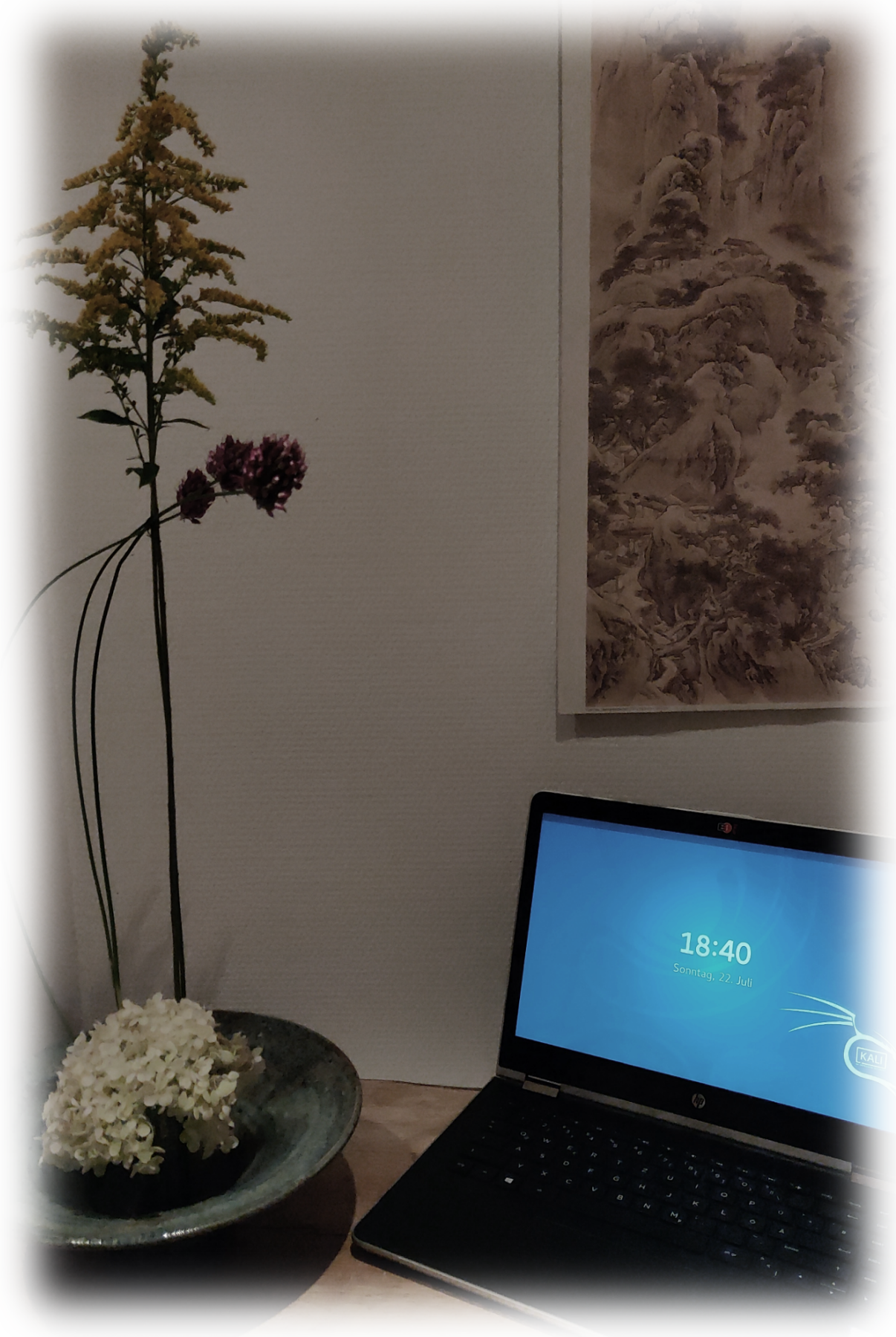
Building Computer Security



Building Computer Security

Thorsten Sick

This book is available at https://leanpub.com/buildingcomputersecurity

This version was published on 2025-08-08

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Tweet This Book!

Please help Thorsten Sick by spreading the word about this book on Twitter!

The suggested hashtag for this book is #buildingcomputersecurity.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

#buildingcomputersecurity

Contents

Preface

This book

Goals

Your next project will be **more secure** than your last project. And this will be almost hassle-free. This is the mission of this book.

This is why I wrote it.

As an experienced engineer, software developer and security guy I want to help you getting started with security features in your projects.

My observation is that there are many specific books for security topics. But almost no "getting started" books.

This is the itch this book wants to scratch.

Release early, release often

One of the reasons I dare to write a book is leanpub's feature to release *early* and *often*. That way I can see progress and get feedback.

Please visit the forum and help me improve the book. Or contact me:

- On Twitter: @ThorstenSick
- On Mastodon: @thorsi@chaos.social

As I have been spending lots of time with security focused engineers, the most important things I want to learn first from you:

- What causes trouble ?
- What kind of technology is currently introduced in normal engineering ?
  - Where does it cause issues ?
- What kind of normal technology (cloud, micro services, IoT) are you using ?

Boiling down to: What do you need from this book ?

80/20 or Pareto Principle

Following the Pareto principle I will focus on finding the *important* 80 percent of security tips first and later filling in the missing 20 percent.

This will result in:

- I start a new topic before I squished the last drop out of an older topic
- Instead of spending hours to fine tune the language and optics of an existing chapter I start a new chapter with dozens of new tips

This does not mean I will not do that sooner or later. But I want to get the low hanging fruit first. I also feel like you expect a book containing lots of good tricks instead of poetry.

Form follows function

With every iteration of the book I will tune the optics, improve the layout and maybe add images.

But adding new chapters is more important.

All this said...

Adding security features to a project can be fun and quite rewarding. Cherry-pick ideas from this book and get started.

Structure of this book

This book is meant to be read twice. First by cherry picking interesting topics and a second time as toolbox when implementing a feature.

For this reason the book is split. The first few chapters offer an introduction to the security landscape. The second part contains chapters helping to solve specific problems.

Do not feel bad if you directly skip to something interesting. Or to a chapter that is currently relevant for your project.

Project phases and audience

A project has different phases. Depending on your institution different people could be responsible for each phase. Everyone can contribute to security. And security will break at the weakest link.

Stages in development are:

- Management
- Architecture/Design/Project Management
- Development
- Testing
- Support
- Analysis (analysis of attacks)

Content

Background

Know your enemies

Describes the types of adversaries your project can face. And their high-level goals, resources, technologies and tactics. You should know your enemies

Attacker's goals

An in depth list of goals and sub-goals the attackers have - things you will want to prevent happening. Block the attack, protect your system.

Principles

High level principles you can use to design a secure system. These principles are abstract - and you will be able to combine several of them in designing your project.

Planning

Threat modelling

A mixed group of people (engineers, architects, managers) meets for a *threat modelling* session. Understanding the strengths and weaknesses of the product and the planned new features. "What would an attacker want ?", "How would he get that ?". Identifying that is the first step to protect the valuable assets. It can sound intimidating. But there are so many different approaches to make it simple and even fun. I collected a range of "best of" approaches in this chapter.

Software Design

Software design - also known as architecture - is the bigger picture of the technological part of the project. A good design can speed up development, simplify testing, and increase security. This chapter covers the security basics of software design. Followed by specific topics in other chapters.

Vulnerabilities

Your software - and maybe even your hardware will have bugs that turn into vulnerabilities. Ready to be exploited by an attacker. The chapter Vulnerabilities will cover the essence and taxonomy of vulnerabilities. Understanding that will make decisions based on vulnerabilities simpler.

Security process

Internal set up of a project will decide if a external vulnerability report is a disaster for the project release cycle or can be handled in a relaxed and simple way. Being prepared for the inevitable is central in this chapter.

Software Design Checklist

The Software Design Checklist introduces you to a source to learn about typical software design flaws. It also starts to create an (incomplete) list of typical flaws encountered it real world failed projects.

UX

How to create a user interface in a secure way - preventing the user from becoming the weakest link.

Updates

As soon as a software reaches a certain complexity you need updates to ship patches. This chapter guides you through the challenge of shipping updates in a secure way.

Passwords

Passwords are the main way the user authenticates himself. Handling that in a secure fashion can turn out to be quite complex - but at the same time there is enough experience around.

Browser

The browser is the main software running on most computers. And the interface for a lot of other technologies. As browsers are complex they are vulnerable. Some things can be done to improve browser security - or the security of connected web admin platforms (router config page).

Censorship

This chapter should be named *Resilience*. But *Censorship* attracts more attention. Our computer systems are very complex. And many do not have enough redundancy to recover from a single failure (accidental or intentional). But they can be built that way. This is covered in this chapter. It currently focuses on personal redundancy. Later it will be extended to also cover how to build resilient systems.

IoT

Things are getting more smart. Embedded Micro-controllers are cheap. Security is very often not existing. This chapter helps to improve that.

CAN

The CAN bus connects micro-controllers in many embedded systems - especially in vehicles. Securing it is a challenge - as it was not designed for security. This chapter will give you some tools - and basics for fuzzing CAN.

Bluetooth LE

Low Energy Bluetooth can be integrated in many systems. Security is hard to implement. But possible. This chapter will give you the tools and basics to improve security.

TLS

TLS is a basic technology to secure data in transit. Some may know it as SSL or HTTPS. And many projects use it.

Crypto algorithms

Which algorithm to use ? In which mode ? Best key length ? This book is basic, so this chapter will be more of a cheat sheet for those essential things. For the curious people: there are resources to dig much deeper.

Programming

Code analysis tools table

There are several tools that will assist you during the coding process. This table will help you to pick the proper ones.

Defensive programming

Some coding techniques will harden your code against attackers. Positive side effects like making your code simpler to debug and more robust and stable are take always that balance the extra effort invested. If you do not yet practice defensive programming you should consider starting it.

Assert

A very powerful tool for debugging and preventing bugs are *asserts*. They are even more powerful when combined with *fuzzing*. When coding, you should start using asserts right now.

Compiling

Modern compilers have lots of less known features to find vulnerabilities during compile or run time. Or even mitigate - making vulnerabilities harder to abuse by the attacker. Covered in this chapter.

Clang LLVM

Clang is a compiler in the GCC league. It adds some more features for static and dynamic testing of your code. Those features are cool enough that you should consider compiling at least the test version using Clang and those settings.

JavaScript security

I am a big fan of automated security tests. And JavaScript has its own pitfalls. But those can be fixed by tools you can integrate seamlessly into your workflow. The JavaScript security will show you the low hanging fruit.

Testing

Testing compiled binaries

To verify that the compiler properly hardened your executable the way you wanted you (or your testers) will want to use some tools. This is described here.

Flawfinder

In addition to your compiler the tool Flawfinder gets you more verification of your code. With zero effort.

CPPCheck

CPPCheck is Flawfinder enhanced. With the small drawback that you will have to invest some time in configuration. This chapter will help you there.

Testing

Is it still testing or already hacking our own product ? There are many tricks to make software testing more aggressive and which will shine some light on vulnerabilities in this chapter.

Fuzzing

Fuzzing is a technology to find unexpected bugs - most of them potential vulnerabilities. Pre-existing tools and DIY scripts can do the fuzzing for you. How to get started is described here.

Tools

Several tools are essential to build secure software. This covers version control systems, SSH and many others that “just” support you in getting your tasks done. Those are covered in this part of the book.

SSH

A very powerful tool that goes way beyond just replacing Telnet. Know it and simplify your work while being more secure. Or - if you build embedded systems - integrate it in there.

Git

Git as a version control system is the core of your development workflow. It is also a perfect spot to add automated code quality tools. You learned about those tools in the other chapters. And this chapter gives you some hints where to get started integrating them into Git.

Bolt-on security

Bolt-on security is what is done of someone missed to do security-by-design. Especially myths make it hard to find the right way to use it. These chapters will bust some myths.

Antivirus tests

How to properly read antivirus tests. As antivirus is neither a silver bullet nor useless a proper estimation of its power should be done first.

Antivirus integration

Antivirus can only scan files it gets access to. Many projects could benefit from an interface between the project and an already installed antivirus. And Microsoft is offering one.

Antivirus sharing samples

There are several reasons to contact an antivirus vendor

- They have a False Positive on your software
- You found undetected malware (files or URLs)

This chapter will help you getting started and give you some internals - the people working in an antivirus lab are on your side and will love to cooperate.

Antivirus: VirusTotal

VirusTotal is the go-to way of quick scanning potential malware and false positives. But has an own tricky personality. Knowing that will prevent you from coming to the wrong conclusions.

Antivirus: Behaviour based classification

Unknown samples must be classified before detection can be created. A quite reliable way to achieve that is behaviour based classification. Especially it is quite simple to interpret the resulting log for malicious behaviour. The basics are covered in this chapter

IOC sources

IOC sources is a collection of lists of so called Indicator Of Compromise Sources. Those are Malware executables, hashes, URLs, registry key, …

The lists can be used for training, learning or (if they are curated) event directly blocklisting.

Offense

Even if you are playing Defense - it is very important to know the tools, tricks and tactics of Offense. The basics are covered here.

Kill chain

A kill chain is a typical structure of an attack. With different steps happening in between “the attacker decided to attack” and “the attack succeeded and is spending the stolen money”. A kill chain can be interrupted at any stage. Knowing kill chains will help the defender.

Recon NG

An early part of manual attacks is reconnaissance of the target. There are several tools helping the attacker doing that. The same tools help defenders estimate their attack surface. One of those is Recon-NG.

Google Dorks

Google Dorks are simple reconnaissance tricks abusing Google. Using them it is possible to learn about vulnerabilities in web pages without event connecting to them once.

BeEF

BeEF - Browser exploitation Framework helps you exploiting browsers. And those are very often the most central application in a person’s life.

Burp Suite

Burp suite is attacking the other way: Connecting to a web page and while crawling it enumerating the vulnerabilities.

Mitmproxy

Mitmproxy is a python proxy to intercept especially HTTP communication. Valuable for development of network communication, testing and pen testing.

Appendix

The appendix contains handy things that will make the reading of the book more enjoyable. Especially:

External References

A list of external references (books, talks and homepages).

Glossary

There are so many new acronyms in the security field. And even there they are territorial - Web security being totally different from Assembler hacking. Have a glossary ready. Here it is.

The author

Who wrote this book and: Why ?

Authors

As the security field is so large I want to have other authors contribute to their field of expertise. This chapter is ready to introduce them. Even if I just slowly started to invite people.

Changelog

Read this first after you got an updated book

License

If you want to do anything else with this book besides reading it - this chapter is for you.

Background

Intro

This book is written for the experienced engineer, manager, tester. And not for experienced security experts. Security knowledge has not been very accessible. There are films - and most are horribly wrong. And there are complex books covering a specific topic in depth.

If you not hyper-focused on security is is quite easy to either get lost in myths or complexity.

I want to get you the foundations to contribute to the security of your product.

The takeaway of this chapter will be:

- You know *who* could attack your technology
- You know *why*
- You will learn the basic *principles* of protection
- You will also learn about the technology available to the attacker

Principles

There is a list of the most basic principles you can build security on. These principles are very abstract, but are the basics for building more specific tools. This is not a complete list.

Bolt-on-security vs security-by-design

Detecting attacks is a typical bolt-on way of security. Anti-Virus, Network intrusion detection, behaviour detection, ... These kind of things are **bolt-on-security**. The other big security philosophy is **security-by-design**.

If you are responsible for building a system use **security-by-design**. In many cases you are not free to do it the perfect way - for example a manager needs an old Internet Explorer to use am esoteric web based application installed on his PC. In this case **bolt-on-security** (filtering files and URLs) is a valid approach.

Disadvantages of Bolt-on-security is: It adds complexity (file parsing, for example).

This can turn the bolt-on-security into a new infection vector.

To protect against it, isolate it from other systems (segmentation): Sandboxing and an own machine for the file processing are options.

Two tactics with bolt-on-security:

- Add it to the outer perimeter of your network to reduce the normal “internet noise” of attacks. This will reduce your important monitor-log reports.
- Add it to the innermost perimeter to detect attacks that bypassed your protection.

A whole part of this book covers bolt-on security. If your are stuck with a given design bolt-on will be a good way to go.

The next principles are part of the security-by-design philosophy

Threat modelling

Threat modelling is a process organized by the designer of the software but including a multi-skill team of people involved in the project. Goal is to identify potential threats to the developed tool as early as possible and to find fixes and mitigations.

It is not complex but can turn out to produce lots of documentation.

There are four basic steps that need to be done to model threats:

- Create overview

- Brainstorm potential attacks
- Rate severity
- Define fixes and create priority to implement them

For those steps there are many tools (methodologies/checklists) to guide your team through.

Which one to pick will depend on mood, team, and especially your project.

An improvised one would be:

- Who will attack your technology ?
- What is the goal of this person (data, money, …)
- Which resources can this person gather ?

There will be a specific chapter on threat modelling

Reduce Attack Surface

Attack surface is everything that can be reached from the outside. These are servers, interfaces, network connections. You can reduce this surface by asking yourself some simple questions:

- Which features do you really need ?
  - Remove unnecessary programs and features
- How to put steel plates on the technologies I have to keep ?
  - Reduce features, reduce privileges, ….
- How to reduce the potential impact ?
- Who do you (have to) trust ?
  - Best case: You do not have to trust anyone

Compartmentalisation / Segmentation

Segment your system into small partitions. Define narrow communication channels. Maybe they can be one-way (data-diodes). Possible segments can be:

- Multiple isolated VMs on a machine
- Multiple isolated processes on an OS
- Multiple isolated network segments
- Sandbox the processes
- Isolated servers
- Split a big application into smaller compartments
  - Example: Browsers have separate processes for tabs, extensions, …

Dropping privileges is a part of that.

Principle of “Least privilege”

Reducing available privileges reduces the impact of a hacked process or account. This reduction can be done on account level (special web server accounts, for example) or on process level (where the process drops privileges during startup - as soon as it does not require them anymore).

Dropping process privileges is possible on Linux. Not on Windows.

At least you can terminate the program if is run by root (as long as it is a potentially vulnerable program).

Landlock

Newer Linux kernels are shipped with the landlock module. Ths allows code to manage the own privileges. Your program should drop all privileges it does not need at startup.Many programming languages do support that:

- C: https://en.ittrip.xyz/c-language/c-sandbox-security
- Rust: https://docs.rs/landlock/latest/landlock/
- Golang: https://github.com/landlock-lsm/go-landlock
- Python: https://github.com/Edward-Knight/landlock

For command line execution use the cmdline tool “landrun”

Updates

After a vendor created a patch it takes the bad guys a few hours to reverse engineer the patch and find the vulnerability which has just been patched. Within days there are attacks. Update everything ASAP. Especially if it is network connected.

More on updates

Flexibility

Whatever you do: Be flexible. You did threat modelling during the design phase, you fixed all vulnerabilities you found. But there will be a hack.

Expect the attacker to find a trick against your tools in the following weeks. Have a plan ready to handle that situation. Expect the attacker to do something unexpected. Plan for flexibility.

Do not make mistakes

“Bitte immer alles richtig machen” in German. This is the core principle especially in Design and Coding. It kind of sounds obvious and boring. But there are ways to engineer fault-reduction with tools and processes.

Those are for example:

- Secure coding
- Compiler mitigation
- Testing
- Fuzzing

De-centralisation

De-centralise everything. Centralised components (servers) can be DDOS-ed. Will your IoT device still work when the server is down ? Which functionality is available in offline mode ? Could you add fallback servers to the plan ? Could the user override the server for a short time ?

Fail gracefully

Do you want your system to fail-close or fail-open ? What does the car’s door lock do when there is a power outage ? Lock or Open ? Plan these failures and the consequences. How can these failures be triggered and by whom ?

Monitor / Incident Response

Expect a hack to happen. But many hacked companies do not notice the breach - or do, hundreds of days later and by accident (server misbehaves or similar).

You do not want to be one of those. You could monitor the devices you own or the devices you sold to customers (IoT, for example) for security incidents. For “monitoring sold devices” scenario: respect privacy !

There is SIEM (Security Information and Event Management) software for that. Connect your technology to a SIEM.

This enables you to respond (“incident response”) when something happens. This has several benefits:

- You can respond (updates, isolate infected devices, ….)
- Learn from this incident for the next projects

Educate users

Users want the option to know what their technology is doing. Having two layers (simple and detailed) of information will very likely be embraced [1].

Your security plan should not depend on users reading, understanding and acting upon this information. But having smart users will add new “sensors” for unexpected attacks to your environment. Especially attacks containing social engineering. Those users would need a fast channel to contact the person in charge for reports on attacks.

Defense in Depth

As protection mechanism will fail - the attackers will find holes - it is better to layer defenses. To protect an asset it is essential to layer **different kinds** of defenses to make attacking more complicated to request different skills.

As for a castle example:

A castle **on a hill** has a **moat**, **walls** and **archers**.

Attackers will have to be:

- Good runners
- Swimmers
- Climbers
- Lucky

To reach their goal alive.

This will narrow down the number of successful survivors/attackers quite well.

The same is true for your computer system. Combine different technologies to protect an asset. Even if you think the one specific layer you just designed offers 100 % protection.

Filter at the endpoint

Thanks to efforts by the EFF, Google and many more the connections between client and server are now end-to-end encrypted (HTTPS). Which is an incredible security improvement. Many bolt-on security solutions offer a feature to scan the content of those connections with MITM proxies. This is only possible if you sabotage the alert system at the endpoints. Because they detect the *attack* a MITM is. Systems having a sabotaged alert have trouble detecting malicious attacks. Scanning as MITM is a very bad idea.

The solution is: Filter at the endpoints. In the browser, When files are written to the endpoint system, …

The user (or the admin - for larger installations) should also have full control about what is filtered.

[1] Depends on your users

Tripwires

Add tripwires to your design. One example is to fill your database with some fake customers with made up mail addresses. As soon as those mail addresses start receiving mails (SPAM), you know the db has been compromised.

The same for fake “password.txt” files in your system. They get accessed ? You got hacked.

Tripwires are useless if you do not have a monitoring system and a plan how to react.

Slow down the attacker

Bank safes have a timed lock. Only after a certain time they open. Thieves will be slowed down. With an alarm system and a police unit to respond this is powerful. Without those to additional ingredients it is useless. Maybe you can build something similar into your systems. Slow down, alerts and response.

Security by Obscurity

A bad idea. There are disassembler tools that help to reverse engineer executables. If any security depends on secrecy it will break as soon as an experienced person attacks it. You are an experienced engineer but not an experienced attacker. The guys you face have the skill set switched: experienced attacker, OK-ish engineer. This is why they became attackers.

Long story short: if your defense depends on “as long as the attacker does not know X” it already failed.

And just in case you think these disassembler tools are out of some fairy tales:

- IDA, paid and free test version. For Mac/Win/Linux
- Hopper, Mac/Linux. Test version available
- Radare2, Mac/Windows/Linux/…

Security Theater

A big show towards the customer faking security features. Most often has an intended negative effect on usability - you want it to be noticed. It is also not necessarily cheap.

One typical example is the water bottle ban at airports.

Please double check if what you are implementing is just smoke and mirrors or some proper solution. And go for the proper solution.

Hollywood threats

Also on the negative side: Thanks to Hollywood movies everyone seems to know how an attack looks like - with lots of show effects and world shattering scale. When discussing security with security-beginners there will be a tendency for the discussion towards those Hollywood threats. Explain to everyone this concept. Have some real-world hacker stories ready and try to steer the discussion towards a productive region. Even if it is less exciting.

Further reading

Ghost in The Wires

Ghost in The Wires by Kevin Mitnick. Life story from Kevin Mitnick. Famous hacker and pen tester. Many described exploits are on the social engineering level. Simple to read. Most of the technology described here does not exist anymore.

Beyond Fear

Beyond Fear by Bruce Schneier is a book where a computer security guy analyses common physical world security practices like airport security. Teaches thinking and critical thinking about security and security promises. Can be read by non-technical people.

Future Crimes

Future Crimes by Marc Goodman is a book covering the experience of a cyber crime police officer. Every chapter is a noteworthy and often entertaining example of a real crime. Entertaining and does not require technical knowledge. Good for training critical thinking and security thinking.

Planning

Intro

Design flaws are as bad as programming bugs: Both are vulnerabilities an attacker can abuse.

A software or hardware design well done can simplify programming and strongly improve security.

Many specific technologies require domain knowledge to be used in a secure fashion. For example a crypto library offers an API to be used by programmers without having them understand the maths. This API can be simple and fool-proof or built-for-experts with lots of ways to make horrible mistakes. In the last case you will require lots of domain knowledge to build a secure application.

This chapter will extend principles from the background chapter into technology specific knowledge.

Updates

As soon as your system reaches a certain level of complexity, you will require updates. Reason for that are the inevitable bugs and feature requests.

And as bugs can be vulnerabilities you will not be offering a secure solution without updates.

The software offered today is highly complex. Caused by the foundation that is used: Operating Systems, libraries, frameworks, …

Even CPU microcode can be updated.

Updates are the solution. And they add more complexity.

Things that a good update strategy covers

- Be able to rollback
- Verify signature
- Secure key handling
- Verify version - only upgrade and no downgrades
- Track distribution
- Be able to stop update roll outs
- Distributing the updates
- Compress updates
- Diff updates
- Incentives to update
- Automatic updates

Things to avoid

- TOCTTOU Time of check to time of use - error
- Break the update chain

Especially breaking the update chain (for example by ruining the update on the client system) is one of the biggest fears.

Response Time

This is the time you have to response to an incident. Normally it is the span between learning about an issue and successfully updating all your customers systems.

If a vulnerability gets known to the *bad guys*, expect a time-to-exploit of a few days. Normal process is that someone identifies the vulnerability and writes an exploit (plus a blog post). The exploit is then stabilized and integrated into attack systems (metasploit for the more white hack side, exploit kits for the bad guys).

If a vulnerability gets reported as "responsible disclosure" by a friendly hacker you will have about 90 days (standard) to release the fix to all customers. If you have good reasons (complex situation where it is hard to get all your users updated) you can maybe negotiate and get some more days. But there is a very good reason for the 90 days: It is a reasonable time to get the fix released and a longer time frame would increase the risk that a bad guy finds the same vulnerability in parallel. This happened more often than you think.

Short: You will not be able to ship the fixes with the next regular update. Be ready to for quick response. Build your processes from support call to all-your-users update accordingly.

One more thing: As soon as the first client gets an update, the bad guys could get it as well. Normally they do a binary diff (using disassembly tools). From that diff they can learn which vulnerability you just patched. The race is on between the bad guys creating a reliable exploit and you getting the update to your last customer.

These *disassembler* tools may sound arcane, but are quite simple to find and sophisticated.

- IDA, paid and free test version. For Mac/Win/Linux
- Hopper, Mac/Linux. Test version available
- Radare2, Mac/Windows/Linux/… %% TODO Add Ghidra

Update strategy details

Be able to rollback

Be able to roll back your changes either automatically or by user interaction if the update breaks something. User interaction could be "press the reset button for 10 seconds". Rollback could be to the *factory system* or *last-know-good* software state. This will reduce your stress when doing emergency updates.

Verify signature

Cryptographically sign your binaries (**public key cryptography** is the keyword).

One way of signing your binaries is to create at least a sha256 hash of your binaries (md5, CRC and sha1 are broken). Create a update config file containing all your files, their versions and their hashes. This update config file should also have a unique version number that is incremented only.

Now sign this config file using **public key signatures**.

This config file will be checked against the public key file on the end-system. If the key matches and the version is bigger than the current one the binaries can be downloaded and verified against the hashes.

Secure key handling

The secret key must be stored on an isolated system. Not a build machine and especially not the download server. You sign the update binary or config config prior to release with this key.

You will want a second emergency key (its public part also distributed to the clients). This one is stored in a safe. If the first key is compromised you can can revoke it and do an emergency update now binding the emergency key to the clients.

Verify version - only upgrade

Only update to a **newer** version. No updates to older versions. If you ever want to roll back a change, release an old version of your programs with a higher version number.

The reason for this is that an attacker could trick clients to install updates with older versions of your program that still contains known vulnerabilities. And hack the systems afterwards.

Track distribution

You want to track your update roll out. And if systems are functional post-update. Especially that.

Be able to stop updates

If you roll out a broken update, you should be able to remove it from your update servers very quick. That way other users do not risk ruining their systems. Automatic roll back (mentioned a few chapters above) will take care for the updated systems.

Slowly increasing the percentage of client systems getting the update offered is also part of this. Start with a few percent, measure success, increase percentage.

Result of these precautions is that everyone (including management) feels much more comfortable rolling out updates. And this will make it simpler to update clients with a new version that contains “only one security fix that is not even abused - as far as we know”.

Distributing the updates

If your updates are big your servers will not have enough bandwidth for to update all clients in time. You could use a CDN for your updates. To further reduce load update in stages:

1. Check first if there is a newer version than the one installed. This is a few bytes only
2. Download config file. Verify it.
3. Download binaries

As you have your files signed, an option to be protected against DDOS-es vs. your update infrastructure would be to also have a fallback to a peer-to-peer network (BitTorrent). Some bigger software applications (games) use BitTorrent. Do not make that the only source ! BitTorrent could be blocked in some environments like companies. Also do not waste the users bandwidth without asking. It could be expensive for some of them if they are connected using a mobile network.

Compress updates

Compressing updates reduces stress on your update servers and makes them more resilient against DDOS attacks. Test common algorithms with your update files. The algorithms have different strengths and weaknesses. Also: De-compressing is very often much faster than compressing. And this is what your clients are doing.

Diff updates

If you are shipping large updates, you want to ship them as diffs to previous versions. Depending on your file types there can be different diff algorithms that are quite effective. Please test before using. For PE files there is "courgette". Always verify after diffing if the created file is the proper one (by hash). If something went wrong, fall back to whole-file download.

That way there is no additional risk.

Incentives

Some users care more about new Emojis than about vulnerability fixes. Maybe bundle some eye candy (new backgrounds, …) with the updates to get the users to update. Sad but true.

Plan for several channels

When creating an updater be ready to add new channels later. Valuable channels to have are for example an "internal" and a "beta" channel.

Automatic updates

Do updates in the background. Without user interaction. Especially if you have corporate users offer an Opt-out hidden in some config. Companies want to test before they deploy to thousands of users.

If the update requires a restart of the software think about waiting for the user to naturally restart it - with a timer. If the user does not restart it after X days nudge him.

Control the update infrastructure

It is essential that you control your update infrastructure. There are several things that can break given time:

- certificates run out
- You lose ownership of your domain
- server needs system updates

Plan ahead and create scripts that test your current situation and warn you weeks ahead before your domain ownership runs out or before you should create a new certificate.

Another thing to control is the certificate that signs the updates. It must not be on the update server. It also should not be on the build server (where lots of people from you company can access it) but on a dedicated system.

Details for things to avoid

TOCTTOU Time of check to time of use - error

Wikipedia on TOCTOU

A possible issue when checking the signature of an update package: If the package is on an external medium and the updater is not copying it locally, the attacker can switch the (legit) package after verification with a malicious package.

1. You verify the (clean) external data's signature
2. Attacker replaces clean software with malicious software
3. Your tool now installs the malicious software from external medium

Solution: Copy the package into a local storage before verification and installation.

Break the update chain

Never break the update chain for your devices. As long as updates work you can fix anything. Focus of your testers should be to (automatically) test updates over several versions.

Passwords

Passwords are one of the most common authentication technologies.

There are different kinds of passwords: Pins, Passwords and Passphrases.

PINs

Pins are short (4-10), contain most often only numbers and are insecure. To still make them a valid security feature add a short delay in between pins entered. After the 3rd wrong entry either lock the system and request a special PIN (like PUK for a Smartphone SIM cards) or add a massive penalty (1 minute) which increases during the next wrong entries. A negative side effect of this is that the system can be locked by a malicious actor…

Passwords

Passwords are longer than PINs. As with the current state of technology (cloud computing and GPUs) standard passwords have to be long to have a decent amount of security. To improve the situation you can push the user to create *good* passwords (high entropy thanks to a large character set, avoiding the top 100+ passwords, …)

Passphrases

Intentionally long passwords. If you create your product for a situation where a good keyboard is available or passwords are stored in password stores you should aim for that - or just do not artificially restrict password length and nudge users to use long passwords in a friendly way.

Entropy matters

Help the user to choose high-entropy passwords. Depending on the system the user is using there are several different approaches to generate high entropy passwords. Smartphone keyboards for example make it very difficult to enter special characters. Maybe go for long passphrases instead ? Also: Avoid the most common top 100+ passwords (google for them). Have a list in your app and block list them. Notify the user not to use those. If you have international users you should also have international top 100+ lists.

There is one exception where it is wise to use common passwords like “12345” or “password”: One time accounts without personal data attached. Things that can be stolen without pain to the user. But as a normal user will not be able to make this distinction - do not tell them that there is a situation when weak passwords are OK…

High entropy is based on length of the password and potential character set being used. This is why so many pages push people to have at least

- Lower case
- Upper case
- Special character
- A Number

In the password. As the code is simple to guarantee at least one of each is in the password it is done that way.

One example of password entropy is the famous Correct horse battery staple

Password hints

Don't

Users will use them and enter some easy to research data.

Preventing copy & paste

Some developers had the idea to prevent copy & paste. Hoping that can prevent malware from accessing the password in the clipboard.

Maybe that **was** true. But current malware has many other tricks in addition. If malware is running on the system, it is game over.

The side effect of preventing copy & paste is that people will use shorter and simple passwords because their password stores will not work properly with your password entry.

And password cracking got a lot better thanks to improved computing power.

TL;DR: Do not block copy and paste

Forgot password

While creating an account enforce and verify a link to a mail address. This mail account is your "Trust Anchor" if the user forgets the password. Create a reset link and mail the user. Make sure not to spam the user. Do not revoke the old password until a new one is set.

Do not send a random password by mail but a (randomly generated) link to a page you have to set a new password.

TTL for passwords

If you force the user to renew the password regularly, store a hash of the last 3 passwords the user used and make sure they are not recycled.

Autofill trap

If you create a web page with a password form: Do not allow autofill without user interaction on your web-page password pages.

Malicious Ads are currently creating invisible forms to steal those passwords.

Princeton research

Stolen passwords

Malicious actors are hacking databases and extract the passwords from there. As users have the tendency to use the same password for different accounts this is troubling.

The project HaveIBeenPwned is collecting those stolen databases from the dark web and makes it searchable for your own credentials.

Hashes from there, cracked to 99%: Hash leaks

Besides locking down your database server you should also “hash and salt” the passwords you store. That way you prevent damage in case of a stolen database..

There are special passwords hashing algorithms that can be used.

Cracking passwords

The technology to crack passwords is quite advanced. Besides Rainbow tables, GPU based passwords cracking and Cloud Computing an old tool is word lists:

GPUs

With powerful GPUs and usable APIs available people started to use a cluster of those GPUs to crack hashes (later they shifted to Bitcoins).

This did beat the cracking capabilities of a CPU by a large factor. But people who invested into that had to specialize and offer hash cracking as a service to others. Reason for that is the investment into a bunch of the most powerful GPUs available (and re-investment a few GPU generations later).

Cloud

Cloud computing offers processing power at a discount (especially when going for the compute-spot market). Using thousands of Amazon VMs in parallel to crack some hashes can be the fastest and cheapest way. Investment into GPUs declined and people switch to cloud way of hash cracking.

Rainbow tables

Rainbow tables are attacker tools you should know about.

To crack hashes (and passwords are stored as hashes) attackers use **Rainbow Tables**. These are large databases with pre-compiled string->hash pairs. That way the attacker invests in storage, but reduces CPU power. If the attacker gets a large database dump with lots of Hashes, it is more efficient to match those two databases once and extract as many matching passwords as possible.

And this is just the Rainbow Table basics. Some more optimization is available.

Relevant for the defender: The attacker just reduced required CPU power and swapped it for HDD space requirements. Common passwords will be in this table and are easy to crack. Non-common passwords or *salted* passwords are better protected.

Word lists

Attackers use word lists to brute force passwords. It is more effective to use existing words and permutes them than to generate all potential character combinations.

Word lists are available online

- Hashpass
- Imsky wordlists
- berzerk0 wordlists
- nerdlist
- SecLists
- If you installed Metasploit, word lists are in */usr/share/metasploit-framework/data/wordlists*
- Assetnote wordlist
- Kali: /usr/share/wordlists

Cracking Archive files

Multi threaded password brute forcing for archives: RARCrack %% rarcrack –threads 12 –type zip crypted.zip

Cracking with wordlists:

```
1 fcrackzip -u -D -p simple_wordlist.txt test_abc.zip
```

Web site scraping for wordlists

CEWL

Cewl example:

```
sudo apt install cewl
cewl -d 2 -m 5 -w docswords.txt https://example.com
```

Keyloggers

Keyloggers exist as hardware and as software variants. Hardware variants are USB dongles attached between keyboard and computer - external and internal. They record keystrokes and are later collected. Some even have WiFi - reducing the risk for the attacker. Those things are simple and can be deployed by cleaning staff.

Software Keyloggers are drivers installed on the OS. Quite often by malware. They log keys and send their data to the mother ship.

One way to defeat Keyloggers are on-screen keyboards. But at least for the software Keyloggers they fail. Those programs do screenshots as well as logging the physical keyboard.

The positive effect of on-screen keyboards is limited. Please think twice if it is the proper solution for your specific security problem or if it is just **security theater**.

IoT: Initial passwords

Especially IoT devices and Routers have initial passwords. Some important wisdom:

- Force the user to change that on first use
- Do never derive it from MAC address or other system specific number
  - No fancy maths will protect you if you do. And MAC is broadcasted on WiFi
- Do not use a increment in the factory to initialise the first password
  - Add proper randomness instead

Salting

A **salt** is a random string/number.

This is added to the password, before it is hashed. The salt is stored together with the username and password-hash in the database - plain text.

By adding a random string/number to the password the user entered before it is hashed, we force the attacker to create a rainbow table for each of these salts.

Create a new random salt for every user. That way even if two users use the same standard password, the resulting hashes will be different.

Make the salt big.

Pepper

Pepper is similar to salt. But pepper is either stored in a different database or in a different configuration file - both the password database and the configuration file have to be hacked.

Or even not stored at all but is small. And then you brute force them when matching passwords to the database. When comparing the hash the computer will have to generate lots of hashes with *password + all possible peppers.*

You can afford the extra computing power - while the attacker trying to hack millions of your accounts can not.

Beyond passwords: “Two-factor authentication”

Technology that extends passwords is already a part of many products. In addition to passwords so called “two-factor authentication” is implemented.

A second factor - and a second communication channel - is required. This improves security.

The second factor is very often based on a hardware device.

Two major concepts are existing.

Time-based One-Time Password (TOTP)

The user has a device that shared an initial secret with the server. Based on this secret and the current time a short lived number is generated. The user enters this number in addition to the normal password to access the server. To verify the server just calculates the same data using the current time and the shared secret.

For the user it looks like an App on the smartphone that generates a 6 digit code. A countdown shows its validity (30 seconds are recommended in the RFC).

The best known app for that is the *Google Authenticator*

RFC 6238

Universal Two-Factor (U2F)

U2F uses a device with a asymmetric crypto implementation. It can export the public key and sign challenges with the private key - which is sealed inside of the device.

The public key is registered at the server. The best known device for U2F is the Yubikey which looks like a USB memory stick. But it can also offer NFC to be used with mobile phones.

FIDO U2F

Yubikey

Backup codes

As mobile phones tend to fall into a toilette or get lost somewhere else your backend will have to offer backup codes as well. Those are generated when the device is paired and can either be stored by the user or printed and locked away.

Those codes should be valid one time only and maybe you also want to offer several codes for download.

Plan a user experience to link a new device to the server using one of those codes. Also better plan a “refresh” button to invalidate the old codes and create new ones in case the printed codes get stolen.

Censorship

The original internet was planned with lots of resilience. Even if parts of the internet have issues, the rest will route around it.

Centralisation has added dependencies on a few companies. This results in accidental outages or planned censorship.

Both are totally contrary to the spirit of the internet. By engineering *resilience* into the software you can protect from outages.

If you are not building a system but administrating one, there are several tools that add redundancy to your current tool set. Especially by disconnecting from centralized services.

The tricky thing: You have to install those programs before the internet fails. And regularly use them to be sure they are operational in case of an emergency.

The programs you currently use will not have to be replaced. Just have those additional programs installed as well.

Mapping censorship

There are several projects mapping internet failures. This can be censorship (Turkey) or power outages causing internet downtime (Venezuela).

OONI by the Tor project

Ooni has globally distributed sensors measuring the reach ability of services. Data can be accessed on a map, by a list of “best of censorship” events and by an API.

The project is Open Source and there is a description how to set up a sensor on a Raspberry Pi.

The whole project is part of the Tor project.

Netblocks

Netblocks is run by a civil society group. Focus is more on written reports based on special internet outage events.

Blocked.org.uk

Censored pages in UK. UK has a voluntary filter for <18 content. This is over-blocking.

This site tests and monitors specific sites.

Censorship countermeasures

Basic countermeasures are based on “have a plan B”. Redundancy.

Messenger

Jabber + OMEMO

Jabber is a chat software. Jabber (aka XMPP) is like E-Mail: You can have an address at any Jabber server, use any kind of Jabber client to access it and contact everyone with a Jabber account.

OMEMO is the encryption technology for Jabber. Just activate it in your client and done.

There a different Jabber clients for example:

- Conversations for Android
- Gajim for Linux and Windows

Jabber is de-centralized and this makes it very resilient.

There is also no central directory of user names. Pro: No one is tracking you or your network. Con: You have to give your friends your Jabber address.

Signal

Signal is a free client for mobile phones with a very good encryption. The Con: There is a central directory. Pro: As soon as you register on Signal, all your address book contacts that also use Signal will be in your Signal-connections-list.

Similar to WhatsApp. But without Facebook.

Threema

Threema is a mobile phone communication app. It is cheap but not free. Also: It is closed source - but is getting security reviews. The encryption and the feature set are good and it is a very good messenger and simple to use. I especially like the (optional) key verification that forces people to meet in person and confirm their identity by scanning a QR code. Intuitive and secure.

Telegram

I am not using Telegram. It has several security issues. But on the other hand: Many people are already using it and it “survived” several censorship attempts already.

Use it but be aware: It is less secure to spying attempts than the other messengers listed here. But the widespread adoption has its own benefits.

Browser replacement

Tor browser

The Tor browser is hiding your tracks on the internet. Just install it. It will look like a Firefox browser with a few extra options. You can ignore them - the defaults are OK.

The Tor trick: Your data is sent encrypted over three "Tor nodes" to hide you from the internet.

Some very restrictive states limit the Tor browser. For those there are extra modes to exit their internet.

The Tor browser can access special ".onion" URLs. That is the Dark part of the web.

This may sound spooky. But even Facebook has a an entry door on the Dark side of the web. Facebook on Tor It is is not as big a thing as you may think.

Install the Tor browser and give it a try. You may not need it now. But maybe later.

Network infrastructure

In some regions the web coverage is under-developed. For some political reasons that is quite common in Germany. The Freifunk project is offering free WiFi under the SSID "Freifunk". Just connect and use the internet.

This project is a good alternative to company owned infrastructure and adds resilience.

If you are responsible for an area where people gather (restaurant, library, ...) think about contacting this project and running an own Freifunk router. Which is basically an AP with special configuration.

DNS

Domain Name System is the phone book of the internet. It converts URLs into server IP numbers (IPv4 or IPv6). Ripping entries out of the DNS pre-configured by your ISP is a simple way to censor specific services.

If it looks like that happened to you:

DNS settings are in your Operating System. Some big companies run their own DNS. Try changing the old DNS entries for one of those:

- Google: 8.8.8.8
- Cloudflare: 1.1.1.1
- Quad9: 9.9.9.9

(Yes, all of them were able to get iconic IPv4 addresses).

Blocking on DNS level happened at the "Censorship in Turkey" event.

File sharing

Using OnionShare it is possible to share files. Just drag some files into the box, click the “start sharing” button and send the generated URL to your communication partner. As long as the server is running the partner can download it by simply using the Tor browser.

Further reading

A large collection of alternative to centralised and surveilled services can be found on prism-break.org

Crypto algorithms

Crypto Algorithms are the basic building blocks of encryption. Above them are protocols ("how do we glue algorithms together to achieve our goal ?"). And above the protocols are libraries offering an easy programming interface to use a protocol.

In many cases the user of the library still has to pick the algorithm to slot into the protocol. A basic knowledge of the algorithms will prevent many bugs.

I will skip the mathematics and try to build simple "Good vs Bad" tables. If you want to dig deeper, please read the books in the links.

A good crypto algorithm is created and tested in challenges. It will get a good peer review by other crypto experts. When mathematics improves (and computers get faster as well) the *estimated time to crack it* will shrink. Moore's law predicts a increase of processing power by factor 2 every 18 months. GPUs, ASICs and Clouds (lots of cheap computers from Amazon Cloud, for example) can also bring down the attack time.

Normally experts will warn months or even years before a realistic attack on an algorithm is possible. That's the time to move on to use a better algorithm.

Be prepared to swap the algorithm used in your products and services

If you rely on a crypto book on your desk, check out the date it was published. During the last years new attacks have been invented and some algorithms and protocols did not age well. If your decisions still are based on a book from 1995 they will be wrong - even if the book was great at that time.

Be aware: This chapter is an extreme simplification. If crypto security is an essential part of your project dig deeper and use the *further reading* section.

Hash functions

Hash functions are one way functions that generate a value of a fixed size from any length of input. One input always generates the same Hash. On the other hand: One Hash can be the same for different inputs.

For security reasons it must be impossible:

- To revert the process and calculate the input from the Hash
- To generate two inputs with the same Hash

Name	Output Length (Byte)	Quality
CRC	*	Broken
MD5	16	Broken
SHA-1	20	Broken
SHA256	32	Good
SHA512	64	Good
SHA-3	variable	Good
BLAKE2	variable	Good

A CRC is not a Crypto Hash function. It is good for validating bit flips in data. But not to prevent or detect human attacks. Still some people use it that way…

shattered.io on breaking SHA-1

SHA256 and SHA512 are SHA-2 hashes. Where the number indicates the bit length.

SHA-3 aka Keccak: Is developed as fallback for SHA-2. SHA-2 is still good. But if something should happen to it, there is SHA-3 with is based on a totally different internal principle.

BLAKE2 was in the SHA-3 contest. Its main feature is speed. There are different variants optimized for different architectures. The core ones are:

- BLAKE2b: for 64 bit platforms
- BLAKE2s: For 8 to 32 bit platforms

If you need speed, give it a chance.

HMAC or MAC

MAC allows *Integrity* and *Authentication*. Simplified: It is a Hash with a shared key involved. That way the parties can authenticate the origin.

Name	Quality
Poly1305	Good
SipHash	Good

Password hash functions

While other hash functions are efficient, for password hashes we want inefficient hash functions. On a normal server it does not matter that much if 1 or 500 milliseconds are wasted on calculating the Hash. But the attacker wanting to break a database dump with hundreds of thousands Hashes gets into lots of trouble thanks to the in-efficient hash.

Name	Comments
Argon2d	GPU attack resistance
Argon2i	side channel attack resistance
bcrypt	Vulnerable to FPGA, ASIC attacks
scrypt	Vulnerable to GPU attacks
PBKDF2	Vulnerable to FPGA, ASIC and GPU attacks

Links:

- Argon
- BCrypt
- PBKDF2

Stream Ciphers

Stream ciphers are used especially in telecommunication to encrypt a data stream (mobile phone voice channel).

Name	Quality
A5/1	Broken
A5/2	Broken
RC4	Broken
Salsa20	Good

Block Ciphers

Block Ciphers are the work horse to encrypt large data. Besides selecting the Algorithm and the key length it is also important to use it in the proper mode. There will be an own table for that.

Algorithms

Name	Quality
DES	Broken
3DES	Mediocre
AES	Good

3DES is rated mediocre because AES offers a better key-length to protection ratio. You should be using that.

Modes

Block ciphers are encrypting the data block-by-block. The **mode** defines *if* and *how* the encryption of one block influences the encryption of the other blocks.

Name	Quality
Electronic Codebook (ECB)	Broken
Cipher Block Chaining (CBC)	OK
Counter Mode (CTR)	Good
Authenticated Enc. .. (AEAD)	Perfect

If possible use AEAD.

Some of those modes need more data in addition to the password to initialize the encryption: a number named *IV* (initialisation vector) or *nonce* (number used only once). This number can be public. But it has an important requirement: Do not recycle it ! This number must be used once only. For the next data you encrypt use a different one. With that requirement: A **counter** would do the job pretty well.

Authenticated Encryption

Combining a MAC and an encryption results in *Authenticated Encryption.*

The most common AE technology is *AES Galois Counter Mode (AES-GCM)*

One important differentiation is if to do MAC first or encryption. It seems during the years - as experience grew - the shift was towards “Encrypt-then-MAC” as best practice.

Name	Quality
Encrypt-and-MAC	Bad
MAC-then-encrypt	Better
Encrypt-then-MAC	Best

MAC and cipher must use distinct keys. But there are foot guns and alternatives to this whole AE complex.

If you go down that road, please check out “further reading” and at least 2 more books. Implementing this is lots of work (months-for experts). Invest your time into smartening up first.

Or use libraries implementing this technology ready-to-use (which I would do).

Asymmetric Ciphers

RSA: you should use 2048-4096 bit key length (security levels of 90 to 128 bits)

Encryption: Cipher text should be padded. RSA-OAEP does this (Optimal Asymmetric Encryption Padding)

For signatures with RSA use RSA-PSS(Probabilistic Signature Scheme)

Key exchange

To do a key agreement/key exchange the state of the art is DH: Diffie-Hellman protocol. It can be used in different flavours.

Name	Quality
Anonymous Diffie–Hellman	Broken
Authenticated Diffie–Hellman	Weak
Menezes–Qu–Vanstone (MQV)	Best but complex

- Anonymous DH is breakable by man-in-the-middle attacks
- Authenticated Diffie–Hellman: breakable by *replay* attacks
- MQV is best but complex

Most often used is *Authenticated DH*

Elliptic Curve Cryptography (ECC)

Elliptic curve based cryptography is an upgrade on many crypto protocols. It offers more security for less key length. And thanks to the smaller keys it is often faster.

If possible replace your RSA and DH with ECC.

EC crypto depends on one special parameter: the curve it runs on. The curve has to be special. You can not make up your own. Instead use one of the standard curves:

- NIST curves
- Curve25519 (which is very common)

Key length

Depending on your computing power, lifetime of your product and adversary this may vary. But sane best-practice key lengths are:

- Asymmetric: >= 3248 bit
- Elliptic Curve to replace classical asymmetric crypto: >= 256 bit
- Symmetric: >= 128 bit

A source for a more specific evaluation of your required key length is here.

Best practice

- Never encrypt without authentication: The cryptographic doom principle

Further reading

Serious Cryptography by Jean-Philippe Aumasson

Serious Cryptography is Mathematics paired with some hands-on. It contains lots of common mistakes being made when using those algorithms and protocols. It should be on your desk when building something with cryptography.

The Mozilla TLS guide

Mozilla TLS guide is a collection of TLS server settings for different situations. A good cheat sheet ranking the current quality of algorithms in a pragmatic (“don’t break the web”) environment.

BetterCrypto.org

BetterCrypto is a project aimed at helping admins. They offer a free manual with lots of specific crypto settings for different applications. It also contains basic algorithm overviews.

Programming

Intro

Programming bugs turned into vulnerabilities since…it feels like forever. With the hacker paper Smashing The Stack For Fun And Profit - which was release somewhere in the 90s - game got more tricky.

As C and C++ are very close to hardware and can mess with memory by design the issue has never been solved properly. And those languages are still around (for good reasons).

The good news: This kind of bugs and similar ones can be avoided, detected, prevented and mitigated. There are tools for that. This chapter covers them.

Requirements for code analysis tools

There is a long list of tools supporting you during the development cycle. Some have specific requirements - and you will not be able to use them in your situation.

Some of my projects are embedded projects - the requirement “compiles on Linux” is quite hard when the release target is a micro controller without an OS.

An option here is to at least test the part of the code that are not hardware depended on Linux.

- Flawfinder: static code analysis
  - Pro: No code modification required
  - Pro: No Linux compilation required
  - Pro: No execution on Linux required
  - Con: No detailed analysis
- Assert: debugging tool
  - Pro: No Linux compilation required
  - Pro: No execution on Linux required
  - Pro: Force multiplier for other technologies
  - Con: Code modification required - debug build only
- Fuzzing: Finding unexpected bugs
  - Pro: No code modification required
  - Con: Linux compilation strongly suggested
  - Con: Linux execution strongly suggested
- Unit tests
  - Pro: Linux or Windows for execution
  - Pro: Compilation for Linux and Windows
  - Con: Code modification required (modular design of code plus writing of unit tests)
- CppCheck: static code analysis
  - Pro: No code modification required
  - Pro: No execution required
  - Pro: More detailed analysis than flawfinder
  - Con: Special build target required
- Clang, static: static code analysis using Clang
  - Pro: Does not require code modification
  - Pro: Does not require execution on Linux
  - Con: Requires special build
  - Pro: Very detailed static code analysis using compiler internal features
- Valgrind: dynamic code analysis (memory as a speciality)

  - Pro: Does not require code modification
  - Pro: Does not require re-compilation
  - Con: Linux execution only
  - Con: Focus is on memory issues
  - Pro: Can run the release binary !
- Clang, dynamic: dynamic code analysis using Clang
  - Pro: Does not require code modification
  - Con: Requires re-compilation
  - Con: Linux execution only
  - Pro: Very powerful dynamic code analysis

Some background

There is a large number of fuzzing and dynamic code analysis tools for Linux. Being able to compile at least core parts of the code and execute them on Linux will make your life much simpler.

The two things that require code modification only modify special debug code. This way you will not mess up your release build.

Unit tests can be run on Linux or Windows. They will even run on embedded systems. But there you will have to handle memory space issues (the binary will be huge) and redirect the report to UART. I would define that as a bonus goal after the unit tests run on Linux or Windows.

Testing

Intro

Testing is the last line of defense before your product is shipped to the customers. Having the ability to find the vulnerabilities here has serious benefits:

- Reduces emergency updates
- Lets everyone sleep better
- Creates a more relaxed development cycle

Testing for vulnerabilities is part automation (=engineering) and part hacking. As it is using existing tools and not looking for new ways to hack systems it does not require 1337 hacking skillz but still kicks the door open to the interesting world of hacking.

Flawfinder

Flawfinder

Flawfinder is a simple static code analysis tool. One of the “grep on speed” kind. It is simple - it should be one of the first tools to use to verify your code. One nice thing: It puts the important messages first. Start checking the code from top to bottom.

As it is simple it will not find all vulnerabilities. You should continue with other tools and code-reviews even if flawfinder gives you green lights.

But it will guide you to areas of your code that are smelly.

Flawfinder has two important modes:

- Finding smelly code that could have vulnerabilities
- Finding code that handles inputs

Finding issues

Fuzzgoat is a vulnerable C program to test your fuzzer.

Running flawfinder on Fuzzgoat results in this kind of log:

```
flawfinder .
Flawfinder version 1.31, (C) 2001-2014 David A. Wheeler.
Number of rules (primarily dangerous function names) in C/C++ rule set: 169
Examining ./main.c
Examining ./fuzzgoatNoVulns.c
Examining ./fuzzgoat.c
Warning: Skipping directory with initial dot ./.git
Examining ./fuzzgoat.h

FINAL RESULTS:

./fuzzgoat.c:1049:  [4] (buffer) strcpy:
  Does not check for buffer overflows when copying to destination (CWE-120).
  Consider using strcpy_s, strncpy, or strlcpy (warning, strncpy is easily
  misused).
./fuzzgoatNoVulns.c:928:  [4] (buffer) strcpy:
  Does not check for buffer overflows when copying to destination (CWE-120).
  Consider using strcpy_s, strncpy, or strlcpy (warning, strncpy is easily
  misused).
./fuzzgoat.c:368:  [2] (buffer) memcpy:
```

```
  Does not check for buffer overflows when copying to destination (CWE-120).
  Make sure destination can always hold the source data.
./fuzzgoat.c:401:  [2] (buffer) sprintf:
  Does not check for buffer overflows (CWE-120). Use sprintf_s, snprintf, or
  vsnprintf. Risk is low because the source has a constant maximum length.
./fuzzgoat.c:427:  [2] (buffer) sprintf:
  Does not check for buffer overflows (CWE-120). Use sprintf_s, snprintf, or
  vsnprintf. Risk is low because the source has a constant maximum length.
./fuzzgoat.c:444:  [2] (buffer) sprintf:
  Does not check for buffer overflows (CWE-120). Use sprintf_s, snprintf, or
  vsnprintf. Risk is low because the source has a constant maximum length.
...shortened...
./main.c:135:  [2] (misc) fopen:
  Check when opening files - can an attacker redirect it (via symlinks),
  force the opening of special file type (e.g., device files), move things
  around to create a race condition, control its ancestors, or change its
  contents? (CWE-362).

ANALYSIS SUMMARY:

Hits = 49
Lines analyzed = 2545 in approximately 0.04 seconds (69403 lines/second)
Physical Source Lines of Code (SLOC) = 1817
Hits@level = [0]   0 [1]   0 [2]  47 [3]   0 [4]   2 [5]   0
Hits@level+ = [0+]  49 [1+]  49 [2+]  49 [3+]   2 [4+]   2 [5+]   0
Hits/KSLOC@level+ = [0+] 26.9675 [1+] 26.9675 [2+] 26.9675 [3+] 1.10072 [4+] 1.10\
072 [5+]   0
Dot directories skipped = 1 (--followdotdir overrides)
Minimum risk level = 1
Not every hit is necessarily a security vulnerability.
There may be other security vulnerabilities; review your code!
See ([http://www.dwheeler.com/secure-programs](http://www.dwheeler.com/secure-pro\
grams)) for more information.
```

Finding inputs

As inputs into a program must be verified, filtered and sanitized it is a handy feature of flawfinder to search for code offering inputs.

Use that tool and spend some time writing code to verify the input data.

```
flawfinder -I .
Flawfinder version 1.31, (C) 2001-2014 David A. Wheeler.
Number of rules (primarily dangerous function names) in C/C++ ruleset: 169
Examining ./main.c
Examining ./fuzzgoatNoVulns.c
Examining ./fuzzgoat.c
Warning: Skipping directory with initial dot ./.git
Examining ./fuzzgoat.h

FINAL RESULTS:

./main.c:142:  [0] (input) fread:
  Function accepts input from outside program (CWE-20). Make sure input data
  is filtered, especially if an attacker could manipulate it.

ANALYSIS SUMMARY:

Hits = 1
Lines analyzed = 2545 in approximately 0.04 seconds (70448 lines/second)
Physical Source Lines of Code (SLOC) = 1817
Hits@level = [0]   1 [1]   0 [2]   0 [3]   0 [4]   0 [5]   0
Hits@level+ = [0+]   1 [1+]   0 [2+]   0 [3+]   0 [4+]   0 [5+]   0
Hits/KSLOC@level+ = [0+] 0.550358 [1+]   0 [2+]   0 [3+]   0 [4+]   0 [5+]   0
Dot directories skipped = 1 (--followdotdir overrides)
Minimum risk level = 0
Not every hit is necessarily a security vulnerability.
There may be other security vulnerabilities; review your code!
See 'Secure Programming for Linux and Unix HOWTO'
([http://www.dwheeler.com/secure-programs](http://www.dwheeler.com/secure-program\
s)) for more information.
```

Tools

Intro

Using tools is one of the things that is typical for the human species [^wrong]. Tools make us doe more things. In a more reliable way. Have “old” stuff automated away and open us to do new things. They reduce human error.

Many tools did not fit in other chapters. But they are part of our daily work with computers. And they contribute to the daily workflow of building software.

Those tools are covered in this part of the book.

[^wrong:] This is wrong. Even birds can do remarkable things and even build their own tools…just follow the biological and cognitive science there. This does not invalidate my point. Be smarter than the chicken wings on your plate.

Bolt on

Intro

There is a rift between security engineer. Those who can plan and build a system despise what they call “snalá±e oil”. It adds complexity (which is bad) and has no perfect guarantee for security (protection is statistics with a detection rate of 99.89%).

Some security engineers are not so fortunate and have to deal with a system that is flawed and buggy and they can not change. For technical or political reasons (manager want to use an outdated internal web based tool that is not supported anymore and only works with and IE from last century). Their only hope is *bolt-on* security tools to manage a deprecated system which is already way to complex.

This chapter deals with the bolt-on side - which commonly uses detection of attacks.

As security can be a stressfull job, this intro closes with “Be excellent to each other”

Anti Virus tests

Abstract

Modern anti-virus tools are a combination of hash databases, pattern matching, behaviour blocking, cloud technology and sandboxes. At least.

Testing them is tricky.

Malware has a short life span. New malware is generated every few minutes (this is called *server side polymorphic*).

Testing malware is tricky as well.

A testing lab needs access to large heaps of current malware, lots of virtual machines, verification tools to identify if the malware is a dud or dangerous, verified clean samples, …

This is the reason it requires teams, racks of computers, terabytes of malware and cleanstuff (for FP tests) and years of experience.

Computer magazines normally let external experts test the AV software for detection and protection and add their own usability and speed tests on top.

External experts are

- av-test
- av-comparatives
- virus bulletin
  - Which is a AV industry internal organisation

You can get some (not very detailed) raw data from those pages directly. Observe how the different tools develop over time, how they currently achieve, how their detection relates to their False Positives.

Also check out who is **not** participating in those tests…. this could be a red flag.

Methods

Basically there are two kinds of tests.

Classic scan

A large folder containing historic and current malware is scanned. The percentage of detected malware and maybe the speed of the scan are the output.

The results will be biased because the focus is on historic malware. The dangerous - current - malware is under-represented.

The tested AV products can not use some of their protection tools like URL block lists, behaviour detection, cloud technology, …

This is not used that often anymore.

For that reason this kind of tests is replaced by:

Real-world protection tests

Current malware is used to “infect” machines where a full AV product is running. The antivirus can use all its features to protect the machine.

- URL block list
- Static file scanning
- Cloud scanning
- Behaviour analysis
- Sandbox

The results of this test are much more significant but the test is harder to run and the malware sample set will be smaller.

This kind of tests is the future.

DIY malware tests

Some people do malware tests (by modifying malware, finding some stuff on the internet, …) and do their own testing. Especially patching malware samples to test with “new and unique” samples is a critical thing. Please do not do that. Patching the malware sample could have broken it: it is not running properly and is causing no harm. **Not** detecting them is the right thing.

Be careful when you see DIY malware tests and always verify with other sources.

Eicar

What you *can* and *should* tests is if you antivirus is properly attached to your system.

Basic questions should be:

- Is the proxy intercepting web traffic to scan it ?
- Is the proxy scanning mails when I fetch them with the mail client ?

- Is the AV scanning files when they are stored on hard-disk (called: On-access scanning)
- Is it scanning when I execute a file ?
- Is it scanning when I copy a file to a USB stick ?

Exactly for these kind of tests there is the EICAR test file. Every AV should detect it. And it is easy to detect. It was created for testing exact this kind of situations mentioned above.

EICAR test file

It is just a few bytes long and you can copy & paste it.

Move it around your system and test which entry points are not protected by your antivirus scanner.

Windows Defender

Windows Defender - pre-installed on Windows - is a kind of fallback if no other AV is installed. It improved its detection rate over the last years and already offers a decent baseline. There are still good reasons to install 3rd party scanner. But from an engineering perspective you can expect:

A current Windows with Defender has a decent scanner already installed

There are many drawbacks of having only the Defender as protection: Malware could start to hide from Defender as priority number one. As it is becoming the most common AV product. And this in turn can weaken the real protection Defender is offering. A typical mono culture effect. It is better not to build a mono culture on your systems.

Appendix

External references and resources

In addition to the references spread in the chapters - most often as “Further reading” - here is an extended list of hacking and security resources.

It is important that you rate those resources for yourself. I have my personal categories.

- ALERT: Resources being fast on alerting. But I do not expect in depth coverage
- DEEP: Slow (months after the incident) but incredibly deep coverage
- SIMPLE: Resources to share with non-tech people
- KNOWLEDGE: General knowledge covering specific technology. Not a source for threat intelligence

All of them have their benefits and you should always use the proper source for the task at hand.

Books

Book: Practical Internet of Things Security

Practical Internet of Things Security by Drew Van Duren, Brian Russell

A very methodical book covering

- Threat modelling
- Design
- Life cycle
- Basic Cryptography
- Authentication/Authorization
- Compliance monitoring
- IoT Incidence Response

A good overview.

Potential improvement: Go deeper into details and specific technology. For example: Crypto chapter is a good start but really using crypto will require more books to be read. Says the one writing a “security overview book”…I know….

Book: “Test driven development for embedded C”

Test Driven Development for Embedded C by James W. Grenning

A pragmatic and step-by-step approach how to develop for embedded systems in C and still benefit from unit tests.

This book will improve your software quality and security.

The Browser Hacker's Handbook

The Browser Hacker's Handbook by Michele Orru , Christian Frichot , Wade Alcorn. Essential knowledge how to hack and secure a browser. Lots of focus on Beef technology.

Book: IoT Penetration Cookbook

IoT Penetration Cookbook by Aaron Guzman, Aditya Gupta

Wide spectrum of offensive technologies. Good to create a checklist what to protect. Be prepared to read more in-depth books for the technologies you are interested in.

Book: Pragmatic Thinking and Learning

Pragmatic Thinking and Learning, by Andy Hunt

Explains your most important tool as a computer expert: Your brain. How to use it, how to learn. A very important book for me. Written for the engineer.

Driving Technical Change

Driving Technical Change, by Terrence Ryan

A book explaining how to introduce new technologies in a team or company. Simple to read but with very essential tricks for the tech guy how to get mental boulders blocking change out of the way.

Anti Patterns

Anti Patterns, Refactoring Software, Architectures, and Projects in Crisis by William H. Brown Raphael C. Malveau, Hays W. "Skip" McCormick III, Thomas J. Mowbray

A book showing typical anti-patterns that just happen in larger (software) projects. Your project will also suffer from some of those. Read this to understand those patterns and learn how to eliminate them.

Conferences

An important criterion for conferences added here is online publication of talks and papers.

When you know where to look you will find lots of awesome conferences with different priorities and a different focus.

CCC

The Chaos Computer Club has several conferences in Germany. Those have different focus and size. If you are interested in computer security, the MRMCD and the Congress are the most valuable ones for you. Talks are recorded and can be found on media.ccc.de

Usenix (scientific security conference)

Usenix is a high tech security and academic conference. Bleeding edge technology but a bit harder to get it into a product than with other conferences.

Usenix 2016

Usenix 2017

Usenix 2018

Usenix 2019

Usenix 2020

Hack.lu

Hack.lu is a security conference in Luxembourg. Talks are recorded, but it seems there is no central storage besides YouTube where you can find them.

OWASP

- German OWASP Day 2019

Blackhat

Blackhat is a security conference with events in US, Europa and Asia (so don't get confused).

- Blackhat USA 2019 Playlist
- Blackhat Europe 2019 Playlist
- Blackhat Asia 2019 Playlist

Shmoocon 2006 - 2020

Shmoocon

- 2006
- 2007
- 2008
- 2009
- 2010
- 2011
- 2012
- 2013
- 2014
- 2015
- 2016
- 2017
- 2018
- 2019
- 2020

Defcon

- Defcon 26
- Defcon 28
  - Main stage on youtube
- Defcon 29

BSides

BSides

BsidesSF 2021

[BSides TLV 2021] (https://www.youtube.com/playlist?list=PLkNlAwTF5yEvS0IDS8zRanqUfTWxadcV4)

Radare Con

2020:

- https://github.com/radareorg/r2con2020

Purdue University seminar

Purdue

Virus Bulletin

Virus Bulletin is an Anti-Virus community magazine, tester and conference.

Especially the conference has in-depth talks covering malware and malware actors. Some of those talks can be found on YouTube.

Virus Bulletin videos

Blogs

Blogs are a good resource for in-depth malware analysis or content of almost any complexity. I suggest you check out which ones cover topics you are interested in and then start monitoring them for changes.

Security perspective from a bank, check out articles, blogs, handbooks, white-papers: Bank indosecurity

Deep analysis and IOCs: Cylance

Team Cymru blog covers threat intelligence.

Malware must die blog contains malware information

Malware don't need coffee focuses on Exploit kits and their distribution.

Cyber crime Magazine focuses more on the crime aspect.

Internet Storm Center

Cloudflare

HITBSecNews

Citizen lab - big game hunting

Also Brian Krebs is very well known for his investigations into the crime part of the topic.

To stay in the world of crime: Europol will also cover it. But not focused on computer security.

The Anti Virus companies also have blogs. Sometimes several blogs. With a focus on end-users or tech nerds. Those can vary in depth.

- Emsisoft
- Kaspersky
- Kaspersky technical
- Bitdefender
- Eset
- Malwarebytes

Just to name a few.

News

Hacker news portals:

- The hacker news
- CNET security
- Bleeping Computer

Podcasts

- Security Now
- The Silver Bullet
- Darknet Diaries

Magazines

Magazines (called Zines) are a part of the hacker culture. And they still exist.

- Pagedout magazine
- POC or GTFO Also available on other mirrors and as books
- Spuz.me pen testing

Videos

- “Administraitor” talks
- Hak5

Workshops and Training

- Open Security Training
- Reverse Engineering for Beginners

CTF

Capture the flags (CTF) are challenges to test your own skills in hacking. A good way to do that without committing any crimes.

- Gruyere CTF
- Vulnhub
- CTF by Samsung
- CTF list

Background and some help:

- Awesome CTF

There are also CTF competitions:

- https://ctftime.org/

Lists and bookmarks

- Offensive security bookmarks (2015)
- Security resource list

Glossary

0-day: An exploit used by the attackers before the vulnerability is known to the defender. They are expensive on the black market. Typical malware does not need 0-days. As many users do not patch their systems.

ABAC: Attribute based access control. A logic decides based on attributes of different objects (subject, resource, action, environment) if access is permitted.

Adware: Malware that shows Ads. As the Ad ecosystem generates money for viewed Ads this is how attackers convert infections to money

Amplification attacks: Are attacks where a 3rd party is used to attack the target with amplified bandwidth. Services emitting more data than they receive are vulnerable. Often uses stateless protocols like UDP because the initiator is different from the victim.

Anycast: Servers share the same IP address. The client connects to the closest one. Relevant if only one of them is hacked and serves malware…

Approved List: A list of url/hashes/… that got approved as being harmless

APT: Advanced Persistent Threat. Advanced malware. Often government backed. Or just very advanced malware.

ASLR: Mitigation strategy to load code into **random** areas of the memory. Makes writing of exploits harder

ASVS: Application Security Verification Standard: This OWASP project lists different ways to verify the security of an application.

Backdoor: A constructed way for an attacker to connect to a vulnerable system and execute code

Beacon: The C2 server on the public internet which is contacted by the implants on the victims local network. Connection is usually done in a regular interval. Depending on the malware this can be from minuets to months. Using this short beacon messages the implant fetches new instructions

Blacklist: Deprecated, see "Block List"

Block List: A list of items like URLs, hashes, … that trigger a blocking

BLE: Bluetooth Low Energy

BLE Characteristic: Value + Descriptor

BLE GAP: Advertisements, connection handling, defining device roles in communication

BLE GATT: Organizes the data offered by the device

BLE L2CAP: Encapsulating data into packets

BLE Service: Several Characteristics combined create a service. A default service for BLE would contain device information

BLE Profile: Profiles combine several services. They standardize specific device classes

Bot: An infected machine that can be remote controlled by the attacker

Botnet: Infected machines centrally controlled by the attacker using a C&C

Bug: Error made in the programming phase. Not all bugs are vulnerabilities

Campaign: Malware campaigns are events where a malware actor spreads malware by phishing, infected sites, … As malware is a business with division of work, one malware type can be spread by different actors in separate campaigns in parallel

Canary: A trigger that will alert in an intrusion attempt. Code canaries for example are values in the code that trigger when being overwritten

CAPEC / Common Attack Pattern Enumeration and Classification: A MITRE taxonomy for attack patterns. https://capec.mitre.org/

C&C aka C2: Command and Control server. Central control over a botnet

Class break: A Class break is a vulnerability that breaks the whole device class. Instead of just one single device. Example: Same master key on all devices leaks

CORS/ Cross-origin-resource-sharing: Mechanism to define exceptions for the SOP (Same-origin-policy) in the HTTP header

CERT: Computer Emergency Response Team. Handles security breaches in organisations or nations.

Command injection: If the attacker is able to inject a command to the system shell (and the data is not sanitized) this is a command (line) injection. See OWASP

Common Platform Enumeration: A unique ID for software products, can be used to query vulnerability databases and software asset management

CRITs / Collaborative Research Into Threats Threat intelligence sharing platform by MITRE. See TIP. https://crits.github.io/

CSRF aka XSRF - Cross site request forgery: The attacker tricks the user to issue a HTTP requests on a site where the user has an open session. This HTTP request triggers an action on the server on behalf of the user.

Dark Web: Intentionally hidden parts of the web. Most common TOR pages

DAST: Dynamic Application Security Testing. Testing by executing the program in a special test harness. Maybe compiler flags like ASan

Deep Packet Inspection: A filter looks into the deeper layers of packets. Complex, and causes issues with encrypted data. Better don't do it.

Deep Web: Parts of the web, that is not indexed by search engines (intranets, everything behind a login, ….)

Dedik: Stolen RDP access to a windows server. Used as first step for Ransomware attacks (2020: 70% of Ransomware attacks started that way). Dedik = Dedicated server

DEP: Data execution prevention. Parts of the memory are marked as non-execute (=NX)

DFIR: Digital forensics and incident response

DGA / Domain generation Algorithm: An malware algorithm generating domain names based on a counter/time to connect to.

Differential privacy: Is a way to collect statistical data on clients/users without compromising the privacy of specific users

Domain Flux: Way to hide C&C servers. A domain name generator creates changing domain names

DOS: Denial Of Service - Attack that exhausts resources of the target (CPU, memory, network, storage)

DDOS: Distributes DOS - Attack is done by many systems in parallel

DNG / Domain Name Generator: Part of a malicious script generating possible URLs for C&C servers by time. By registering some of them the attacker can reduce take downs and still get connecting bots.

DTLS: Datagram TLS - TLS for short UDP messages (see: IoT)

DVA / Damn Vulnerable Application (Also: Damn Vulnerable Web application): A insecure application to train hacking on or test tools

Dynamite phishing: Automated spear phishing. The malware silently collects mail communication to later fake new phishing mails for new victims

EDR /Endpoint Detection and Response: Endpoint protection with a deeper analysis system and an array of potential responses to attacks

ELF: Linux executable file format

EMET: Enhanced Mitigation Experience Toolkit for Windows hardening. Now integrated into Windows Defender Exploit Guard

EPP / End Point Protection: Passive Endpoint protection

Evercookies: Are snippets stored in the browser like cookies. Storage is JavaScript Local storage, Flash storage, ... hard to delete those data snippets. AKA Zombie Cookies

Exploit: Attack on a vulnerability

Exploit Kit: A ready to run malware spreading tool. Normally uses infected web pages to spread the exploit.

Fast Flux: Using Round-Robin DNS to hide a C&C server. One URL gets multiple IPs that way.

Flaw: Error made in the design phase. Not all flaws are vulnerabilities

FP: False Positive. A harmless URL or file is detected as malicious

Firewall: Network filter. Can reduce allowed traffic on a network connection. Smarter firewalls can maintain some state and decided based on who initiated a connection - internal or external computer ?

Heap Spraying: Loading the exploit code several times into different memory areas. Also uses *NOP slides*

Honeypot: A Honeypot is a simulated vulnerable system to attract attackers. The attacks will then be analysed to learn about the attackers.

HVCI: Hypervisor Protected Code Integrity, aka “memory integrity”, a Windows feature HVCI

IDS / Intrusion Detection System: Monitor in a network to analyse that for malicious activity. Using rules it can notify admins if anomalies are detected

IPS / Intrusion Prevention System: IDS with the ability to automatically stop potential attacks (firewall up, terminate processes, …)

Implant: A tool used by the attacker to maintain access to an exploited system. Normally has some form of connection to the attacker’s infrastructure to be controlled

IOC: Indicator of Compromise. A file-hash, URL, registry key or similar feature left by the malware when a system is infected.

JWT: JSON web token. A “cookie” for authentication with added data. Stored on the client. The JWT is signed by the server.

Kerckhoffâ€™s Principle: A cryptosystem should be secure, even if everything about the system, except the key, is public knowledge.

Key rotation: Periodically change the encryption key

Kill chain: Phases of an attack

Living off the land: A tactics where the attacker uses pre-installed software for the attack. This reduces detectable malware on the system

MAEC5: Malware sample description language. Covers static features like hashes and behaviour. MAEC5

MBEC: mode-based execution control: Windows memory integrity feature, requires CPU support

Malware: Malicious software

MaaS / Malware as a service: Organised crime split malware creation and using malware against a target into different roles. Malware creators offer malware “as a service” to the second group

MISP: Open standard for threat information sharing. Also a sharing platform. Focus on IOC. MISP. See also. TIP

MISRA C: Guidelines for safe C programming. Tools (compiler and static code analyser) support it. Some of the safety issues are also security issues. Check it out. Low hanging fruit.

MITM: Man/Monster in the Middle. Network connections are broken up in the middle by the attacker to spy or modify network packets

MITRE: Non profit organisation doing security research

MITRE Att@ck: A taxonomy for attacker TTPs. see: MITRE

mutual TLS / mTLS: Both sides, client and server, authenticate to each other using certificates

NOP slide: Increases the target the instruction pointer has to hit to get to the exploit code. Done by adding NO operation codes to the beginning.

Open redirect: A vulnerability where a legitimate website can be tricked into HTTP redirecting a visitor to a url supplied by the attacker

Patch gap: Gap between a security patch in a 3rd party library and a patched release of the main product using this library

Patch Guard: Windows kernel patch protection, protects some memory structures from being modified

PE: Portable Executable file. On Windows .exe and .dll (and some others). Starts wit a DOS Header (MZ)

Persistence: Attackers can do a hit and run - stealing data - or try to maintain access to the target by running implants there or getting the credentials. That way the "attack survives a reboot". This would be called "persistence".

Persistence(system): System level persistence can be gained by running code on a single system. Often done as a hidden rootkit in kernel mode

Persistence(network): Persistence by controlling several systems by either stolen credentials or compromise of a core infrastructure in the network.

Phishing: Attack using social engineering to get credentials. Can involve malware or exploits as well

POLA / Principle Of Least Authortity see POLP

POLP / Principle Of Least Privilege: Users are granted the smallest set of privileges they need to get their assigned tasks done

POMP / Principle Of Minimal Privilege: see POLP

QUIC: A Google protocol based on UDP to replace TCP. By adding TCP features to UDP

RAT / Remote access trojan: An implant to gain persistence. Can remote control OS functionalities.

RBAC / Role based Access Control: Every user has a role. Those roles come with permissions

RBN / Russian Business Network: Rogue ISP offering bullet proof hosting (take downs by police not easy)

RCE: Remote Code Execution. An attacker is able to execute code without physical access to the system. Very bad.

Regular Expression DOS: Non optimized regular expressions can require lots of CPU. If the attacker is able to submit an own Regex, it is possible to DOS the target system OWASP

Reproducible Build: The software build chain must create hash-identical software on different build systems. That way some supply chain attacks can be mitigated.

Reverse shell: A command shell running on a hacked PC. To get out through NAT/Firewall it connects back from the infected PC to a server the attacker controls.

ROI: Return on investment. A business metric. But as attackers want to make money, make them pay for it by detecting their tools, sharing this knowledge. And make it very expensive for them to attack.

ROP / Return Oriented Programming: If an executable is compiled with DEP it is not possible to insert new shellcode to execute. ROP uses *existing* code from the victim-executable and chains those ROP gadgets into a new order to achieve the goal.

Round-Robin DNS: Old system for load balancing that does not work with modern clients or IPv6. TTL is short lived.

SafeSEH: A windows compiler feature to mitigate attacks abusing the Exception Handler. Attackers can modify the the exception table. The cause an exception. By that they then can control program flow. SafeSEH mitigates that.

SAST: Static Application Security Testing. This can be compiler warnings or special tools

SBOM Software Bill of Materials

SCA: Software Component Analysis: Check external software components for vulnerabilities and license compliance issues

Self guided malware: Malware for air-gapped systems. Does not need a connection to a control server. Has all the propagation/infection logic built in. Not a common thing. But Stuxnet was one of those.

Session fixation: Vulnerability caused by not creating a new session on login but recycling an old one. The attacker injects his token into the victims browser. As soon as the victim logs in, the attacker has parallel access to the account.

Session Hijacking: The login token of a session is stolen. While this is valid the attacker has access

Shellcode: The payload in an exploit. Shellcode either pops a shell or establishes a connection to the attacker in the internet, … This is where malicious things happen.

Side Channel: Using side effects to hack a system. For crypto a side channel can be timing differences between a right and a wrong password.

SIEM / Security Information and Event Management: Collect and analyse different security relevant data into logs. Actions are triggered based on this data.

Sigma rules: Sigma rules are for logs what Yara is for files https://github.com/Neo23x0/sigma

SKF: Security Knowledge Framework: OWASP expert system to build and verify secure software https://owasp.org/www-project-security-knowledge-framework/

SNI / Server Name Indicator: Part of the header to indicate which one of the hosted servers to contact. Could be encrypted if ESNI (Encrypted SNI) is used.

SOC / Security Operations Center: The team running SIEMs in an organisation.

SOP / Same-origin-policy: Web browser security principle: Elements of a web site must be from the same server

Spam: Unwanted Advertising mails. Or similar messages on other channels. Often sent from hacked PCs to abuse their bandwidth.

Spear Phishing: Targeted phishing. Learn about the target first to create better social engineering attack

SQLI / SQL Injection: SQL *commands* can be sent through the user interface to the database backend. Instead of just *values* (which was the developer's intention)

Stack Clash: A vulnerability on Linux/BSD systems. Stack Guard Page should protect against Stack-Overflows. But it can be tricked into overwriting memory. A compiler switch can protect against Stack Clash.

Static Code analysis: Analyse code without executing it. Compiler warnings are the best known example. Also linters.

STIX2: A data format to exchange threat intel. More focused on Attack Groups/Identity/Threat actor/Campaign. STIX2

StrongNaming: Authentication for .NET libraries. Signed binaries with version verification and pinned key.

Sub domain hijacking aka sub domain takeover: The attacks is able to claim an abandoned web host with still existing DNS entries. Now it is compromised….

Supply chain attack: The attacker manipulates a software, library or compiler used by the victim to attack the victim or the victims's customer

Suricata: Open Source IDS Suricata

Threat intelligence: A service offered by security companies (or peer-to-peer). Can start with sharing IOCs and end with detailed reporters on attacks and threat actors. Focus is on Advanced attacks

TIP / Threat intelligence platform: Threat intelligence sharing between organisations. See MISP or CRITs by MITRE

TLP / Traffic Light Protocol by DHS: defines threat intelligence information sharing policy. From public to secret. Using 4 coloured levels

TOR: The Onion Router. Overlay network over the internet to ensure anonymity of clients and servers

Trust boundary: Who do you trust with what ? If your project integrated 3rd party compilers, tools, library, … you are already trusting someone. Knowing *who you trust* and *how far* is the trick.

Trusted Computing Base (TCB): The part of a system that **MUST** work properly to ensure security. Keep this small.

UAF / Use After Free vulnerability: Typical memory corruption bug

VBS. Virtualization Based Security, a Windows feature to isolate parts of the memory from the OS. VBS

Vulnerability: A bug or a flaw that has security implications

Warning fatigue: A psychological aspect for secure UI design. If you display to many warning UIs your users will be trained to ignore them.

Waterhole attack: Targeted phishing attack using a hacked/manufactured homepage the victim is known to visit and trust

Weaponizing: Making an exploit easy and reliable to use.

Whitelist: Deprecated term. See "Approved List"

XSS, Persistent: An attacker can store a script on the web server that is for another user rendered. As it is a script it will execute in the context of the victim's page.

XSS, Reflected: The attacker can add a script to a url parameter which is part of the rendered page. This script will then run in page context

XXE: XML External Entity injection: The attacker uses modified XML sent to the server to access internal data like files

Zero Trust: All network requests must be authenticated. Even those originating from the own network

The author

I have to tell you my origin story. Because you should know where we are going if you join me by reading this book. I hope it is not boring as no radioactive spiders are involved….

Trouble shooter and one-man-team somehow happened to be my role. This and team expert for security. I am a software developer and engineer who also took the role of architect/project manager in security related projects. Which gave me a good and wide perspective on things in the IT world.

After studying Computer Science (Dipl. Ing FH at University Ravensburg Weingarten, a German title) I went to Avira. An Anti-Virus/Endpoint bolt-on-security company. I handled core detection projects as part of teams. I was focused on the engineering/architect and developer roles. Amongst the things I did is:

- An AI SPAM filter in C: String processing in C….
- A full Anti-Virus engine. Cross-compilable. For WinCE, Linux, PalmOS and Symbian. C
- A generic module to detect malware in homepages. Building a kind of DOM. String processing. In C.
- Management security consulting
- Browser extension development - self learning phishing detection. JavaScript
- A government founded research project:
  - Split an OS into several virtual machines for segmentation
  - Scan into these virtual machines without installing anything (Volatility)
  - Classify malware based on behaviour (Cuckoo Sandbox)
- Create the architecture of a security/privacy focused Chromium based browser
- Went to the Embedded and IoT world at Feo, another company
- Mvoing to Avast I wrote a simulation environment to experiment with advanced attacks named PurpleDome

Right in the middle of doing all those things I started to sort my knowledge and experience. Resulting in this book.

Currently I am Lead Security expert at Primion where I can use all my knowledge.

You can reach me

- On Twitter: @ThorstenSick
- On Mastodon: @thorsi@chaos.social

Thorsten Sick

The origin story: External brain

This book was already written once. As my external brain. Just for me. I collect my knowledge in “external brains” a private wiki. My security knowledge external-brain just grew to a stage where I thought “well, you just wrote a book”. And after finding *leanpub* which fits my style of tackling projects I decided I can transfer my external brain (written for me) into a book (written for tech people world wide).

And this is currently happening.

Authors

My goal is to write an anthology. Contributions from experts in their fields are welcome. Everyone contributing a chapter gets half a page of biography.

I just do not want to start asking for contributions now. Not until the book has a well tested and established structure. I would hate to force external authors to re-write their chapter several times just because the structures is still developing….

Credits

Credits are for all the contributors.

Changelog

I want to “release early - release often”. For this reason I will add a changelog to make it simpler for you to find the new sections.

Aug 2025

- RSS feed space started
- AV behaviour based classification extended
- Principles extended
- Anti Virus sharing samples extended
- Google dork extended
- Browser security extended
- Cryptographic algorithms extended
- Passwords extended
- Threat modelling extended
- TLS extended
- Secret scanning extended

April 2023

- ZAP updated
- Passwords updated
- Vulnerabilities updated
- Python updated
- Git hardening updated

December 2022

- Added Secret Scanning
- Improved ZAP
- Improved Git hardening

November 2022

- Updating my Author-page
- Update in Vulnerabilities
- Updated Git hardening chapter
- Adding ZAP

July 2022

- Small extensions to video.txt, testing_compiling.txt, presentations.txt
- Extended external references
- Text cleanup based on vale
- Improved behaviour classification
- Added CAPEv2 chapter

October 2021

- Fixed safari links (now learning.oreilly)
- Cleaned up and extended glossary

June 2021

New:

Big additions:

- Added Nmap chapter

Small additions:

- Extended Glossary
- Behaviour based classification
- Kill chain
- Passwords
- Vulnerabilities

April 2021

- Extended glossary
- Extended “design”
- Extended python %% TODO: Fix the whole book thing

February 2021

- Extended glossary
- Updated python chapter

October 2020

- Extended Behaviour based classification
- Added Antivirus detection chapter
- Extended glossary
- Extended Kill chain
- Extended External References
- Extended SSH

August 2020

- Extended External References

May 2020

- Extended Glossary
- Extending Git hardening (turning it into CI/CD + Security)
- Replacing whitelist/blacklist with *approved list* and *block list*

April 2020

- Reworked layout. Removed headers from chapters
- Extended Glossary
- Extended Glossary

March 2020

- Extended passwords
- Extended external references
- Extended fuzzing
- Extended Glossary

February 2020

- Glossary extended

January 2020

- Extended *external references*
- Extended *glossary*
- *Programming/compiling*: visual studio added
- *Testing-compiling* extended
- Decission: I will remove the target audience from all chapters. Also the author as long as it is just me. 3rd party authors will get the credits.

December 2019

- Thug chapter extended
- TLS chapter extended

November

- Spelling and quality improvements
- fixed ../../images/tls_simple.png
- Adding Recon-NG
- Small fuzzing upgrade
- Upgrade in *vulnerabilities*
- The part *planning* got a proper review.

August/September/October 2019

- Improved *mitmproxy*
- Improved *browser*
- Added *kill chain*
- Added the practical parts to *thug*
- Extended *glossary*

July 2019

- Extended Glossary
- Updated Thug
- Extended Browser
- Extended security process
- Added mitmproxy
- Extended “Threat modelling” with MITRE attack

June 2019

- Chapter for thug added - a honey client to investigate malicious web pages

May 2019

- Some cleanup
- Extended “security process”
- Added *python security*

April 2019

- Added “Threat modelling”
- Extended Glossary
- Added “git hardening”
- Added “JavaScript security”

March 2019

- Extending **testing compiled binaries**
- Most readers read PDF ⇒ Focus on PDF layout now.
- Old PDF setting: A5 (14.8cm x 21.0cm) to get a book style size
- New PDF setting: A4 (21.0cm x 29.7cm) for more table space and better screen readability
- Tables set from default to wide
- extended **content**
- extended **glossary**
- extended **browser**
- extended **security_process**
- Full quality check for “background” section
- Added **Censorship**

February 2019

- Cleaning up the author page. Adding Mastodon and Twitter
- Extended **Attacker’s goals**
- Extended **know your enemies**
- Small extensions to **principles**
- Extending *TLS*
- Extending *browser*
- Starting *vulnerabilities*
- Starting *security process*

January 2019

- Added **Software design checklist** (initial version)
- Added **Google Dorks**
- Added **Glossary**
- Added part **Appendix**
- Added basic **beef** chapter
- Added basic **burp suite** chapter
- Added **IOC sources** chapter
- Extended **TLS**
- Updated **content**
- Intro for background added
- Intro for planning added
- Intro to programming added
- Intro to testing added
- Updated samples
- Intro for tools
- Reworking first chapters: re-write, remove or move to the end of parts. Reason: I want to get people to encounter the core book faster.
  - Structure: removed unnecessary things
  - The origin (moved to the author)
  - stages of learning moved to part “psychology toolbox” (which is not active yet)
  - random encounter removed
- More glossary entries

December 2018

- Enhanced book list of defensive programming
- **SSH** chapter added
- Added **Code Coverage** chapter
- Improved *Fuzzing* chapter
- Added Vagrant to *compiling* chapter
- Extended *CppCheck*
- Small things in:
  - *Antivirus sharing samples*
  - *clang*

November 2018

- Enhanced *defensive programming*
- Added *design*
- Enhanced “attacker’s goals”
- Enhanced *passwords*
- Enhanced *clang*
- Cleaning up the book, adding parts
  - Chapters got moved around
- Enhanced *This book*
- Enhanced *The author*
- Enhanced *structure*
- Clean up crypto algorithm tables
- Enhanced *antivirus testing*
- Added *antivirus testing* to the sample
- unhooked external references
- Kehrwoche: Aspell for all text parts in “sample”

October 2018

- Added *crypto algorithms*
- Re-worked *Asserts*
  - It got an own chapter
  - Python added
  - JavaScript added
- Extended principles
- Added *testing* chapter. Especially for unit testing and bug bounties (basics)
- TLS got a diagram and minor improvements
- *Clang* chapter added
- Added *crypto algorithms* to Sample

September 2018

- *CAN bus hacking*
- *Bluetooth LE (BLE)*
- Added *code analysis tools requirements* table for an overview
- URLs now in footnotes
- Content chapter added
- Chapter *Antivirus Behaviour classification* added
- added *code analysis tools requirements* to Sample

August 2018

This is the holiday release: Focus is on improving text quality of existing chapters.

- Extended “Principles”
- Added new chapter “browser security” (not finished yet)
- Added new chapter “IoT security” (not finished yet)
- Quality improvements in
  - defensive programming
  - know your enemies
  - principles
  - structure
  - this book

July 2018

- Added *attacker’s goals*
- Added *antivirus-tests*
- Added *antivirus-integration*
- Changed PDF to A5 for a typical book-size PDF
- Added basic *fuzzing* chapter
- Extended “Defensive programming”
- Extended “principles”
- Extended “TLS chapter”
- Extended “External references”
- Added *antivirus-sharing-samples*
- Added *antivirus virustotal*
- Added *Cppcheck chapter*
- Added *Testing compiling* chapter
- Added *kill chain* chapter to offense

June 2018

- Added *TLS chapter*
- Extended *principles*
- Added *passwords*
- Added basic *compiling*
- Improved *external references*
- Extended *update*
- Added flawfinder chapter

May 2018, initial release

- Added *principles* chapter
- Added *updates* chapter
- Added *Know your enemies* chapter
- Added *UX* chapter
- Added *structure* chapter
- Added *external references* chapter
- Added “The Author” chapter
- Added “This book” chapter
- Added “Defensive programming chapter” for default defensive programming

License