Principles of Security Engineering

https://iuk.one/1033-1004

Clemens H. **Cap**

ORCID: 0000-0003-3958-6136

Department of Computer Science

University of **Rostock**

Rostock,Germany

clemens.cap@uni-rostock.de

Version 2

Overview on 8 Principles

1. Fail-Safe Defaults
2. Separation of Privileges
3. Least Privilege
4. Complete Mediation
5. Economy of Mechanism
6. Acceptability
7. Kerckhoff
8. Proper Information

Principle of Fail-Safe Defaults

Definition

The **normal** situation is to **deny** access.

Every access needs

- Good reason.
- Explicit permission
- Positive action

Example: Restrictive umask.
Set restrictive defaults for file-system permission.

Example: Initialize variables.
Initialize all variables (and permissions) with safe default values.

Examples of Bad Defaults

Counterexample: Engineering backdoor.

- Often: Circumvention of checks as default during development.
- Left in by mistake or bad intent.

Counterexample: Will add security later.

- No resources left for adding solid security.
- Architecture does not permit simple "add on".

Counterexample: Therac-25

- Rather an example for safety and for security.
- Patient placement mode allowed high radiation to be engaged.
- Race conditions allowed the activation of improper default values.
- Results: Dead patients.

Principle of Separation of Privileges

Variant 1: Components

A system is split into **several components** with specific tasks.
Tasks only have the privileges for their respective task

Variant 2: Multiple Conditions

A system should **not grant** permissions based only on a **single** condition.

More detailed break down:

- **Conjunction** of Requirements
- **Disjunction** of Rights (rather: exclusion)

Example for Privilege Separation / Components

Example: UNIX daemon architecture

- UNIX daemons often are split into different processes.
- Main program forks.
- Proc 1: Drops privileges, communicates with users, implements features.
- Proc 2: Stays privileged but only does security relevant parts.
- Maintenance (and introduction of bugs) to the larger part does not compromise the smaller part.

Examples for Conjunction of Requirements

Example: A user may `su` to `root` if she

- knows the root password **AND**
- is part of the group wheel

Example: Multi-factor authentication.

- **Knowledge**-Factor: Something I know (a password) **AND**
- **Property**-Factor: Something I own (a hardware token) **AND**
- **Biometric** Factor: Something I am (a fingerprint)

Example for Disjunction (Exclusion) of Rights

Example: Audit Integrity.

- Have maximum rights as root and edit (almost all) files **XOR**.
- Review and clear log and audit files.

Principle of Least Privilege

Definition

A subject should be given **only** those privileges **needed** to complete its task.

Need-to-know principle.

Core aspects:

- **Over time:** Do not give privileges longer than required.
- **Over privilege:** Do not give more privileges than required.
- **Revocation:** Do not forget to take back privileges.

Privilege Bracketing

An entity requiring special permission for some task should

1. **acquire** permission at the **latest** possible moment — **Late opening** of bracket
2. **render** at the **earliest** possible moment — **Early closing** of bracket.
3. **render in all** possible cases — **No escape** from bracket.

Examples for Least Privileges

Example: Mail server

- Needs right to access incoming network port.
- Needs right to access spool directory.
- Does not need any additional rights.

Example: `sudo`

- A system administator does not log on as root for the entire day.
  She then does a `rm -Rf *` while in the wrong directory...
- For every single command requiring `root` a `sudo` is used.

Example: Specific file system permissions.

- If we need to append data, do not provide permission to write data.

Example for Privilege Bracketing (1)

We design a program using privileges.

```
function Example () {
  acquire_some_privilege();  // open privilege bracket
    do_some_task();
  revoke_some_privilege();  // close privilege bracket
}
```

Src. 1: Stereotypical program requiring a permission.

We realize that `acquire_some_privilege();` may throw an exception.

- When the privilege is not granted.   This is planned operations.
- When "something" goes wrong.   This is unplanned operations.

We therefore add an exception handler.

Example for Privilege Bracketing (2)

We assume that exception handling is good.
Thus, we protect a large portion of the program by a `try-catch` block.

```
function Example () {
  try {
    acquire_some_privilege();   // needed an exception handler
      do_some_task();
    revoke_some_privilege();
  }
  catch (exception) {
    handle_exception();
} }
```

Src. 2: Program requiring a permission after first "improvement".

If `acquire_some_privilege();` throws after granting the privilege,
the privilege never is revoked again.

Example for Privilege Bracketing (3)

As a result of this observation we "improve" the program by tightening the try-catch block.

```
function Example () {
  try {acquire_some_privilege();}
  catch (exception) {
    handle_exception();
    revoke_some_privilege();    // this is fine now
    return; }
  do_some_task();               // but what if this throws ??
  revoke_some_privilege();
}
```

Src. 3: Program requiring a permission after second "improvement".

Example for Privilege Bracketing (4)

try-catch-finally comes to our rescue.

```
function Example () {
  try {
    acquire_some_privilege();
    do_some_task();
  } catch (exception) {handle_exception();}
  finally {revoke_some_privilege();}
}
```

Src. 4: A good place to revoke a privilege is in the `finally` part, since it is guaranteed to execute.

Note: Similar problems show up with early returns from within a function.

Principle of Complete Mediation

Definition

Every requested operation with a possible security impact **must be tested** for being acceptable with respect to the security policy.

Fig. 1: Famous folklore for violating complete mediation in real world premises.

Lack of Complete Mediation: TOCTOU Attack

Example: TOCTOU (time-of-check to time-of-use) attack

- A file system opens for a particular access.
- There is: "open for read", "open for write", "open for append"
- Assume the system checks permissions on open API calls.
- Attacker opens a file and keeps it open for a long time.
- Attacker can do a `write` months after `write` permission has been revoked.

Lack of Complete Mediation: Unattended System

Unattended Terminal:

- Alice logs on for her telebanking system.
- Alice enters a transaction.
- Before completing the transaction she is called on the phone.
- Mallory approaches the terminal, changes the account number and presses "Okay".

Questions for you:

- Which strategies can prevent such attacks?
- Are these strategies always practical?
- What about a reasonable compromise?

Principle of Economy of Mechanisms

Definition

Security mechanisms must **KISS**.

KISS = Keep it simple and stupid.

Further break-down according to NIST

- **Simplicity** Complex things go wrong and cannot easily be tested & debugged
- **Operational ease of use:** Users love convenience
- **No unnecessary** security mechanisms.
  This adds complexity.
  This reduces convenience and maintainability.
- **Minimize the number** of components to be trusted.
  Ideally: A single security monitor.

Principle of Acceptability

Definition

A security mechanism should **not**
make the resource **too difficult** to access for the entitled user.

Example: ssh

- ssh allows a log-in mode using passwords.
- ssh allows a log-in mode using locally stored private keys.
- The key mode is more acceptable for the user.

Kerckhoff Principle of Open Design

Definition

Security of a mechanism should not depend on the secrecy of its design or implementation.

Aspects:

- Security must not depend on ignorance of the security mechanism.
- Security must depend only on possession of one specific item.
  Examples: password, hardware-token, software-token, fingerprint, face
- Design of a security mechanism should be published, available for review and attack.
- We want the systems to fail early and before deployment.
- Always assume the enemy knows the system.
- Quite old principle. First publication in 1883 in
  Journal des sciences militaires, vol. IX, pp. 5-38.

Violations of Kerckhoff Principle

Famous examples:

- **GSM hash functions A3, A8, COMP128.**
  Design kept secret, is bad and gets broken.
- **GSM A5/1 and A5/2 encryption.**
  Design kept secret, is bad and gets broken.
- **MIFARE REFID tag security system.**
  Design kept secret, is bad and get broken.
  Practical consequences: See this report.
  Breaking of the algorithm: Masters Thesis, which broke the algorithm

Principle of Proper Information

Definition

In **design-mode** provide the **maximum** amount of information to the user
in **operational mode** provide the **least** amount of information to the user.

Task

The security parameter of a system is the binary logarithm of the number of brute-force attempts needed to crack the system.

Two systems allow 128-bit user-names and 128-bit passwords.

System 1 uses the error message "Login incorrect"

System 2 uses the two error messages "User-name not found" and "Wrong password".

Calculate the security parameter of system 1 and system 2.

Role of the Principles

Goal: Provide general guidelines for designing systems.

Problem: Some principles are contradictory.

Consequence: Good design always is a compromise.

Tasks

Comment on the following practices from the perspective of our principles:

- umask 0022
- umask 0777
- Using 3-factor authentication for StudIP
- Using 3-factor authentication for banking log-in

Find examples of designs where there are contradictions between the principles.

Read about the **Heartbleed** security bug in OpenSSL
and comment on the lessons learend in view of these principles.

Appendix

Contents of Appendix

List of Codes

List of Figures

Terms of Use (1)

Terms of Use (2)

Disclaimer: Die sich immer wieder ändernde Rechtslage für digitale Urheberrechte erzeugt für mich ein nicht unerhebliches Risiko bei der Einbindung von Materialien, deren Status ich nicht oder nur mit unverhältnismäßig hohem Aufwand abklären kann. Ebenso kann ich den Rechteinhabern nicht auf sinnvolle oder einfache Weise ein Honorar zukommen lassen, obwohl ich – und in letzter Konsequenz Sie als Leser – ihre Leistungen nutzen.

Daher binde ich gelegentlich Inhalte nur als Link und nicht durch Framing ein. Lt EuGH Urteil 13.02.2014, C-466/12 ist das unbedenklich, da die benutzten Links ohne Umgehung technischer Sperren auf im Internet frei verfügbare Inhalte verweisen.

Wenn Sie diese Rechtslage stört, dann setzen Sie sich für eine Modernisierung des völlig veralteten Vergütungssystems für urheberrechtliche Leistungen ein. Bis dahin klicken Sie bitte auf die angegebenen Links und denken Sie darüber nach, warum wir keine für das digitale Zeitalter sinnvoll angepaßte Vergütungssysteme digital erbrachter Leistungen haben.

Zu Risiken und Nebenwirkungen fragen Sie Ihren Rechtsanwalt oder Gesetzgeber.

Weitere Hinweise finden Sie im Netz hier und hier oder hier.

Citing This Document

If you use contents from this document or want to cite it,
please do so in the following manner:

Clemens H. Cap: Principles of Security Engineering. Electronic document. https://iuk.one/1033-1004 24. 4. 2021.

Bibtex Information: https://iuk.one/1033-1004.bib

```
@misc{doc:1033-1004,
   author      = {Clemens H. Cap},
   title       = {Principles of Security Engineering},
   year        = {2021},
   month       = {4},
   howpublished = {Electronic document},
   url         = {https://iuk.one/1033-1004}
}
```

Typographic Information:
Typeset on April 24, 2021
This is pdfTeX, Version 3.14159265-2.6-1.40.21 (TeX Live 2020) kpathsea version 6.3.2
This is pgf in version 3.1.5b
This is preamble-slides.tex myFormat©C.H.Cap

List of Slides

Legend:
- continuation slide
- slide without title header
- image slide