


School of Computer Science & Engineering

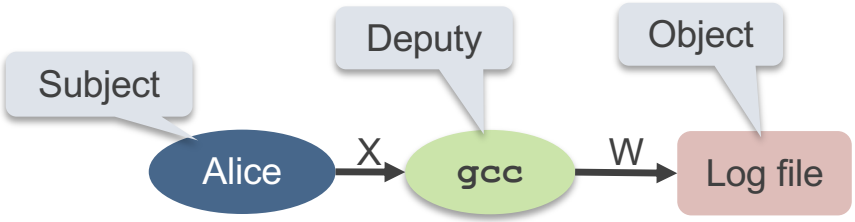
**COMP9242 Advanced Operating Systems**

2026 T3 Week 07 Part 2

# Security Fundamentals

Gernot Heiser

Incorporating material from Toby Murray



# Copyright Notice

**These slides are distributed under the**
**Creative Commons Attribution 4.0 International (CC BY 4.0) License**

- You are free:
  - to share—to copy, distribute and transmit the work
  - to remix—to adapt the work
- under the following conditions:
  - **Attribution:** You must attribute the work (but not in any way that suggests that the author endorses you or your use of the work) as follows:

    *"Courtesy of Gernot Heiser, UNSW Sydney"*

The complete license text can be found at
http://creativecommons.org/licenses/by/4.0/legalcode


© Gernot Heiser 2019–25 – CC BY 4.0

# Today's Lecture

- Security Intro
- Access-Control Principles
- ACLs vs Capabilities
- Mandatory Access-Control Policies


© Gernot Heiser 2019–25 – CC BY 4.0

# Computer Security

**Protecting *my interests* (that are under computer control) from *threats***

- Inherently subjective
  - Different people have different interests
  - Different people face different threats
- Don't expect one-size-fits-all solutions
  - Grandma doesn't need an air gap
  - Windows insufficient for protecting TOP SECRET (TS) classified data on an Internet-connected machine

Security claims only make sense
- wrt *defined objectives*
- while *identifying threats*
- and *identifying secure states*

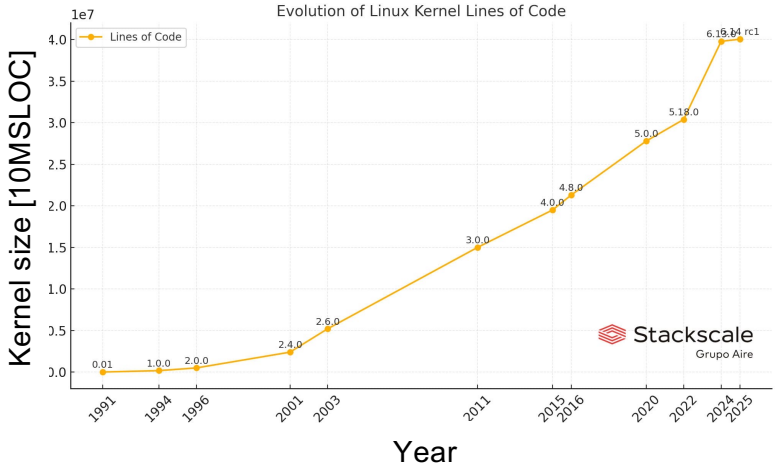

© Gernot Heiser 2019–25 – CC BY 4.0

# State of OS Security

- Traditionally:
  - Has not kept pace with evolving user demographics
    - Focused on e.g. Defence and Enterprise
  - Has not kept pace with evolving threats
    - Much security work is reactive rather than proactive

Some things are getting better:
- more systematic hardening of OSes
- Better security models in smartphones compared to desktops

Other things are getting worse:
- OS kernel sizes keep growing
- Fast growth in attacker capabilities
- Slow growth in defensive capabilities



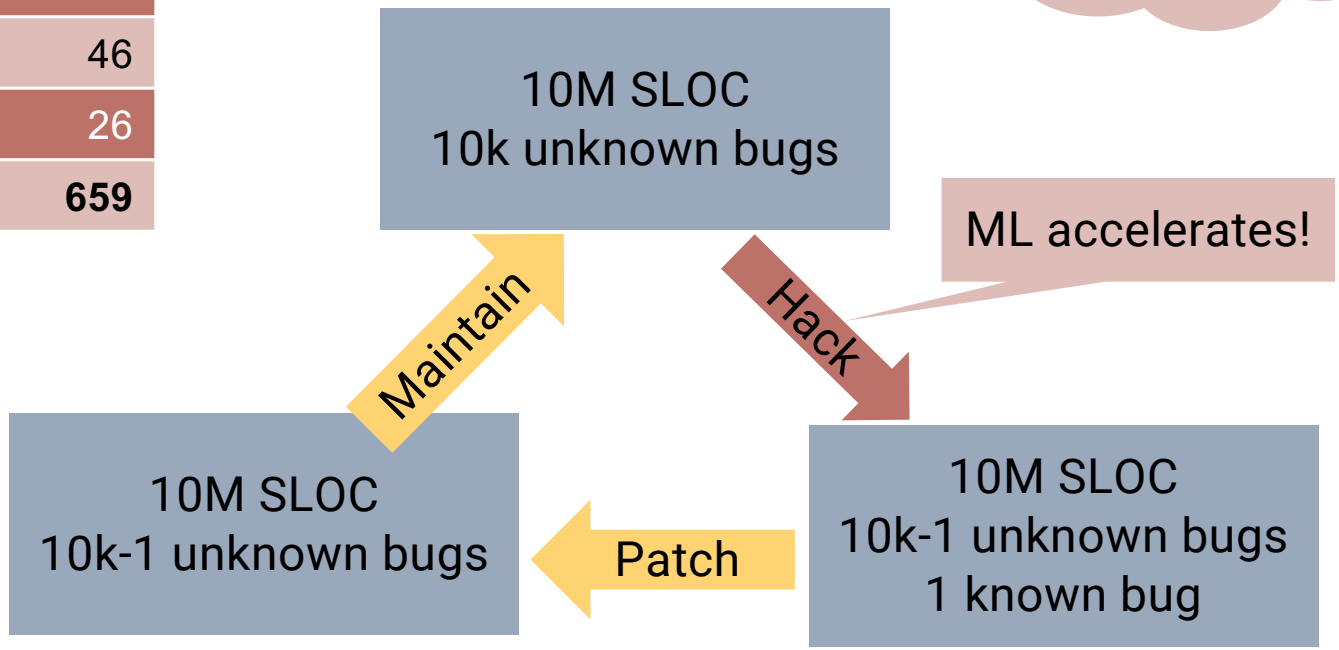


© Gernot Heiser 2019–25 – CC BY 4.0

# Standard Approach: Patch-and-Pray

| Year | Critical CVEs |
|---|---|
| 2026 (to 29/10) | 537 |
| 2025 | 46 |
| 2024 | 26 |
| **All years** | **659** |

A losing proposition!





© Gernot Heiser 2019–25 – CC BY 4.0

# Security Design Principles

Saltzer & Schroeder [SOSP ’73, CACM ’74]:

- **Economy of mechanisms** – KISS
- **Fail-safe defaults** – as in any good engineering
- **Complete mediation** – check everything
- **Open design** – no security by obscurity
- **Separation of privilege** – defence in depth
- **Least privilege** – aka *principle of least authority* (POLA)
- **Least common mechanisms** – minimise sharing
- **Psychological acceptability** – if it’s hard to use it won’t be

Mainstream OSes violate most of them! Especially KISS, POLA


© Gernot Heiser 2019–25 – CC BY 4.0

# Common OS Security Mechanisms

- Access Control Systems
  - control what each process can access
- Authentication Systems
  - confirm the identity on whose behalf a process is running
- Logging
  - for audit, detection, forensics and recovery
- Filesystem Encryption
- Credential Management
- Automatic Updates

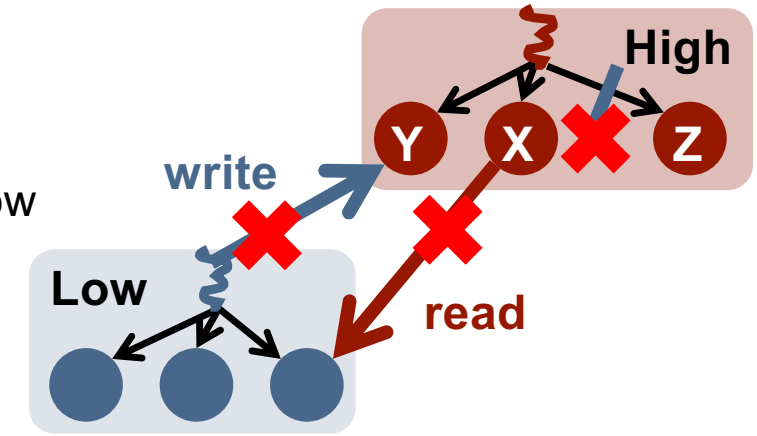
Fundamental mechanism


© Gernot Heiser 2019–25 – CC BY 4.0

# Security Policies

- Define what should be protected, and from whom

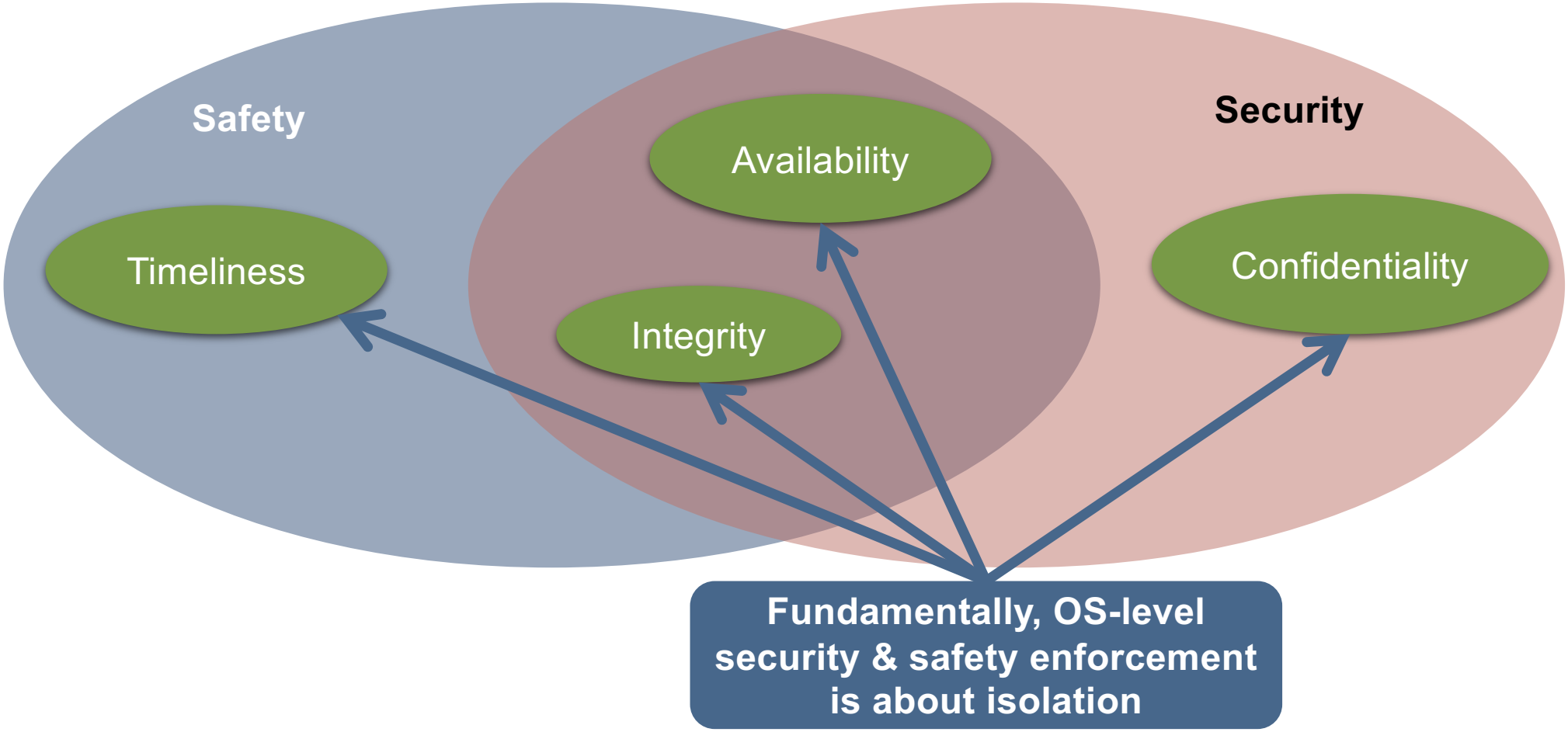
- Often in terms of common security goals (*"CIA properties"*):
  - **Confidentiality**
    - X should not be learnt by Low
  - **Integrity**
    - Y should not be tampered with by Low
  - **Availability**
    - Z should not be made unavailable to High by Low





© Gernot Heiser 2019–25 – CC BY 4.0

# Security vs Safety





© Gernot Heiser 2019–25 – CC BY 4.0

# Assumptions

- All policies and mechanisms operate under certain **assumptions**
  - **e.g.** TS-cleared users can be trusted not to write TS data into the UNCLASS window
  - some *trusted* entities behave as expected
- Problem: implicit or poorly understood assumption

Good assumptions are
- *clearly identified*
- *verifiable*!


© Gernot Heiser 2019–25 – CC BY 4.0

# Trust

- Systems always have **trusted** entites
  - whose misbehaviour can cause insecurity
  - hardware, OS, sysadmin ...

Trusted computing base (TCB):
The set of all trusted entities

- Secure systems require the TCB to be **trustworthy**
  - achieved through **assurance** and **verification**
  - shows that the TCB is unlikely to misbehave

Minimising the TCB is key for ensuring correct behaviour


© Gernot Heiser 2019–25 – CC BY 4.0

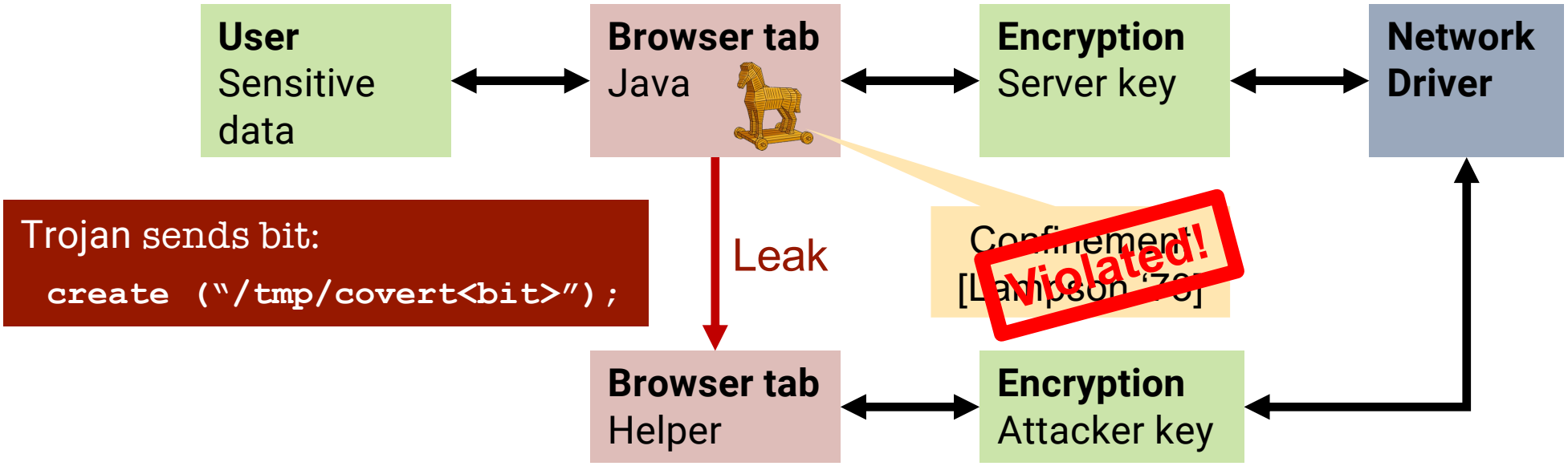
# Assurance and Formal Verification

- **Assurance**:
  - systematic evaluation and testing
  - essentially an intensive and onerous form of quality assurance
- **Formal verification**:
  - mathematical proof
- **Certification**: independent examination
  - confirming that the assurance or verification was done right

Assurance and formal verification aim to establish correctness of
- mechanism design
- mechanism implementation


© Gernot Heiser 2019–25 – CC BY 4.0

# Covert Channels: Bypassing Access Control




Helper reads bit:

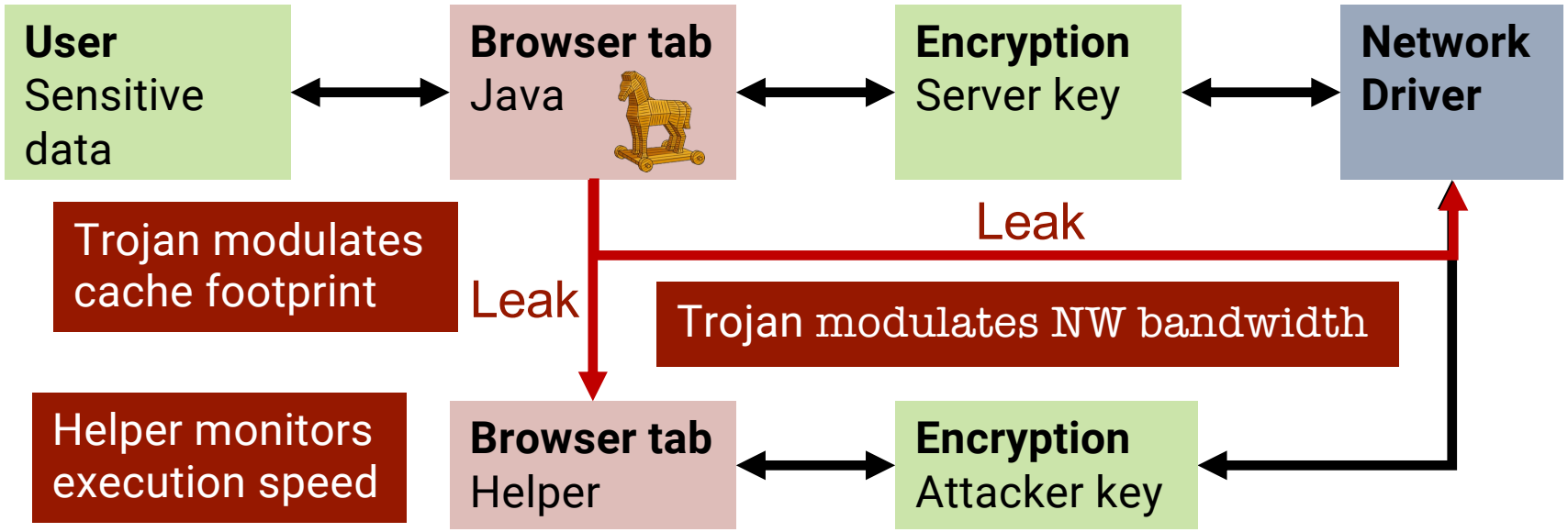
```
if (create ("/tmp/covert0")) bit=0
else if (create ("/tmp/covert1")) bit=1
else retry;
```

**Covert storage channel:**

- Uses attribute of shared resource
- Controlable by access control


© Gernot Heiser 2019–25 – CC BY 4.0

# Covert Channels: Bypassing Access Control




**Covert timing channel:**
- Outside access control
- Very hard to control/analyse!


© Gernot Heiser 2019–25 – CC BY 4.0

# Covert Timing Channels

- Created by shared resource whose effect on timing can be monitored
  - Cache, network bandwidth, CPU load...
- Requires access to a time source
  - Anything that allows processes to synchronise
  - Generally any relative occurrence of two events

Usually based on some hidden state (e.g. caches)
⇒ timing/storage channel distinction is not deep!

Channel bandwidth may matter: is 1b/s ok?

- Leaking a video vs
- Leaking a server's SSL key

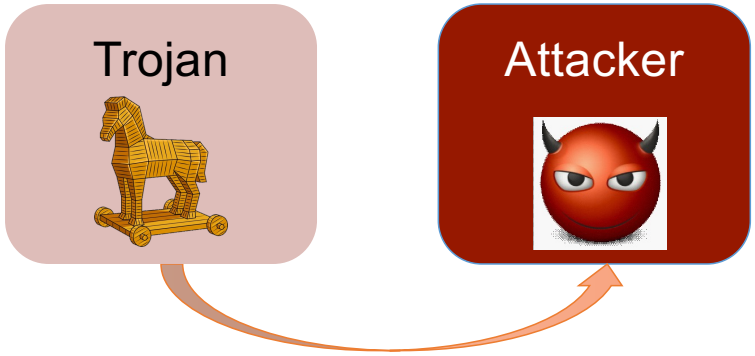
**Other physical channels:**

- Power
- Temperature (fan speed)
- Electromagnetic emanation
- Acoustic emanation…


© Gernot Heiser 2019–25 – CC BY 4.0

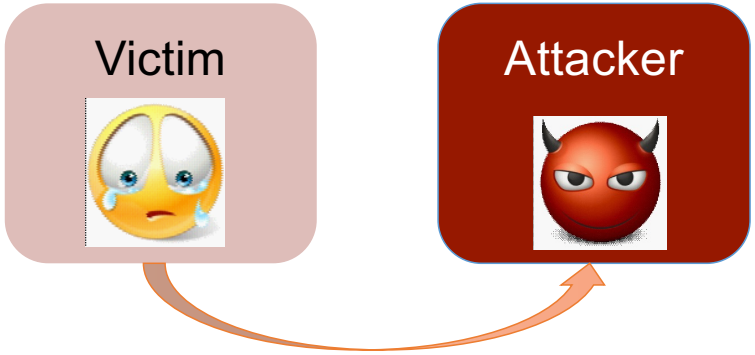
# Covert Channels vs Side Channels

## Covert Channel




- Trojan intentionally creates signal through targeted resource use
- Worst-case bandwidth

## Side Channel




- Attacker uses signal created by victim's innocent operations
- Much lower bandwidth


© Gernot Heiser 2019–25 – CC BY 4.0

# Summary of Introduction

- Security is very subjective, needs well-defined objectives
- OS security:
  - provide good security **mechanisms**
  - that support users' **policies**
- Security depends on establishing **trustworthiness** of trusted entities
  - **TCB: set of all such entities**
    - **should be as small as possible**
  - Main approaches: assurance and verification

The OS is necessarily part of the TCB


© Gernot Heiser 2019–25 – CC BY 4.0

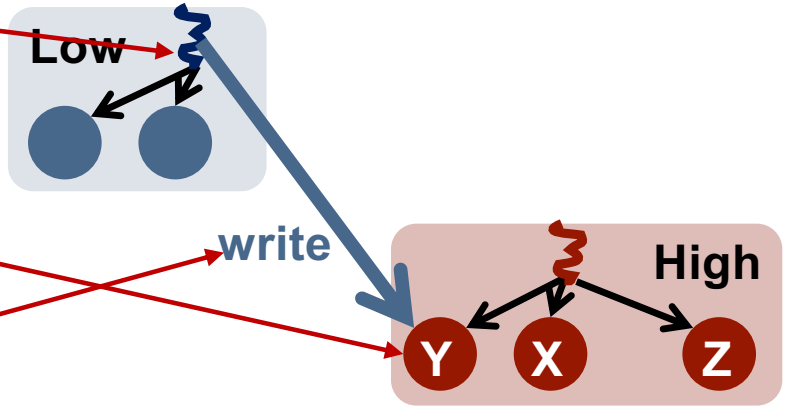
# Access-Control Principles


© Gernot Heiser 2019–25 – CC BY 4.0

# Access Control

**Who** can access **what** in which **ways**

- The “who” are called **subjects** (or **agents**)
  - e.g. users, processes etc.
- The “what” are called **objects**
  - e.g. individual files, sockets, processes etc.
  - includes all subjects
- The “ways” are called **permissions**
  - e.g. read, write, execute etc.
  - are usually specific to each kind of object
  - include those meta-permissions that allow modification of the protection state
    - e.g. own





© Gernot Heiser 2019–25 – CC BY 4.0

# Access Control Mechanisms & Policies

- Access Control **Policy**
  - Specifies allowed accesses
  - And how these can change over time
- Access Control **Mechanism**
  - Used to implement the policy

Certain mechanisms led the selves to some policies but not others

Some policies cannot be expressed with the OS's mechanisms!


© Gernot Heiser 2019–25 – CC BY 4.0

# Protection State: Access-Control Matrix

Defines system's protection state at a particular time instance [Lampson '71]

Subjects are also objects!

| | **Obj 1** | **Obj 2** | **Obj 3** | **Subj 2** |
|---|---|---|---|---|
| **Subj 1** | R | RW | | send |
| **Subj 2** | | RX | | control |
| **Subj 3** | RW | | RWX<br>own | recv |


© Gernot Heiser 2019–25 – CC BY 4.0

# Representing Protection State

Storing full matrix is infeasible
- huge but sparse
- highly dynamic

| | Obj 1 | Obj 2 | Obj 3 | Subj 2 |
|---|---|---|---|---|
| **Subj 1** | R | RW | | send |
| **Subj 2** | | RX | | control |
| **Subj 3** | RW | | RWX<br>own | recv |

**Obj 1**

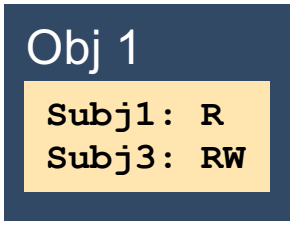
```
Subj1: R
Subj3: RW
```

Columns are access-control lists (ACLs)


© Gernot Heiser 2019–25 – CC BY 4.0

# Access Control Lists (ACLs)

- Subjects usually aggregated into classes
  - e.g. UNIX: owner, group, everyone
  - more general lists in Windows, recent Linux
  - Can have negative rights
    eg. to overwrite group rights
- Meta-permissions (e.g. own)
  - control class membership
  - allow modifying the ACL




Used by all mainstream OSes


© Gernot Heiser 2019–25 – CC BY 4.0

# Representing Protection State

How about rows?

| | Obj 1 | Obj 2 | Obj 3 | Subj 2 |
|---|---|---|---|---|
| **Subj 1** | R | RW | | send |
| **Subj 2** | | RX | | control |
| **Subj 3** | RW | | RWX<br>own | recv |

Set of rights a subject has – the subject's *protection domain*

"Object capability"

Subj 3

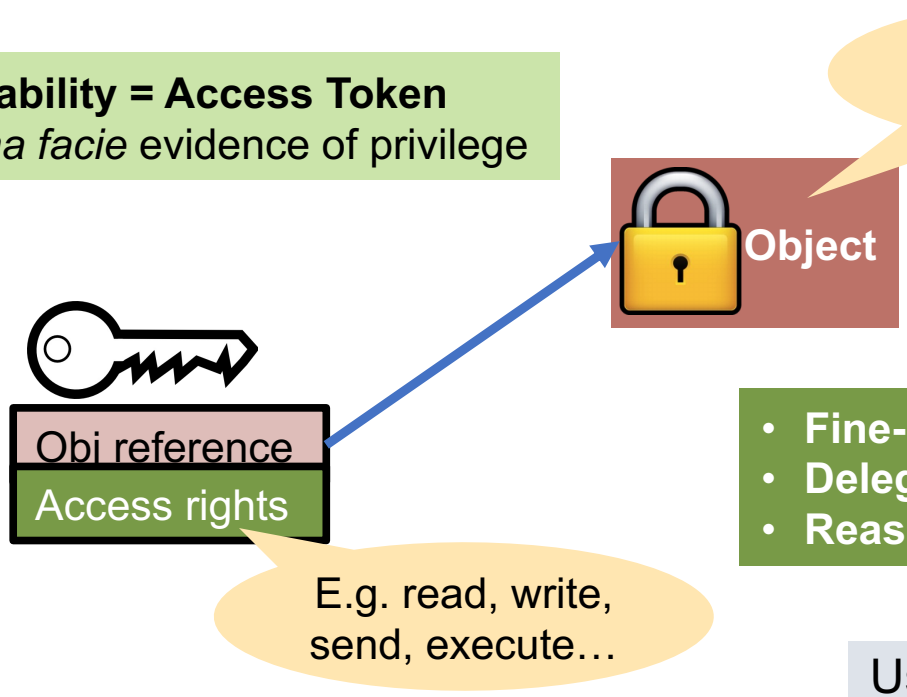
```
Obj1:  RW
Obj3:  RWX, own
Subj2: recv
```

Represented as a *capability list* (Clist)


© Gernot Heiser 2019–25 – CC BY 4.0

# (Object) Capabilities (aka Ocaps, Caps)

**Capability = Access Token**
*Prima facie* evidence of privilege




Linux "capabilities" do not have these properties (not *object* capabilities)!

- **Fine-grained access control**
- **Delegation of rights**
- **Reasoning about information flow**

Any operation is invoking a capability:

```
err = cap.method( args );
```

Used in very few commercial systems:
- IBM System/38→AS/400→i-Series
- KeyKOS [Bomberger et al, 1992]
- L4 microkernels, Google Fuchsia


© Gernot Heiser 2019–25 – CC BY 4.0

# Implementing Ocaps: Hardware

- Cap can be copied like data ⇒ “delegation”
- Tag is reset when modifying word
- CPU has capability registers

Revocation is hard!

i-node #,base address, …

r, w, c, d, …




Special bit in memory

- Historic capability machines:
  - IBM System/38 → AS/400 → i-Series
  - Hydra
- Revived with CHERI


© Gernot Heiser 2019–25 – CC BY 4.0

# Implementing Ocaps: Software-Usermode



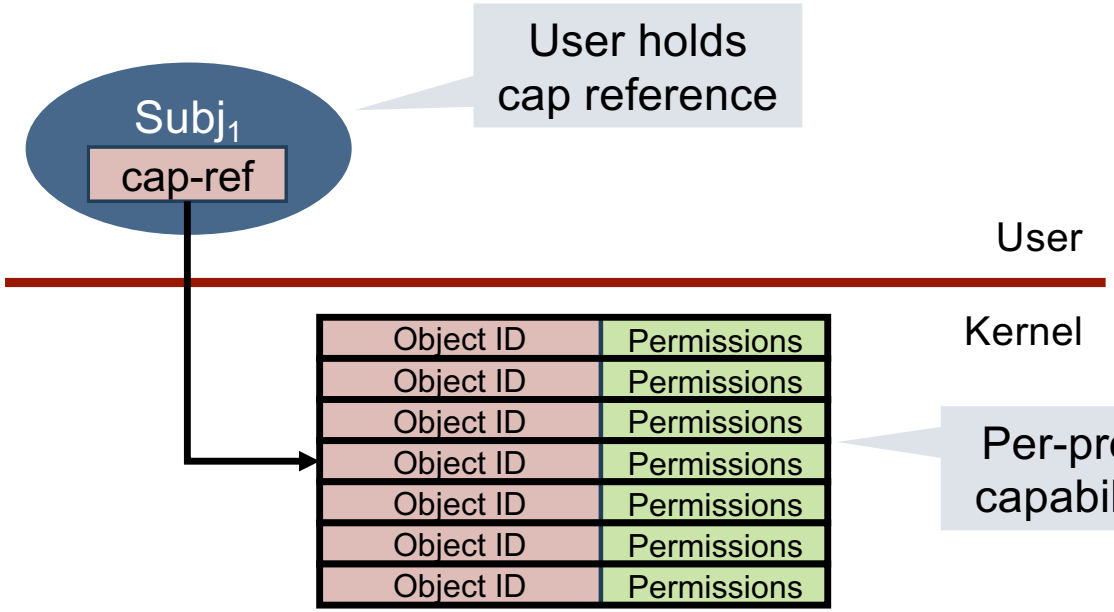

Revocation is hard!

- "Sparse capabilities"
- Cap can be copied like data ⇒ delegation
- Signature mismatch when modifying cap
- OS has object table, holds signatures and permissions

- Amoeba [Tanenbaum '81]
- Monash *Password Capability System* [Anderson '86]
- Mungi [Heiser '98]


© Gernot Heiser 2019–25 – CC BY 4.0

# Implementing Ocaps: Software-Kernel




- "Partitioned" or "segregated" caps
- Delegation is system call
- Revocation is easy

- Mach [Accetta '86]
- EROS [Shapiro '99]
- Modern L4 kernels
- Unix file descriptors


© Gernot Heiser 2019–25 – CC BY 4.0

# ACLs vs Capabilities


© Gernot Heiser 2019–25 – CC BY 4.0

# ACLs & Object Capabilities – Duals?

- In theory dual representations of access control matrix
- Practical differences:
  - Naming and namespaces
    - Ambient authority
    - Deputies
  - Evolution of protection state
  - Process creation
  - Auditing of protection state


© Gernot Heiser 2019–25 – CC BY 4.0

# Duals: Naming and Name Spaces

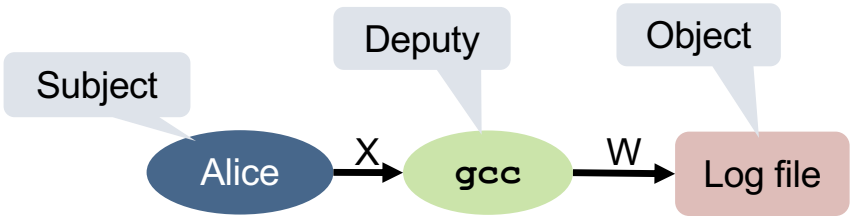
- ACLs:
  - objects referenced by **name**
    - requires separate (global) name space
    - e.g. open("/etc/passwd",O_RDONLY)
  - require a subject (class) namespace
    - e.g. UNIX users and groups
- Capabilities:
  - objects referenced by **capability**
  - no further namespace required –
    object discovery orthogonal to access control
  - cannot even *name* object without access

Covert storage channel?

Least common mechanisms!


© Gernot Heiser 2019–25 – CC BY 4.0

# Duals: Confused Deputy




**Unix:**
- Log file is group **`admin`**
- Alice not member of **`admin`**
- gcc is set-GID **`admin`**

```
alice$ gcc -o Log_file source.c
```

```
static char* log = "/var/gcc/log";
int gcc (char *src, *dest) {
    int s = open (src, RDONLY );
    int l = open (log, APPEND);
    int d = open (dest, WRONLY);
     …
    write (dest, …);
}
```

Clobber log!

- ACLs separate naming and permissions
- Deputy depends on *ambient authority*: Uses own privileges for access

*Confused deputy is* inherent problem of ACLs!


© Gernot Heiser 2019–25 – CC BY 4.0

# Deputy With Ocaps




```
alice$ gcc -o Log_file source.c
```

Invalid cap

```
static cap_t log = <cap>;
int gcc (cap_t src, dest) {
    fd_t s = open (src, RDONLY );
    fd_t l = open (log, APPEND);
    df_t d = open (dest, WRONLY);
    …
    write (d, …);
}
```

Open fails!

**Object capability (Ocaps) system:**

- gcc holds **`w`** cap for log file
- Alice holds **`r`** cap for source, **`w`** cap for destination
- Alice holds no cap for log file

- Caps are both names and permissions
- Presentation is explicit, not ambient
- Can't name object if don't have access!

Linux "capabilities" don't help!

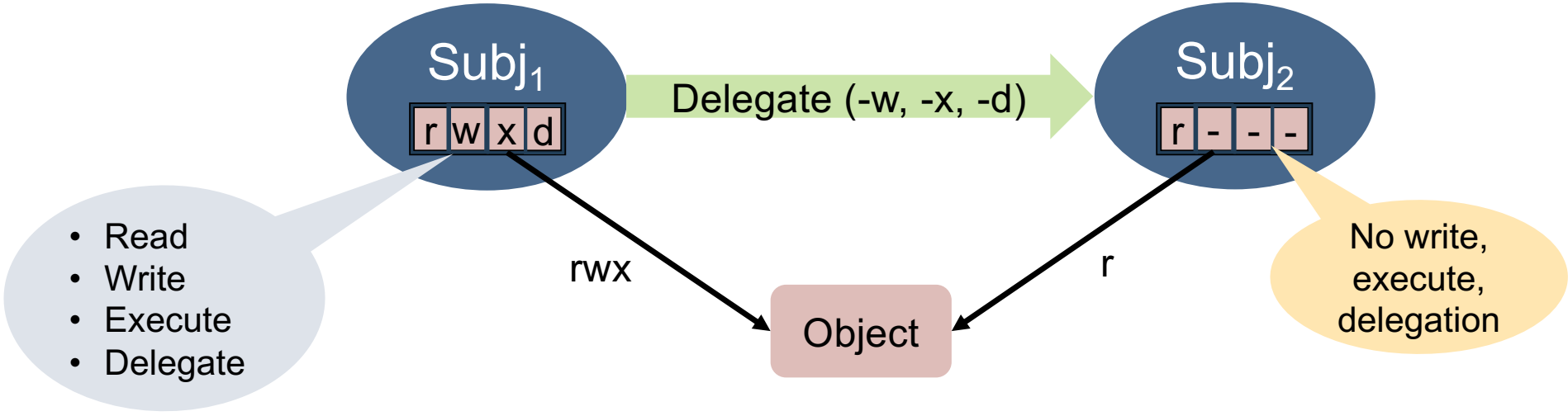

© Gernot Heiser 2019–25 – CC BY 4.0

# Duals: Evolution of Protection State

**ACLs**: Protection state changes by modifying ACLs

- Requires certain meta-permissions on the ACL

**Capabilities**: Protection state changes by delegating and revoking caps





© Gernot Heiser 2019–25 – CC BY 4.0

# Duals: Evolution of Protection State

**ACLs**: Protection state changes by modifying ACLs

- Requires certain meta-permissions on the ACL

**Capabilities**: Protection state changes by delegating and revoking caps

**Caps support reasoning about information flow:**

- A can send message to B only if A holds cap to B
- A can obtain access to C only if it receives message with cap to C


© Gernot Heiser 2019–25 – CC BY 4.0

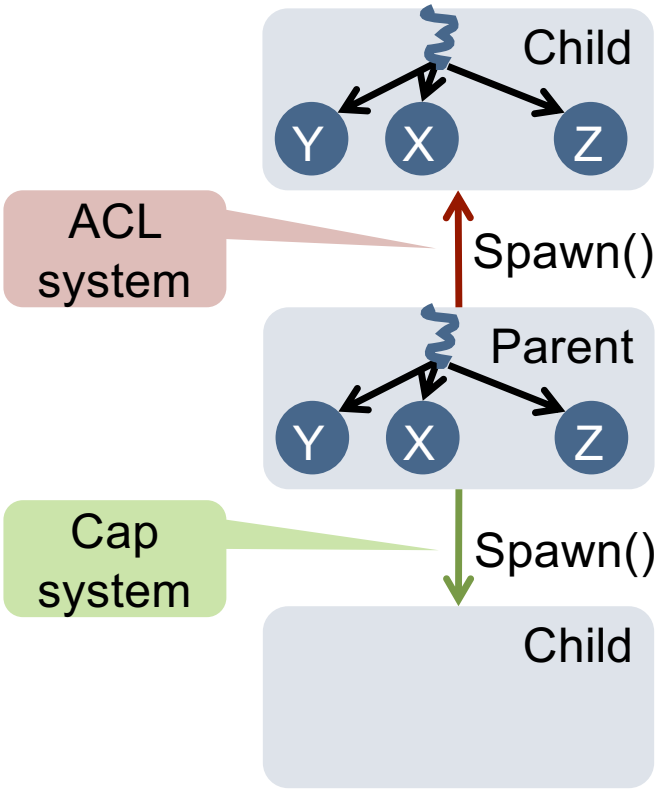
# Duals: Process Creation – Child Permissions

**ACLs:**
- Permissions defined by child's subject
- Generally inherit all of parent's rights

Max privilege

**Caps :**
- Child is created with no rights
- Parent gets cap to child, can delegate as needed

Least privilege





© Gernot Heiser 2019–25 – CC BY 4.0

# Duals: Auditing of Protection State

- Who has permission to access a particular object (right now)?
  - ACLs: Just look at the ACL
  - Caps:
    - sparse or tagged caps: hard/impossible without full memory scan
    - partitioned caps: doable (maintain derivation tree)
- What objects can a particular subject access (right now)?
  - Capabilities: Just look at its capabilities
  - ACLs: may be impossible to determine without full scan

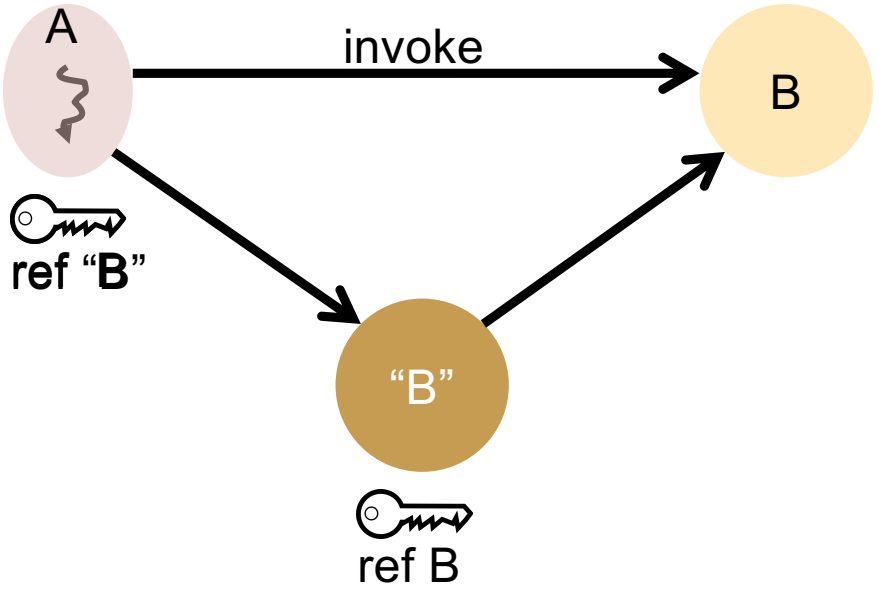
"Who can access my stuff?"
vs
"How much damage can X do?"


© Gernot Heiser 2019–25 – CC BY 4.0

# Interposing Access

**Caps are opaque object references (pure names)**

- Holder cannot tell which object a cap references nor the authority
- Supports transparent interposition (virtualisation)

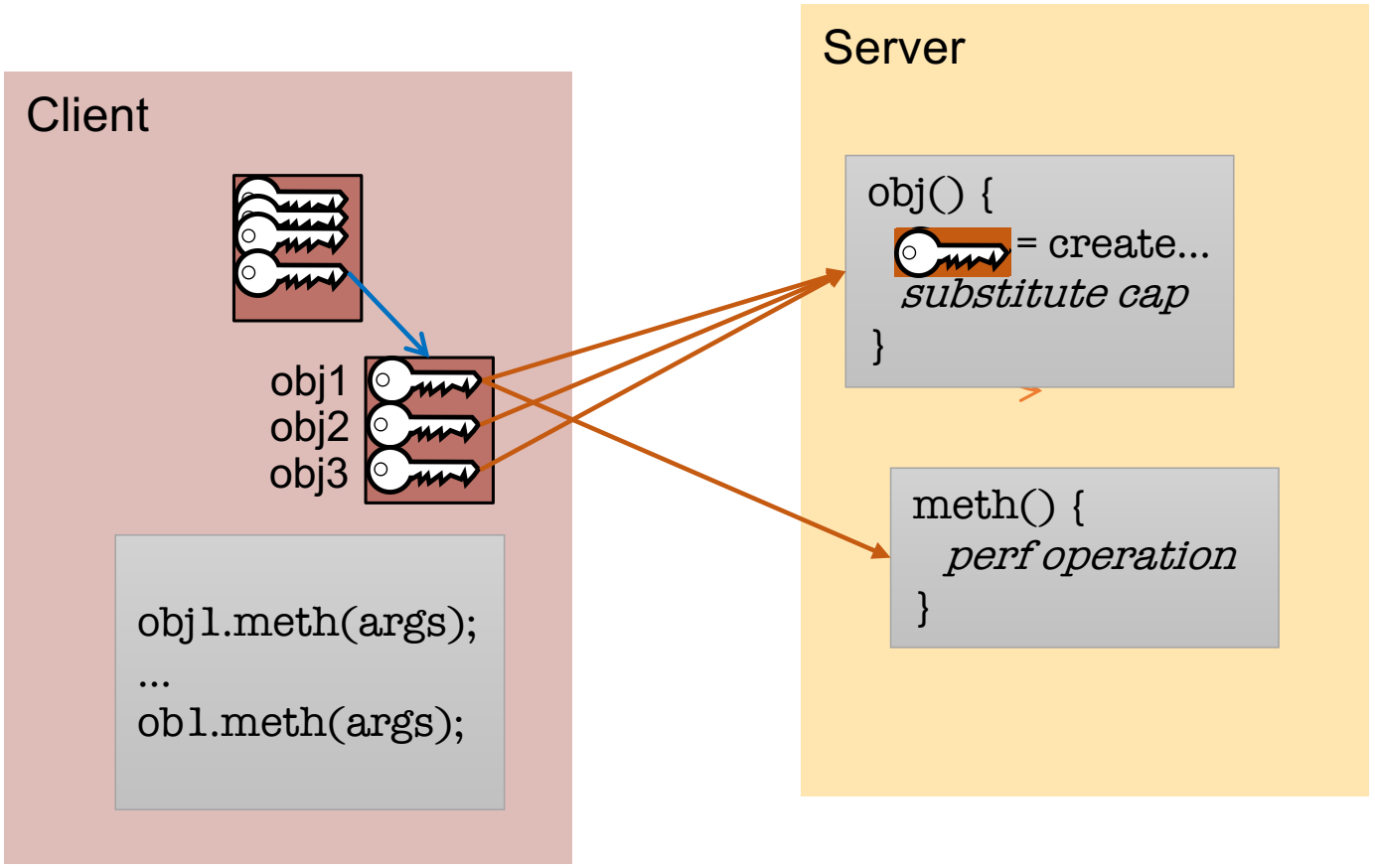



**Usage:**

- API virtualisation
- Reference (security) monitor
  - Security policy enforcement
  - Info flow tracing
  - Packet filtering…
- Secure logging
- Debugging
- Lazy object creation


© Gernot Heiser 2019–25 – CC BY 4.0

# Example: Lazy Object Construction





© Gernot Heiser 2019–25 – CC BY 4.0

# Duals: Satzer & Schroeder Principles

| Security Principle | ACLs | Capabilities |
|---|---|---|
| Economy of Mechanism | Dubious | Yes! |
| Fail-safe defaults | Generally not | Yes! |
| Complete mediation | Yes (if properly done) | Yes (if properly done) |
| Open design | Neutral | Neutral |
| Separation of privilege | No | Doable |
| Least privilege | No | Yes |
| Least common mechanism | No | Yes, but… |
| Psychological acceptability | Neutral | Neutral |


© Gernot Heiser 2019–25 – CC BY 4.0

# Reminders

- Arista Advanced Operating Systems Prize for top performer in this course
- OS Hall of Fame for straight HDs in OS, AOS, OS Thesis
- Honours theses at Trustworthy Systems
  https://trustworthy.systems/students/theses
- **John Lions CS Honours Award for thesis in OS**
  https://www.scholarships.unsw.edu.au/scholarships/id/1993/7299
  **Deadline: 30 Novemberfor T1/27!**


© Gernot Heiser 2019–25 – CC BY 4.0