Computer Security and Privacy
Principles of computer security (part I)

Carmela Troncoso
SPRING Lab
carmela.troncoso@epfl.ch

Some slides/ideas adapted from: Philippe Oechslin, George Danezis, Emiliano de Cristofaro, Gianluca Stringhini

Basic principles to build Security Mechanisms

READING: J. Saltzer and M. Schroeder. *The Protection of Information in Computer Systems*. Fourth ACM Symposium on Operating Systems Principles (October 1973)
(Intro & Section 1)

8 + 2 principles at the core of security engineering practices

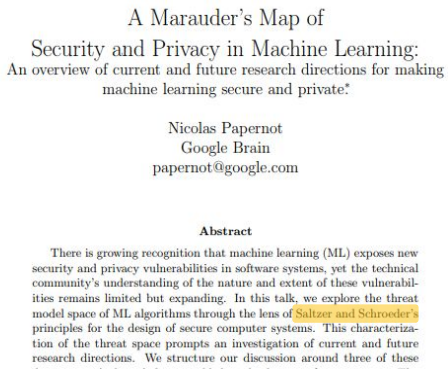
*"Principles **guide** the design and contribute to an implementation without security flaws"*

https://www.acsac.org/secshelf/papers/protection_information.pdf

The security design principles that we will see in this lecture will be principles that one must always try to follow.
Of course it may not be possible to follow all of them at the same time, but at least one must consider them.
If one principle is not followed, one must have a good reason to not consider it, and one must understand the risks associated with not following this principle.

Why should you care about principles from 1973

READING: J. Saltzer and M. Schroeder. *The Protection of Information in Computer Systems*. Fourth ACM Symposium on Operating Systems Principles (October 1973)
(Intro & Section 1)



A Marauder's Map of
Security and Privacy in Machine Learning:
An overview of current and future research directions for making machine learning secure and private*

Nicolas Papernot
Google Brain
papernot@google.com

Abstract

There is growing recognition that machine learning (ML) exposes new security and privacy vulnerabilities in software systems, yet the technical community's understanding of the nature and extent of these vulnerabilities remains limited but expanding. In this talk, we explore the threat model space of ML algorithms through the lens of Saltzer and Schroeder's principles for the design of secure computer systems. This characterization of the threat space prompts an investigation of current and future research directions. We structure our discussion around three of these

Keynote at Workshop on Artificial Intelligence and Security

2018

3

The security principles that Saltzer and Schroeder laid down in 1973 are not only widely used to build secure systems, but are nowadays guiding principles to design secure and privacy-preserving machine learning systems.

These security principles are the bread and butter of every security engineer. Learning and internalizing them is essential to design and implementing secure systems.

1 - Economy of mechanism

"Keep the [security mechanism / implementation] design as simple and small as possible"

Why?

It needs to be easy to audit and verify.

(operational testing is not appropriate to evaluate security)

[Penetration testing is valuable]

"**Trusted Computing Base" (TCB):** Every component of the system on which the security policy relies upon

4

The first principle says that the security mechanisms must be as simple as possible. (Also known as the KISS principle – Keep it Simple Stupid)

When security mechanisms are small and simple it is easy to check everything it does and verify that the operations are correct.
Reasoning about the security of complex mechanisms with many dependencies, and auditing them, is extremely hard. How do you know you have taken into account all of the possible options?

Operational testing, such as one would do to test the functionality of the system, cannot provide any security guarantees. Security is not about normal operation, and not even about faulty operation (errors in the implementation or caused at random by the environment). It is about what can go wrong when the adversary uses particular inputs on the system, and one cannot test all possible inputs…
This is not to say that penetration testing, whereby one tries to find flaws in the design or implementatoins is useless. The more things you try the more sure you are you have covered a big surface of "what could go wrong". But it cannot give you assurance that nothing can go wrong.

The (desirably small) security mechanism in which the system relies on to maintain the security policy is called the ***Trusted computing base (TCB)***

The "Trusted Computing Base" (TCB)

Every component of the system on which the security policy relies.

Hardware / firmware / software

The TCB is **trusted** to operate correctly for the security policy to hold

The only proper use of the verb "to trust" in Security Engineering:

"X trusts Y will do Z"

If something goes wrong within the TCB **the security policy may be violated**

…and if something goes wrong outside the TCB?

The TCB must be kept small to ***ease verification*** (economy of mechanism) and ***diminish the attack surface***

5

The (desirably small) security mechanism in which the system relies on to maintain the security policy is called the ***Trusted computing base (TCB)***

By definition, if something goes wrong outside of the TCB security is not affected! The TCB contains every component that must operate correctly for the security policy to hold. If something goes wrong outside, the TCB is still working and thus the security policy cannot be violated.

However, if the TCB gets compromised or fails, then the security of the system cannot be guaranteed anymore

2 – Fail-safe defaults

***"Base access decisions on permission rather than exclusion"*[SS75]**

If something fails, be as secure as if it does not fail

→ errors / uncertainty should err on the side of the security policy

Do not try to fix!! (e.g., automated doors: if they cannot close, stay open)

Whitelist, do not blacklist

→ lack of permission is easy to detect and solve

Examples:

- Security door: if no permission, do not open
- Form input: if no permission to write in X, do not write anywhere

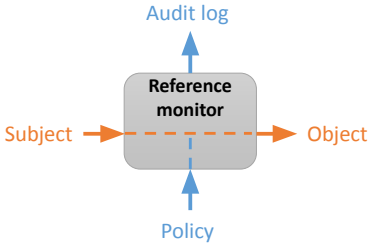
The insight behind this principle is that permissions are given after thought. If a permission is given, we know that the principal is allowed to do the action and this action is safe. On the contrary, the space of non-permissions is infinite and we do not know exactly what happens. We cannot list them all and there will be uncertainty. By basing decisions on permission, you are sure that no matter what is executed, had permission and cannot harm.

Achieving the goal of security under failure with blacklisting is hard. One can never make sure th at all harmful events are blacklisted.

Do not try to fix the errors. If something went wrong, the safe step is to go back to a known safe state. Once there is an error, one does not know what is the state of the system, so one cannot be sure that steps forward towards a solution are safe.

3 – Complete mediation

"Every access to every object must be checked for authority" [SS75]



mediates ALL actions from subjects on objects and ensures they are according to the policy

Difficult to implement

- Performance?
  - Checking everything is slooooooow
- Time to check vs. time to use
- Modern distributed systems
  - You can only check what you see!

7

Ideally, all actions should be checked before allowing them to happen.

The **reference monitor** is the component (both in design and implementation) that mediates actions and ensures they are according to the policy.

Implenting the ideal reference monitor in reality is very hard:

- If you check every action there would be a big performance hit (think how many memory access per second your computer does!)
- It is not straightforward to decide when the reference monitor should operate: between a check and an execution the state may have changed. How do we ensure the check is correct?

- In distributed systems resources are spread across machines and even networks. Subjects and objects may not even be on the same machine! Where should the reference monitor be? If we don't distribute it, not all checks can be run. If we distribute it, it becomes very complex and hard to know if it is correct

4 – Open design

"The design should not be secret" [SS75]

Kerckhoff
La Cryptographie Militaire
(*1883*)

"The design of a system should not require secrecy"

"The enemy knows the system"

"one ought to design systems under the assumption that the enemy will immediately gain full familiarity with them"

Shannon
Communication Theory of Secrecy Systems
(*1949*)

Baran
Security, secrecy, and tamper-free considerations
(*1964*)

"The Paradox of the Secrecy About Secrecy"

"Without the freedom to expose the system proposal to widespread scrutiny' by clever minds of diverse interests, is to increase the risk that significant points of potential weakness have been overlooked"

https://www.rand.org/pubs/research_memoranda/RM3765/RM3765.chapter2.html

These are three views on why secrecy is not a good underlying requirement for security:

Kerchoff: every secret in a system creates a potential failure point. Secrecy, in other words, is a prime cause of weakness. Openness, helps having less points of failure.
Shannon: designing assuming the adversary does not know the system gives advantage to the adversary, as he is out of the threat model
Baran: designing in secret is hard, as it prevents conversations that could help identifying weaknesses and better security mechanisms

4 – Open design

"The design should not be secret" [SS75]

Kerckh
La Cryptograp
(188

When you design… algorithms are public! Only key elements are kept secret

Crypto: only keep the key secret
Authentication: only keep password secret
Obfuscation: only keep the used noise secret

non
ry of Secrecy Systems
(*1949*)

"The Paradox of the Secrecy About Secrecy"

Baran
Security, secrecy, and tamper-free considerations
(*1964*)

https://www.rand.org/pubs/research_memoranda/RM3765/RM3765.chapter2.html

4 – Open design

"The design should not be secret" [SS75]

Open design results in better & easier auditing

Linus' law: *"given enough eyeballs, all bugs are shallow"*

Secrecy is unrealistic!!

Way to build a bad threat model!

Famous failures closed design:

- DVD encryption
- GSM encryption

Raymond
The Cathedral and the Bazaar
(*1997*)

Key principle behind the academic discipline devoted to understanding computer security

10

Open designs can be revised by experts to revise who find vulnerabilities and propose ways to fix them.

DVD Encryption (1996): the algorithm was called Content Scramble System (CSS). Propietary 40-bits algorithm (US export law at the time forbid larger keys). It's use was tied to a license – only if one has the license, obtained under a non-disclosure agreement, one can decrypt the content. Every licensee got a decryption key, and in every DVD the encryption key of the content is encrypted to all the licensees, so that all can read the content. Eventually, one careless reader developer did not encrypt properly their decryption key. Hackers could get it and with that read any DVD to enable copies. There were many secrets, when one was leaked the full system broke (more: https://www.schneier.com/essays/archives/1999/11/dvd_encryption_broke.html)

GSM Encryption (1987): A5/1, with a key length of 54 bits (short, so that it could be found via brute force by secret services). It was reverse engineered, and since then many attacks, some of them very serious have been found. As of today, one can decrypt A5/1 in real time without knowing the secret key.

*The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary (*E. Raymond), describes the open design process: software is developed by a group of engineers, but made available for others to check, comment, and improve.

5 – Separation of privilege

"No single accident, deception, or breach of trust is sufficient to compromise the protected information" [SS75]

*A **privilege** allows a user to perform an action on a computer system that may have security consequences, e.g., create a file in a directory, access a device, write to a socket for communicating over the Internet.*

Require multiple conditions to execute an action improves security

Examples: two keys to open a safe, two-factors to authenticate

Problems

- Availability?
- Responsibility?
- Complexity!

11

The separation of privilege principle indicates that having only one entity/process/user/… responsible for a (critical) security task is undesirable. If that responsible fails, the system's security is compromised. Instead, one must try to divide responsibility so that if one entity gets compromised the full system does not break.

While having two or more conditions helps with security, it has downsides:
- Now one needs all conditions to use a service. What happens when you don't have your phone to log in on your bank account?
- When the conditions are related to people, this dilutes responsibility: no one in particular is responsible. This may also lower security, and also makes it more difficult to establish liabilities.
- Having to meet more than one condition increases complexity! How are they combined? When should they be considered? Does order matter? Remember "economy of mechanism" principle.

Recap

Economy of mechanism. Keep it simple!

Fail-safe defaults. If there is a problem, your move should comply with the policy

Complete mediation. Verify *every* action

Open Design. Make the design (and implementation) of your mechanism available

Separation of Privilege. Try to never rely on *only one* entity or action

6 – Least privilege

"Every program and every user of the system should operate using the least set of privileges necessary to complete the job" [SS75]

Rights added as needed, discarded after use

Damage control
- Minimize high privilege actions & interactions

"Need-to-know" principle
- Examples
  - Guest accounts @ EPFL
  - Data minimization principle (Data Protection)

What principle is this related to?

13

The least privilege principle indicates that one should be very careful when assigning permissions to principals. They should always have the minimal set of permissions necessary to carry out their task. This helps limiting how much damage can a principal cause in the system. If a principal does not have privileges, any error/misbehaviour of this principal cannot create huge damage on the system.

The least privilege principle is very related to the fail safe principle. It ensures that if there is an error or compromise the compromised principal cannot execute unauthorized actions, helping staying on a safe state.

7 – Least common mechanism

"Minimize the amount of mechanism common to more than one user and depended on by all users" [SS75]

"Every shared mechanism represents a potential information path between users and must be designed with great care to be sure it does not unintentionally compromise security"

Remember "Economy of mechanism"

- (Design) Interactions make it hard to validate the security design
- (Implementation) Interactions may lead to unintentional leaks of information
  - Unintended channels: use of /tmp, shared cache

Note that this refers to mechanisms! Not to the code. Reusable code that has been tested many times is helpful for security

Having common mechanisms makes the system more complex (against economy of mechanism).

At the design level, interactions are difficult to model completely. It is hard to guarantee one has considered all of the possible cases and thus the validation holds in reality

The number of common flows and unintentional information channels introduced by the use of common resources when implementing systems tends to grow: for efficiency, computers share a lot of resources!

8 – Psychological acceptability

"It is essential that the human interface be designed for ease of use, so that users routinely and automatically apply the protection mechanisms correctly" [SS75]

Hide complexity introduced by security mechanisms

Security mechanisms should not make the resource more difficult to access than if it was not present

Mental model of the (honest) users must match security policy and security mechanisms

Cultural acceptability – not all mechanisms are acceptable everywhere

(Authentication) Face recognition not suitable in cultures that cover their face

(Safety) Register of everyone who sleeps in a dorm

In order for security mechanisms to succeed, users need to feel comfortable with them:

- They need to be able to use them, and they cannot cause a burden. When the mechanisms are complicated and hard to interact with, users start following insecure practices.
- The threat model against which the mechanism is designed must be understandable by honest users. If users do not perceive the threat, they are unlikely to make good use of the mechanism.
- The mechanism must not go against any social/cultural norm. Otherwise, users will ignore it or try to circumvent it.

Extra principles from physical security
9 - Work factor

DIFFICULT TO TRANSPOSE TO COMPUTER SECURITY!!

"Compare the cost of circumventing the mechanism with the resources of a potential attacker" [SS75]

It helps **refining** the threat model!

Quantifying **cost** is hard**?**
- cost of compromising insiders?
- cost of finding a bug?
- monetization?

Difficult to quantify

16

This principle is useful to reason about security and may help to better define the threat model (making very explicit the capabilities of the adversary), but it is very hard to completely transpose to computer security.

Cost is hard to define and quantify for many cases. What is the cost of corrupting an employee? Or of finding a bug? Time? Expertise?. Also at the end of the day it is about cost vs benefit. How many people will be able to use the exploit? What will be the benefit for the adversary that uses the exploit? And what if the cost of individuals is small, but many individuals can use the exploit.

In this sense, computer security is very different from physical security mechanisms where in general the cost is easy to see (breaking a lock, climbing a wall) and the benefit is limited to the people that are physically present.

Extra principles from physical security
10 - Compromise recording

DIFFICULT TO TRANSPOSE TO COMPUTER SECURITY!!

"Reliably record that a compromise of information has occurred [...] in place of more elaborate mechanisms that completely prevent loss" [SS75]

Keep **tamper-evidence logs,**

they may enable recovery (integrity)

Logs **are not magic**:

What if you cannot recover? (if confidentiality mechanisms were in place)

How to keep integrity?

Logs may be a vulnerability (Privacy)?

Logging the log? (Availability)

Logging is not a guarantee that the compromise is detected.

17

Creating secure logs in a digital environment is much harder than in a physical environment.

The main reason is that you need a new security mechanism for them!

Having a log that helps detecting an attack does not guarantee that the system can recover. For instance, if the attack deletes keys, even if you know the attack happened and who made it you cannot recover information

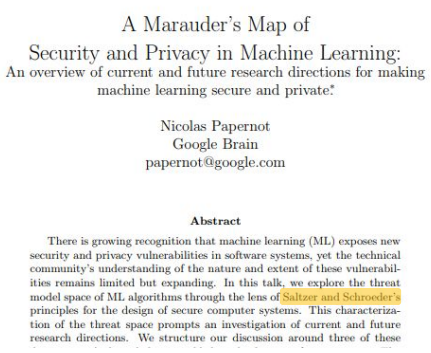
How do we keep their integrity: if the adversary attacking the system can tamper the log, then there is no use in having a log

Logs may contain information that leak the content of confidential exchanges (e.g., a record of a patient talking to an oncologist very frequently reveals the patient health status even if the patient's records are confidential)

Availability is crucial: how do we make sure that the log is not deleted or that access to the log is not prevented?

Thinking about the principles helps designing good logging systems

Why principles are important?



A Marauder's Map of
Security and Privacy in Machine Learning:
An overview of current and future research directions for making machine learning secure and private*

Nicolas Papernot
Google Brain
papernot@google.com

Abstract

There is growing recognition that machine learning (ML) exposes new security and privacy vulnerabilities in software systems, yet the technical community's understanding of the nature and extent of these vulnerabilities remains limited but expanding. In this talk, we explore the threat model space of ML algorithms through the lens of Saltzer and Schroeder's principles for the design of secure computer systems. This characterization of the threat space prompts an investigation of current and future research directions. We structure our discussion around three of these

Least privilege. Let the ML learn as little as possible so that information cannot be extracted

Least Common Mechanism. Get samples labelled from different origins

Psychological acceptability. Users must be able to understand why models classify or misclassify an input

Work factor. The cost of the attack, e.g., in terms of number of calls to an API, matters for its relevance

Compromise recording. Ideally we would like to be able to log all steps inside the algorithm

Summary of the lecture

Principles allow us to identify safe and unsafe *patterns* in when designing security mechanisms

- Do not use principles as a blind checklist!
- Use principles as tools to weight design decisions.