2028 - Hacker-AI and Cyberwar 2.0+
(Securing our Future: Proactive Resilience through Separated Security Measures)
ISBN: 9798375252568

1. Why do we have Vulnerabilities in our Computers?

Why do we have vulnerability? Because we have attackers who ignore our good intentions. So we blame attackers, hackers, and all who want to make our lives less secure. Or we could blame the developers who put them in; they cause our problems with broken security systems. Well, why don't we blame the computer scientists and operating system designers who made it so easy to create and exploit vulnerabilities? This third answer option is what I would choose.

Computer soft- and hardware are prone to unauthorized misuse due to user negligence or other technical "issues". So there is another option: we could blame users.

But if misuse happens because of systems' vulnerabilities and that causes damage directly or indirectly to its owners and users, the manufacturer should be liable for these failures. But wait, software and computers are special. Who is responsible? That's not so easy to determine. There might be an embarrassment and bad reputation factor from security failures, but there is no legal accountability anywhere. Manufacturers fix the security and move forward. How can that be any different?

And is it right to blame developers for having created a vulnerability? No, I don't think that is fair. I think vulnerabilities are normal. If we acknowledge that, we could change the foundation slightly, resulting in less catastrophic outcomes from vulnerabilities. We could use the Internet for that.

There is no day without multiple reportings on computer/software vulnerabilities, on how easily attackers use exploits or damages created from data sabotages, ransomware or spyware. Often these reports are very specific in what applications or systems are affected and what the consequences of this security hole could be or have been. If we were lucky, the underlying issues were fixed; news reports assure us that an update is already applied automatically or should be installed manually immediately. Sometimes practical advice on what affected users should do if no fix is available or accept that problems would happen if they are being attacked. Because security is so broad, these reports explain possible data leaks into side channels or unexpected privacy or access-control behavior. Also, we can often read about the outrageous amounts of damage these vulnerabilities have created. We have accepted this as normal.

The main takeaway from security reports for the public, there is no reliable security and no protection from harm. All we can do is retroactively fix what

we are aware of. Marketing and sales pitches try to convince us if we pay for additional protection, it will help; it often does, but is expensive protection (really) more reliable?

Acknowledging Complexity

Asking why computers have these vulnerabilities sounds strange, even ignorant, given the huge complexity of computer technology. As a technologist and ex-C++ turned Python developer (for the laypeople among you, both are computer languages), I am actually in awe of how few security problems we have. I try to imagine how many lines of code have been written and updated. We have millions of applications and code libraries. Why do we have only a few security problems, or don't we look hard enough?

I am unaware that anyone knows how much software has been developed or how many different hardware platforms we have. Or how many different system platforms are still in use. I am also unaware that anyone is keeping track of how many standardized software solutions we have in an extremely complex international, multi-lingual, and diversified marketplace. And creating new solutions is the job of software developers. The more developer we have, the greater the diversity.

However, it is not easy to define software developers: are they writing code for websites, or are they engineers writing code to be compiled for a CPU? In a google search, I found numbers between 4 and 27 million. But the exact number is unimportant; even a factor of 2 or 5 on this estimate is irrelevant. We can assume that about 10 million people are directly involved with software development.

And how many software libraries and executables were developed and published? There are also scripts and macros relevant to user's security. The most deployed software is published in standardized packages; they are regularly used, (thereby) tested, updated, or frozen in time. I read a large number (100 million different software files) found over the years by antivirus manufacturers. If it is a tenth of this number, it would be too much to check out, but what if the number of "potentially" dangerous software (files/versions) is (much) higher? The relationship is easy: the more developers there are, the more unique software solution we will create and get.

When talking about software, I mean modifiable instructions applied to or used in hardware. These (instruction) processors could be microcontrollers or CPUs. They are all unified by an operating system (OS) taking (exclusive) control. Each soft- or hardware component impacts security, i.e., they could covertly be misused by attackers, but for us consumers, both soft- and hardware works perfectly together in serving us. And for the developers, technology is carefully planned and architecturally designed for concrete products. Products use standard (Swiss army knives-like) components capable of doing so much

more but then utilized in a very limited manner by the main components that deliver features to users. Even if not used, removing not used 3rd party hard or software features is considered ridiculous, too dangerous, and entirely unpredictable. It is much better and more stable to leave seemingly unnecessary / hidden features within the product - but this could come with a price, i.e., possible utilization of features never being considered within the context of the product.

Additionally, software is a publication with instructions designed to run on different hardware configurations. Furthermore, software amazingly interoperates with each other via standardized interfaces. When these standards or interfaces were designed, anyone considered that or thought carefully about the consequences of doing that. There are too many misuse scenarios, and many are handled in other layers. Any problem arising from its application, including security-related issues, is handed down to the context that is using it. The original designers could still claim that the utilization, even misuse, is not their responsibility, although the feature utilization got much easier. This is not a critique; it is simply how it is. However, some technologies have inherent limitations - their use requires special permits in certain contexts or countries.

In the meantime, thousands of technical standards are available. Or is it in the meantime ten-thousands? Each standard usually has different implementations. A lot of effort went into designing these standards, keeping them interoperable between different implementations, and making them unambiguous. However, some feature descriptions were less specific, and they might be interpreted by different soft- or hardware implementations slightly different. Over time these glitches are found and removed. So, over time, these standard implementations do get better so that they don't cause undue errors or damage. But still, they are not identical; they could contain surprises, even under very specific circumstances that an attacker could create if he knows about this vulnerability.

Developers often use extremely complex solutions only to get some simple features implemented quickly. These complex solutions are like black boxes; there is no reason to understand them or acknowledge that we could do much more with them. But from a security point of view, they offer dormant features that could be utilized under circumstances that will be discussed later. Right now, they are ignored because humans look for low-hanging fruits when they put their minds on accomplishing a goal, e.g., hacking a system.

The problem with security is that it results from software's complexity; this applies even in relatively simple situations. "The method of simplifying situations is a standard tool to gain confidence in the reliability of solutions." But this common sense statement, built from about 10 main terms, has many unmentioned assumptions based on what we exactly (could) mean with these 10 terms. It is even hard to develop a comprehensive list of things we should check before we commonly agree on this statement.

Based on well-earned experience, most developers are looking for surprises / problems in implementing other technologies. This negativity bias extends even to the tools they use within their development. All these problems and surprises are pre-security concerns; they are only relevant for regular bugs within the software. Detecting that software behaves unexpectedly is an art on its own, and its removal requires fixes so that users retain their trust in that software. Claiming or demanding that software is bug-free is unrealistic because of the inherent complexity of software solutions.

It sounds strange, but maintaining security is much simpler than keeping complex software bug-free. Security vulnerabilities are currently handled as software bugs, which is true, but there is a different quality in security - preventing intentional damage by some unauthorized assailant.

Who is Responsible for Vulnerabilities

I already mentioned possible culprits that could be blamed: attackers, developers, system designers, and users. However, the answer to who is responsible and thereby obligated to provide a fix is not easy.

Developers write software, and they test their work products. Developers should understand what problem their solution must face, and they should be educated and vigilant enough to avoid problems. Therefore, if someone could be made responsible than the developers who wrote to code. But this seems to be too shortsighted and superficial.

Developers include backdoors, and they are using them in their testing. Some of them are forgotten or insufficiently removed. Additionally, developers use, accept or define assumptions that could easily be faked or simulated by attackers (spoofed) when they circumvent expected methods of protection or authentication. Leaving these kinds of vulnerabilities in the code is certainly the developers' fault and shouldn't happen.

Educating themselves about the assumptions of protection or how it is being conceptually implemented are essential methods within the responsibility of developers when they implement or improve security. Additionally, some companies like Microsoft have developers working in teams of two, one writing the code while the other observing and being skeptical about possible problems. Still, software writing is a human endeavor with many flaws. Therefore, automated code review software will detect patterns that could lead to vulnerabilities.

The question we could ask is: is that enough? It helps, but vulnerabilities are still found independently. Unfortunately, code-review tools cannot be used to remove (all) vulnerabilities proactively. Some optimists may hope that Artificial Intelligence (AI) will improve these review tools and remove them all. Finding some vulnerabilities is achievable while removing all vulnerabilities is an entirely different challenge.

Most algorithms are written with expectations that they are not being misused unexpectedly. Moreover, it is hard for developers to know or decide what constitutes misuse; this depends too often on the context. The required data for misuse determination could be outside the algorithm's scope, and then how could the extended algorithm know that the received data are genuine and not being manipulated? Because the attacker knows what data the algorithm is using, it is for developers a waste to start an arms race with assailants on misuse detection.

Instead, security-related concerns are simplified into simple true/false decisions; thereby, more complex security problems are kept outside. Examples: Can a user read, modify or execute a file? Is a user a member of a user group associated with some of these rights? Does a user has permission to use restricted resources? Etc.

Ideally, there should be no methods to bypass or circumvent these restrictions. However, impersonating users with covertly stolen credentials is doing that, and it is comparatively easy. There are even three categories of attacks on how this can be done:

- users' negligence in protecting their credentials or passwords is making this too often too easy, or
- technical flaws help attackers to get their rights/permissions elevated or role changed or
- users were duped into providing access credentials unknowingly to the assailant.

Unfortunately, attackers could utilize too many technical or deceptive measures without following the required identification and authentication steps. Access control could be a single source of failure. The system will be in serious trouble if this security barrier is breached.

Because this book is not designed to educate readers on hacking, I mention only some high-level concepts that could do the trick for the attackers. The most direct and popular way is to gain sysadmin or system rights for a task, e.g., by starting it as a sub-task that would reduce inherent rights from its parent process. Another way is to interfere with or manipulate the code responsible for denying access. The last method is much more difficult but still doable.

The key takeaway should be that once the attacker gains sysadmin rights, no protection could hold him back from doing whatever he wants.

Unfortunately, too many methods give attackers these rights or permissions. Cybersecurity tries to get these methods under their control by using additional information to determine when it hurts/harms users and when it is done to enable or benefit users. If these rules are too flexible or generous, they could also be misused by attackers. The problem is that too much happens in the shadow, covertly, and users do not need to be involved by default when it

comes to security - because decisions/tasks must be automated. However, defenders have a chance when tasks are detected as "anomalies". But this anomaly detection is based on (fixed) rules that could be ignored or bypassed. Using AI and pattern learning, attackers could determine which rule can be manipulated by desensitizing the learning algorithm.

Back to the question: Who is responsible for computer vulnerabilities and should mitigate them? It seems to be a matter of opinion. I would propose: vulnerabilities should be accepted, but not their exploits. We already do this, but we are not serious enough about that approach. This implies that we should better focus on detecting and removing exploits of computer vulnerabilities with a combination of technical, organizational, and societal measures. Preventing exploits (the how is explained in later chapters) is the primary line of defense. Detecting and fixing vulnerabilities is the second line because it takes more time.

On the technical side, we already make it more difficult for attackers to find and exploit vulnerabilities in the first place. This protection is done with regular updates or patches, vulnerability assessments, and security tools and technologies, such as firewalls, intrusion detection systems, and antivirus software. Theoretically, proactively identifying and addressing vulnerabilities could make it more difficult for attackers to gain access and exploit them. But how can we proactively identify vulnerabilities? Some hope AI can do that. In reality, we are always in a race where we react to vulnerabilities and let many attackers get away with what they did with the vulnerabilities. We don't know who did an attack or who came up with the exploit without telling us about it. What if that is being changed? (More about that later)

By detecting and responding to attacks/exploits quickly, organizations could minimize their impact and potentially prevent attackers from achieving their goals. Additionally, organizations could implement measures to mitigate the effects of a successful attack via recovery plans and data backup systems to minimize the damage caused by an exploit. But these measures are better suited as redundant backups when the primary security has failed.

Are users responsible for vulnerabilities? On the organizational side, awareness of the dangers of vulnerabilities and exploits among employees, customers, and other stakeholders is already being raised. Users are warned not to fall for traps set out by attackers. By educating people, organizations hope to create a culture of security that reduces the likelihood of successful attacks. Many businesses implement policies and procedures to ensure that employees, contractors, and other stakeholders know their responsibilities and obligations concerning cybersecurity. Some people in cybersecurity may think it is good to have this culture of mistrust. I disagree; humans are then only forced to compensate for technical insufficiencies. Technology should facilitate sufficient and timely transparency and anomaly detection if there is a reason for mistrust; it should make our life easier and not unnecessarily more difficult and distrustful.

Then on the societal level, we should increase the legal and regulatory pressure on organizations and individuals who use exploits to gain unauthorized access to systems or steal sensitive information. By enacting and enforcing laws that criminalize exploits, governments can create a disincentive for attackers by increasing the risks of getting caught, punished, or excluded from those who could provide solutions or services. Some countries have these laws but catching the perpetrator is the bottleneck in making criminalization more deterrent. However, we don't want the good guys among the hackers discouraged from finding vulnerabilities.

Creating a vulnerability happens accidentally while developing and using exploits is done intentionally; therefore, making developers responsible for vulnerabilities or even blaming them for that is wrong. Developers are helping us to become more efficient with technology. Much better is to punish the use of exploits with a multi-faceted approach that combines technical, organizational, and societal measures to detect who is using it intentionally to benefit from it. Identifying and fixing vulnerabilities could reduce the incidence rate, but detecting who used vulnerabilities intentionally creates a more effective deterrence. Because then, consequences of successful attacks based on legal and regulatory pressure on attackers, organizations, and individuals can make exploits more difficult and less attractive.

Layers and Components

Layering and components are two key concepts in software engineering that simplify the development process. Layering involves organizing different parts of a software system into distinct layers, where each layer has a specific responsibility and communicates with other layers through well-defined interfaces. This reduces complexity by allowing developers to focus on one layer at a time and providing clear boundaries between parts of the system.

Components are self-contained units of functionality that can be easily reused in different parts of a software system. By breaking a complex system into smaller, modular components, developers can make the system more flexible and easier to maintain. Components can be developed, enhanced, and tested independently. It makes software easier to update. Also, we can improve or covertly modify individual system parts without affecting the rest (which is good and bad).

In both layers and components, interfaces are critical in how software interacts with each other. They define the boundaries between layers or components and what/how data and functionality are called. With interfaces, features are isolated from each other so that they can be developed, enhanced, and tested more independently. Components are highly cohesive, i.e., they contain only related features. This helps to make components more modular and easier to

understand, maintain, and reuse. Reusability is easier when components are designed with more generic concepts. Reusable code is more flexible in different contexts and applications, which helps developers to save time and effort when building new systems.

Access restrictions to interfaces in layers and components play a crucial role in protecting software systems. These restrictions isolate layers or components from each other, making it more difficult for attackers to penetrate systems. With reusability, security features are more consistent throughout the system rather when implemented on an ad-hoc basis.

While layering and components can provide security benefits by establishing clear boundaries and making it harder for less sophisticated attackers to penetrate a system, they can also make it easier for advanced attackers to reverse engineer the system and identify vulnerabilities when they bypass these restrictions.

Reverse code engineering, which involves analyzing the source code of a software application to understand how it works, can be used by attackers to identify vulnerabilities or make modifications to the system more easily. Once reverse code engineering is mastered by attackers, it is currently hard to imagine that any restriction could constrain them. On a positive note: there are methods to contain attackers that use reverse engineering - discussed later.

Could OS Security be Strong Enough to Protect Devices?

In July 2022, Apple announced for their iOS 16 iPhones a "Lockdown Mode" as an "extreme, optional level of security for the very few users" who may be "personally targeted by some of the most sophisticated digital threats" [https://www.apple.com/newsroom/2022/07/apple-expands-commitment-to-protect-users-from-mercenary-spyware/].

I am regularly asked if Apple could protect their phones from malware or spyware threats. Unfortunately, this question cannot be answered fairly because Apple could have and use some secret breakthrough technologies to deliver on this promise. However, based on the publicly available information, there are not enough details to answer this question positively. The measure they have to announce within the Lockdown Mode is very reasonable, but they do not seem sufficient.

There is a simple rule in security: A single vulnerability makes all existing security measures useless. Can they guarantee that there is not a single vulnerability exploitable by hackers? I have not seen that Apple is trying to provide that proof. Their new security approach is certainly based on solid engineering, but I have not seen anything indicating they have a breakthrough.

We know from click-free vulnerabilities in malware like Pegasus from NSO-Group. Pegasus is a commercial smartphone-based malware for governments

to spy on diplomats, human rights activists, dissidents, journalists, or legitimately applied to criminals. NSO sold this software not as a widespread malware to leave hidden backdoors on all smartphones but as a solution to be used against a few people on the radar of governments. Still, click-free malware can infect a device without the need for a user to click on a link or take any other action; they are based on exploiting vulnerabilities in system's software or operating system - the problem is not limited to Pegasus alone.

For iOS 14 and iOS 15, I saw several release notes with "leads to arbitrary code execution", which is an admission that the attacker gained sysadmin rights that they could have used to plant backdoors that are even usable after these security fixes were made. These security fixes remove known malware and prevent the vulnerability used by attackers in other exploits as well.

In anticipation of a more extended discussion within the next chapter, backdoors could be implemented in operating systems so that no software test could detect them. There are good reasons for having features that could facilitate this.

The biggest issue that I have with Lockdown is:

1. Would Lockdown detect malware already installed on the phone?
2. How well are (rarely used) exception-handling routines protected against exploitations?
3. Why does Apple think that only a few people (from exceptional groups) are prone to advanced malware?

I will return to some of these questions later. But I want to point out here other Pegasus-type malware might have left backdoors on phones so that attackers could use compromised devices much faster and more effortlessly. Additionally, the assumption that Pegasus software could only be used on a few users' smartphones is wishful thinking if nations are planning winnable cyberwars. The limitation of Pegasus was due to the huge amount of data that needed to be analyzed in the background manually. With more data reduction, automation, and anomaly detection tools, (total) mass surveillance via Pegasus-type spyware could already be a reality.

Without strong OS/device security, there is no protection against surveillance and misuse. Regulation would only handcuff the good guys to gain the knowledge exploited by the bad guys.