OS Security

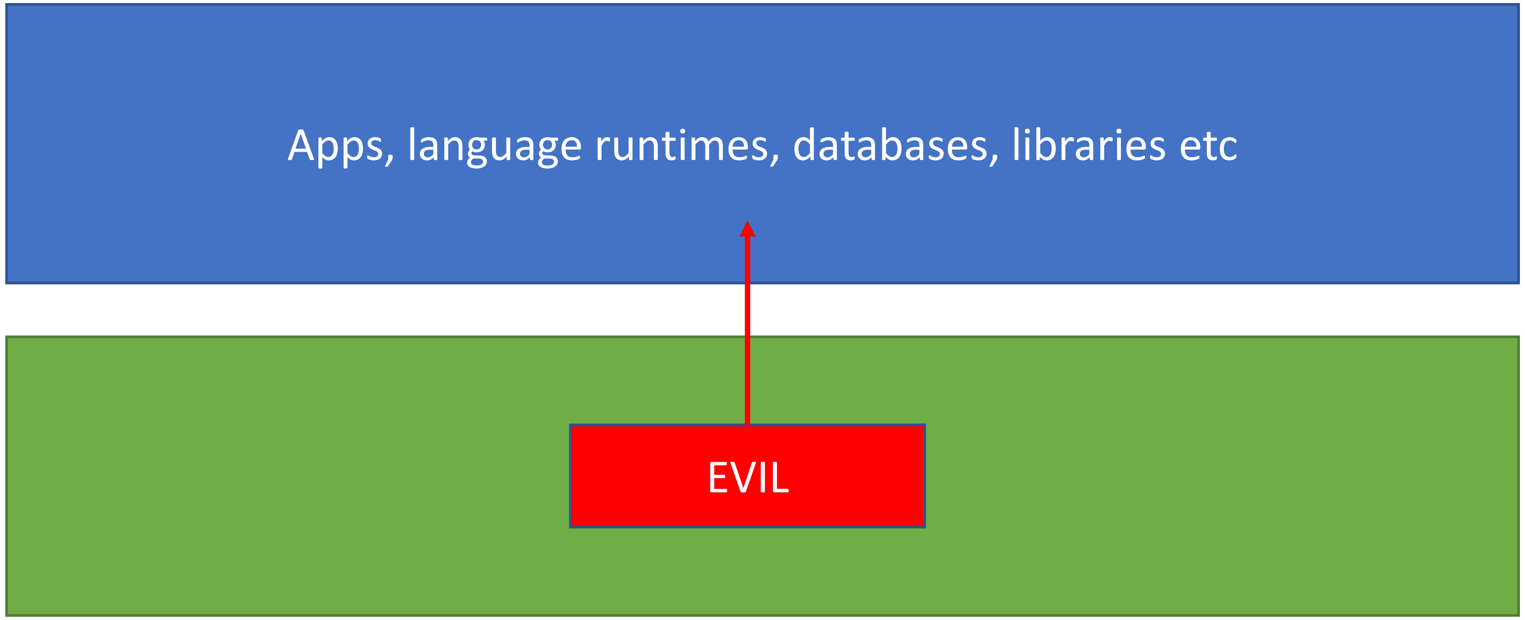
Questions Answered in this Lecture:

- Why does security matter for operating systems?
- What are some design concerns with security abstractions?
- What are instances where (poor) OS security has caused problems?

Announcements

- P4b due Friday (phew!)
- P4a grades in the works, p3* should be up already
- Course evals!

Why is OS security important?



Security of OS affects everyone

- **If your OS is compromised, you can't trust *anything***
- That includes:
  - Compilers
  - Libraries
  - Text Editors
  - Any process!

- Oh by the way, the OS controls hardware ☺

Why is it hard?

- **Operating Systems are complex**
  - The bigger the codebase, **bugs more likely**
  - More “entry points” (attack surface)
- Support *many* programs (multi-programming), and one being insecure can’t break things!
- Security is a “cross-cutting” issue. It’s hard to separate out and “assign” it to one developer

Protection: What's at stake?

- Access to *any* process's memory
- Access to *anything* on persistent storage
- Kill processes, muck with scheduling
- Access the network in any way
- Control/manipulate devices
- Change available resources for processes
- Information returned from the kernel!

Security Goals

- **Confidentiality**: information can be hidden from others
- **Integrity**: My stuff doesn't change arbitrarily!
- **Availability**: If something should be available, don't let others bring it down
- Also: we want to share resources/state, but in a *controlled way*. E.g. only my group member can access mapper.c

Security goals can be achieved with policies:

- “Only users in my group may read this file”
- “By default, every process has distinct page tables”
- “Only the user with UID 0 may add device drivers to the kernel”
- Etc etc

Design Principles

- Following some guidelines will help (but not guarantee) result in secure systems

Economy of mechanism

- "Keep it simple stupid! (KISS)"
- Simplicity reduces bugs, makes it easier to envision misuse, fewer "entry points"

How many ways to get in?

Fail-safe defaults

- Default to security!
- Default configurations, options, behaviors should be the *most* secure by default, not the other way around

Mediation

- If possible *every* action taken in the system should be mediated (checked to see if it adheres to our security policies)
- This is **often not possible** because we have other design constraints (performance) that we have to meet

Open Design

- Assume attacker can pick apart your system
- Note this doesn't mean you have to publicize your code/system design
  - But you should **assume that attacker has managed to get it anyhow**
- Corrolary: Security by obscurity does not work!

Separate Privilege

- Critical actions require (>=) two sets of credentials
- E.g. something you know with something you have
- Something you know with something you *are*
- Use a two-man rule…

Principle of Least Privilege

- Only give privilege to users/entities/processes that is *necessary*. No more.
- You may trust a particular user, but do you trust them not to be compromised?
- Example: “ping” program needs privileged access to network card. Should we allow elevated privileges when ping runs?

Least Common Mechanism

- For each entity in the system, e.g. users or processes, use different state or mechanisms to manipulate them
- Every process gets its own page table
- What about shared libraries?

Acceptability

- Security cannot come at the cost of too much complexity
- If barrier to entry is too high, it won't be used. Corollary: users are lazy.
- Example…PGP
- How many people have Ubikeys?
- How many have burned a one-time pad to a CDROM?

Safety not Guaranteed!

seL4: Formal Verification of an OS Kernel

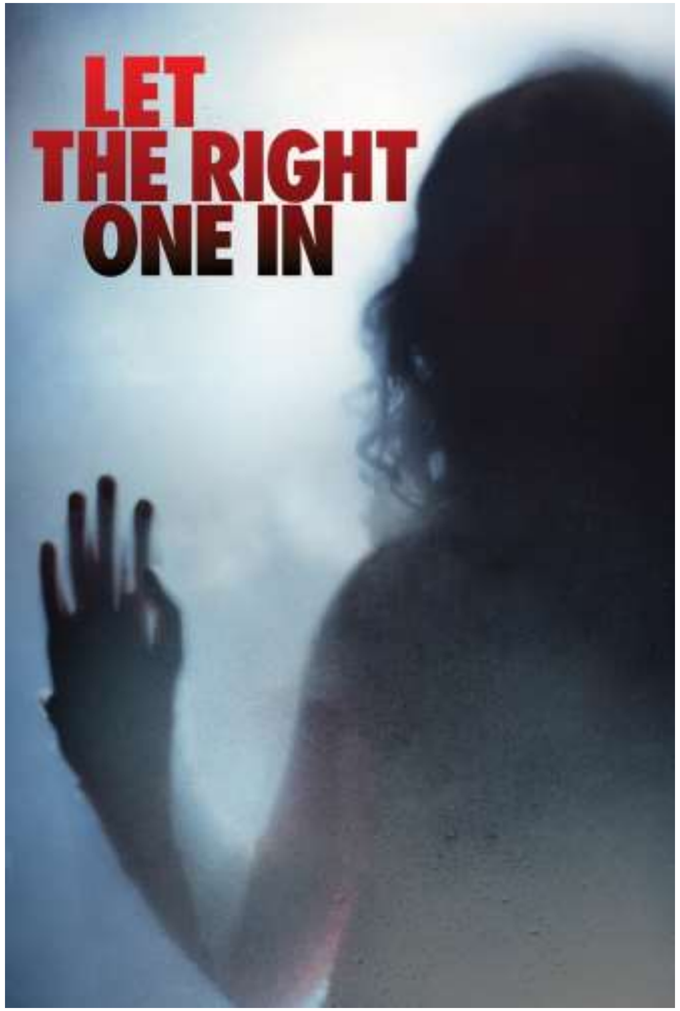
Gerwin Klein[1,2], Kevin Elphinstone[1,2], Gernot Heiser[1,2,3]
June Andronick[1,2], David Cock[1], Philip Derrin[1*], Dhammika Elkaduwe[1,2‡], Kai Engelhardt[1,2]
Rafal Kolanski[1,2], Michael Norrish[1,4], Thomas Sewell[1], Harvey Tuch[1,2†], Simon Winwood[1,2]
[1] NICTA, [2] UNSW, [3] Open Kernel Labs, [4] ANU
ertos@nicta.com.au

Abstract
Complete formal verification is the only known way to guarantee that a system is free of programming errors.

We present our experience in performing the formal, machine-checked verification of the seL4 microkernel from an abstract specification down to its C implementation. We assume correctness of compiler, assembly code, and hardware, and we used a

proach is to reduce the amount of privileged code, in order to minimise the exposure to bugs. This is a primary motivation behind security kernels and separation kernels [38, 54], the MILS approach [4], microkernels [1, 12, 35, 45, 57, 71] and isolation kernels [69], the use of small hypervisors as a minimal trust base [16, 26, 56, 59], as well as systems that require the use of type-safe languages for all code except some "dirty" core [7, 23]. Similarly, the Common Cri-

Authentication



Authentication

- At some point we need to answer the question can person X perform action A?
- But how do we identify a *person or a* ***principal*** in the OS context?
- It's not, after all, the *person* that's invoking system calls, or dereferencing pointers to deadbeef virtual addresses!
- Some entity on the system (**agent**) is doing it on the person's behalf
- Process = agent

Identities

- Thus, we need some way to attach an identity to a process. We can stash this somewhere (struct proc?) when the process is created.
- Ultimately, this means we have to pass more information to fork()
- But how do we *know* this person is this person?

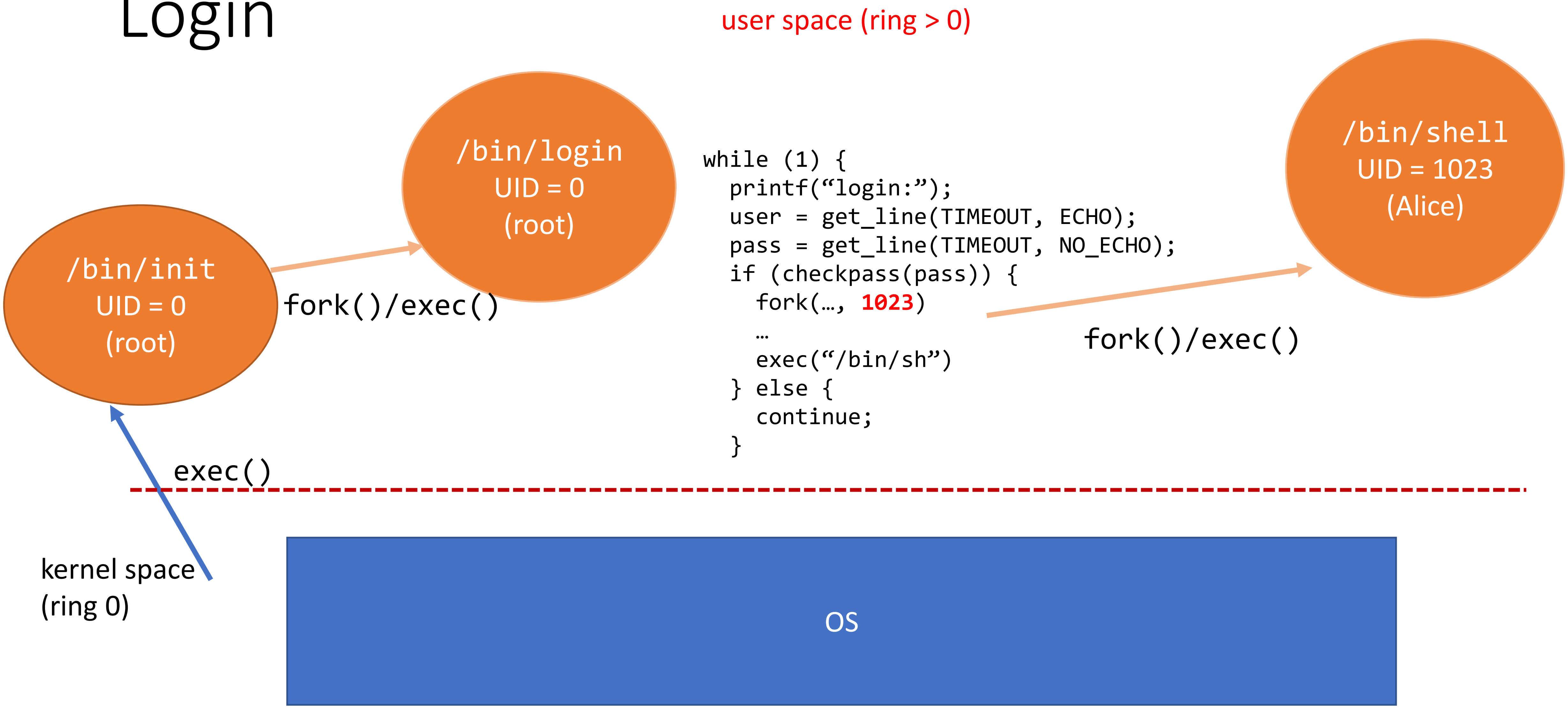
Authentication by…

- Something you know — **Password, PIN, shared secret, the Macarena**
- Something you have — **Keycard, USB key, credit card, key, barcode, signed letter**
- Something you *are* — **Fingerprint, Iris, facial structure, voice, thermal signature, skeletal structure,**

Passwords

- System asks for keyword
- User types it in
- Do they match? Access granted.
- Do we need to store passwords? What do we *really* care about?

Login




```
while (1) {
  printf("login:");
  user = get_line(TIMEOUT, ECHO);
  pass = get_line(TIMEOUT, NO_ECHO);
  if (checkpass(pass)) {
    fork(…, 1023)
    …
    exec("/bin/sh")
  } else {
    continue;
  }
```

Access Control

- Now that we've authenticated someone on the system, how can we determine whether or not they have access?

	ObjA	ObjB	Objc …
Principal 1	Rw-	rwx	Rw-
Principal 2	---	---	---
Principal 3	R-x	R--	R--

Does not scale!

Access Control

- Now that we've authenticated someone on the system, how can we determine whether or not they have access?

	ObjA	ObjB	Objc …
Principal 1	Rw-	rwx	Rw-
Principal 2	---	---	---
Principal 3	R-x	R--	R--

Are you on the list?

Gate
Access List

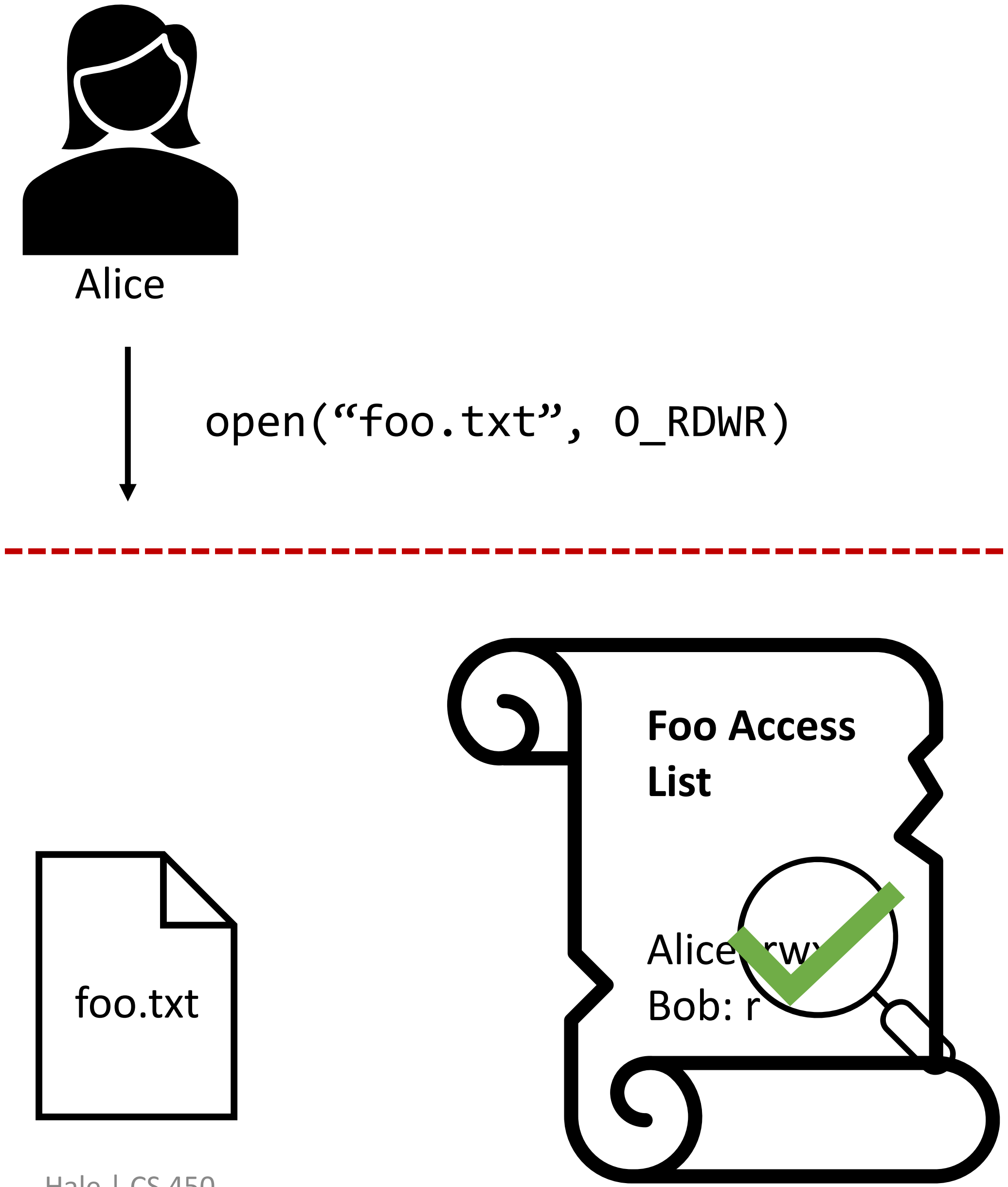
Alice
Bob
rlie

Karen

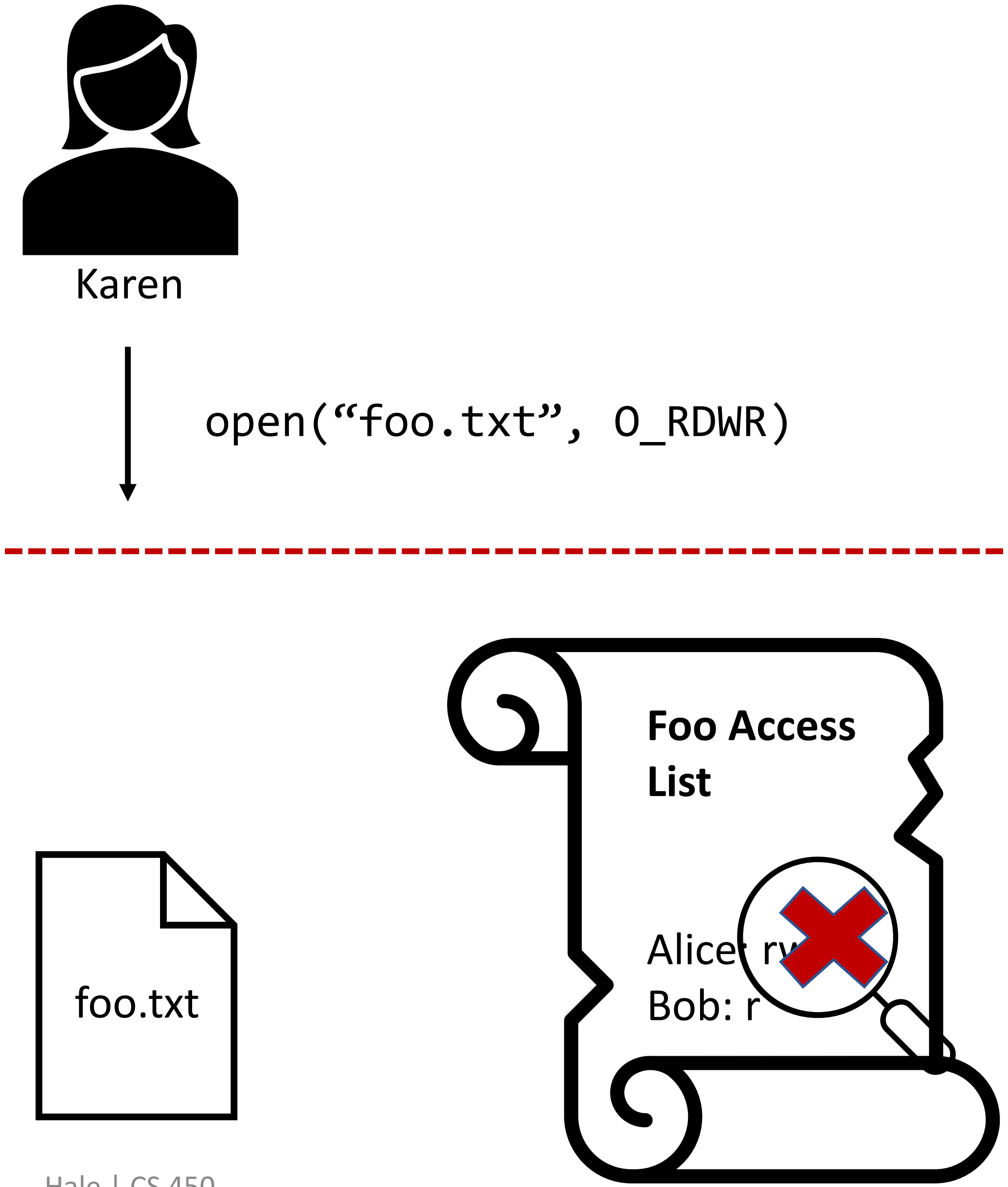
Access Control Lists

- For each resource for which we need access control manage a list
- List contains allowed principals
- If requesting agent is not on the list…no beans

In the System:



In the System:



But…

- Where do we store ACLs? (Think back…)
- How much space do we have for them? How are they structured?
- What if we don't have enough space? How do we avoid overhead?

```
❯ ls -ltrh README.md
-rw-r--r-- 1 kyle kyle 176 Nov 24 16:38 README.md
```

what's this?

I can read it

rwx

I can write it

rwx

I can execute it

`rwx`

`rwx`

this user

`rwx` `rwx`

this user — users in this group

rwx	rwx	rwx
this user	users in this group	anyone else

which user?

```
rwx   rwx   rwx
```

this user | users in this group | anyone else

Store UID,
GID of user who created
file in inode. Compare
against requester (who called open())

which group?

which user?

`rwx` `rwx` `rwx`

this user | users in this group | anyone else

This is a bit string…

111	**111**	**111**
`rwx`	`rwx`	`rwx`
this user	users in this group	anyone else

This is a bit string…

101	**101**	**101**
`r-x`	`r-x`	`r-x`
this user	users in this group	anyone else

This is a bit string…

111	**000**	**000**
`r-x`	`---`	`---`
this user	users in this group	anyone else

Is there a base 2^3 number system?

Yes! Octal (base 8)

r-x	---	---
this user	users in this group	anyone else

Can we use it to specify access control?

```
chmod 101 000 000 file.txt
chmod 5    0   0  file.txt
chmod 500 file.txt
```

r-x	---	---
this user	users in this group	anyone else

Can we use it to specify access control?

```
chmod 110 100 100 file.txt
chmod 6   4   4   file.txt
chmod 644 file.txt
```

```
rw-   r--   r--
```

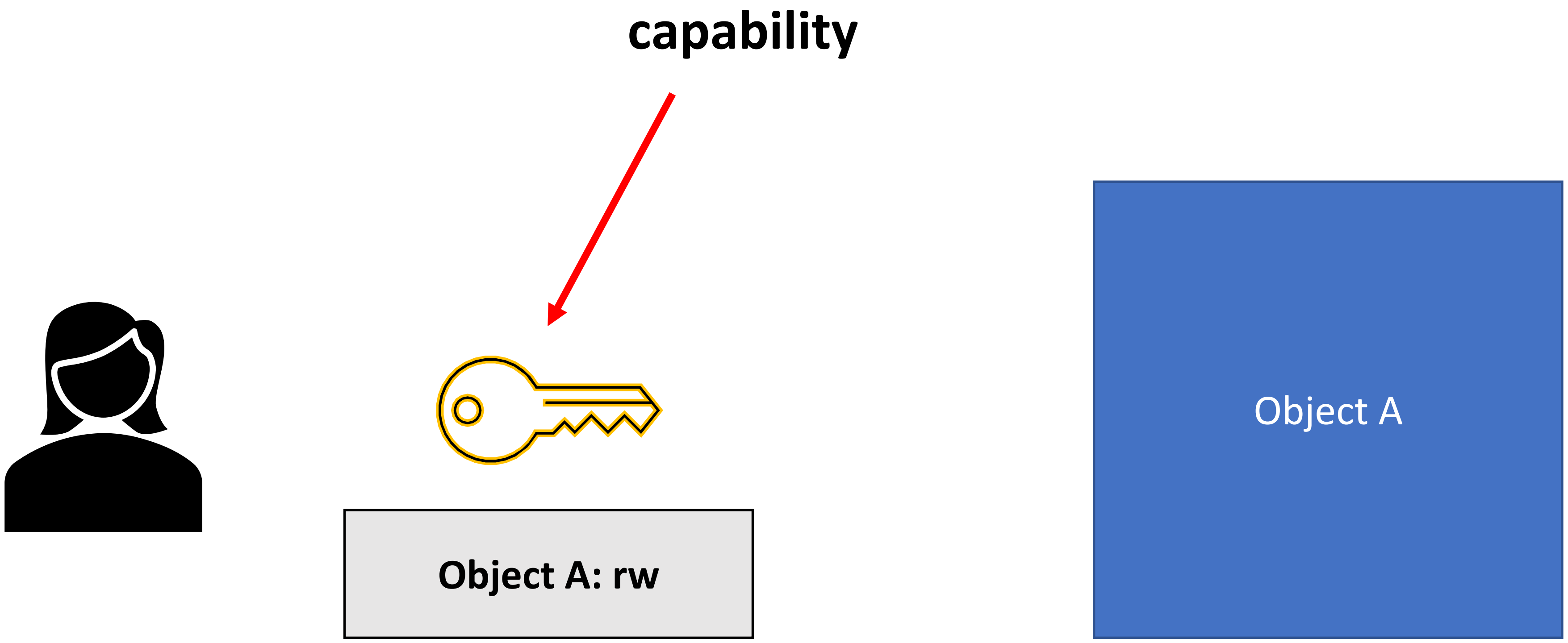
this user | users in this group | anyone else

ACLs

- **Used in most commercial OSes** (in some form or other)

- But…
  - How do we enumerate all resources a user has access to?
  - How do we make ACLs make sense across systems? (namespacing)

Capabilities



Questions

- How are capabilities created?
- Where are they stored? (consider files…how many?)
- Can they be copied? Why/why not?

Capability Implementation

- Capabilities must not be ***forgeable***
- Store them somewhere in the PCB (only kernel can access)
- Research Examples: Hydra, Cheri, Mungi
- Not common in real systems (e.g. KeyKOS, IBM System/38)

Hybrids

- Consider how open() works on UNIX systems
- Is this only ACL?

How to think about security?

- Adversarially…
- Assume the worst!
- If I were trying to break this, what would I do?
- You must *really understand the code you write!*

How much do you trust?

TURING AWARD LECTURE

Reflections on Trusting Trust

To what extent should one trust a statement that a program is free of Trojan horses? Perhaps it is more important to trust the people who wrote the software.

KEN THOMPSON

INTRODUCTION

I thank the ACM for this award. I can't help but feel that I am receiving this honor for timing and serendipity as much as technical merit. UNIX[1] swept into popularity with an industry-wide change from central mainframes to autonomous minis. I suspect that Daniel Bobrow [1] would be here instead of me if he could not afford a PDP-10 and had had to "settle" for a PDP-11. Moreover, the current state of UNIX is the result of the labors of a large number of people.

There is an old adage, "Dance with the one that

programs. I would like to present to you the cutest program I ever wrote. I will do this in three stages and try to bring it together at the end.

STAGE I

In college, before video games, we would amuse ourselves by posing programming exercises. One of the favorites was to write the shortest self-reproducing program. Since this is an exercise divorced from reality, the usual vehicle was FORTRAN. Actually, FORTRAN

Want to Learn More?

- CS 458: Intro to Infosec
- CSP 544: System and Network Security
- CS 528: Data Privacy and Security
- CS 549: Cryptography