# 3
# How Software Works

By this point in the book, we've already mentioned the term "digital world" quite a few times. This world is none other than the one shaped by software from various manufacturers, running on servers, your personal computer, or your *smartphone*. This software forms the core of what we use daily in the digital world—what we know as applications, platforms, or online services.

Just as with manufacturing a car or a kitchen knife, software is built from raw materials and advanced production processes. In its case, the raw material consists of text files known as *source code*, where programmers write instructions that are later translated into a language machines can understand. Therefore, if the final quality of a car largely depends on the materials used and the professionals who build it, software quality similarly relies on the experience and expertise of the programmers writing the source code, as well as the refinement of the development process[24].

Because we've been building applications for a relatively short time, **the professionalization levels of the software industry are not comparable to those of more classic trades** with many more years of experience, such as architecture or aeronautics. Romans were building bridges over 2,000 years ago. In contrast, the first programmable calculator was only built less than 100 years ago[25]. Since we've had relatively little time to develop software, it's obvious there's still much room for improvement. Despite this, we manage daily to produce incredible applications that make us feel

like we're living in the future. However, we must admit that the reliability and resilience of these digital products are far from what we'd like them to be.

## "WhatsApp Is Down"

Think for a moment about the last time a service you use was unavailable for a while. Social media, messaging apps, or even our online banking occasionally go down and stop working without warning. The primary requirement for a bridge or a building is that it doesn't collapse while you're using it. Yet, online services, no matter how well-built or maintained, occasionally go dark when you least expect it.

These temporary outages usually occur because, unlike a bridge or a building, software undergoes structural modifications throughout its lifespan[26]. Just as our personal computers install patches and restart occasionally, the software underpinning online services also receives regular maintenance to fix bugs and add new functionalities. Applying patches to these gigantic constructions made of different pieces of software doesn't always go smoothly. Even when teams perform tests, small, seemingly innocuous, routine changes are often the culprits behind major, untimely outages.

Occasionally, other, less structural changes can also cause problems. In 2012, a routine password change for the guest Wi-Fi at Google's offices triggered a series of cascading failures that took an internal password storage application offline. Without this application, some employees couldn't access certain services and were unable to work. The application's recovery process had never been tested and didn't work on the first attempt. This forced people out of bed on the other side of the world, and it even required drilling through the lock of a safe to retrieve a master key needed to get the application back online[27]. This example perfectly illustrates the unpredictability of seemingly innocuous changes and

the exposure to incidents, even for companies with mature and well-tested management processes. Occasionally, chaos finds an unforeseen angle through which to slip in, even in the best-run operations.

**Software resilience and security are difficult requirements to meet in isolation, but far more challenging to achieve simultaneously**. Particularly secure software can complicate recovery tasks in case of an incident—for example, by forcing response teams to obtain relevant authorizations before acting (or literally requiring them to drill open a safe), thus slowing down their agility and increasing the duration of outages. On the other hand, both resilience and security are particularly invisible requirements because when they're present, nobody notices them. And yet, for any user, both an outage and a security breach are intolerable.

Perhaps the most reliable method for delivering stable and reasonably secure software is to subject it to the test of time. Some mission-critical systems, where a user cannot afford a sudden crash or erratic behavior, use software versions that would be considered obsolete by consumer standards. If we visit a nuclear power plant or the bridge of a warship, we're likely to find computers running operating systems several versions behind the latest available in stores. This is because software used in such critical environments must undergo very strict testing and certification processes, requiring time before a specific version can be deemed reliable. To someone who loves always using the very latest technology, these might seem like old and obsolete systems. In reality, their resilience, stability, and security likely far surpass many of the more modern products we use. Cutting-edge software is sometimes released under the urgency of business generation needs and, consequently, suffers from lower stability and a higher probability of having yet-undiscovered vulnerabilities.

## Freedom or Security

However, even if time is invested in thoroughly testing software, it's unlikely that all vulnerabilities can be eliminated. There's a clear asymmetry in the software world: an application can be declared insecure after a vulnerability is observed, but even if we find no flaws, we cannot confidently declare that none exist. It's impossible to find proof certifying an application is secure[28]. There are multiple examples of vulnerabilities that have survived in the shadows for years until being discovered and exploited[29] [30].

Users, however, demand that their applications be secure, and their tolerance for failures is very low. The public announcement of a security breach in a product is usually met with outrage and a feeling that the responsible party has committed gross negligence. Companies know this. In Chapter 2, we discussed the need for many companies to perpetuate business models based on people continuing to visit and use their services with increasing frequency. For these companies, it's paramount that users feel secure sharing more and more data, creating content, and increasing their reliance on these services. Because of this, companies like Apple, Google, Microsoft, and Amazon invest immense amounts of money in having the best cybersecurity teams in the world, but also in marketing campaigns to boast that their products are more secure than those of the competition.

Nobody likes having to issue a statement notifying their customers of a data breach. This need to instill trust and avoid negative headlines has forced companies to take drastic measures and launch increasingly closed software ecosystems. The clearest example is Apple and its *App Store*. By implementing a review process for all software before it receives permission to be installed on an iPhone or iPad, Apple has almost entirely eliminated any possibility of users being infected with malicious software[31]. The collateral effect is that Apple also prevents the installation of

*undesirable* software that could directly compete with its business model (e.g., an App Store from another manufacturer)[32]. Most iPhone users are unaware that they've sacrificed some of their freedom to achieve a substantial improvement in security. In cybersecurity, nothing comes for free.

In Chapter 8, we will discuss this compromise further, along with other characteristics of iOS-based devices (the operating system for iPhones and iPads) and how other ecosystems have partially or entirely copied this model to enhance users' perceived security.

## The Bear-Repelling Rock

Software manufacturers go to great lengths to make their users feel secure using their products. This feeling often stems more from the influence of marketing and brand image than from concrete evidence or facts.

At the beginning of *The Simpsons* episode titled *Much Apu About Nothing* [33], a bear appears in the neighborhood, causing panic among residents. Residents quickly demand urgent and drastic solutions from the mayor to prevent such an event from happening again. Mayor Quimby responds to the popular demand by creating a disproportionate anti-bear patrol, consisting of surveillance vehicles and even fighter jets. This patrol is soon after praised by Homer for its effectiveness, with him arguing that he hasn't seen a bear in the neighborhood since. Lisa, very intelligently, explains to him that the absence of bears cannot be attributed to the patrol's presence, and that she herself could attribute the bears' disappearance to a simple rock she holds in her hand.

This conversation between Lisa and Homer about the reasons one feels secure is reflected in some of the techniques companies use to sell software. Beyond their own campaigns emphasizing how vital it is for them to keep their users secure, there's an entire industry of specific software dedicated to selling users concrete solutions to improve their security. Examples include

Antiviruses, personal Firewalls, Virtual Private Networks (VPNs), Cookie Cleaners...

These promises primarily seek validation through an appeal to authority. That is, their target user won't have the capacity or knowledge to verify these promises, so they must trust the company behind it and its historical prestige. A well-placed logo can speak more about a product's perceived quality than its actual effectiveness. This practice of leveraging brand image to convey quality is widely used across all sectors. However, unlike the textile or food industries, in the software world, it's much harder for a user to examine the true quality of the product they're using.

Beyond marketing, we can say that in almost all cases, there's a genuine interest behind each product to improve its users' security posture. Except for specific cases where the security gain is marginal or products border on being scams[34], we can say an antivirus is objectively useful for protection against some common threats. We can also assume that companies like Google or Apple have an honest interest in protecting their users from certain threats—at least from those they aren't directly causing and that aren't instrumental to achieving their business objectives.

On the other hand, it's important to point out the side effects of using services advertised as secure or products like antiviruses, VPNs, and other security software. An overreliance on this perceived security can lead us to relax and let our guard down. **We must not become complacent, believing that a service provider is invulnerable or that we're protected simply by using a VPN or an antivirus**. Downloading and executing files too carelessly, trusting in the coverage of this type of software, or being swayed by promises of immunity that have been sold to us, can lead to some unpleasant surprises. In Chapter 10, we'll discuss the limitations of specific software like VPNs and their implications for security and privacy. For now, suffice it to remember that bear-repelling rocks, unfortunately, do not exist.

## Fixing Leaks

The only robust strategy for keeping software secure and resilient is regular maintenance. Software manufacturers must regularly publish updates that fix bugs, improve performance, or eliminate vulnerabilities. Without updates, software slowly withers. For someone seeking a strong security posture (like the reader of this book), it's of vital importance to evaluate the quality of product maintenance and how long that support is expected to continue.

Most manufacturers offer long-term support versions. Sometimes, newer-featured versions of the same software are also released in parallel, but with maintenance periods that end sooner[35]. In both cases, we must pay attention to the duration for which the manufacturer commits to continuing to publish these patches. Once this maintenance ceases, we cannot rely on that software remaining secure to use.

Furthermore, it's also important to consider the effort required from the user to keep their software updated. There are various ways to design this task, ranging from the most rudimentary to the most automated. Generally, a user interested in disciplined updating should favor software that updates automatically in the background, without requiring any action on their part and with an almost invisible user experience. This is the case, for example, with most modern browsers or applications downloaded on mobile devices (Android or iOS). In these cases, once the software is installed, the user is rarely required to take any additional action for periodic patch application.

At the other extreme, it's advisable to avoid software that requires the user to manually download patches or merely reminds the user that an update is pending installation. Occasionally, having to endure an occasional restart will be inevitable, but we must guard against the possibility of indefinite procrastination or being tricked into downloading a patch from a fraudulent site.

Regarding online services, the maintenance of the software that keeps them running is obviously the service provider's responsibility. End-users have little oversight here, and it will be difficult for them to easily identify which services are poorly maintained. They should therefore limit themselves to using providers with apparent reputations[36] who can presumably afford the necessary resources to keep their software updated.

There are various factors that play a relevant role in a company's ability to provide secure and resilient software to its customers. Among these, the company's size and complexity are probably the most relevant factors. Unfortunately, as we'll see below, some companies have to contend with both factors at different points in their lives.

## Move Fast, Break Things

Although for several years now all companies claim that the cybersecurity of their products is one of their priorities, some face significant challenges in fulfilling this promise. Certain factors limit their opportunities to perform routine tasks like patching software with due diligence or thoroughly and carefully testing products before making them available to customers.

Firstly, the urgency to generate profits or start selling a product does not favor software security or resilience. *Startups* or companies whose investors exert pressure to shorten development cycles as much as possible—to grow and generate revenue quickly—are not the best environments for creating secure software. As we've previously mentioned, delivering secure software requires careful design, review by many eyes, and detailed testing that tends to significantly extend development cycles. This philosophy runs counter to the mindset of many startups, which is to iterate their products quickly until they find their market fit. During its first 10 years of operation, Facebook proudly displayed the slogan *Move Fast and Break Things*[37]. It wasn't until 2014 that Mark

Zuckerberg announced its change to the phrase *Move Fast, but with a stable infrastructure*[38], thus illustrating the shift in mindset needed to continue ensuring the reliability of their products and not lose user trust.

On the other hand, even when this shift in mindset occurs and companies reduce the speed at which they deliver new products, difficulties in keeping their software secure sometimes persist. Earlier in this chapter, we discussed how software development techniques are still immature and constantly evolving. This means certain products can become obsolete in less than 10 years, creating genuine "technical debt" that is difficult to replace once deployed to customers. This type of software—still functional but obsolete and difficult to replace in some areas—is known as *legacy* software. More mature service providers need to continue maintaining this legacy, building new market-demanded developments on top of and around it. It's neither efficient nor practical to start from scratch with every new development, which means old and modern software are destined to coexist.

This coexistence of legacy software with more modern products forces companies to perform balancing acts that don't favor the delivery of secure and resilient products. Furthermore, for particularly large companies, organizational complexities can lead to the creation of software with unnecessarily fragmented designs or environments where responsibilities might not be sufficiently clear among different teams [39]. In these companies, this complexity is compounded by external pressures from shareholders and regulatory bodies to precisely ensure that critical services are always available[40]. The fear of causing those dreaded outages we previously mentioned creates additional difficulties for patching and restarting services or applying maintenance with due diligence.

## Zero-Day Vulnerabilities

We've introduced the need for software to have regular maintenance to correct vulnerabilities. However, there's a type of vulnerability for which this regular maintenance is useless and for which no solution exists: these are called *Zero-Day* vulnerabilities.

**A *Zero Day* is a type of vulnerability unknown to the software manufacturer that may be actively exploited by an attacker**. Because of this, the manufacturer has "zero days" of lead time to publish a patch before malicious actors begin exploiting it. Conversely, a typical vulnerability is usually known to the manufacturer before malicious actors discover and exploit it. In these cases, the vulnerability can be kept secret until the corresponding patch is released, giving users more time to protect themselves and delaying its discovery for as long as possible.

Generally, we can summarize the main characteristics of a Zero-Day vulnerability as follows:

1. **Danger**: A Zero Day grants its possessor the ability to attack without victims being able to defend themselves. While there are Zero Days of varying severity, the term usually refers to those that provide the necessary privileges to manipulate software at will.

2. **Unknown Status**: Being a completely unknown vulnerability, it's impossible to fix or defend against it until it's either used or shared with the manufacturer for resolution. It's difficult to anticipate how many Zero Days might have been discovered for specific software or if someone with sufficient interest still has one at their disposal, unused.

3. **Difficult Response**: Even after being exploited for the first time [41], the vulnerability's lifespan until fixed by the manufacturer can be short or long, depending on many

factors. The impact generated by its use or the victim's notoriety can create enough noise to attract the attention of the media, researchers, and the manufacturer itself. Conversely, a more discreet use can extend its invisibility, potentially affecting more victims long-term. Therefore, manufacturers often release patches as quickly as possible once they gain knowledge of the vulnerability. However, due to the dynamics mentioned in previous sections, the application of the patch by all potential victims could be delayed for days or weeks. If the patch requires manual user actions, the window of exposure can be even longer.

**Zero Days constitute the most powerful and terrifying weapons in the digital world**. The more popular the affected software, the higher its value due to the volume of potential victims. There are organizations worldwide dedicated both to trading Zero Days[42] and storing them like nuclear warheads, waiting to fulfill their purpose—whether that be mere deterrence of enemies or complete devastation. In some of the most widely used applications globally, rewards from manufacturers like Google for reporting the most severe Zero Days can reach a million dollars[43].

The public figure reading this book will undoubtedly soon wonder what options they have to protect themselves against such a powerful threat. The short answer is: none. Fortunately, the extremely high value of these digital weapons, combined with the fact that their value drastically decreases after their first use, reduces the likelihood that someone will "expend" one of these vulnerabilities on us. However, despite the difficulty in protecting oneself from a Zero Day, not all is lost. In Chapter 23, we will explore some of the most drastic techniques for individuals significantly exposed to being victims of such advanced attacks.

## Malware and Spyware

One of the most frequent threats affecting software is malicious programs, or malware. This type of software doesn't always require vulnerabilities to infiltrate a system; it also exploits user oversights or ignorance to trick them into executing and installing it. Occasionally, users even grant additional permissions to malware when it requests them.

Encountering malware is relatively easy. Simply check your spam folder and observe (from a distance) the attachments that typically arrive in those messages. Their appearance varies, but it's always seemingly harmless: a file that looks like a PDF document but isn't, a compressed ZIP file containing an executable, or a link to a website asking you to install something additional to view content. Every day, a new trick is invented to convince users to run malware on their devices. In Chapter 19, we'll discuss some of these attacks and offer tips to avoid falling for these tricks.

Depending on how targeted the malware is, its objectives and impact will vary greatly. In most common cases, the malware may not know who you are and is simply seeking to gain control of your PC to use it remotely. Most often, your device ends up being used to commit other criminal acts (e.g., using your PC as a springboard to attack other users [44]) or simply to encrypt your files and demand a monetary ransom for them (what's known as *ransomware*).

Another type of malware, known as *spyware*, is usually more targeted at better-selected victims. Its purpose is to access all user activity on a specific device. Recently, this type of malware has gained notoriety thanks to *Pegasus*, a spyware that reportedly victimized several political figures and government authorities[45]. The capabilities of such software, once installed and granted the necessary permissions, are practically limitless. From tracking the user's location and logging everything they type, to sharing any information leaving or entering your phone with the tool's

administrators: direct messages, emails, and voice calls. This software can even potentially activate the device's microphone at any time to record conversations happening around it[46].

In Chapter 21, we will delve deeper into this type of software, how mobile devices have increased opportunities to spy on public figures, and in Chapters 22 and 23, we will discuss some advanced protection strategies.

## A Superman in Every Application

Until now, we've primarily discussed vulnerabilities. That is, flaws in the software's construction that allow malicious actors to manipulate it for their benefit. We've also discussed software specifically designed to do harm. However, not everything that can go wrong in terms of software security and resilience is due to these types of flaws or to malware. **In many cases, the abuse of normal software functionality is the cause of incidents.**

Almost all software, in one way or another, uses the concept of a user account. It's also common for different users to be able to use the same product simultaneously or in shifts. Some of them will inevitably need special privileges to configure the software and potentially affect other, less privileged users. These privileged users, commonly known as *administrators*, have the ability to configure the software at will and govern the accounts and data of all other users. As software grows in complexity and functionality, it may become necessary to establish different privilege levels for various users and even implement a separation of duties to prevent abuses of power. For example, in a corporate system for high-value payments, it may be necessary to prevent the same person who requests a transfer from also being able to execute it. By introducing a second user to validate the request, committing fraud becomes more complex, requiring the collusion of at least two people and ensuring process traceability.

Unfortunately, no matter how well-implemented a privilege system is or how exquisitely a separation of duties is designed among individuals—at the end of the day, there will always be at least one user with maximum privileges capable of configuring the permissions and separation of duties for all other users. No matter the type of software, it will always be necessary to trust one or more omnipotent individuals. It's inevitable.

The mere existence of such superusers exposes the software to total compromise and manipulation if an attacker obtains the credentials for one of these privileged accounts. In that case, it won't matter how well-designed the software is or if it has no vulnerabilities. It all comes down to an attacker's ability to log in as an administrator. In Chapters 13 and 14, we will discuss credential management and user account authentication processes in more detail. For platforms used by thousands or millions of users, access to such privileged accounts must be managed with particular caution, as a potential compromise would leave all application users completely unprotected. On the other hand, if the compromise of a specific account is a sufficient prize because it belongs to a public figure, we can say that the user's account itself will be equivalent to an administrator's.

Numerous incidents occur as a result of privileged user account theft. The ease and impunity with which attacks can be launched against specific individuals en masse mean that, occasionally, malicious actors gain access to credentials to use one of these accounts. When attacking a person rather than a machine, the usual approach is what's known as *Social Engineering*. These types of attacks attempt to deceive the victim by impersonating someone they trust or by confusing them into performing an action that benefits the attacker. As an example, some recent incidents may still resonate with the reader:

- In 2014, intimate photos of various celebrities (primarily actresses) were stolen and leaked online. These images included nudes and other private content. The root cause

of this incident was unauthorized access to the celebrities' *iCloud* accounts. Attackers used social engineering and *phishing* techniques to steal passwords and access the accounts [47].

- In 2020, Twitter suffered an attack where several accounts belonging to prominent personalities like Elon Musk, Barack Obama, and Bill Gates, or to companies such as Apple and Uber, were compromised to promote a fraudulent Bitcoin scheme. Hackers posted messages urging followers to send Bitcoin to a specific account, promising the sender double the amount they sent in return. Among the factors identified as the incident's root cause was the use of social engineering to trick certain Twitter employees and gain access to the company's internal systems [48].

While some authors cite the fact that almost all companies have no choice but to trust a handful of employees[49], not all is lost. Some modern services are designed in such a way that even an administrator of the service itself cannot compromise user data. This type of software is designed on the premise that it could be breached by an unauthorized person, a disgruntled employee, or even a law enforcement agent with a court order in hand. In Chapter 16, we will discuss some modern platforms that offer capabilities like end-to-end encryption, which prevents an administrator from reading what users send. Nevertheless, despite the existence of such services that claim to be hacker-proof or FBI-proof, we must maintain a natural skepticism towards these types of assertions. Building platforms that maintain the confidentiality and privacy of their users against a court order is not a simple mission.