TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science Institute for System Architecture, Operating Systems Group

Security - Introduction

Benjamin Engel

Dresden, 2010-12-21

Overview

- Basics: Security policies and mechanisms
- Bell – La Padula & Biba
- Access control
- Capabilities
- Naming

- Information Flow (Control)
- Non-Interference
- Formal methods: verification and evaluation

Confidentiality, Integrity, Availability

Confidentiality: Data is only accessible to those with appropriate rights; no statement about integrity

Integrity: Data is either unmodified (authentic) or tampering is provable; no statement about confidentiality

Availability: Timely access to resources is guaranteed to authorized users

Secure System[*]: A secure system is a system that starts in an authorized state and cannot enter an unauthorized state.

Security Policy[*]: A security policy partitions the states of the system into a set of authorized, or secure, states and a set of unauthorized, or nonsecure, states.

[*] *Matt Bishop: Computer Security – Art and Science*

Security Policies vs. Mechanisms

Security Policy:

- A security policy states what is allowed, and what isn't.
- e.g.: SELinux policy, /etc/passwd

Security Mechanism:

- A security mechanism is a method, tool, or procedure for enforcing a security policy
- e.g.: Capabilities, ACLs, MMU ...

Principle Of Least Authority – POLA

“Every program and every user of the system should operate using the least set of privileges necessary to complete the job.”

(Saltzer and Schroeder, 1974)

Bell – La Padula Model

- Developed in the 1970s, demand for access control mechanisms solving problems of security in computer systems

- Main focus on *Confidentiality*

- State transition system: Define a set of secure states, transition function ensures to stay in this set (enter no insecure state)

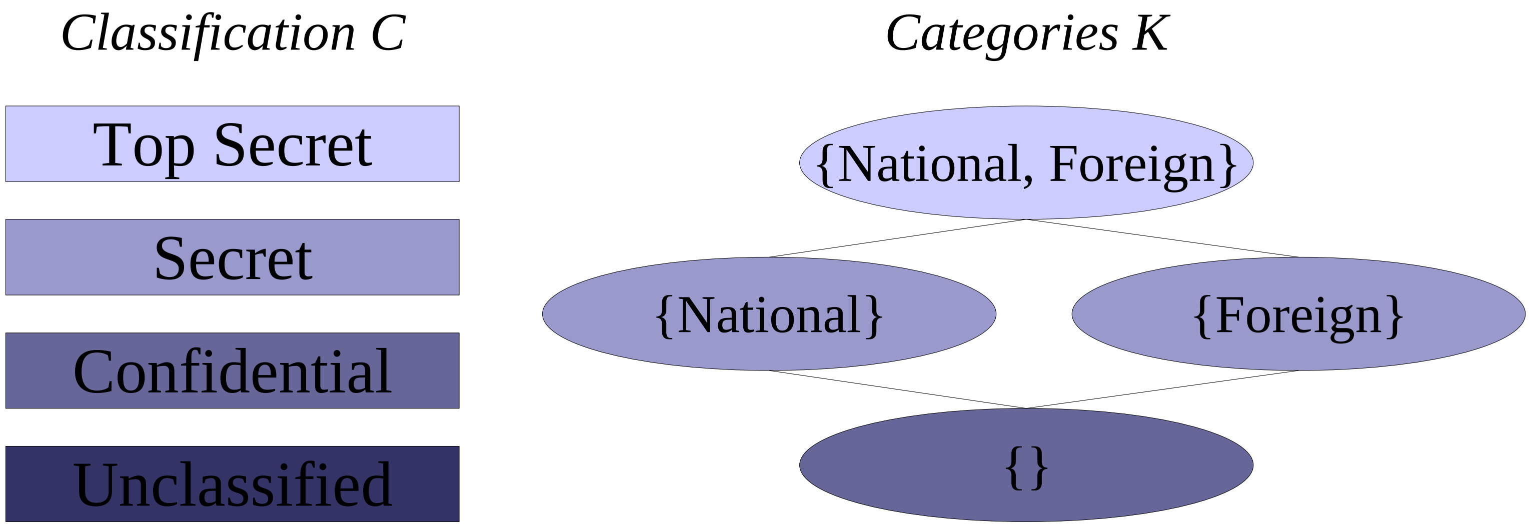
Elements of the Model

Set	*Elements*	*Semantics*
S	$\{S_1, S_2, \ldots S_n\}$	*Subjects*; processes in execution
O	$\{O_1, O_2, \ldots O_m\}$	*Objects*; data, files, programs, subjects
C	$\{C_1, C_2, \ldots C_q\}$ $\{C_1 > C_2 > \ldots > C_q\}$	*Classifications*; clearance level of a subject, classification of an object
K	$\{K_1, K_2, \ldots K_n\}$	*Needs-to-know categories;* project number, access privileges

Classification and Categories

Subjects and objects have a *security label* (C,K) consisting of a *security level* C and a *category set K*, both are orthogonal to each other

dominates relation: $C_1 \geq C_2$ && $K_1 \supseteq K_2$



Bell – La Padula Transition Rules

- Example: Label L_1 (Top Secret, {National}) dominates Label L_2 (Unclassified,{})
- Simple Security Condition: S can read O if S dominates O (**no reads up**)
- *-Property: S can write to O if O dominates S (**no writes down**)
- Declassification through **trusted subjects**

No reads up – no writes down

Bell – La Padula Example

Classification : p.noel > lutins > enfants
Categories : cadeaux, rodolphe, lettres

No reads up – no writes down

- enfants, {} write p.noel, {lettres}

Bell – La Padula Example

Classification : p.noel > lutins > enfants

Categories : cadeaux, rodolphe, lettres

No reads up – no writes down

- enfants, {} write p.noel, {lettres} ✔

Bell – La Padula Example

Classification : p.noel > lutins > enfants
Categories : cadeaux, rodolphe, lettres

No reads up – no writes down

- enfants, {} write p.noel, {lettres} ✔
- lutins, {lettres, cadeaux} read enfants, {lettres}

Bell – La Padula Example

Classification : p.noel > lutins > enfants

Categories : cadeaux, rodolphe, lettres

No reads up – no writes down

- enfants, {} write p.noel, {lettres} ✔
- lutins, {lettres, cadeaux} read enfants, {lettres} ✔

Bell – La Padula Example

Classification : p.noel > lutins > enfants

Categories : cadeaux, rodolphe, lettres

No reads up – no writes down

- enfants, {} write p.noel, {lettres} ✔
- lutins, {lettres, cadeaux} read enfants, {lettres} ✔
- lutins, {lettres} write enfants, {lettres}

Bell – La Padula Example

Classification : p.noel > lutins > enfants

Categories : cadeaux, rodolphe, lettres

No reads up – no writes down

- enfants, {} write p.noel, {lettres} ✔
- lutins, {lettres, cadeaux} read enfants, {lettres} ✔
- lutins, {lettres} write enfants, {lettres} ✘

Bell – La Padula Example

Classification : p.noel > lutins > enfants

Categories : cadeaux, rodolphe, lettres

No reads up – no writes down

- enfants, {} write p.noel, {lettres} ✔
- lutins, {lettres, cadeaux} read enfants, {lettres}✔
- lutins, {lettres} write enfants, {lettres} ✘
- p.noel, {} read lutins, {rodolphe}

Bell – La Padula Example

Classification : p.noel > lutins > enfants

Categories : cadeaux, rodolphe, lettres

No reads up – no writes down

- enfants, {} write p.noel, {lettres} ✔
- lutins, {lettres, cadeaux} read enfants, {lettres} ✔
- lutins, {lettres} write enfants, {lettres} ✘
- p.noel, {} read lutins, {rodolphe} ✘

Bell – La Padula Example

Classification : p.noel > lutins > enfants
Categories : cadeaux, rodolphe, lettres

No reads up – no writes down

- enfants, {} write p.noel, {lettres} ✔
- lutins, {lettres, cadeaux} read enfants, {lettres} ✔
- lutins, {lettres} write enfants, {lettres} ✘
- p.noel, {} read lutins, {rodolphe} ✘
- p.noel, {rodolphe} read lutins, {rodolphe}

TECHNISCHE UNIVERSITÄT DRESDEN

Bell – La Padula Example

Classification : p.noel > lutins > enfants

Categories : cadeaux, rodolphe, lettres

No reads up – no writes down

- enfants, {} write p.noel, {lettres} ✔
- lutins, {lettres, cadeaux} read enfants, {lettres} ✔
- lutins, {lettres} write enfants, {lettres} ✘
- p.noel, {} read lutins, {rodolphe} ✘
- p.noel, {rodolphe} read lutins, {rodolphe} ✔

Bell – La Padula Summary

- **Information flow** policy, that preserves *confidentiality*
- Very simple model, proof of model's security properties is trivial, practical proof is hard
- No integrity concerns in the model (use Biba)
- Shortcomings:
  - Too simple, many scenarios cannot be expressed by this model (e.g. device drivers has to be used by all security levels)
  - Purely confidentiality centric
  - Central, system wide, policy (global labels)

Biba Model

- Developed in the 1970s (after Bell – La Padula)
- In contrast to the Bell – La Padula model, it focuses on data *integrity*
- Many similarities to Bell – La Padula:
  - Facilitates also a state transition system
  - Objects are ordered by **integrity** levels
  - Rules are inverse to BLP (no reads from lower integrity levels, no writes to higher ones)

No reads down – no writes up

Access Control

- **Information flow** describes how data is spread throughout the system
- **Information flow control** states which flows are allowed (policy) and restricts distribution of data accordingly (mechanism)
- In contrast **access control** states who can access what using which operation
- Prominent example : **A**ccess **C**ontrol **M**atrix

DAC, RBAC, MAC

- Discretionary access control
  - privileged instance (e.g. owner) related to an object decides who is allowed to access it, permissions might be passed to other subjects

- Role-based access control
  - Operations are permitted based on roles, not directly on subjects
  - Powerful enough to simulate DAC and MAC

- Mandatory access control
  - system rules, that cannot be altered by an individual user (SELinux, AppArmor)

Access Control Matrix: ACL vs. Caps

Access Control List

Capabilities

	O_1	O_2	O_3	...	O_n
S_1		r,w	r		
S_2	x		w		
S_3	w,x				m
...					
S_m	a	c,d			

read, write, execute, append, create, delete, map, ...

TECHNISCHE UNIVERSITÄT DRESDEN

Access Control List

- Tied to the objects (classic example: file access rights in Unix/Windows)

- For each object (or group of objects): which subjects are allowed to perform which operation

- Changing of permissions easy: right at the object

Capabilities

- Bound to the subject (compare: ticket system)
- States which permissions a subject has on specific objects
- Hard to express group relations (indirection)
- Changing (revoking) permissions is difficult ... "Whom I gave access rights to foobar?"
  - Tracking of granted permissions
  - How to invalidate a ticket once given it away

Capabilities – Name and Rights

- Designate/name a specific object plus access rights to that object
- Sole possession of a cap is sufficient to prove ones authority to perform an operation
- Implementation using hardware support, memory protection mechanisms or cryptography

Capability Systems

• KeyKOS:	Persistence, one run 17 years
• EROS:	Extremely Reliable OS
• Coyotos:	Towards formally verification
• Amoeba:	Transparent distributed system
• SeL4:	First formally verified Microkernel
• Fiasco.OC:	Successor to Fiasco
• NOVA:	Microhypervisor

Capabilities: Kernel vs. User

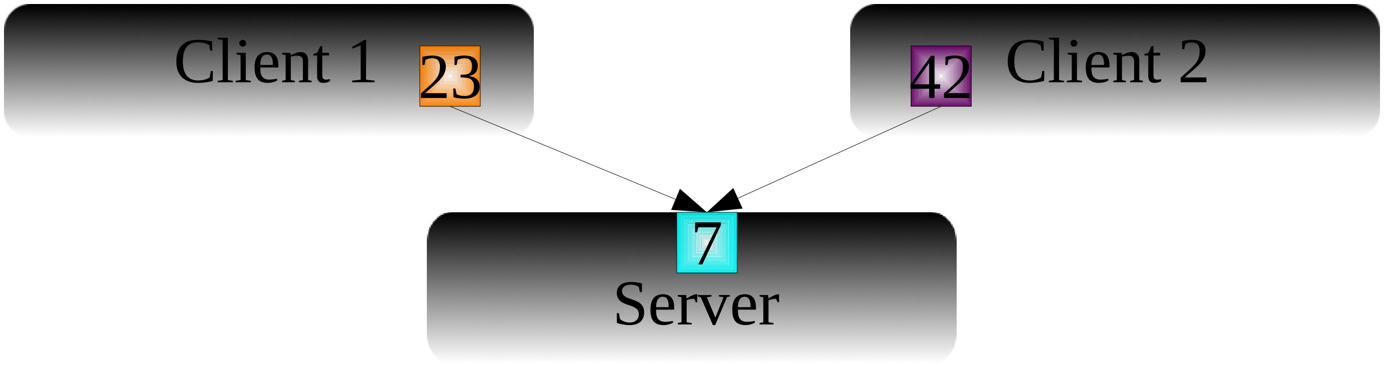
- Kernel
  - Protected by kernel
  - User gets only a handle
  - Compare to File Descriptor in Unix
  - Easier to revoke

- User
  - protect against tampering (Amoeba: a cap is a 128 bit value, protected by cryptography)
  - Persistency: user responsibility, for the kernel it's just a value

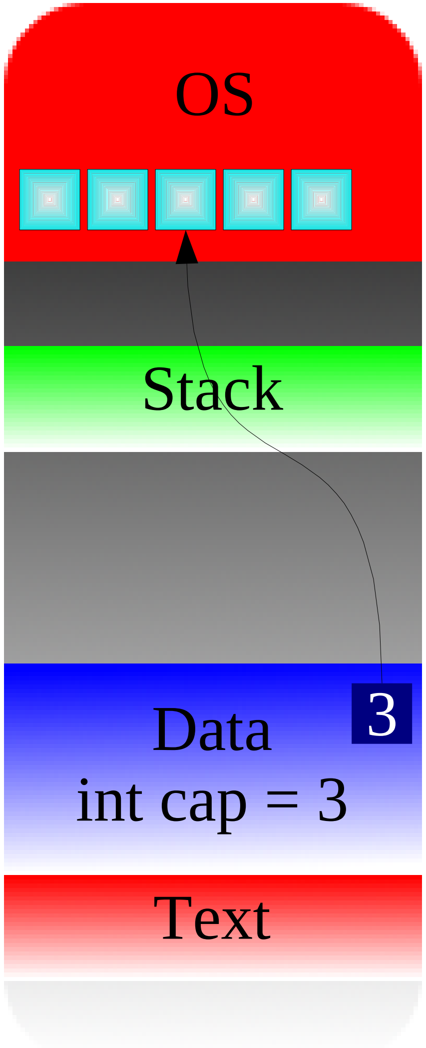
Capabilities – Usage

- Server offers its service by
  - Creating a portal (=kernel object)
  - Get a new (portal) capability at cap index 7
  - Send the capability at index 7 to its clients

- Clients receive the capability locally at index 23 or 42 and send messages to this portal



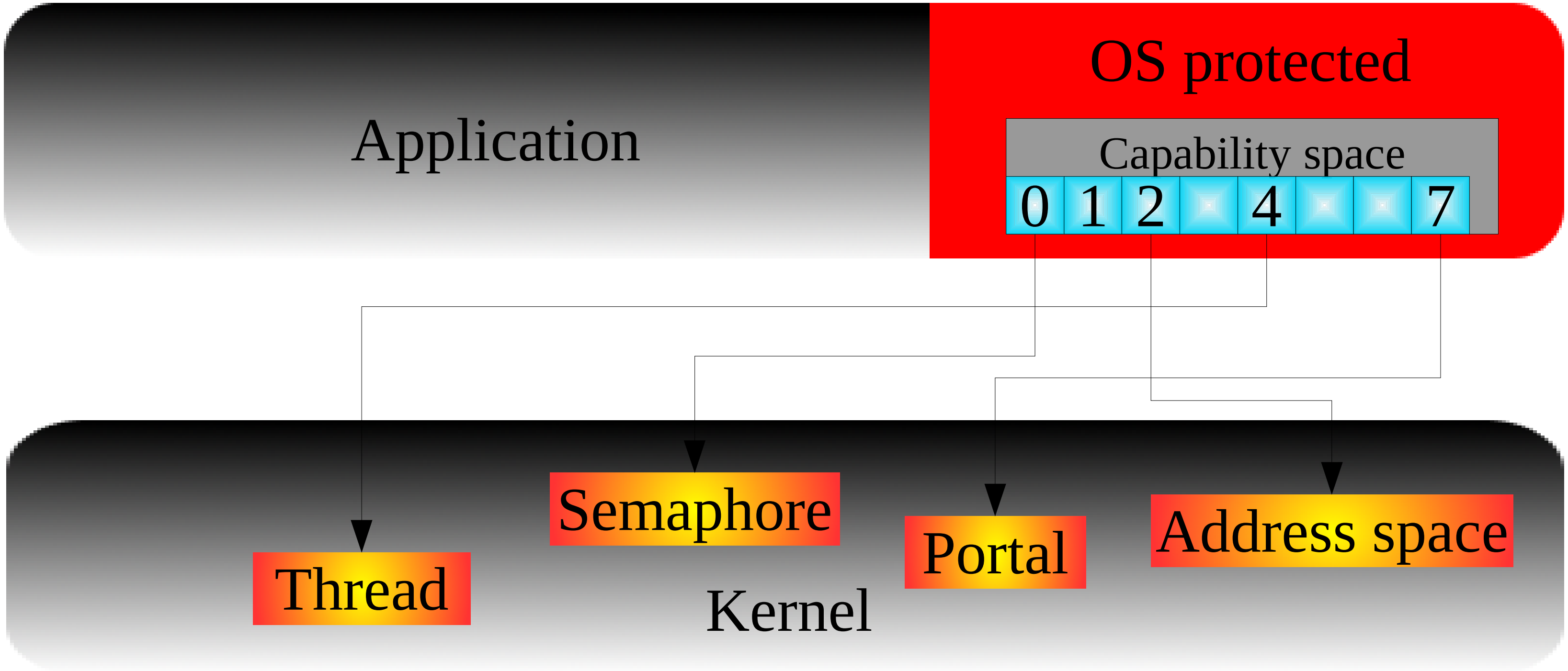
Capabilities – Implementation



- Within the address space of a task, accessible by the OS only, is a capability space
- Double indirection: user gets an index (3) into an array of pointer to kernel objects
- When creating new kernel objects, a new capability is created, user needs to specify where to put the handle
- Backed by kernel memory

Capability Space

- Application has references to kernel objects
- Referred via index into cap space
- Caps might be transferred to other tasks

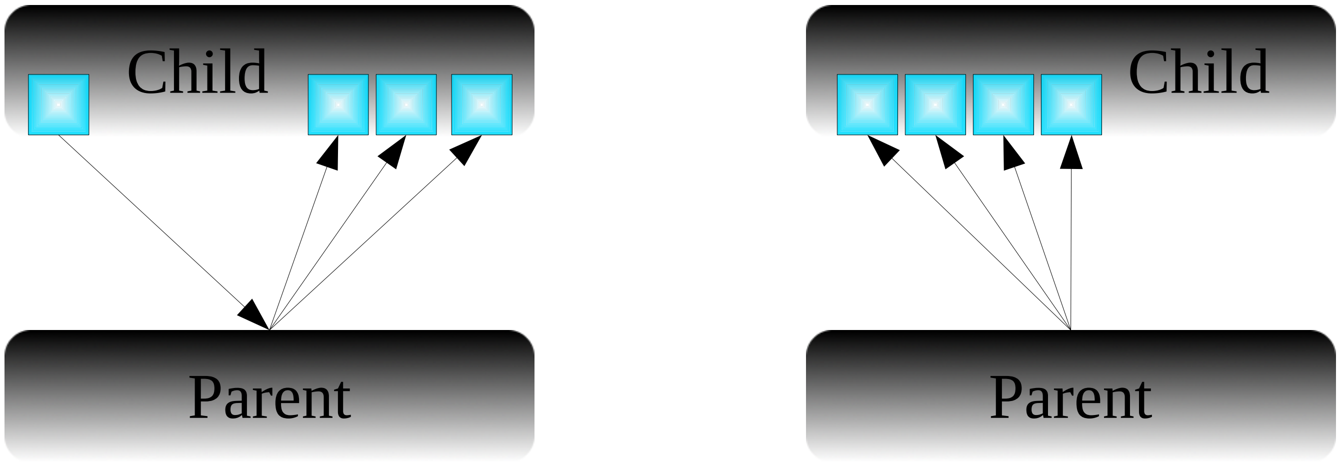


Capabilities - Bootstrapping

or:

How do new applications get their (initial) capabilities?

Task Creation



- Child is created with only one cap
- Further caps are requested from the parent or someone else (servers, ...)

- Predefined set of initial caps at well-known cap space indices
- Receive further caps via request + map

TECHNISCHE UNIVERSITÄT DRESDEN

Receiving Capabilities

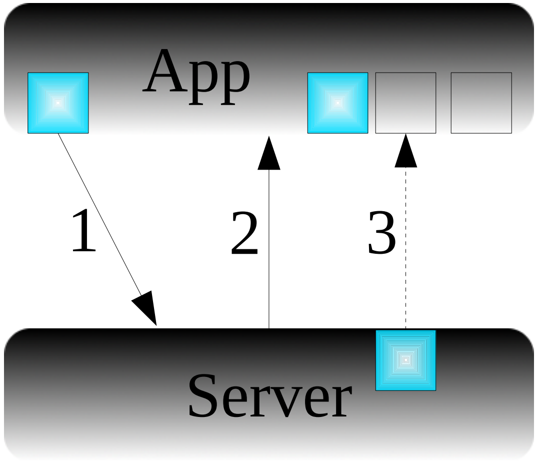
- Initial Task Creation
  - The creator possesses the capability to the newly created task
  - Task cap is very powerful, allows to place new caps in its cap space

- Receive via IPC
  - Prepare receive window, send a request to someone (parent, server, ...) asking for caps
  - During reply the requested caps will be mapped to own cap space

Initial Task Creation

- Initial set of caps
  - Parent: capability to your parent
  - Mem_alloc: memory allocator
  - Log: logging facility
  - Thread: first application thread
  - Rm: region manager / pager
  - Factory: factory to create objects
  - Task: the task itself

Receiving via IPC



1 App invokes an IPC-Gate, thereby calling the server behind this gate
2 Server replies, sending the requested cap along
3 During reply the kernel transfers/copies the specified capability to the receiver

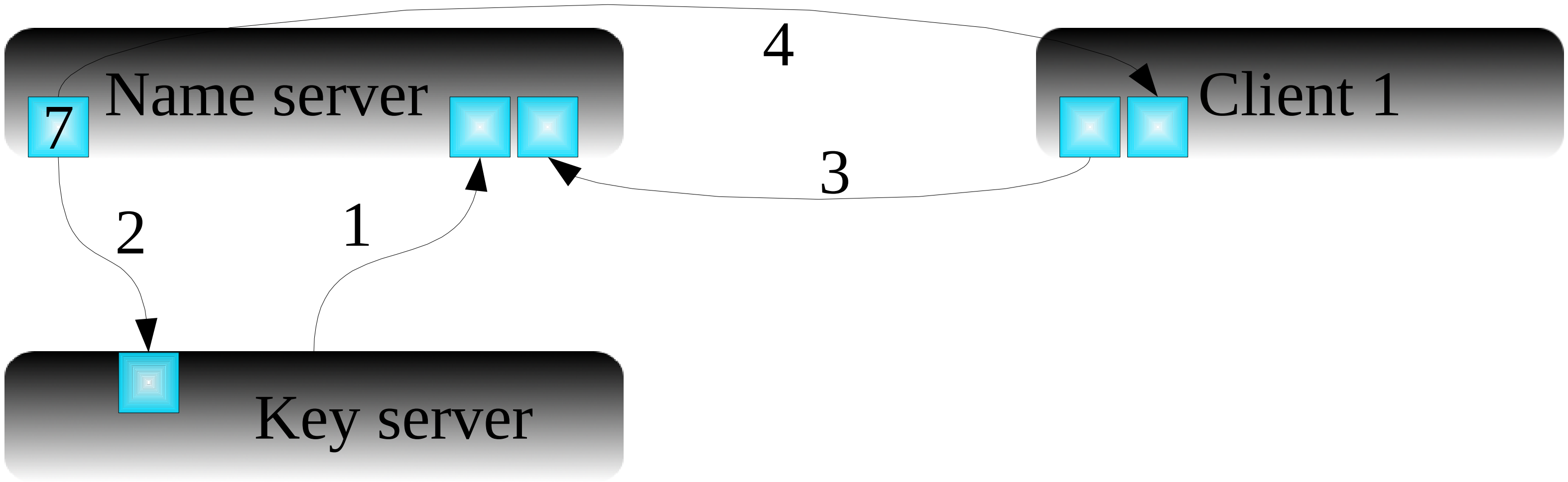
L4Re Example

```
-- abbreviation
l = L4.default_loader;

-- new communication channel
c = l:new_channel();

-- start the server
l:start ( { caps = { service = c:srv()}},
  “rom/server” );

-- start the client
l:start ( {caps = { server = c}},
  “rom/client” );
```

Service discovery

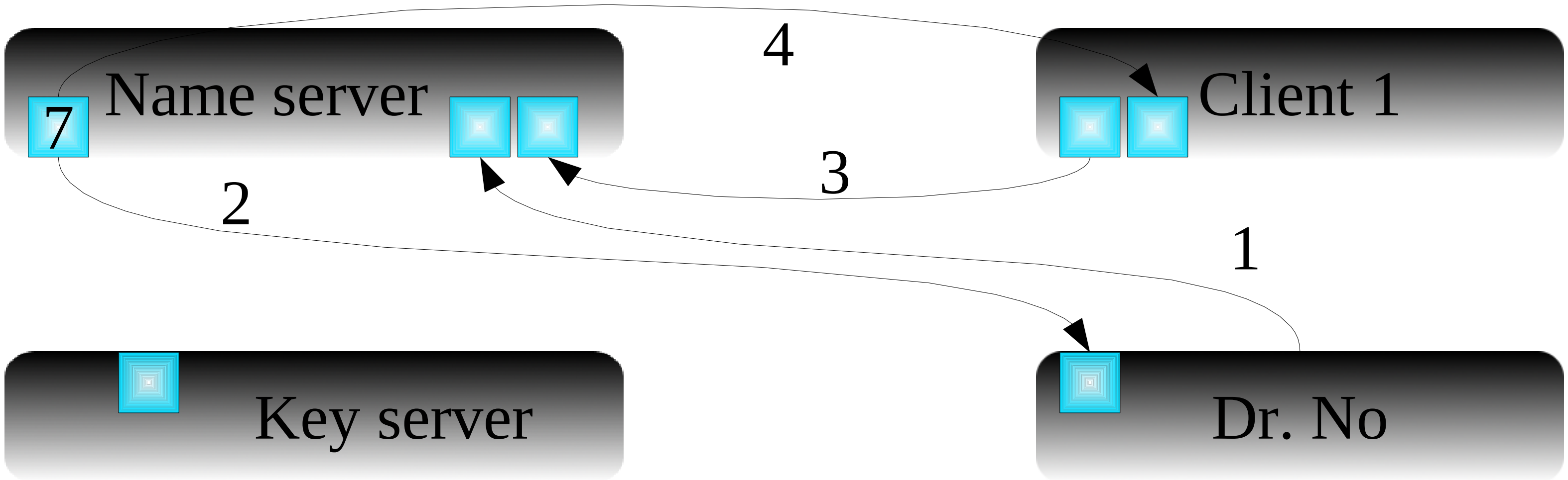
- Whom do I ask ?
- What do I ask for ?

Name Resolution – Example



- Key server registers itself at the name server, sending a cap along the message
- Name server receives name + cap
  - Mapping “Key server” → cap 7
- Client queries Name server, receives cap to the key server
- Client contacts key server for service

Name Resolution – Havoc



- Dr. No contacts the name server, registers itself under the name “key server”
- Key server tries to register itself, but fails since the name “key server” has already been taken
- Client queries, gets a cap, contacts “key server” (impersonated by Dr. No) → GAME OVER

One Step Back

- Naming issues are coupled with security
- Where to get capabilities from
- How to name objects
- How does service discovery work

Names are resources, have to be managed

Local vs. Global Name Spaces

- Global name spaces
  - All instances share the same view

    There is only one global key server, impersonation doesn't work
  - Classical in monolithic systems
  - Easy to configure
  - Recap: BLP security levels → global

- Local name spaces
  - Instances have private name spaces
  - Forwards *principle of least privilege*
  - Common examples: BSD jails or chroot

Problems with Global Names

- Communication
- Example: L4 thread ids were globally visible
- Everyone can send IPC to everyone
  - Clans and chiefs
  - Reference monitor
  - Ports, endpoints, gates, portals, ...
  - Language based approaches (Sing#)
- **D**enial **o**f **S**ervice attacks are possible
- No full isolation (covert channels)
- Solution: local names = name spaces

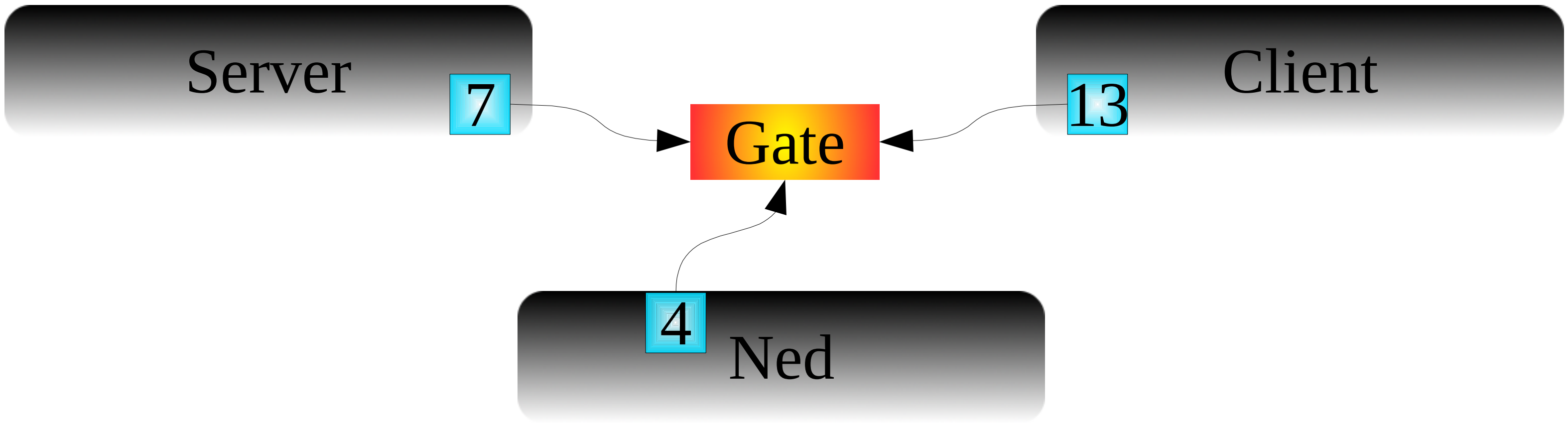
TECHNISCHE UNIVERSITÄT DRESDEN

Local Names

- Task local name space
- Initially populated by task's creator
  ... whom you have to trust anyway
- Mapping from name to capability
- Additional entries through querying
  - Name server
  - Parent → hierarchical name resolution (compare with DNS)

- Not perfect: Receiving a capability, how to figure out if I already have it (cap compare)?

Populating Local Name Spaces



- Ned creates a new Gate, receives a cap (4)
- Map this cap with server rights to Server's address space (7), add a new entry in Server's name space: “server” → cap
- Map the same cap with client rights to Client's address space (13), add name space entry there too, “service” → cap

Review & outlook

- Review
  - Security models (Bell – La Padula, Biba)
  - Access Control Matrix
  - Capabilities
  - Naming

- Next lecture
  - Information Flow
  - Non-Interference
  - Software verification