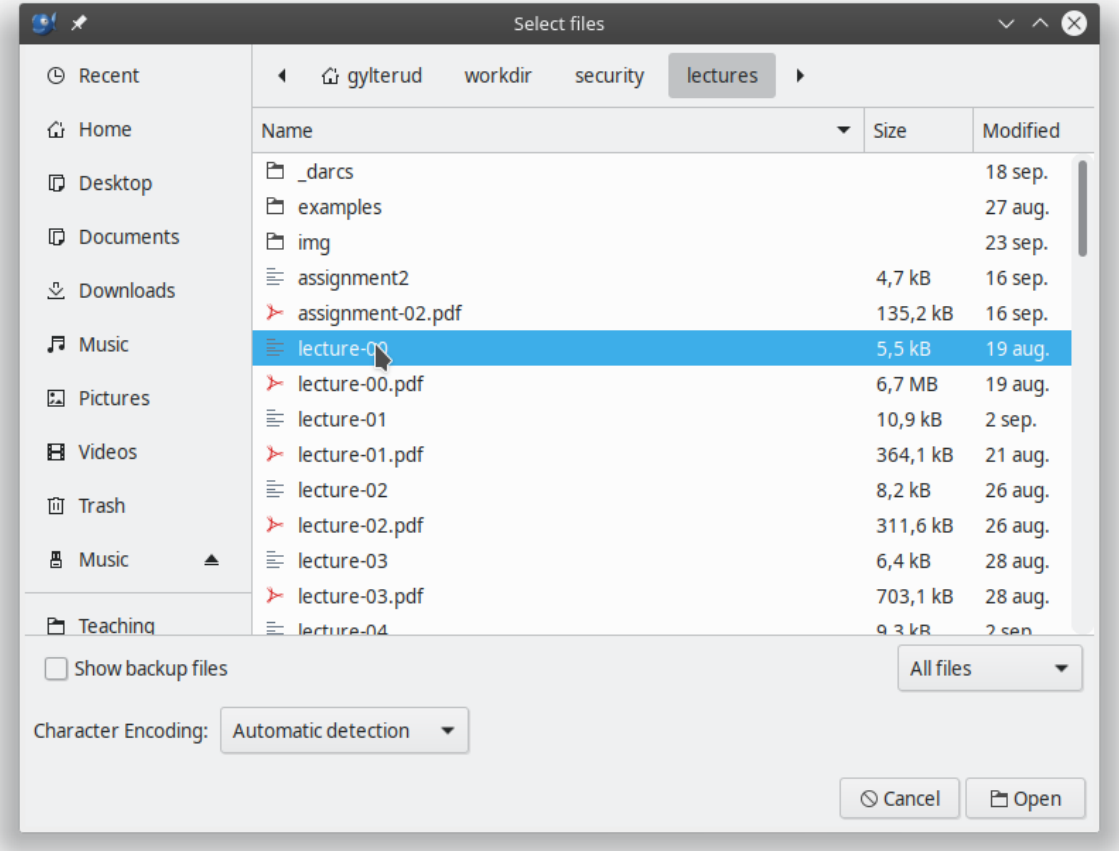
INF226 – Software Security

Håkon Robbestad Gylterud

2019–10–07

Capability based security

File opening



Access control

Basic security concern on any multi-user system:

- How to controll access to objects?

Access control

Basic security concern on any multi-user system:

- How to controll access to objects?

More specifically:

- Preventing access
- Limiting access
- Granting access
- Revoking access

Principle of least priviledge

Principle

A process/object/user/service/$\cdots$ should only have as much priviledge as needed to perform their intended task.

Access control lists

An *access control list* associates to each object a list who can access the object.

- **Example:** File permissions.

Access control lists (ACLs) is the conventional solution for access control.

The Confused deputy

The confused deputy

Typical failure of ACLs:

- Priviledged process (deputy) is **tricked** into performing a bad action when acting on behalf of less priviledged process.

This pattern was identified by Noam Hardy, when thinking about a security hole in a system's Fortran compiler

Hardy's confused deputy

A fortran compiler located in `/sysx/fort`:

- Accessible for all users
- Has special priviledges to `/sysx/` to:
  - Write usage statistics to `/sysx/stats`
- Takes input/output files from command line args

Hardy's confused deputy

A fortran compiler located in `/sysx/fort`:

- Accessible for all users
- Has special priviledges to `/sysx/` to:
  - Write usage statistics to `/sysx/stats`
- Takes input/output files from command line args

Question: What happens if user gives output file `/sysx/stats` or worse `/sysx/bill`?

Hardy's confused deputy

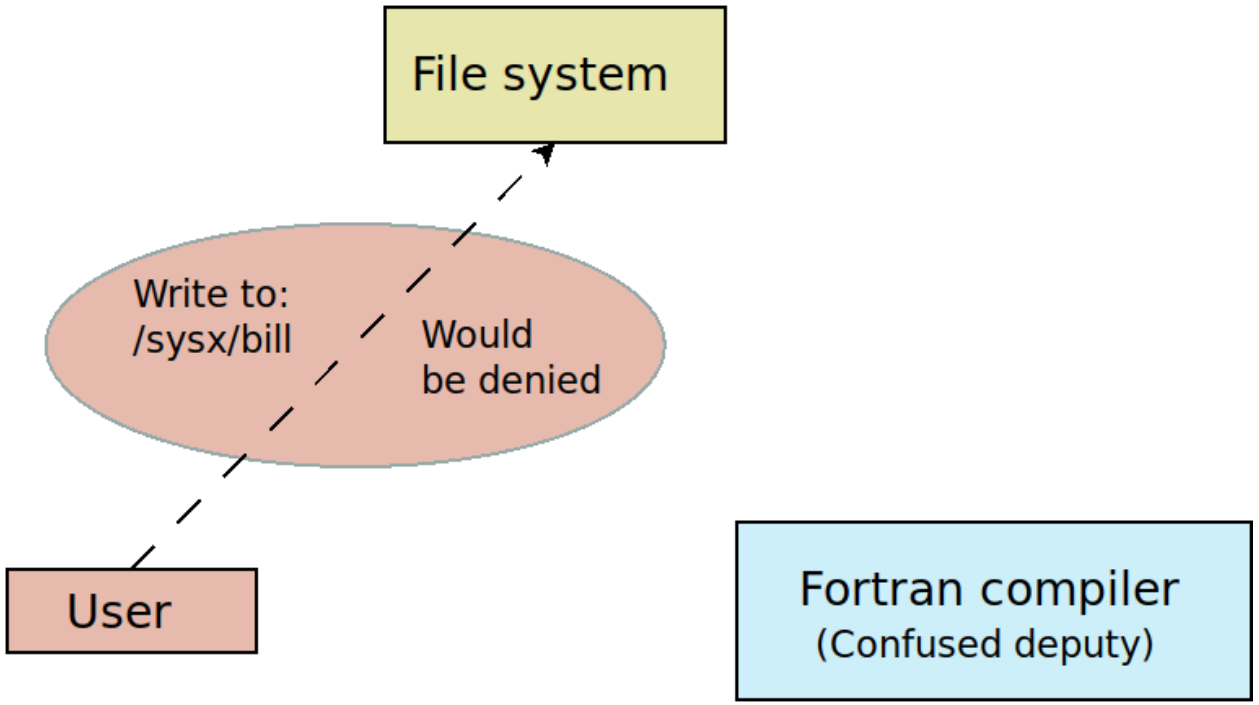


Figure 2: Direct access would be denied

Hardy's confused deputy

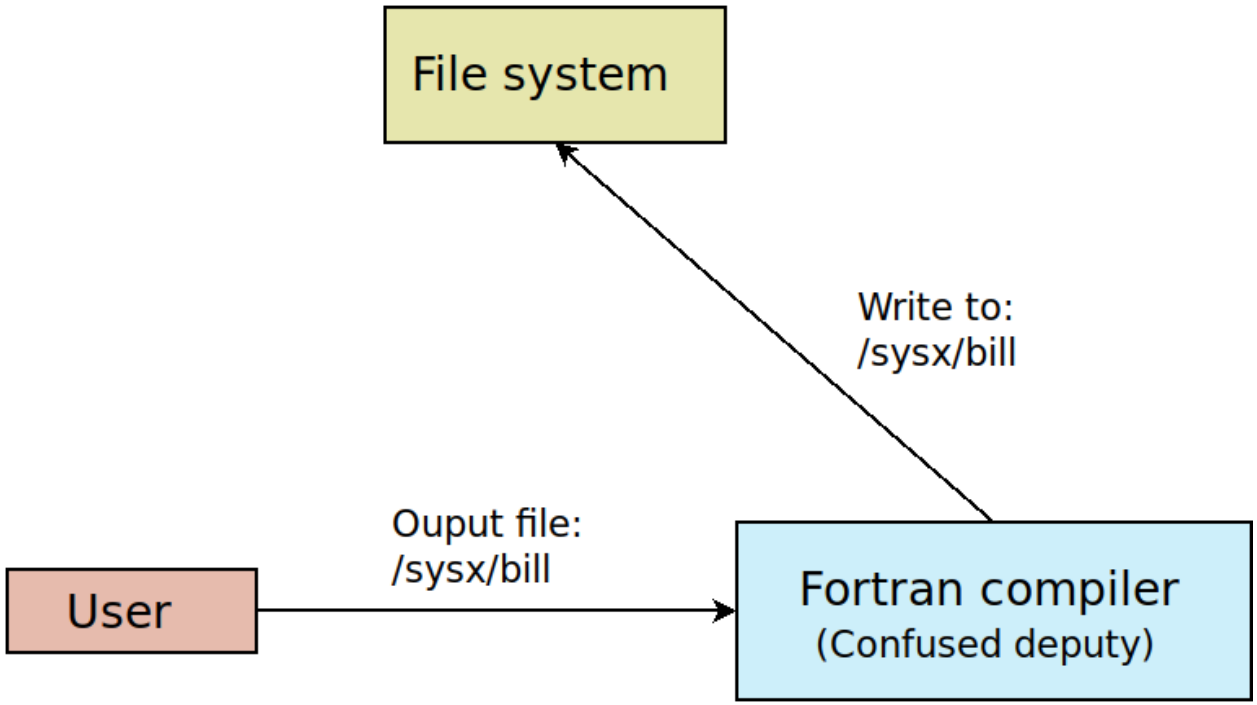


Figure 3: Fortran compiler acts as a confused deputy

Hardy's confused deputy

Why did this failure occur?

- `fort` has legitimate access to `/sysx/stats`
- Users have to specify input/output files

`fort` inadvertedly becomes a *confused deputy*.

How can we solve this?

Ambient authority

Hardy identifies cause as **ambient authority**:

- An ACL system **tries as hard as possible** to allow operation using session cookie, UID/GID etc.
- Processes **do not have prove** their access permit, which is stored in the (ambient) environment.

Cross-site request forgery: The confused deputy browser

Remember, cross site request forgery is possible when:

- User is logged into site A (with a session cookie).
- Site A will perform actions when the user makes request (send message, transfer money, $\cdots$)
- User visits site B, which makes the browser send requests to site A. (`form.send()` or `<iframe>` or `<img>` or $\cdots$)

In this case the confused deputy is the browser.

Capabilities

Capabilities

A *capability* consists of:

- A **reference** to an object
- A set of **permissions** for that object

Capabilities

A *capability* consists of:

- A **reference** to an object
- A set of **permissions** for that object

A capability is used **whenever a resource is accessed**.

Capabilities

A *capability* consists of:

- A **reference** to an object
- A set of **permissions** for that object

A capability is used **whenever a resource is accessed**.

Simple to check if an operation is allowed:

Example: `read(capability)`:

- Reads from the object pointed to by `capability`
- if `capability` allows reading.

Using capabilities

Restricting access to programs:

- Give only the capabilities needed.
- What capabilities should given to:
  - a word processor?
  - a web site?
  - a system login manager?

This allowes very fine grained applications of the *principle of least priviledge*.

Capability properties: Unforgeable

If a capability can be forged, it is useless as a security measure.

Capability properties: Unforgeable

If a capability can be forged, it is useless as a security measure.

Two approaches to unforgeability:

- Enforced by supervisor (operating system, virtual machine, compiler, $\cdots$)
- Unguessable capabilities (random tokens, cryptographic signatures, $\cdots$)

Enforced by supervisor

In an OS, the kernel can keep a **table of capabilties** for each proces.

- A capability is **just an index** in the table.
- Since the process cannot access its table, it cannot forge capabilties,

Example: File descriptors on Unix.

Unguessable capabilities

Unguessable capabilities relies on entropy and cryptographic security to prevent forging:

- A capability can be referenced by a random number.
- A capability can be signed.

Unguessable capabilities

Unguessable capabilities relies on entropy and cryptographic security to prevent forging:

- A capability can be referenced by a random number.
- A capability can be signed.

Unguessable capabilities must be used when tranferring capabilities over networks.

Capability properties: Transferrable

Capabilities could be transferrable:

- If I have a capability, I should be able to transfer it to you.

Capability properties: Transferrable

Capabilities could be transferrable:

- If I have a capability, I should be able to transfer it to you.

Capabilities do not care who uses them. Access control is decoupled from *identity*.

This is what prevents possibly confused deputities.

Example: Hardy's confused deputy

Before: Fortran compiler takes file names from user.

Now: User transfer their capability for the output file to the fortran compiler.

Example: Hardy's confused deputy

Before: Fortran compiler takes file names from user.

Now: User transfer their capability for the output file to the fortran compiler.

- When writing ouputs the *user given* capability is used.
- When writing to `/sysx/`, the compiler has a separate capabilities.

Example: Hardy's confused deputy

Before: Fortran compiler takes file names from user.

Now: User transfer their capability for the output file to the fortran compiler.

- When writing ouputs the *user given* capability is used.
- When writing to `/sysx/`, the compiler has a separate capabilities.

Question: Why does this prevent the compiler from overwriting `/sysx/bil` based on user input?

Capability properties: Transferrable

Question: How would we implement transfer for unguessable capabilities?

Capability properties: Transferrable

Question: How would we implement transfer for unguessable capabilities?

Question: How about for supervisor enforced capabilities?

Capability properties: Transferrable

Question: How would we implement transfer for unguessable capabilities?

Question: How about for supervisor enforced capabilities?

Question: Do capabilities implement mandatory access control (MAC) – or discretionary access control (DAC)?

Abstraction

Capabilities are described by *what you can do with the object* (permissions, or interface).

Not: What *is* the object?.

In principle, the following are treated the same:

- The capability of reading from a file.
- The capability of reading from a network connection.

This means capabilities can be a means of abstraction.

In principle, the following are treated the same:

- The capability of reading from a file.
- The capability of reading from a network connection.

This means capabilities can be a means of abstraction.

Example: File descriptors in UNIX.

Enforced by language: Memory safe capabilities

In a memory safe **object capability system** can be obtained by

- **endowment**: Alice might have intrinsic capabilities given to her at her creation
- **creation**: Alice gets capability to access an object she creates.
- **introduction**: Alice transfers a capability to Bob

This approach **relies on the memory safety** of the language.

Example: Banking

Bank account capabilities:

- Deposit `D`
- Withdraw `W`
- Read balance `R`

Example: Banking

Bank account capabilities:

- Deposit `D`
- Withdraw `W`
- Read balance `R`

Attenuation:

- Alice wants Bob to transfer her some money.
- Alice has a `(D,W,R)` capability to her own account.
- Alice creates a new `(D)` capability to her account and transfers it to Bob.

Example: Banking (alternative)

(Example from E programming language)

Instead of bank accounts, we could have a capability `purse` which **references an amount of money**.

- Anyone can create an empty purse.
- `transfer(src,dst,amount)` transfers between purses.

Example: Banking (alternative)

When Bob wants to transfer $10 to alice:

- Bob creates an empty purse
- Bob transfers $10 from his main purse to new purse.
- Bob sends the purse with $10 to Alice.
- Alice transfers to her main purse.

Capability properties: Revokability

The creator of a capability should be able to revoke it.

Revokation can be temporary, partial.

Example: CSRF-tokens as capabilities

CSRF-tokens can be viewed as capabilities:

- Denotes an object (form target) and permission (POST,GET,$\cdots$)
- Unforgeable (unguessable)

Tokens are principle transferrable.

Capabilities for collaboration

Capabilities can be useful for collaboration:

- Run a program with capabilities to acess shared resources.

Universal persistence

Some capability based system feature *universal persistence*:

- **Program state remembered**, along with capabilities. So that a program is never "restarted".

Universal persistence

Some capability based system feature *universal persistence*:

- **Program state remembered**, along with capabilities. So that a program is never "restarted".

This solves the problem: How are capabilities retained when a program restarts?

- When a user logs in, the login manager reconnects them to their running programs.

Capabilities summary

A *capability* consists of:

- A **reference** to an object
- A set of **permissions** for that object

A capability is a unforgeable, transferrable token of authority.

History

A lot of papers, systems and languages are based on capabilities.

- Dennis & van Horn 1966, coined the term "Capability"
  - Ideas implemented in MIT's PDP-1
- Several systems in the 70's (GNOSIS, KeyKOS, Cambridge CAP)
- More recently:
  - E language / Joe-E (Java subset)
  - Capsicum (FreeBSD)
  - Genode
  - Google Fuchsia

Other things called “capabilities”

There are several things called “cabilities” which are *unrelated* to capabality based security:

- POSIX capabilities
- Docker capabilties

Muddiest point

Answer on `mitt.uib.no`.

Next time: Capsicum and Chromium

Capsicum is in implementation of cababilities in FreeBSD implemented as an extension of file descriptors.

Have a look at the Capsicum paper linked from the syllabus page on MittUiB.