state of the art in user authentication is FIDO (Fast Identity Online), which turns the user authentication around. A user generates a private-public key pair and sends the public key to the server. The private key is protected using a pin or biometric authentication on the client, and the public key is saved at the server. Every unique website has a different public-private key pair, so a compromise of the public key at the server does not compromise the user account, since the adversary cannot do much with the public key without possessing the private key.

Trusted Execution Environments and Virtual Machines

Traditionally software running on mainstream computers has been classified into user applications (like browsers, file explorers, etc.) and operating system that hosts these applications. The operating system is the supervisor that manages all the hardware resources on the platform and selectively grants them to the applications. Most people are familiar with Windows, Chrome OS, and macOS. Since the operating system runs at a higher-privilege level, by the virtue of managing resources, it also enforces access control. The applications run at a lower privilege level

from the operating system, thereby insulating the operating system from the applications. As the threat landscape has evolved, it turns out that operating system–level access control is no longer sufficient. There is a trend to run applications in an environment that is more secure than the operating system itself. These are not traditional applications like Notepad but purpose-built applications for security, like a user login service. These specialized environments, isolated from the operating system, are called trusted execution environments (TEEs). These are highly secure environments running extra secure applications. Trusted execution environments may run on a separate controller as a peer to the host operating system, albeit with higher privileges than the host operating system. The alternative is to have TEEs run on the same controller as the host operating system in a time-sliced fashion and with higher privileges. Virtual Machine Monitors (VMMs) are used to achieve the latter, while security controllers are used to achieve the former. These TEEs are protected from the operating system and user applications and provide higher security than the operating system itself.

Traditionally one platform could run one operating system, but it turned out that one operating system was not able to consume all the resources of the platform. Virtualization was then invented to solve the problem of running multiple operating systems. Virtualized systems run virtual machines that are containers for operating systems. All the virtual machines on the OS are managed by another layer of software called a Virtual Machine Monitor (VMM). Since the VMM can isolate the VMs from each other, VMs have become one way of instantiating a TEE. VM-based TEEs are commonly used in commercial OSs in the market.

Secure Boot

Secure boot is the process of loading and executing mutable code after verifying the first mutable code by hardware. Mutable code is code that can be modified (before execution) in non-volatile storage like a disk or a solid-

state drive. Subsequent mutable code is verified by the previously verified mutable code, thereby forming a chain. It is commonly used to protect from malware attacks that modify firmware/software in persistent storage and is a common industry practice now. The main goal of secure boot is to ensure that only the system firmware and OS from a trusted source execute on the client. As explained previously, the OS makes sure that the applications and drivers are signed and sources are verified.

Most commodity hardware provides mechanisms for secure boot. They might either have a non-mutable code embedded in the hardware (in a ROM) or have a security controller that is responsible for verifying the mutable code. Clearly the code for the security controller itself must go through secure boot, and for that non-mutable code is typically stored in the ROM. Figure 6-2 shows a typical scenario of secure boot.

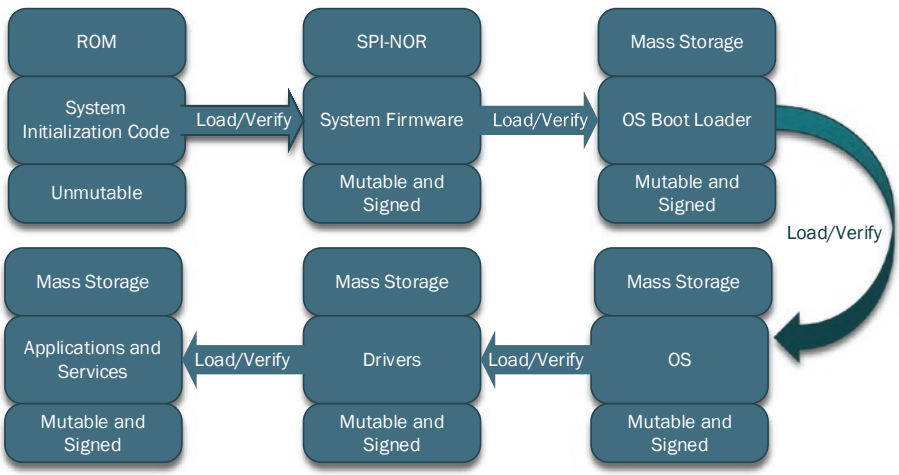

Figure 6-2. *Secure Boot*

The preceding diagram shows a typical secure boot session. In each node in the diagram, the topmost bubble shows where the code is stored, the middle bubble shows what the code is, and the lowermost bubble shows whether the code is mutable and signed. Most client platforms

boot from a program stored in read-only memory (ROM). This program is considered unmutable since it is programmed in the memory at the factory and cannot be modified after that. Once this program executes, it loads the next program from persistent storage. The persistent storage can be limited (few megabytes) storage in the form of a SPI-NOR or be a big storage drive sized to the order of terabytes. The program in the persistent storage is considered mutable since the persistent storage can be modified by an adversary. As a result, the ROM program checks the cryptographic signature of the program in memory before executing it. This ensures all the programs that execute are cryptographically verified.

Secure I/O

Human beings interact with the computer using I/O peripherals. The security of the I/O peripherals is of utmost importance. Let us take a common scenario of money transfer; Alice wants to send 100 dollars to Bob, and she fills out a bank form on her favorite browser. Mallory implants a malware in the path from her keyboard to the browser and in the path from the browser to the display. When Alice types Bob, the browser receives the name Mallory. Although the browser wants to display Mallory in the window, the malware makes it display Bob. When Alice clicks Submit, the money gets transferred from Alice's account to Mallory's account. Most operating systems own and manage the I/O channels like keyboard, mouse, display, and so on. However, this makes both the OS and the I/O devices vulnerable to malware in the OS or in the device itself. Connecting the device directly to a TEE (trusted execution environment) protects the TEE and the device from OS malware. This connection can be a logical connection where there is a security protocol between the device and the TEE or a physical connection where the TEE directly manages the physical port connected to the device. Secure storage is also another form of secure I/O where the TEE manages the storage.

Digital Rights Management

Digital Rights Management (DRM) came into prominence with the Digital Millennium Copyright Act (DMCA). It criminalizes copyright infringement or attempts to evade protections put in place to control access to copyright works. More commonly, it is used to protect videos and music from unlicensed consumption. Most client computing devices provide mechanisms for the user to be able to access licensed content while deterring the user from accessing unlicensed content. These security mechanisms commonly work with the help of a TEE. Typically, the client first enrolls with a content provider. Once the server has identified and authenticated with the client, it provides the encrypted content (movie or music) to the client along with the license. The TEE then decrypts the content and coordinates with the operating system to play the content on the selected media device. The hardware and the software on the client ensure that the licensed user can access the content but cannot copy the content for redistribution or for use on another device. An astute reader will notice this is one of the fields in computer security where the owner of the device itself is not completely trusted with the data present on the user platform, since the user is not the owner of the movie but only a consumer of it. From the perspective of the content industry, as they are pouring billions into new content, they need these DRM mechanisms to protect their investments.

Communication Security: Security of Data in Motion

Most computers converse over an untrusted channel on the Internet. Even though corporate and home networks are considered more trustworthy due to the restricted physical access to the data cables and data signals on which the data travels, the trend is going toward open networks where the

clients are expected to reduce their trust in the network channels and take appropriate cryptographic and security measures to ensure that the data can travel securely over untrusted networks.

On the network, the security protocols used must ensure the confidentiality, integrity, and/or replay protection of data. There are really no protocols available today that can protect against denial of service in an adversarial network. In other words, if Alice sends a message to Bob and Mallory is sitting on the adversarial network, Mallory can drop the message, and there is nothing Alice or Bob can do about it except Bob informing Alice that he did not receive a certain message and Alice resending it. The following three protocols are commonly used to ensure the security properties of the data on the network.

Transport Layer Security

TLS (Transport Layer Security) is the second generation of the Secure Sockets Layer (SSL) cryptographic protocol that is designed to protect data being sent over an untrusted network. It is commonly used between a web browser running on a client and the server providing the service. The use of TLS has expanded to email servers, chat servers, voice calls, and even media streaming in some cases. TLS provides confidentiality and integrity of the messages using cryptographic asymmetric and symmetric key mechanisms. In common scenarios, web browsers do a bidirectional authentication with the server, that is, the server authenticates the client device and the client authenticates the server.

TLS (Transport Layer Security) and encryption on the Internet in general are seen as a double-edge sword by the government and the regulatory bodies across the world. The same secure conduit that allows users to protect their data from adversaries on the network is also used by malware to send malicious data across the network while avoiding the prying eyes of the government and regulatory agencies. Governments want

to be able to monitor the data, and privacy advocates do not want any loopholes in privacy protocols – the debate is ongoing.

Figure 6-3 shows the Open Systems Interconnection (OSI) layers of a network stack. Although data protection applies to all these layers, most security solutions use TLS in the transport layer and IPSec (Internet Protocol Security) in the network layer while resorting to purpose-built protocols in the application layer. The cryptographic primitives used by these protocols remain the same, while the messaging formats and the number of messages in the protocols change. It is also common to see data being encrypted multiple times as it travels down the stack and getting decrypted as it goes up the stack on the receiving side.

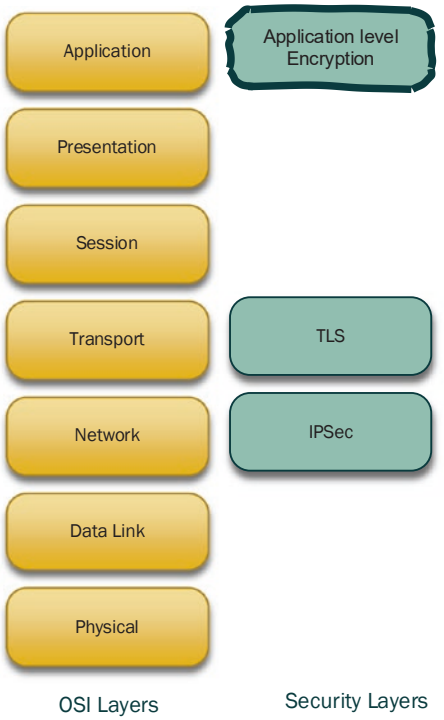

Figure 6-3. *Communication Security*

Virtual Private Network

Virtual Private Networks (VPNs) provide a mechanism to extend corporate, school, and home networks to remote, untrusted networks such that a device connected from a remote network is functionally connected to the private network. The device encrypts all the network data, and the data travels through the untrusted Internet in an encrypted fashion. Once the data reaches the edge of the private network, the data is decrypted at the edge and sent to the nodes in the private network. More specifically, it allows enterprise users, corporate users, and users of big organizations to connect to their parent networks through untrusted networks, like the Internet. There are three big categories of VPNs that are commonly used:

1. Connecting host to network as described in the preceding
2. Connecting two networks together through an untrusted network
3. Connecting two disparate networks following different network protocols or IP addressing schemes

Over the years, many protocols have been used to accomplish VPN, but today Internet Protocol Security (IPSec) and versions of TLS are used for setting up these secure channels. The use of TLS for VPN is managed by the operating system in contrast to the application managing a TLS session. In other words, with application ownership of TLS, every application will have one or more unique TLS sessions with one or more servers, and it is the application's responsibility to set up and tear down the session and make sure that the data being sent is sent through the TLS session. In contrast, a TLS VPN covers the whole client. All applications on the node can send data naturally without worrying about the TLS, and the

OS ensures that the data is always sent through the TLS connection. These (application and VPN) are two separate ways of using TLS, which are not necessarily interchangeable for given usage.

IP Security

IPSec, Internet Protocol Security, is a protocol that works at the IP layer and secures the data in the network channels. One of the foundations of the VPN is it works at the network layer to provide a secure communication channel from the source node to a network. Like TLS, the fundamental mechanisms in IPSec are the same, a key establishment/exchange protocol followed by data transmission that is encrypted and integrity protected with symmetric keys. Unlike TLS, IPSec works at the network layer, while TLS works at the transport layer. The difference is what part of the data header and data payload is encrypted and integrity protected. Like TLS, IPSec also has various modes for authenticating endpoints and protecting data.

Writing Secure Programs: Where Do We Start?

A lot has been written about secure programing and secure software. In this section, we talk about the fundamentals and provide some pointers to find more information. First, no program or code runs in isolation. It always depends on its environment, a set of libraries, a set of APIs, and sometimes software running on a remote server that this program interacts with. Hence, the security of the environment has a direct bearing on the security of the program itself. Even in these scenarios, there are certain fundamental security tenets that most programmers can use:

1. Every program has inputs and outputs; it is important to make sure that all the inputs are checked for an allowed range and any input out of range is rejected.

2. Establish boundaries of trust. This will ensure that a vulnerability in one part of the program will not be used to compromise other parts of the program.

3. Programs that use cryptography should never implement their own cryptographic functions. It is strongly recommended to use existing cryptographic libraries for cryptographic primitives. It has been repeatedly shown that cryptographic functions are extremely hard to get right, so it is recommended to stick to proven libraries that have survived the test of time.

4. Memory allocated in the heap or the stack should be carefully managed and range checked and eventually freed. Memory overflows are one of the topmost causes of vulnerabilities. Programming languages that provide automatic memory management and garbage collection, so-called manage environments (e.g., C#, Java, Python), are more resilient to these kinds of attacks than languages that expect the programmer to explicitly manage memory (e.g., C, C++).

5. It is the responsibility of the programmer to ensure that any logs generated do not have any secrets, since logs are generally not access controlled.

6. Compiler warnings are our friends. Sometimes compiler warnings point us to hidden vulnerabilities; always try to fix the compiler warnings before shipping the code.

7. Adhere to the principle of least privilege. If a function or subroutine does not need access to a certain variable, restrict it from the function to prevent any unintended modifications of the variable.

8. When there are multiple people working on the same program, have a secure coding standard so things remain simple and do not get cloaked in multiple styles or standards of coding.

9. Run static and dynamic analysis tools to remove any inadvertent errors that are not caught by compilers.

10. It always helps to have a second pair of eyes review the code.

11. Lastly, always have a recovery plan ready. Attackers will find vulnerabilities, and they will compromise your program. There must be a way to fix the vulnerability and update the new program in the field.

Summary

Computer security has become an integral part of computer science. It not only impacts our data; in some cases, it can impact our physical safety. As the threats in the ecosystem are evolving, the industry is developing new countermeasures to diffuse these threats. However, there is no silver bullet that can counter all threats, and we need a mixed set of tools in our arsenal to protect us from these emerging threats. The fundamental cornerstones of computer security, access control and cryptography, are likely to evolve in coming years. As outlined previously, passwords are on their way out, albeit slowly, and are increasingly likely to get replaced with biometrics-based techniques. We can expect use of more encrypted network channels, VPNs (Virtual Private Networks), or TLS (Transport Layer Security) as the network data increases. The need for DRM (Digital Rights Management) is going to increase as the media industry pours billions into new and exciting content. Privacy will be the key debate for the next decade due to multiple economic factors like advertisement revenue that enables service providers to provide *free services* to users in exchange for user information. Finally, state actors are likely to use cyber warfare to complement traditional warfare, and there will be an increased need for encryption mechanisms that can be *managed by* law enforcement authorities. One thing is clear: computer security as we know it today will transform in a positive manner in the coming years.

References and Further Reading

- A. Acquisti, C. Taylor, and L. Wagman. "The economics of privacy." *Journal of Economic Literature*. 2016, doi: 10.1257/jel.54.2.442

- C. Dwork and A. Roth. "The algorithmic foundations of differential privacy." *Found. Trends Theor. Comput. Sci.*, 2013, doi: 10.1561/0400000042
- C. Paar and J. Pelzl. *Understanding Cryptography*. 2010
- J. Daemen and V. Rijmen. "The Design of Rijndael." *New York*, 2002
- M. E. Smid and D. K. Branstad. "The Data Encryption Standard: Past and Future." *Proc. IEEE*, 1988, doi: 10.1109/5.4441
- NIST. "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions." 2015
- N. H. Function *et al.* "Description of SHA-256, SHA-384 and SHA-512." *ACM Trans. Program. Lang. Syst.*, 2016
- R. L. Rivest, A. Shamir, and L. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems." *Commun. ACM*, vol. 21, no. 2, 1978, doi: 10.1145/359340.359342
- W. Diffie, W. Diffie, and M. E. Hellman. "New Directions in Cryptography." *IEEE Trans. Inf. Theory*, vol. 22, no. 6, 1976, doi: 10.1109/TIT.1976.1055638
- D. Johnson, A. Menezes, and S. Vanstone, "The Elliptic Curve Digital Signature Algorithm (ECDSA)." *Int. J. Inf. Secur.*, 2001, doi: 10.1007/s102070100002
- G. M. Lentner and P. Parycek. "Electronic identity (eID) and electronic signature (eSig) for eGovernment services – a comparative legal study." *Transform. Gov. People, Process Policy*, 2016, doi: 10.1108/TG-11-2013-0047

- D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile." 2008
- E. Skoudis and L. Zeltser. *Malware: Fighting Malicious Code*. Pearson, 2003
- C. Fachkha and M. Debbabi. "Darknet as a Source of Cyber Intelligence: Survey, Taxonomy, and Characterization." *IEEE Commun. Surv. Tutorials*, 2016, doi: 10.1109/COMST.2015.2497690
- H. Orman. "The Morris Worm." *Secur. Privacy, IEEE*, 2011
- P. Kocher *et al.* "Spectre attacks: Exploiting speculative execution." 2019, doi: 10.1109/SP.2019.00002
- J. Corbet, A. Rubini, and G. Kroah-Hartman. "Linux Device Drivers, Third Edition." *Linux Device Drivers, Third Edition*, 2005
- A. Kadav and M. M. Swift. "Understanding modern device drivers." 2012, doi: 10.1145/2150976.2150987
- S. Ghorbani Lyastani, M. Schilling, M. Neumayr, M. Backes, and S. Bugiel. "Is FIDO2 the kingslayer of user authentication? a comparative usability study of FIDO2 passwordless authentication." 2020, doi: 10.1109/SP40000.2020.00047
- J. Gerhardt-Powals and M. H. Powals. "The digital millennium copyright act." *ACM SIGCSE Bull.*, 1999, doi: 10.1145/384267.305937
- S. Rose, O. Borchert, S. Mitchell, and S. Connelly. "Zero Trust Architecture." *Nist*, 2019.