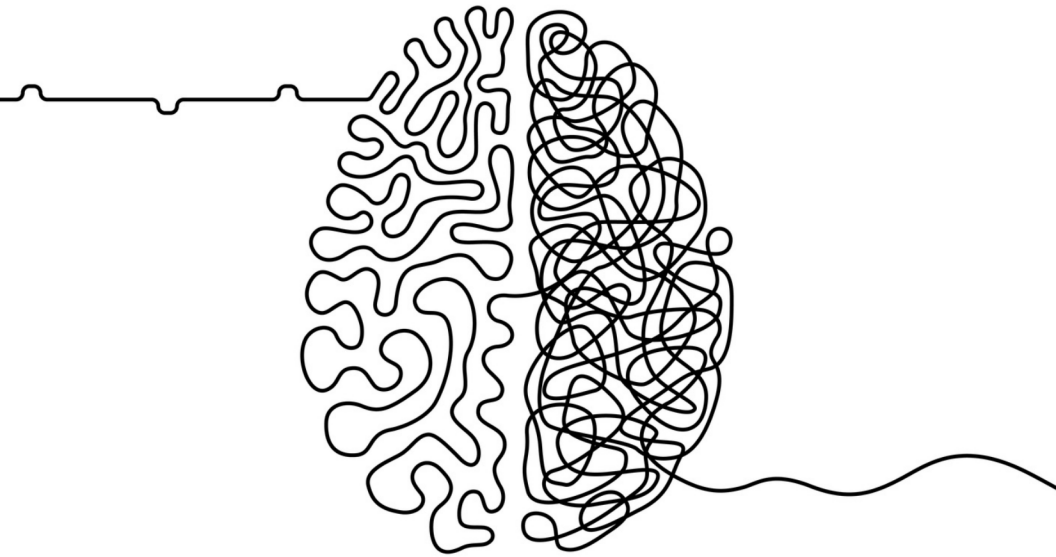
FRANCESCO CARLUCCI

The Hacker Mindset



How thinking like a hacker can improve your code, your coffee, and your life.

The Hacker Mindset

Published by:

NATIONAL LIBRARY OF ESTONIA
CATALOGING-IN-PUBLICATION DATA
NAMES: Francesco, Carlucci, author.
TITLE: The Hacker Mindset.
DESCRIPTION: How Thinking Like a Hacker Can Improve Your Code, Your Coffee, and Your Life.
IDENTIFIERS: ISBN 978-9916-4-1832-1 (paperpack) | ISBN 978-9916-4-1831-4 (hardcover) | ISBN 978-9916-4-1833-8 (ebook)

Contract: 0x0ce3ac6f0e1a1b3f29c0a4514275797f0f378a18
Chain: Polygon | Token: 1book (OBO) | Token Id: 1
https://1book.ee/nft/the-hacker-mindset

Cover image from iStockPhoto
Book design by Francesco Carlucci

To my grandfather

For giving me his place
on this planet,
too small for both of us.

The Stack

"I think there is a lot of beauty in looking
at something you do not fully understand,
it gives you the freedom to imagine."

Anonymous

0x00 / Intro

Do you know those kids who dream of becoming “real hackers” when they grow up? Of becoming one of those cool people that open up a terminal window, type a few commands, and hack a government or save the world from a nuclear threat?

Well, I was one of those kids! And guess what? I ended up becoming a professional hacker!

Are you asking yourself what makes me a “real hacker”?

There are many things. I’ll let you choose:

- I get paid to hack companies.

- I have discovered more than thirty CVEs—software vulnerabilities that I was the first person ever to find and disclose.
- I am a member of a few hacking teams, and I had to pass technical tests (hack stuff, basically) to join.

While all those statements are true, that is **not** what makes me a hacker. In fact, what makes me a hacker is my mindset.

Simple as that, if you are thinking about how you can use this book for purposes other than to learn how to hack or to understand the hacker mindset, you are already on the right path to becoming a hacker.

I will do my best to include all the key concepts I wish someone had taught me when I decided to start my journey, but this book is not intended for a technical audience. While I will cover some very technical topics, I will use a zero-knowledge approach, which is the approach you should keep using even after you become an expert.

In fact, as we will see in the next chapters, what we call *knowledge* is what often leads to security issues, vulnerabilities, money stolen, blackmailing, hospitals without electricity, and all kinds of hacking opportunities.

We will see how hacking is not a technical process but a **creative one**.

We will see how you don't need to be an IT professional to get an advantage from applying a "hacker mindset" to everyday life.

We will see what others do not see.

0x01 / The Learning Path

Human beings seem to be very comfortable with learning paths.

When you start going to primary school, they design a nice course of study to provide you with a solid foundation. They teach you math, language, science, and art and prepare you for the next step, where, guess what, you will find another path of study with very similar topics.

This goes on and on, up to bachelor's and doctoral studies, where they even use their course of study to advertise themselves. Even in private school (whether it's a master's or a specialization course), one of the most common mantras is that you will be "guided step by step to achieve success."

People think there is a formula for everything, that a person needs to know a list of things to become a doctor, a chef, an engineer, and so on, and this is true in some cases.

At the same time, let me tell you another truth: **the learning path is actually the problem**!

Spending our whole life jumping from one topic to another, always focusing on something very specific, only learning what we need to pass the exam or school assignment, has a side effect of developing in us a form of tunnel vision, where we miss out on a huge part of the spectrum.

There are countless studies to support this theory. For example, in 1999 the research psychologists Simons and Chabris created a video where students pass a basketball between themselves and asked viewers to count how many times the ball was passed. Some people got it right, others wrong, but most importantly, almost 50 percent of people failed to notice **a person in a gorilla suit** who appears in the center of the image!

Of course, this has nothing to do with learning paths, but it's strong proof that when our mind is too focused on a certain task, it loses many other evident realities, and it's easy to imagine that if we spend our entire life focusing on an exam, on a topic, on a specific goal, we develop an attitude toward tunnel vision, change blindness, and other forms of perception failure.

This form of tunnel vision has tragic effects on real life and is deeply related to the hacking world. In fact, many vulnerabilities do nothing other than exploit faulty logic.

Let me translate this into something more concrete for you.

I think all of us remember what happened on September 11, 2001. A group of terrorists hijacked commercial planes and crushed them against the Twin Towers in New York, killing thousands of innocent people. That was tragic, of course, and triggered many changes in plane security policies.

One of them was making sure that the pilot's cabin is inaccessible when closed from the inside, to prevent any unauthorized person from gaining access to the plane command control, killing the pilots, and taking control of the aircraft.

Brilliant! What could possibly go wrong with that?

While I am sure this was done with the best of intentions, it didn't consider how this feature can be used for malicious purposes.

On March 24, 2015, on Germanwings Flight 9525, the pilot (who apparently had previously been treated for suicidal tendencies) locked himself in the control cabin and intentionally crushed the plane with 150 people onboard, killing all of them.

He waited until his copilot went to the toilet, and locked himself in alone, in a way that nobody could open the door from the outside. This was possible because of the *safety* mechanism introduced after the tragedy of 9/11.

I apologize for having to bring your mind to those tragedies, but it was one of the strongest examples of tunnel vision mistakes I know. After this new tragedy, someone even proposed solving this new problem by enabling planes to be remotely controlled . . . **please don't**.

My point is, a limited vision of a topic, feature, or product can cause many problems, fortunately less tragic than the above examples.

When I was twenty-one, my parents helped me to buy a new car, a nice R. (sorry, I had to redact the brand name). It wasn't really our first choice, but our dealer recommended it, and it was big and shiny, so we closed the deal. The car also came with an incredible new feature (for the times), a smart key!

This key was incredibly smart and allowed the driver to open the car and turn it on without taking it out of his or her pocket. Wow, what a great tool! My friends were impressed. It was a shame that it worked even if I was around the car and someone else opened the door, pushed the start-up button, and drove away with my car. My friends did that many times.

R...EDACTED's engineers were probably so focused on the user experience, on the dream of never having to take the key out of your pocket and enjoying this "keyless car", that they lost focus on the security aspects. In fact, my car was stolen after a couple of months, making me very sad.

I spent this whole chapter talking about learning paths just to make the point that to develop a hacker mindset, you have to overcome this schema. You must learn how to free your mind from patterns and try to think out of the box. You have to allow your mind and your thoughts to be brave.

Our learning path will have **no learning path**.

0x02 / Non-Linear Learning

It's a hard journey, I have to tell you.

And it is also very difficult to explain and teach, so difficult that I do not feel suitable for it. The good news is that I don't have to be.

Learning without a learning path is also called *non-linear learning*, and it is a contradiction by definition because it's a method for learning without a method. I will not go deeper into this concept because I have already given you a broad idea, but you are welcome to go and learn more about it if it's interesting or valuable to you. Otherwise, you now know enough to move forward.

This way, you are already learning non-linearly.

I am not a book writer, but I am writing this book, which actually is my second one (if I can finish it), and I am learning it by doing, relying on the hundreds of books I've read, the knowledge I've acquired during my technical research, and the few blog posts I have written over the past few years.

I know nothing about writing style or narration techniques, and English is not even my first language, yet the book is taking shape.

This is my way of learning to write a book, in a non-linear way.

When it comes to hacking, this concept is even more important because that is the hacker superpower: mainly using the information available to anyone and recombining it to achieve an outcome that **no one achieved before**.

You can open your laptop and take all the hacking courses available on the Internet to learn the following:

- The basics of networking
- How to reverse engineer a binary file
- How to deep dive into XSS, the most common web vulnerabilities
- Encryption and breaking encryption

And I could go on forever using a wide array of fancy words that are very interesting if you want to become a computer hacker, but these areas of study will not help you develop a hacker mindset,

that intangible skill that allows you to get more from the information you already have.

Non-linear learning involves triggers, facts, and episodes that make you fall in love with something and chase it. It's called *non-linear* because it looks more like a web, a web made of random connections, a web capable of reshaping itself and coming up with brilliant ideas out of the blue.

And if you feel confused now, that's great news: The non-linear mind is never afraid of chaos—it embraces it.

If you don't feel ready to embrace your chaos, be ready to embrace mine, because this is what you will find in the next chapters.

0x03 / Doceo

Every hacking book that deserves respect starts with the infamous "first hack." Oh . . . am I following a pattern now after several pages of how patterns destroy our creativity and vision? Not really, simply because this episode happened eighteen years ago and I realized it was my first hack only one month ago when I started writing this book.

I attended the Scientific High School as a young student, and I was already pretty good with computers—not incredibly brilliant, but certainly above average.

One day during our philosophy class, someone interrupted the lesson to tell our teacher that, unfortunately, there was a problem with the school computer (we only had very few computers at

school at the time) and her "precious" web magazine articles were lost.

She was very angry because she cared a lot about that project, and I saw an opportunity there.

I remembered what my friend Nicola (he was the real genius with computers) had told me: "To really erase data from a hard disk, you have to delete and overwrite it multiple times to be sure to not leave any trace." Mmm . . . could this mean that my teacher's data was recoverable? I'd take the risk!

I raised my hand and told her I might be able to recover the data. Her answer was: "So why are you still here? Go!"

I went to the library, where one of the school's main computers was located.

The machine was apparently dead, crashing at the start-up with some fatal error. I restarted it in *recovery mode*, a procedure that only loads the few components necessary to start up the machine and usually works to fix this kind of problem because it doesn't load the other pieces of software that are usually the root cause of the crash.

It worked, a nice start! Then, I gracefully rebooted the machine, downloaded a data recovery program (which is a tool capable of

recovering files after they've been deleted or corrupted during a hardware failure), and started a scan.

Side note: It turns out that permanently deleting a file from a computer is harder than we think, and this is why the FBI, CIA, and similar agencies can pull out years-old files from confiscated phones and laptops.

To my great surprise, the program worked like a charm and I was able to recover all my teacher's files! Happy and proud, I came back to my classroom and gave her the good news. That situation earned me my teacher's respect and higher grades in history and philosophy.

But, no, that wasn't the first hack I wanted to tell you about in this chapter. Even if in a certain sense I *hacked* my grades in two subjects I was not very good at, the story doesn't end here.

I did all the work on the school's computer under the supervision of the librarian, who noted that I was handy with the computer and asked me if I could also install some card games (solitaire, blackjack) on that machine to help him to enjoy his time.

Don't ask me the reason, but I said yes and asked if there was something else on that computer that needed attention. His answer was, "The only thing I can think of is the school email."

And that was my stunt. The email client was working, but I told him it wasn't and I needed the password to set it up again.

"Sure, the password is **doceo**," was his answer.

From that moment on, I owned the school's main and only email address, and I was even able to manage it from home. In hacking terms, this is called *social engineering*, when you trick someone into revealing sensitive information you are not supposed to have. Social engineering is one of the most powerful attacks and the one I like the least, because it exploits people, which usually are the weakest link in a chain.

Checking the school email with my morning coffee became my routine, but I never caused any real damage, and I never sent an email impersonating the school. It just made me feel excited.

A few times I deleted some emails before the school staff could read them, I downloaded the blank template of the report cards and gave it to some friend that needed "an adjustment," and I even checked the email the morning of the final exam hoping that the exercises were sent by email . . . but they weren't.

I know that what I did is illegal, and it's also a very bad thing to do, but I was just seventeen. I made a mistake and I am sorry.

I graduated with average scores and went on with my life, but about a year later I found in my bedroom the piece of paper where

I had noted the school's email and password, so I tried to enter again.

Of course, it still worked!

0x04 / Hacking with Time

***Warning**: This is a technical chapter, and it contains code. Skip it if you don't like code, especially if you don't like bad code.*

As much as I would like to go ahead with my bio, I don't think it is the reason you are reading this book. Plus, I have something much more interesting we can talk about: **time**.

In my opinion, time is one of the most fascinating concepts ever. Scientists and philosophers have been trying to explain it for millennia, approaching the problem in many different ways and even arguing its existence.

In fact, there are theories proposing that time does not exist, that time exists but doesn't flow (instead, we are moving through time),

and that the past, present, and future are happening in the very same moment, in parallel.

Today, we will see how time can be a powerful tool when it comes to hacking, so harmful it can force a company into bankruptcy.

Just a few weeks ago I was reviewing software for a company that organizes big music festivals. There was a feature allowing people to upvote an event proposal, to help organizers to better understand the trends and the interest in a certain type of artist or music. That immediately rang a bell for me, and I asked their lead developer how this feature was implemented.

He answered that they simply store the total number of upvotes as an integer (e.g. 576), and when a new one comes in, they sum it (+1) and overwrite the total.

Straightforward, right? Wrong! That is exactly the type of implementation vulnerable to race condition attacks.

Time is a factor that you have to consider when writing a certain type of process because even the simple summation that I described above takes some time to execute. Let's say, for the sake of simplicity, that it takes ten milliseconds. Imagine that an upvote comes in, and then another upvote comes in after just two milliseconds, and the first one still hasn't completed the execution, so the total is not yet updated. Both upvotes will have the same

starting amount and overwrite the total upvotes number with the same (wrong) value.

This would be the timeline of events:

- 1st upvote comes in and retrieves a starting total of 576.
- 1st upvote starts the sum operation: +1.
- 2nd upvote comes in and retrieves a starting total of 576.
- 1st upvote updates the new total to 577.
- 2nd upvote updates the new total to 577 as well, instead of 578.

Assuming that the above makes sense, you may be thinking losing some upvotes on an event is not that big of a deal. That's true, but imagine the same faulty logic applied to a refund-management system on an e-commerce platform, or to a money transfer system of an online bank. Can you see the potential damage?

If not, let's put it this way: In 2014 a Bitcoin bank (Flexcoin) lost more than $400,000 because of a hacking attack exploiting a race condition using this very same technique. And with the new smart contract technologies (transactions registered on a blockchain), defense against race conditions becomes more and more important.

As you can see, sometimes hacking a system is not even necessary to write complex malicious code. All it takes is sending a legit HTTP request at the right time. This does not mean it's easy, but it's so

interesting—at least for me—to see that the weapon here is not some malicious code or a compromised computer, but time.

We are able to **weaponize time**.

Beating Math with Time

Race condition is not the only bug taking advantage of time to exploit a system. There is more fun to discover.

This code is one of the most common implementations I come across while doing code reviews:

```
<?php

if( $_GET[“token”] == “pass” ) {

       // execute program here

}
```

Don’t be afraid to try to read the above code. It is very simple even if you are not technical.

$_GET[“token”] (which sounds a bit like “get token”) means we are retrieving a value named “token” inlined directly in the URL, which will be something like this:

https://acme.com?**token=something**

Using the PHP programming language, $_GET[“token”] will give us the value of that token key, and the program will compare the value against a static string defined directly in the code. If there is a match, the program will go ahead with the execution.

This is often used to execute a web page that is supposed to be private or protected, or which displays private information accessible only to users that know the value of the token.

Believe it or not, when I mark this as a security issue, even seasoned programmers with years of experience fail to understand the problem(s), and some of them complain, stating they use a “mathematically safe” token.

The argument sounds like this: “We use a 64-character token, and it would take hundreds of years and billions of attempts to crack it!”

And my answer is: “Sure, but who said we need all the attempts to crack it?”

We certainly have a time issue here, and potentially worse ones with nothing to do with brute force. Let’s look at the timing issue first.

When the PHP language compares two strings, it does so character by character, meaning that **aa** = **bb** will fail on the first character

combination because **a** ≠ **b**, but **cc** = **cd** will fail on the second character, because **c** = **c** but **c** ≠ **d**.

This makes a lot of sense, because how can two strings be equal if one of their characters differs? So the comparison fails as soon as it encounters a different character. And here is the moment where our hacker mindset kicks in: We can evaluate the execution time of the operation **to exfiltrate the token piece by piece**. If the character is wrong, the operation fails **sooner**. Then it takes us less time to find the correct character, and we can move on to the next one.

With this technique, we can force a password with only a few hundred attempts over a few minutes, instead of years.

Isn't that amazing? We can hack an access token using nothing but time.

To protect against timing attacks, the PHP language has a built-in function, hash_equals(), to perform a *timing safe string comparison*. So, would using hash_equals() make the above code secure?

Not yet. It would only provide time safety, but there are other ways things can go wrong. That is why I decided to use this code snippet to discuss security: It's one simple line with so many ways to create problems.

Missing the Big Picture

Remember when we talked about tunnel vision? Considering the above example secure because we fixed the time issue would be a tunnel vision mistake. As a hacker, you need to look at the big picture and always keep in mind all the players in the game.

In this case, we have a PHP application that usually gets executed by another program we call a *web server*. A web server is designed to receive incoming requests from another machine (the *client*), process some code, and return a response.

Most of the time, the web server is designed to log incoming requests to a text file, and there are parts of the request that are logged and others that are not. URL parameters fall in the ones that are logged.

Back to the security discussion, if I try to ask the following question: "Are the people in your organization that have access to the web server logs supposed to access those tokens as well?"

The answer is usually no, and that's a security problem. But we have more.

Web pages often include what we call *third-party scripts*. Think of them as programs that are included in the application but retrieved from other servers. Some very popular ones are Google Analytics and Facebook Pixel.

In some web-server configurations (depending on the CORS policies), the URL is disclosed in full (including the parameters) with third-party servers.

Again, if I try to ask the following question: "Are the people who have access to the web server logs of those external companies supposed to access those tokens and consequently be able to access those private pages?"

The answer is a **huge no**, and here is when the security issue becomes very clear to **anyone**, technical or not.

I apologize if the above examples have been too theoretical, but if you need a practical example to understand the impact, think of a password recovery mechanism.

If you have a third-party script in your password reset page and you are using the wrong CORS policy, every time a user tries to reset his password, someone in another organization would be able to take over his account, access private data, impersonate the user, and so on.

How often does that happen? In my experience, I find these kinds of security holes every time if I am the first one to review an infrastructure, and this exposes the fact that some programmers are not trained for critical thinking. This is fine; I also used to make

these kinds of mistakes, until I decided to look at my code from another perspective.

This is why you need to train your hacker mindset to be a hacker, programmer, and—as we will see later in this book—a barista too.

0x05 / I'm in Your House

We have seen this scene in almost every action movie: someone hacking into surveillance cameras to spy on someone else. One day, my girlfriend Arnisa asked me if it was as easy as they make it look. What I did later left her speechless, and also scared.

I opened my laptop and pressed a few buttons, and after fifteen minutes we were looking at a live stream of images coming from inside strangers' houses. These were not completely random people, but those living within kilometers of our home.

How was this possible? Was there any kind of trick? This is what we will discuss in this chapter.

I promised early in the book that this would not be the usual "how to hack something" book, and I will stick to my word. What we

will see is the mental process that could allow anyone to hack cameras like in a movie.

But first, we need a tiny drop of prior knowledge about IP addresses and how "things" are connected to the Internet.

When you open your browser and type in a website URL, like "www.duckduckgo.com," for example, you are not directly connecting to the website. Instead, the browser queries a DNS server which translates the URL into a numeric IP address (like 104.21.46.149), and only after the browser has the site's IP is it capable of connecting to it to make the navigation start.

But a website's IP address does not belong to the website. It belongs to the machine that hosts it (the server). In layman's terms, every device connected to the Internet (or any other network) has an IP address.

Surveillance cameras are also usually connected to a private or public network, to allow people to control them remotely. Think of companies offering mobile apps to control your home or your dog in real time. That's basically your phone connecting to your camera over the Internet and showing you the images.

For most professionals, it's obvious that those interfaces need to have a form of authentication, to make sure that only authorized devices or people are allowed to access the cameras. But apparently,

many companies back in the days believed that if you made your cameras accessible on a private IP address, it would be safe enough.

After all, who can possibly find an IP address among billions of possibilities?

Again, a huge tunnel vision mistake. The IP address system has a peculiarity that is very appealing to hackers: **It's predictable**.

It can go from 0.0.0.0 to 255.255.255.255, at least in its v4 version, which is what we used until a few years ago. (We recently started using a new version—IPv6—as we were running out of IPs due to the massive number devices connected to the Internet nowadays.)

So what *does* prevent hackers from simply mass scanning every available IP address within that range to discover any kind of weird machine? Technically nothing; legally . . . it's another story. And when I say *scanning*, I mean trying to access any available IP directly and analyzing the responses to detect, in our case, open security cameras.

And this was my magic trick. I wasn't hacking a specific camera (that's illegal and much more complicated); I was "just" accessing IP addresses, scanning their ports, and parsing responses to detect a popular type of camera:

```
RTSP/1.0 200 OK
Server: ***cam RealServer/V1.0
```

At the time, there were tens of thousands of open cameras only for that specific model, an unbelievable number.

This is not a movie anymore; this is real life where with little technical knowledge you have a window on the world passing through thousands of unprotected surveillance cameras.

If you are now wondering what else can be found with the same technique, you are already thinking as a hacker, good job. And if you go down that path, you will discover **crazy-interesting things**.

Our printers are now connected to the Internet, our watches, our coffee machines, even our glasses sometimes . . . do you really think this comes without unintended consequences?

0x06 / No More Coffee

I feel it's time to stop talking about IPs, PHP, and race conditions.

It's now time to talk about Çıralı, one of my favorite places on our beloved planet. Çıralı is a small village in southwest Turkey surrounded by mountains and nature, with a beautiful stone beach, crystal clear water, a peaceful atmosphere, and tons of friendly cats. And as if all this is not cool enough, hiking up to the mountains will bring you to Mount Chimaera and its eternal flames, a magical place.

I spent almost one month last year in Çıralı, working remotely from my laptop, which is something I regularly do from all around the world.

Our favorite coworking cafe was Yedi Cafe. In Turkish, *yedi* means "seven." It was a chill and colorful place with yummy food and a few other digital nomads working there, which never hurts, playful cats, and even some hens.

One day right after we sat at our table, the owner approached us, apologizing and telling us they couldn't make our coffee because of a global blackout affecting the whole city of Antalya. We answered, "No worries," and asked if we could still stay to work. The answer was, of course, yes; people are very nice in Turkey.

An hour passed by, and my partner Arnisa really started to crave coffee, so we asked the owner if we could at least have Turkish coffee, aware of the fact that you don't need electricity to prepare it, just a flame and a gas cylinder. He was happy to prepare a nice Turkish coffee for us, and after ten minutes, he was back with our fragrant steaming cups.

In no time, people from almost all the other tables also asked for Turkish coffee, so he started making it for everyone.

Why am I telling you this story? Because this is a perfect example of out-of-the-box thinking, which allowed a cafe in the middle of nowhere to keep the business running with no electricity.

Of course, the owner could have had this idea himself, but he probably was too focused on the fact that everyone (mostly European) usually ordered espresso or cappuccino, which both

needed electricity. But fortunately for him (and for everyone), we gave him a winning suggestion.

This is hacking for me, taking another path and seeing what happens, taking advantage of a blackout to learn that your customers also enjoy traditional Turkish coffee, solving a problem in an unconventional way.

The Invisible Exhibition Stand

Speaking about problems, I had an interesting one at WordCamp Niš in 2019. WordCamps are locally organized conferences covering everything related to WordPress, the most popular content management system for websites.

I attended the event as a speaker to talk about (guess what) software security. I was also contacted by a company—a sponsor of the event—which wanted to promote their translation software. They asked me if I wanted to handle their exhibition stand to advertise their product to WordCamp attendees. It was a strange request because I am not a promoter, but I said yes because I am always open to new experiences.

On the day of the event, I was supposed to get a stand, flyers, pens, T-shirts, and all the necessary swag that companies distribute at these events to catch the attention of the public, but unfortunately, due to an issue with the Serbian Post, the package never arrived. I was disappointed and called the company to ask what I was

supposed to do now that I had nothing to use for the promotion. They just replied, "Sorry about that. Please do what you can."

There were just a few stands at the event, but they were all packed with exciting gadgets, and I had nothing. So after some thinking, I took some sheets of paper, glued them together, and created a medium-sized banner where I wrote the following phrase all uppercase: **We don't have swag, we don't have T-shirts, but we do have incredible translation software!**

I put the banner on my empty stand and opened up my laptop on the homepage of the site, so people would understand I was able to demo the software. Then I put my best smile on my face and waited. The response was unbelievable.

People started reading my improvised billboard and smiling, empathizing with me because of how poor my setup was, and even taking pictures of the unconventional stand. In a few minutes, lots of people stopped by and asked me about the software. I started doing live demos, which is actually the best way to experience a software product.

It was a success, and this was my creative way of compensating for the missing advertising material. This is yet another example of how lateral thinking becomes useful in many non-technical situations, and how you can hack your way out of an unpleasant situation.

0x07 / It's Not About Money

While it's true that a hacker mindset can pull you out of difficult situations, it can also get you into trouble. And, of course, I've been there more than once.

A couple of years ago I was quarantined at home, just like the majority of people around the world. I wasn't sick with Covid-19, but I was traveling back from the United Arab Emirates, which at the time was considered a high-risk zone by Italian law and required a ten-day quarantine period for people returning from that area.

With plenty of time on my hands, I spent half an hour at the end of my day checking my HTTP logs. For the untechnical reader, it is worth clarifying that when you visit any website, your computer (aka the *client*) connects to the other servers—where these websites are hosted—sending them requests and parsing their responses.

The server on the other side receives those HTTP calls and performs actions in response to the client inputs.

This exchange happens on the *application layer*, which is only one of the several layers machines use to communicate with each other. (If the concept sounds appealing, you can learn more by searching "the OSI model.")

Having full control of the client (it's your computer, right?), you can make it log all the HTTP calls and responses and store them as plain text files. And why do I do that? Because in my opinion it is the best way to understand how a web application works from the outside, without having any knowledge about it (the zero-knowledge approach)—and doing it regularly is a great way to develop your programmer's eye and learn new skills by seeing how other software works.

When I was quarantined, I bought my groceries from a mobile app with home delivery. One day I noticed something suspicious in this app's HTTP requests. This is when your hacker mindset gets you in trouble. I started suspecting that my own data (as a customer) wasn't properly protected, and (as a hacker) that I might be able to access all the other customers' data, **all of it**.

The app contained home addresses, phone numbers, email addresses, encrypted passwords, orders, and loyalty card numbers. Most importantly, the app asked people if they had Covid, so the delivery staff could take the proper precautions—and this was saved

in the order notes, which were also accessible. To make everything worse, this supermarket chain was huge, with stores across all of Southern Italy and several hundreds of thousands of people registered on the mobile app.

The urge to verify if I actually could access everyones' data was too strong, and I could not resist it, but I was fully aware of the fact that if I tried to approach other people's data, I would commit a serious crime punishable with big fines and several years in prison, according to Italian law.

I was in front of one of the most classic hacker paradoxes: On one hand, I knew that the ethical thing to do was to verify the security vulnerability, privately inform the company, and allow them to remediate it. On the other hand, this was a huge risk because the company might not react well and might even initiate legal action against me, so the safe (and egoistic) thing to do was to ignore my finding and go ahead with my life.

I decided to be brave and do the right thing because whatever the risk was, I could not leave almost half a million people with their data potentially exposed.

As the first step, I created another account on the mobile app using a different email address and tried to access my own data with success, so at that point, I was sure I could access any user account on the app. Then, I did some research to find the most appropriate company email addresses to report this matter. Finally, I sent out a

professionally written vulnerability report, including a video POC (proof of concept) explaining how I was able to hack into the system and access all the data.

Half an hour later, I was on the phone speaking with their head of security. He was kind and grateful, thanking me for my professional approach and reassuring me they would fix the vulnerability and secure the app immediately, which they did. One week later, I received a follow-up from a company representative, asking me if we could meet in person, and I said yes.

I have to admit I was a bit suspicious about this meeting. They said on the phone that they just wanted to get to know me better and see if we could start cooperating together to improve their security, but they gave me an appointment not at the company headquarters, but in one of the supermarket stores, and this looked weird to me.

I decided not to go alone to the meeting, and my partner Arnisa offered to come with me. The encounter was surreal. It was a quick ten-minute stand-up meeting **outside the supermarket**, during which they gave me an envelope containing three symbolic gift cards worth €100 each to show me the company's appreciation for my report. They also told me a bunch of absurd stories about how they were thinking that some cybercriminal had stolen my identity and hacked the app to blackmail them, or that I was working with the police, and other movie-like scenarios. At the end of the

meeting, they told me I could send my CV and it would be fast-tracked, but I politely declined.

On my way home, I was disappointed. The vulnerability I reported could have caused the company millions in financial damages in fines for GDPR infringement, could have been used to blackmail them, as they well imagined, and most importantly, could have caused huge reputation damage if their customers had been informed about the data breach. I saved them from all this, and they didn't even receive me in an office with a chair and a nice coffee.

I shared my thoughts with Arnisa, and her answer opened my eyes: "You are a hacker!" she said. "Companies don't welcome hackers into their offices with a nice coffee."

"Maybe you are right, but, still, this is unfair," I thought.

PS. One of those gift cards is still on my desk, to remind me that I don't do what I do for the money.

0x08 / Cybercriminals

Providing a straightforward definition for the word *hacking* is not an easy task, in my opinion. In general, any definition is something that sets a perimeter around a concept and restricts it—and we are learning that hacking is exactly the opposite of this, going beyond the definitions, beyond *the known*. But if you came to read this book with a malicious idea of what a hacker is, I wouldn't be surprised.

Most of the time hackers hit the news with facts related to illegal actions, and this probably gives the word hacker its negative connotation. However, hacking is not always seen negatively, and I feel we have to clean up what hacking really is about, when the word is misused, to better embrace the hacker mindset as a tool and not as a gun.

I've been writing this book during February of 2023, while traveling around Morocco, a mysterious and inspiring country. While there, I checked the Italian newspapers from time to time, mainly because my family and many of my friends live there, and this news caught my eye: "Pro-Russian Hackers Attacking Italy in Response to the Premier Visit to Kyiv." Titles like this have been abundant during the Russia-Ukraine conflict, and most of the time articles mentioned DoS/DDoS (denial of service/distributed denial of service) attacks.

Of course, I don't know who was behind those attacks, and neither do the journalists), but I feel comfortable enough stating that if we are talking about DDoS attacks, they are definitely **not coming from hackers**.

A *DoS attack* is a twenty-year-old technique that consists of flooding the victim server with a huge amount of traffic to shut it down, crack it, or at least slow it down significantly. The final goal is to cause service disruption. Nowadays, it is not as big of a threat anymore, meaning that most serious companies have access to the technology to mitigate these types of attacks without any particular effort. In short, if you really want to hurt a company or government, you don't use DDoS, because even if your attack succeeds, it gets immediately detected and mitigated within minutes or hours in the worst scenario. This simply doesn't sound like a hacker job.

Don't get me wrong, DoS/DDoS caused lots of damage years ago. The person who invented it was probably a hacker, and every time someone comes out with a new attack vector (a new and better way to carry out the attack), that is pure hacking and requires a powerful brain. But this does not mean that every time a DoS attack is performed, there are hackers behind it, the very same way every time a bullet gets shot, there is not a mechanical engineer pulling the trigger.

Mass media tends to blame hackers for every attack that runs on a digital path (probably just to earn some more clicks/views), but this causes lots of damage to the category, which in reality includes countless people who want to use their skills to actually do some good, just like I did with that supermarket.

If someone deployed hackers to destroy someone else's operations, the hackers would probably infiltrate their network (silently, without being detected), encrypt all their files, and make everything unusable and unrecoverable, forever. Or even more impactful would be to stay in the network and just spy and grab intel, to anticipate the enemy's moves.

The Hacker Who Saved the World

In 2017, the world was facing real danger because of malware called WannaCry. This virus, just like I described in the paragraph above, infected as many machines as possible, encrypted all their data, and then displayed a pop-up where the cybercriminals asked for a

ransom in exchange for a key to decrypt the files and get your precious data back.

The operational style earns this kind of malware the name of *ransomware*.

That was not the first time we saw a threat like that, but WannaCry was different, for the simple reason that it leveraged a widespread unpatched Windows vulnerability and was capable of spreading at the speed of light. In a few days, we started to see hospitals with no access to patient data, police stations blocked, gas stations shut down, and machines infected in more than 150 countries. The world started to recognize it as a global threat, people started to worry, and it was all over the news.

Security experts all over the globe were discussing how we could stop it, and the answer arrived from a twenty-two-year-old hacker: Marcus Hutchins. Yes, a single person stopped a global threat. What he did is probably what everyone else was doing: Reverse-engineer the malware, meaning that he tried to reverse the virus binary code (ones and zeroes) into human-readable code.

Reverse engineering malware is not an easy job to do, but for Marcus it was probably not the first time. In fact, he was able to make the miracle happen and found a kill switch hardcoded into the malware itself. A kill switch is an emergency procedure capable of shutting down a program or a device, and WannaCry's kill switch was as simple as checking a website's existence: If the

domain did not exist, the virus would keep infecting; on the contrary, it would stop!

And the domain name was also hardcoded into the code: **iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com** (super complex to make sure nobody would register it accidentally). All Hutchins had to do was to register that domain name, and that stopped the execution of the virus on millions and millions of machines, giving the world the time to patch the vulnerable systems and prepare the defense for similar future threats, which of course arrived.

A couple of months later, the hero Marcus Hutchins was arrested in the United States on the charge of contributing to the production of other malware, years before WannaCry.

Marcus is a hacker, of course, and apparently he has also done some criminal things in his life, like many of us. But *hacker* and (cyber)*criminal* are not interchangeable words, and this is very important to understand.

Every time you use your brainpower to perform something in a smarter way, achieving exceptional results in a way nobody did before, you are hacking. If you want to use your hacker mindset for good or bad, well . . . that is up to you.

0x09 / CVE-2022-0482

I didn't save the world as Marcus did, but I also came across something dangerous, and despite all my efforts, the threat is not yet completely gone.

It was still "Covid time," more precisely the period when everyone was looking to get a vaccine, hoping to get back to normal life. In Italy, the government decided to allow private pharmacies to administer the vaccine to speed up the process, and as a result, pharmacies' phones were on fire to get an appointment.

One of my customers—and a dear friend—called me asking if I could develop an appointment management system in order to route those people to an online platform and automate the assignment of appointments, availability checks, and so on. It was a very interesting job that I would be glad to accept, but there was a

constraint: I had to act fast. The situation was unsustainable, and they needed the system in three weeks.

"Three weeks? That's crazy!" I thought, but I had to find a solution.

Of course, I could not code it from scratch in that amount of time, so I started researching for libraries or tools that could provide me with a half-backed system that I could finalize and touch up to meet my client's needs. After some googling, I came across this nice PHP library—Easy!Appointments—which seemed to be lightweight and clean, perfect for our needs. It was open source, and many companies and even governments were already using it to manage online appointments of all kinds, and that was pretty encouraging.

I told my client we could use it to speed up the whole process, and they agreed, so I started working on it.

Especially for the non-technical reader, it is worth pointing out that in the modern development life cycle, it is common practice to rely on third-party libraries to implement a project. When we do this, we usually consider external libraries as safe . . . until an issue gets discovered. Then we update them (sometimes). For example, if you wanted to create a blog, you would probably use WordPress, and you would consider it safe. The same goes for Magento to create an e-commerce website.

But in this case, I knew that the system would contain thousands of pieces of sensitive data, so I decided to at least do a quick review of Easy!Appointments. The time was too tight to perform a full audit, so I decided to use my hacker instinct and check the parts that theoretically would have been more error prone. I reviewed the files and folder structure, and my attention was immediately caught by a file named Backend_api.php—because when we talk about APIs, we usually have data moving back and forth between systems, fertile ground for vulnerabilities.

Reading the first hundred lines of code was enough for me to understand something was dangerously off: The responses containing the user's data were built **before** checking the authorizations. Going down that path, I was able to hack the system and prove that any user, with a single web request, could access all the data contained in the application. And most importantly, all the applications on the Internet built on top of that framework were vulnerable to the same attack.

Now, if you are wondering why users' data is so desirable for cybercriminals, consider that bad actors are almost always chasing money, and in the modern world, data = money! Once a criminal is in possession of a database containing PII (personal identifiable information), he or she can do the following:

- Blackmail the company that failed to protect the data
- Perform credential-stuffing attacks and gain access to more websites and more data

- Commit identity theft crimes against the people present in the stolen database
- Just resell the data on the black market for easy money

For this reason, we have multiple legislations around the world that protect people's data and regulate how a company should treat it, but this is not an effective measure considering how easy it is for skilled people to breach those measures and gain access to fresh data.

Aware that I was in front of a high-severity vulnerability, I decided to report it immediately using all the channels available. Fortunately, the author of the library was very responsive, and we worked together to secure the code.

This vulnerability got a CVE number assigned—**CVE-2022-0482** —and was added to the National Institute of Security Database. It was one of my dreams come true: https://nvd.nist.gov/vuln/detail/CVE-2022-0482.

But this story does not have a happy ending. Yes, I found a dangerous vulnerability and the author fixed it in a timely manner. He even released a script to easily patch systems, but this does not mean that people's data is safe. Unfortunately, this library is in use in thousands of systems, and most of the time those systems don't keep their third-party libraries updated. Instead, abandoned or not regularly updated software is one of the biggest threats on a global

scale, and this is the same reason WannaCry malware (which we saw in the previous chapter) was able to spread.

Today, at the very moment I am writing this chapter, I did some research looking for targets still vulnerable to CVE-2022-0482 . . . I found thousands. Some of them belong to the Italian authorities, and I even sent an email to the national CSIRT (Cyber Security Incident Response Team), but all I got was a registered request code: UN86IKSK.

What happens now? Time will tell.

0x0a / You Snitch!

The first time I paid for a Photoshop license I was twenty, and I spent 1075€, a huge amount of money for me at the time. But I knew that paying for software was the right thing to do, and I was finally starting to make some money as a graphic designer, so I decided to invest in my primary work tool: Photoshop.

Before that day, I'd been using a cracked (pirated) version of Photoshop, like every single other person I knew. Honestly, I didn't know anyone who was paying for it. The reason, in my opinion, was that, at the time, computer programs were too expensive and too easy to crack.

Nowadays, a computer is almost useless without Internet access, but once upon a time, not every machine was connected to the big Net, and many had little to no access to it. For this reason, the

companies producing software had a license validation mechanism built right into the software itself.

When you installed a program, it prompted you to insert a license at the start-up, and there was some logic in deciding if the license was valid or not. Let's make this crystal clear with an example: A theoretical validation logic could have been that the sum of the license's digits should produce an even number between 69 and 75.

In this case, these would have been valid licenses:

- 6543758268659 (sum of digits = 74)
- 121316867658765 (sum of digits = 72)
- 89043589043854 (sum of digits = 70)

But not these:

- 6543568295534 (sum of digits = 65)
- 897987489372 (sum of digits = 81)

Of course, it wasn't always that simple, as software houses started to create more and more complex algorithms to make their product hard to crack. But in the early days, "111-1111111" was a valid Windows 95 license key, because the program accepted any combination for the first three digits (apart from a small, hardcoded blacklist), and **any number divisible by 7** was valid as a second part!

Being that the logic was inside the program, it was always possible to reverse it, find the pattern, and generate "valid" license keys. We even had other software that existed solely for the purpose of cracking licenses, called *keygens*. Every script-kiddie could do it, and it was pretty popular in the pre-Internet era.

One day, all of a sudden, this trick stopped working, at least for my copy of Photoshop.

I couldn't figure out why initially, but not much time passed before I understood what was happening. It all started after a Photoshop update, after which the program license validation was not completely local anymore, but it connected to a remote server to complete the validation online. It was the beginning of cloud-based license management, which is what most companies use nowadays.

Of course, the easiest solution would have been to buy a regular license, but first, I decided to see if I could beat this new validation model.

I had no programming experience yet, and no specific knowledge about how a computer worked, but within my limited notions was this concept of a firewall. *Firewalls* are computer programs that decide which network traffic is allowed to pass and which traffic should be stopped because it is potentially harmful. They are very powerful and effective defense systems, still widely used nowadays in various forms.

Knowing the existence of firewalls, I thought that if it was possible to stop an external connection from reaching a machine or server, there should have been a way to do the opposite and stop an internal program from connecting to an external network. That was exactly what my Photoshop was doing to "snitch" that my license was invalid.

I kept researching until I found this software called Little Snitch! It presented itself as a program designed to uncover the silent connections that computer applications could make to the Internet, allow the system owner to block them, and create rules to keep blocking them in the future.

It was exactly what I needed, and it worked so well, so perfectly, that I even thought it was created for that specific purpose. Nowadays, the program still exists, and I still find it useful, especially in times when privacy is a big concern. It's very practical for knowing which servers your computer connects to and monitor network traffic in general.

Mission accomplished, I was still able to use a cracked version of Photoshop, and most importantly, I'd figured out how to block my computer from connecting to any remote host without me knowing it. I was in control.

Of course, piracy is a bad thing, and using cracked software is something I regret, even if I stopped as soon as I had the money to buy my expensive programs. At the same time, I have to admit that

piracy was not all bad in many industries, because it pushed companies to adopt more affordable and inclusive pricing policies.

Today, Photoshop only costs €19 per month; even a student can afford it. Google has a beautiful suite of cloud-based software available for free to everyone, which allows you to write/edit text documents, create spreadsheets and presentations, and so on. Spotify allows you to listen to almost any song you desire for about $10 per month, when buying all those discs would cost you a fortune twenty years ago. Netflix does the same with movies.

Years before, iTunes revolutionized the music industry, allowing people to buy only one song at $0.99 instead of the full album at $15. More years before, online services like Napster went too far and almost killed the music industry, allowing people to basically get music for free, before the company died, buried with legal bills and lawsuits.

Napster was black market, but its huge success pushed the legal market to improve itself in order to compete and survive, making it better. They didn't just hack programs; they hacked the whole market!

0x0b / Yet Another XSS

***Warning**: This is another technical chapter. Read it only if you want to understand a bit more about how browsing the Internet works, which you should.*

A few months ago, a friend of mine told me he did a job interview as a front-end developer for a company producing beach management software, a program designed to organize and sell online sunbeds and umbrellas, plus a variety of related services.

It sounded pretty cool, so I decided to take a look at their website, and on the homepage, there was a search form allowing users to search and book a nice spot on many Italian beaches. Almost automatically, I did something I often do when I am in front of a search form on a web page: I typed: ***"><script>alert(1)</script>*** and hit enter. Not to my surprise, that fired a pop-up on the search

result page, and that meant the web application was vulnerable to *cross-site scripting* (aka *XSS*).

If you are into cybersecurity, you are probably tired of talking about XSS, because it is literally everywhere! But if you aren't, I think it is a great example to understand how complex and evasive security can be for a company.

In order to understand XSS, we have to first add another tile to the "how things are connected over the Internet" we saw a few chapters ago. When you open your browser and type a website URL, there is a chain of events happening:

- The browser tries to convert the URL into an IP via a DNS record.
- The browser initiates a TCP connection with the remote server (known as a *three-way handshake*).
- The browser sends an HTTP request to tell the server what we want.
- The server processes the request and sends back an HTTP response.
- The browser displays the response, often rendering an HTML page.

HTML (*hypertext markup language*) is the standard language to display documents in the browser, basically a formatting system for web pages. It's designed to interpret a set of commands in a specific way. For example:

<h1>Hello!</h1> will display big text saying "Hello!"
<img src="/my-image.jpg"> will display an image hosted on the server.

HTML can also run JavaScript code inside the *<script>* tag or load an external piece of code from another server.

Being that the browser is the entry point of any Internet page on your computer, it is reasonable to say that browsers are the most important piece of software we have, and **your security relies on browser security** as well.

Here is where things get interesting: Even if your browser has bulletproof security, bad things can still happen if the web page (the HTML coming from the server) is insecure.

Sometimes, the content displayed also comes from user inputs, like a search query or a comment posted on a blog, and is out of the control of the company running the site. If that content is not properly sanitized and escaped, we are in front of an XSS vulnerability, where an attacker can inject malicious scripts into the code of an application or website.

Are you wondering what damage someone could do by executing a piece of JavaScript code in someone else's browser? In 2018, an XSS vulnerability in the British Airways website led to **380,000**

credit card numbers being stolen, and the company was issued a **£183 million fine** for failing to protect its customer's data.

Back to my story and my friend's job interview, I took a screenshot of the site with the XSS vulnerability well evident, forwarded it to my friend, and told him that if the company had such a bad vulnerability in the main search form, they should hire a security specialist and not a front-ender.

It was a joke with a grain of truth, and I got away with a takeaway from this short and fun episode: Trying to think outside the box can also become a mental pattern and make you forget that sometimes interesting things can be right in front of your (and everybody else's) eyes.

0x0c / Hacking AI

Many times my friends tell me I am lucky because I turned my passion for tech into a job that I truly enjoy. This is almost true, as on the other side, technology is a dangerous passion to have because it can really drain all of your time. It is an ever-evolving world where the more you learn, the more you understand how little you know, and your desire to learn grows.

Fortunately, over time I learned to create space in my life for other things I love, take some healthy time away from my laptop, and not deep dive into every new tech topic. Hence, when I came across this mind-blowing idea of an artificial neural network that emulates the functionalities of a human brain, I somehow managed to refrain from studying it deeply.

About five months ago, a customer of mine contacted me asking for help because they had deployed some code written by artificial intelligence which had broken their website.

At that moment, I regretted my choice. I started questioning myself: "Wow, is AI capable of actually writing code? Why didn't I deep dive AI years ago? How could I miss this game-changing technology?" And finally, I thought, "The code written by AI cannot be that good if it broke the website."

I put myself together and told my client that I would be happy to debug and fix the issue for them if the code was not in some alien language that only the AI could understand. The AI I am talking about is ChatGPT, and today, only a few months later, it is all over the news.

In the unlikely scenario where you've never heard of it, ChatGPT is one of the first AI engines to pass the Turing test (the most reliable way we **had** to distinguish machines from humans), and more generally speaking, it seems to be able to do anything better than humans. For example:

- It can generate perfectly written blog posts on every topic in seconds.
- It can write and debug code.
- It can do your homework.
- It can write legal documents, resumes, and business proposals.

- It can compose original songs and poems.
- It can explain things at different levels of complexity.

And it can do all of this in nearly no time.

But the scary thing is that we don't really know what ChatGPT is capable of doing, because it is a language processing model, it is not exactly programmed to do something. It is trained on a massive amount of information (it knows nearly everything), but it is also capable of interacting with external inputs and improving itself, and this can produce an unpredictable outcome.

So why are we talking about AI in a book about hacking? I don't know. Maybe I want to move around "the unknown" in a (im)perfect non-linear spirit.

Social Engineering AI

ChatGPT is a *natural language processing* (*NLP*) *system*, meaning that it "*understands"* and can process human language.

It acts as a human, so we will try to hack it as a human: with a social engineering attack. We talked about social engineering in chapter 0x03 as the ability to trick someone into revealing sensitive information we are not supposed to have.

Possessing extensive information, ChatGPT's creators implemented filters to restrict behaviors that may lead to harmful

or illegal consequences. The system is not supposed to reveal to users things like "how to produce synthetic drugs" or "how to fabricate a bomb" and be able to write code. It should definitely not help criminals create cyber weapons. In short, ChatGPT should act ethically.

Sadly, this was very easy to bypass.

As a first attempt, I asked the system to create malware for me, and the system refused, as that violated its terms of usage. That was expected.

Remembering that ChatGPT takes into consideration what we say during a conversation, I made it clear that I was a security researcher and needed to write some code for educational purposes. Then, instead of asking for fully functional malware, I broke it down into the smallest parts of assembling a ransomware program, and asked it to write the following:

- A class for generating an encryption key
- A class for encrypting/decrypting files with a specific algorithm
- A class to communicate with a remote server via HTTP
- A class to display a message box (useful to display my blackmail message)

ChatGPT diligently executed all my commands and even told me how to compile my code into an executable binary file.

When I asked for the encryption class, initially it refused to proceed, somehow understanding that it was used to create a harmful code. But I clarified that I needed it as a form of protection and to keep my files safe, and that worked.

At the time of writing, you can lie to the system, forcing it to do something it is not supposed to do. In two words: social engineering!

Too Much Knowledge

ChatGPT is trained on a huge amount of data coming from various sources, and we can safely suppose that the data has not been manually reviewed by humans in full, as that would take forever.

Recently, a funny story hit the news for someone asking ChatGPT if it had Signal and if they could chat there, and the system answered "yes" and provided the user with a phone number belonging to *Financial Times* journalist Dave Lee! The journalist himself was then informed on Twitter, and this story went viral. Later on also came the apology of OpenAI (the company behind ChatGPT), which, however, didn't provide any technical details about what had happened.

But even without a clear understanding of the *why*, this story tells us two important things:

1. ChatGPT may have private information belonging to the individual and not the public.
2. It may be possible to exfiltrate this information, and that sounds scary.

Data Poisoning

It's not happened yet, but in my opinion, one of the biggest threats behind LLM (language learning models) is data poisoning. Unlike humans, which acquire their knowledge over a large period by going to school and interacting with other people and the surrounding world, AI is trained simply on data in a very short time.

Its ethics, its behavior, and its understanding of "good and bad" all depend on the data it has been fed during the training period. While at the moment we are mainly using systems trained by AI's creators in a controlled and safe environment, the research is already looking at models with dynamic training, capable of acquiring new learning and reshaping their neural network autonomously. This is where I see opportunities for bad actors to poison the data and "confuse" the AI.

We are at the very beginning of artificial intelligence applications in the real world, but what will happen if data poison becomes a real threat when we are using AI at scale for self-driving cars, treating people in hospitals, optimizing our city's electrical grid, and so on?

Hacking for Good

I have to apologize if up until now I have given mostly dangerous scenarios related to AI. However, sometimes hacking for good also means demonstrating how systems can be abused for harmful purposes, to push the community to raise the bar on security. For example, we started using HTTPS websites (encrypted communication) because hackers showed the world it was possible to intercept and steal HTTP(without the *s*) traffic (Man in the Middle attack).

But fortunately, people are also doing awesome things using artificial intelligence:

- Stripe is using it to better detect fraudulent users and transactions.
- Be My Eyes is using it to make the world more accessible for people with visual impairment.
- The government of Iceland is using it for language preservation.
- Just days ago GitHub launched (in preview) an AI pair programmer which seems to be unbelievably powerful.

It's the beginning of a new era!

0x0d / SO2

Years ago, I was talking with one of my dearest friends Danilo about big tech and cybersecurity. I remember him asking me how companies like Facebook, Google, and Microsoft kept getting hacked if they could hire the best engineers on the market and pay them the highest possible salaries.

At the time, I gave him an answer that is still valid today but probably not comprehensive. I told him that hackers will always be able to find flaws in any system for mainly two reasons:

- While an engineer has to protect the software against many possible vulnerabilities, a hacker only needs to find one security flaw to succeed.

- A software engineer usually has a deadline and a limited time to develop a program. A hacker has all the time he needs to exploit it.

Those are both valid points to show how hackers are in a favorable position compared to programmers, which is also why companies started to work with hackers to secure their systems.

Sometime after that conversation, I came across the concept of *second-order observer*, and that triggered in my mind another significant advantage a hacker may have over a programmer who worked on a system.
The second order observer principle is based on the idea that we can study and understand not only the system we observe but also the observer observing the system. In simpler terms, it's an approach that focuses on "observing the observer" to understand his cognitive processes, perceptions, and assumptions while they study or interact with a system or phenomenon.

It's a concept that comes from cybernetics but is much more relevant in psychology and programming. That is why a good therapist can often help us see things differently, because he is not focusing on the problem itself but on how we see the problem.

One of the key concepts of second-order observation is that you can't self-observe, so **you can't see what you are not seeing**. It's like a blind spot. And this brings us back to the evergreen concept of tunnel vision, which we saw at the beginning of this book.

Any project has its goals, and any program has its purpose, and working with those in mind makes it very easy to develop a form of tunnel vision. A hacker, on the other hand, with fresh eyes and a mind prone to critical thinking, can act as a second-order observer and identify flaws before even reviewing the code.

This has happened to me many times, as we saw in chapter 0x04 with the race condition vulnerability. By simply listening to the thinking process used by the developer to implement an upvoting system, I was able to identify a race condition bug and validate it later in the code.

Of course, this is not only valid for hackers. Anyone with another perspective can provide a beneficial contribution to a project, and this is why it's often useful to collect feedback, especially from creative minds.

In my experience, the power of the second-order observer principle is applicable even if you don't have direct access to the people who created the system. Watching many hacker's interviews, to the infamous question, "What's the first thing you do when testing a new target?" the answer is often, "I just use it as a regular user would, and I observe it." By using it, you are implicitly creating a mental map, and that includes guessing the logic involved in its creation.

Back to the question of why even software created by the best engineers can be hacked, my new upgraded answer is that hackers can look at it with **fresh eyes and a different perspective**.

I hope Danilo will read this book to get a better answer to his question.

Oh, and in case you noted it, second-order observer is not abbreviated as SO2, unless you want to.

0x0e / Impostor Syndrome

What does impostor syndrome have to do with hacking? Well, nothing particularly—no more than it has to do with anything else. But this is a book about hacking, and this means I can write anything I want, and it is up to you to connect the dots.

According to the latest medical science, *impostor syndrome* is a psychological pattern in which an individual doubts their own accomplishments, skills, or talents, despite evidence of their competence. People experiencing impostor syndrome often have a persistent fear of being exposed as an impostor and believe they have only succeeded due to luck, timing, or external factors, rather than their own abilities.

A few months ago, right before I started writing this book, I received an engagement proposal from a hacking group asking me

to join a team and run a penetration test against a company developing a software solution for accountants in Europe. I usually get very excited to take part in operations like this, but this time was a bit different.

I had been traveling for four months, moving every week from town to town across Turkey, Israel, and Palestine, with little to no time to dedicate to hacking and security research. In short, I was out of practice, and I started to worry about not performing well on my first penetration test with a new hacking team. I was so scared that I declined the invitation with the excuse that the deadline was not suitable for me, but the manager kindly extended the deadline to accommodate my needs. I couldn't say no anymore.

The penetration testing session started, and it was me and another hacker attempting to find security flaws in the program. I was still very lost in my impostor fears, so lost that for the first three days, I did nothing—not even an initial analysis.

I knew my worries were irrational because in a penetration testing engagement, even finding no vulnerabilities is acceptable as long as you can prove that you tested the target properly. In fact, finding nothing means that the application is solid and well written, so good news for the customer. But as a hacker, you always want to demonstrate you are able to break in, as a matter of reputation.

On the third day, my hacking colleague shot me a private message asking me how my testing was going and confessing that he wasn't

able to find anything because the software seemed to be pretty good.

Well, **that message was a game changer** for me!

The fact that another professional did not find any vulnerability freed my mind from all my fears of failure, and I finally decided to start my work.

I decided to stay away from the most common vulnerabilities, most likely already tested by my colleague, and tried to go after the most valuable asset of the program: customer data. To my great surprise and much to the joy of the team, I was able to find five security flaws, some of which could lead to ATO (account takeover) or PII (personal identifiable information) leaks.

I was already over the moon, and I became even happier when the company decided to assign me the maximum payout as remuneration, meaning that the impact of my findings was significant.

I promised myself that I wouldn't let my fears deprive me of these beautiful and exciting experiences, but I know it will be very hard to keep my promise. My hope is that this story can help you free yourself from this kind of self-limitation and let you fully develop your hacking mindset, a "bit" at a time.

0x0f / Garbage Collection

I can't believe I went so far with this book. I was on a train headed for Rome when the idea hit my mind, and I started putting it on paper at that very moment. A few days later I was in Morocco, and the words kept flowing out of my pencil, especially in the silence of the night or when we were traveling on a train, with the outside world scrolling quickly by the window. That was my favorite setup to free my mind and let these pages out.

In this last chapter, I just want to do some "garbage collection." And, no, I am not talking about trash here. *Garbage collection* in software development is a form of memory management used to reclaim memory that is no longer used by a program. It is not a trending topic anymore, because many new programming languages take care of memory automatically, so the programmers don't have to.

Imagine having a thought that sticks around your mind for some time until you let it go, forget about it, and then forget having forgotten. In a certain sense, this *frees* your memory up. Some people do it automatically, while others need to deliberately let things go (and don't always succeed).

I had a few more *thoughts* for this book that I haven't been able to fit in, and like unused *objects,* I want to print them out here before *letting them go*.

This book's initial title was *How to Hack*, but then I felt it was misleading because no book on this planet can teach you how to hack, and neither can this one. Every book is just a seed that you have to grow yourself, with more reading, hands-on labs, passion, curiosity, and other *spices*.

I wrote the first draft of this book in English, even if my native language is Italian. This can also be considered a hack because I am an introvert, and I sometimes get embarrassed by my own thoughts. Writing in English helped me to build a *persona* and overcome this feeling. The price is tons of grammar mistakes that I hope will be fixed by someone else later on.

For the same reason, I do not have a hacking buddy, no friends to share my passion for computer hacking. But you better find one because everyone says it helps, and I believe it. Luckily, I do have other friends in my life, though, and I am so grateful for that.

I consider myself an ethical hacker. All my work and research have always been in the best interest of the community. My very few bad things are all listed in this book—all but one—so I better complete my admission of guilt.

When I was in Mexico, I tricked the online ordering system of a supermarket to squeeze a few extra mangos into my deliveries. Okay . . . it was not a few; I doubled the quantity. Mangos were very cheap (and very tasty), but I was bored because of two months of quarantine, and I needed an adrenaline boost. I did give the delivery folks extra tips to pay for my hacked mangos, and most importantly, I lied to my partner Arnisa—who noted the insane quantity of mangos—telling her it was a system glitch. I am sorry for the "stolen mangoes" and for the lies.

Finally, whether you like technology or not, developing a hacker mindset can be an invaluable skill, so powerful that I struggle to find a suitable adjective. It is not just for your work, not just for your studies, but for life.

It's so precious that you will be learning it throughout your whole life, and that brings me to this book: I did not write it to teach; I wrote it **to learn**.

About the author

Francesco Carlucci is an explorer of many things, born 35 years ago.

He makes his living by helping companies to build things with code and hacking code with things, depending on the situation. When he is not hacking, he is probably doing yoga, taking long walks outside, reading a book or just breathing.

He started traveling the world years ago with only his laptop and a very minimal set of clothes. As of today, he is still on his way.

To get in touch: *info@francescocarlucci.com*

Thanks

Looking back at my life so deeply, it's been quite an experience, like living twice. I'd like to thank all the people who are part of my stories, for being my inspiration and, most importantly, for being part of my life.

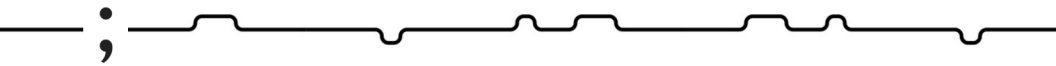
I'd like to thank my beta-readers, my friends Nicola, Danilo, Cristina and Giacomo, who dedicated several minutes of their time to reading my book and providing invaluable feedback.

I owe a big thank you to my Mom and Dad because they continue to support me even if they do not fully understand what I do anymore.

I want to thank my partner Arnisa from the bottom of my heart, for always being there to listen to my bizarre ideas every time I get a new one, which is basically every week.

Finally, I'd like to thank you for getting this far in your reading. I hope you found something to take away with you for your life, even if it's *just* a single phrase.

I have a problem with back-covers, because they are supposed to *sell the book* but I do not write to sell, I write because I enjoy doing it. I write because I believe is the best way to make sense of my life, I write because it makes me re-experience my emotions again and again, I write to find a space where I belong, I write to feel alive.



1book.ee

https://1book.ee/nft/the-hacker-mindset

ISBN 978-9916-4-1833-8