

Systems Security

COMSM1500

Race condition vulnerabilities



Plan

- Race condition
- Example of race condition vulnerabilities
 - access system call
 - How (not to) implement an OS reference monitor
 - dirty cow

What is a race condition?



What is a race condition?



What is a race condition?



What is a race condition?



Resource
10



What is a race condition?

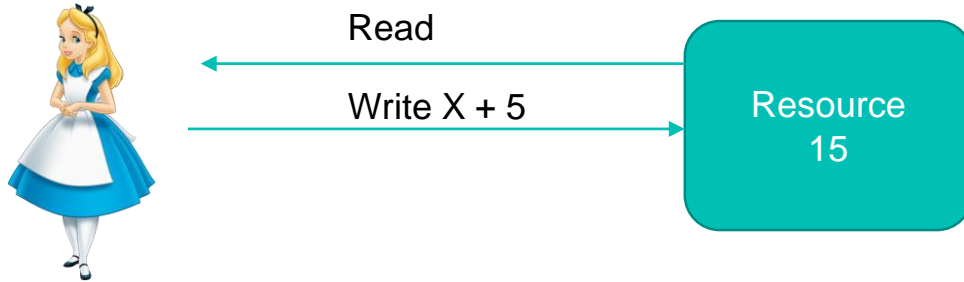


Read

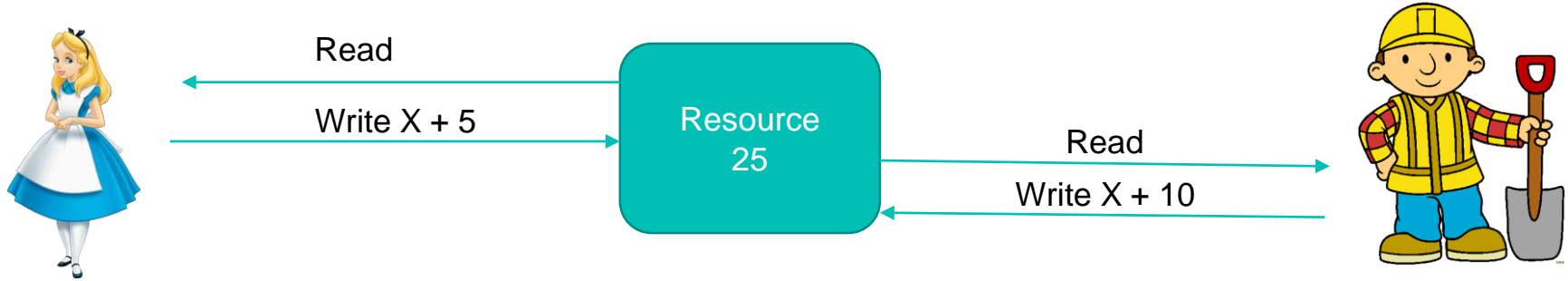
Resource
10



What is a race condition?



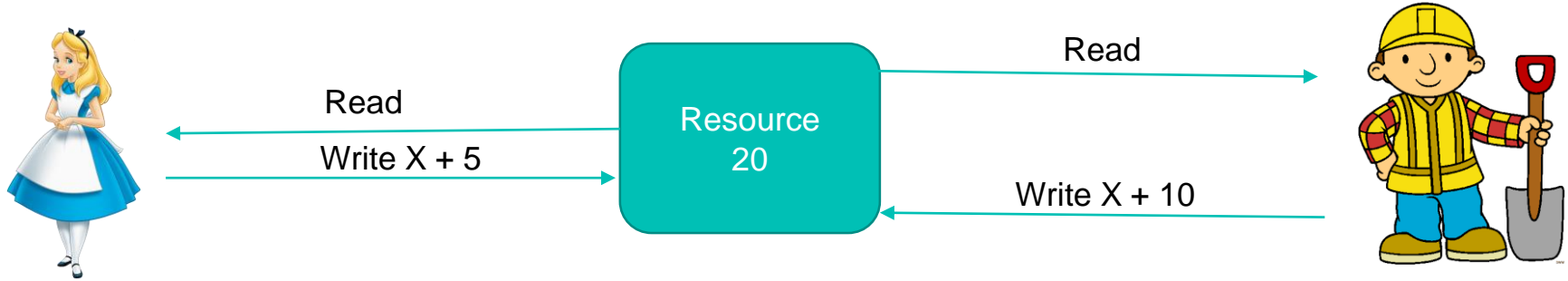
What is a race condition?



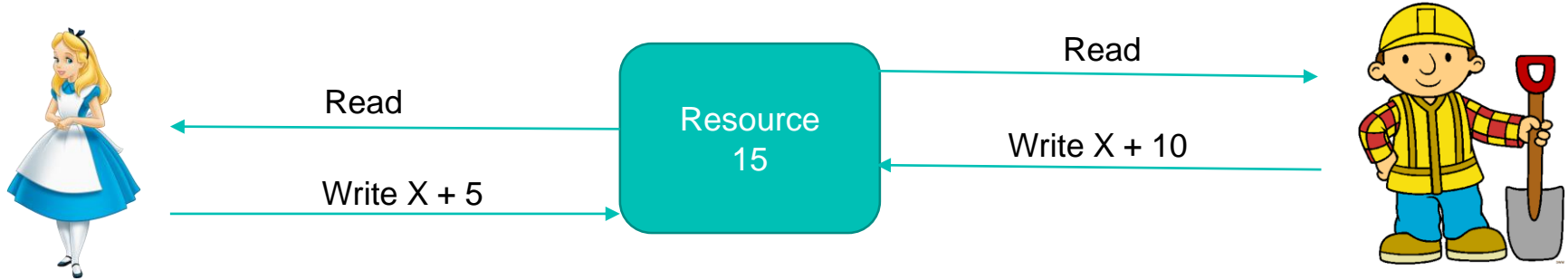
What is a race condition?

- All good here!

What is a race condition?



What is a race condition?



What is a race condition?

```
fcn witdraw(amount)
    balance = getBalance()
    if (balance > amount)
        balance = balance – amount
        saveBalance(balance)
    else
        print “not enough fund”
```

What is a race condition?

- Race condition:
 - System where the output is dependent on the sequence or timing of events
- Many possible permutations
- ... you have seen this before
- It is a well understood problem, with solutions
 - Synchronization
 - Transactions
 - etc...

Race condition vulnerabilities



Example 1: access system call

```
if(access("tmp/X", W_OK)) {  
    f = open("tmp/X");  
    write_to_file(f);  
} else {  
    printf("You do not own the file");  
}
```

Example 1: access system call

```
if(access("tmp/X", W_OK)) {  
    f = open("tmp/X");  
    write_to_file(f);  
} else {  
    printf("You do not own the  
    file");  
}
```

- setuid: root
- want to make sure the “real” user own the file

Example 1: access system call

```
if(access("tmp/X", W_OK)) {  
    f = open("tmp/X");  
    write_to_file(f);  
} else {  
    printf("You do not own the  
    file");  
}
```

- setuid: root
- want to make sure the “real” user own the file
- access return either or not the operation is permitted to current user

How can this be exploited?



Example 1: access system call

```
if(access("tmp/X", W_OK)) {  
    f = open("tmp/X");  
    write_to_file(f);  
} else {  
    printf("You do not own the  
    file");  
}
```

- Path hard coded
 - Program will only write to /tmp/X

Example 1: access system call

```
if(access("tmp/X", W_OK)) {1  
    f = open("tmp/X");2  
    write_to_file(f);  
} else {  
    printf("You do not own the  
    file");  
}
```

- Path hard coded
 - Program will only write to /tmp/X
- Symbolic Link
 - /tmp/X -> /etc/config
- 1: fail
- 2: success

Example 1: access system call

```
if(access("tmp/X", W_OK)) {1  
    f = open("tmp/X");2  
    write_to_file(f);  
} else {  
    printf("You do not own the  
    file");  
}
```

- Exploited race condition
- Changed value between check and use
- time of check to time of use
– TOCTOU

access man

Warning: Using `access()` to check if a user is authorized to, for example, open a file before actually doing so using `open`(2) creates a security hole, because the user might exploit the short time interval between checking and opening the file to manipulate it. **For this reason, the use of this system call should be avoided.** (In the example just described, a safer alternative would be to temporarily switch the process's effective user ID to the real ID and then call `open`(2).)

access man

Homework/exam question:
access system call is vulnerable
to race condition. Explain how it
can be exploited.

Warning: Using `access()` to check if a user is authorized to, for example, open a file before actually doing so using `open(2)` creates a security hole, because the user might exploit the short time interval between checking and opening the file to manipulate it. **For this reason, the use of this system call should be avoided.** (In the example just described, a safer alternative would be to temporarily switch the process's effective user ID to the real ID and then call `open(2)`.)

Example 2: implementing OS reference monitor

What is a reference monitor?



What is a reference monitor?

We will see this again in the lectures on access control!

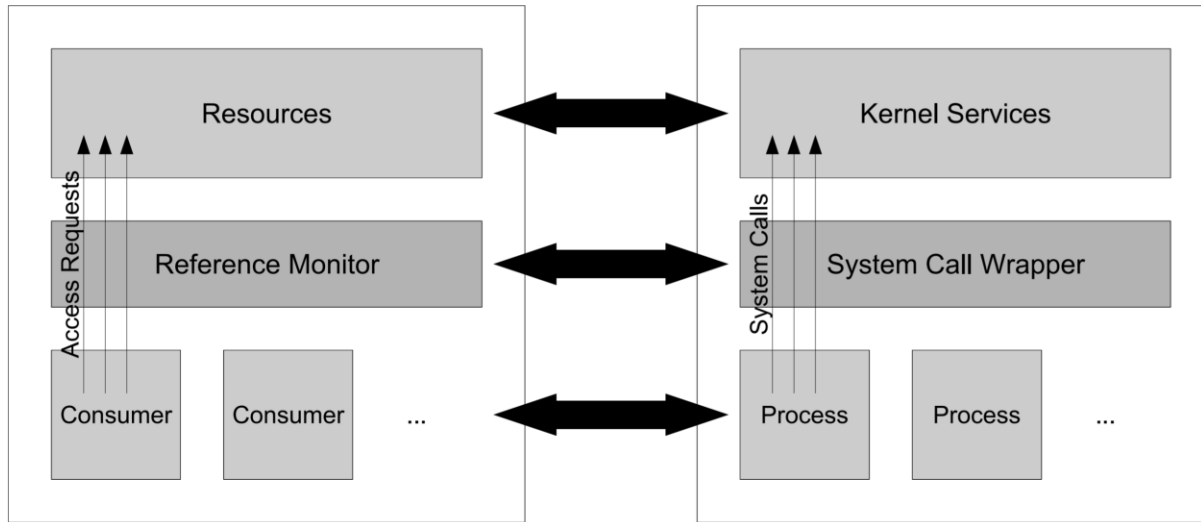


Example 2: implementing OS reference monitor

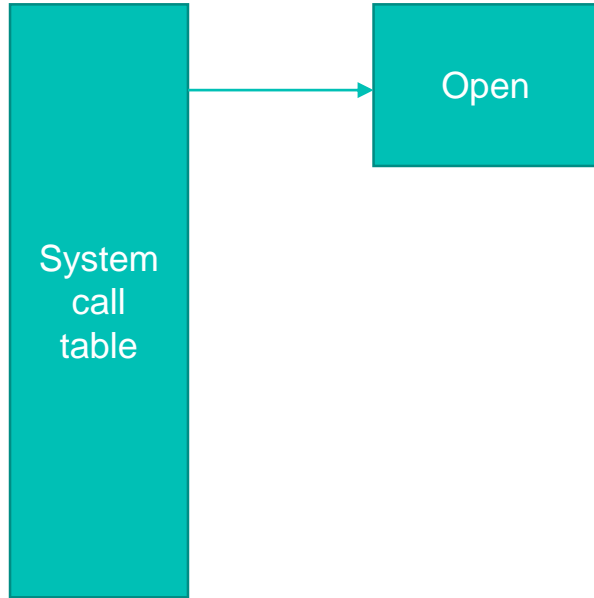
- a **reference monitor** is a secure, always-used and fully-testable module that controls all software access to data objects or devices

Example 2: implementing OS reference monitor

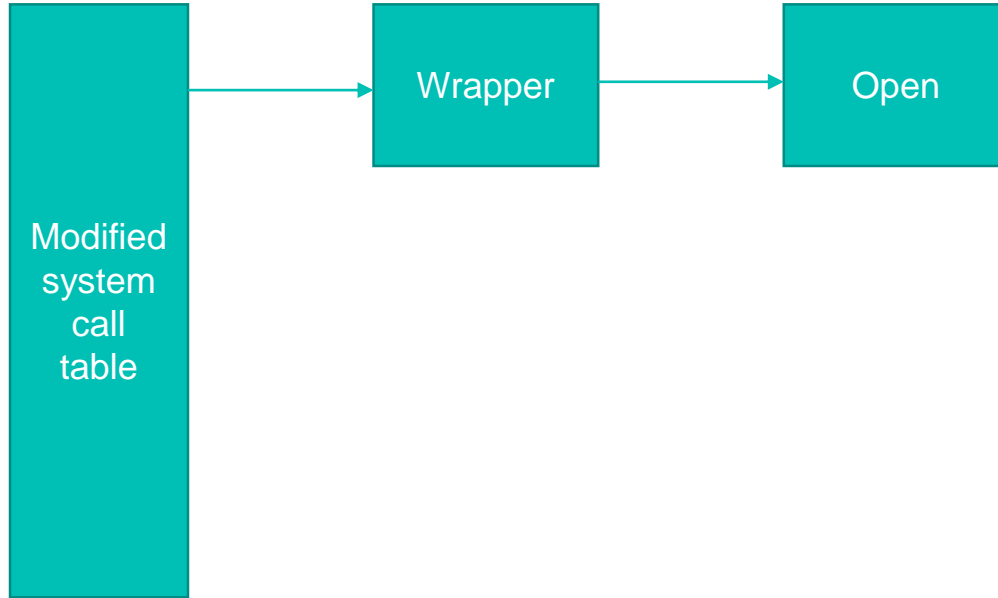
- a **reference monitor** is a secure, always-used and fully-testable module that controls all software access to data objects or devices



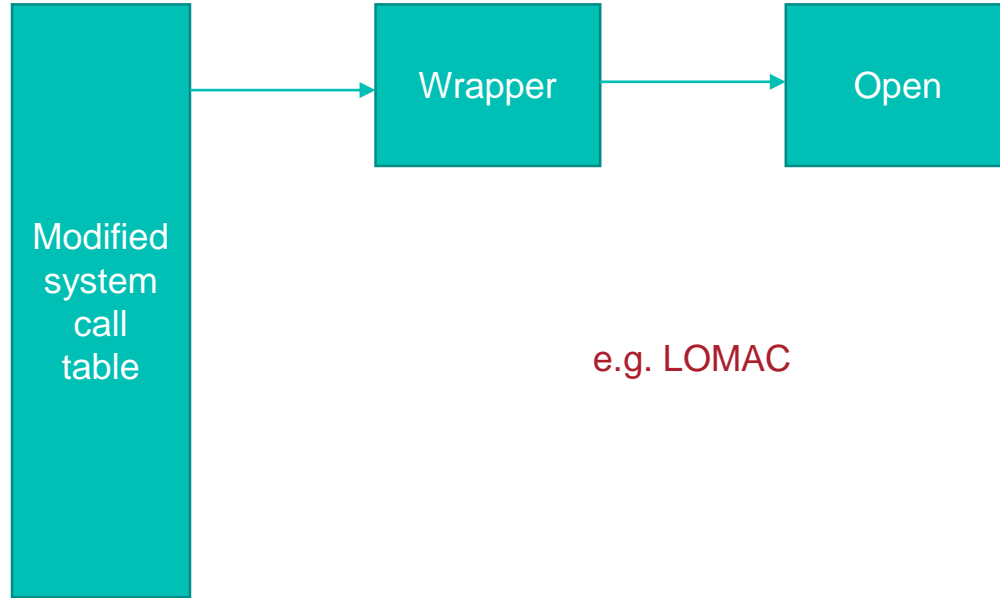
Example 2: implementing OS reference monitor



Example 2: implementing OS reference monitor

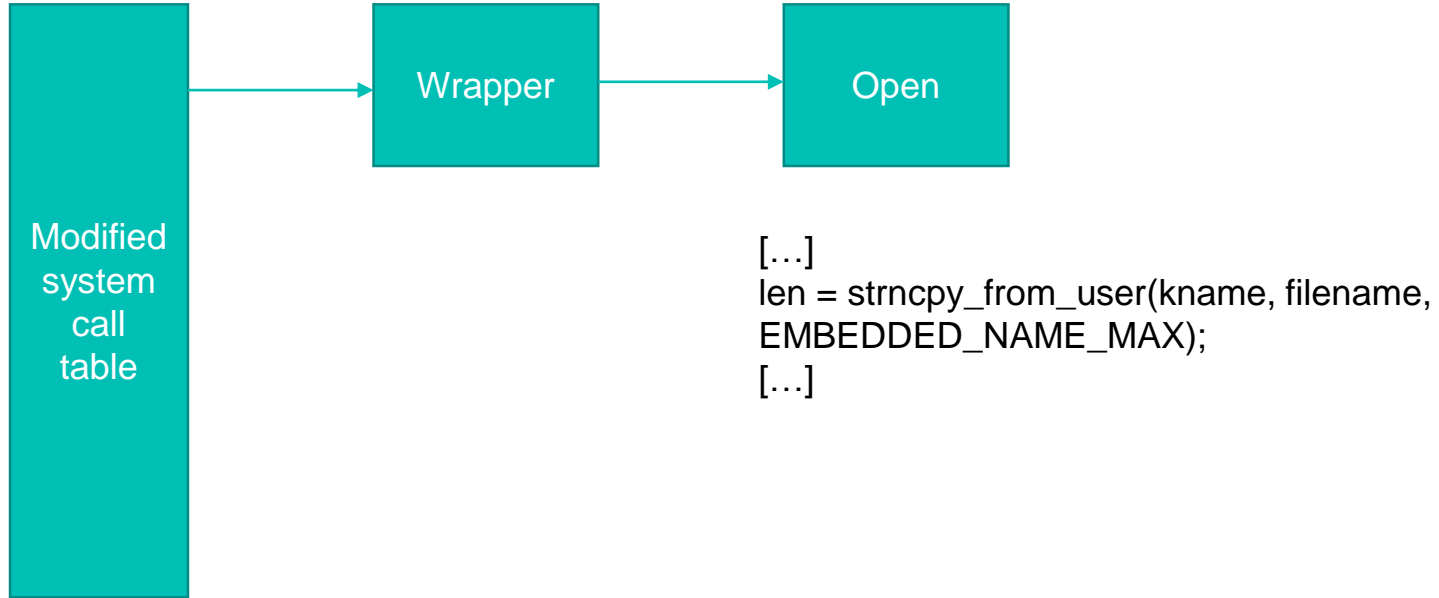


Example 2: implementing OS reference monitor

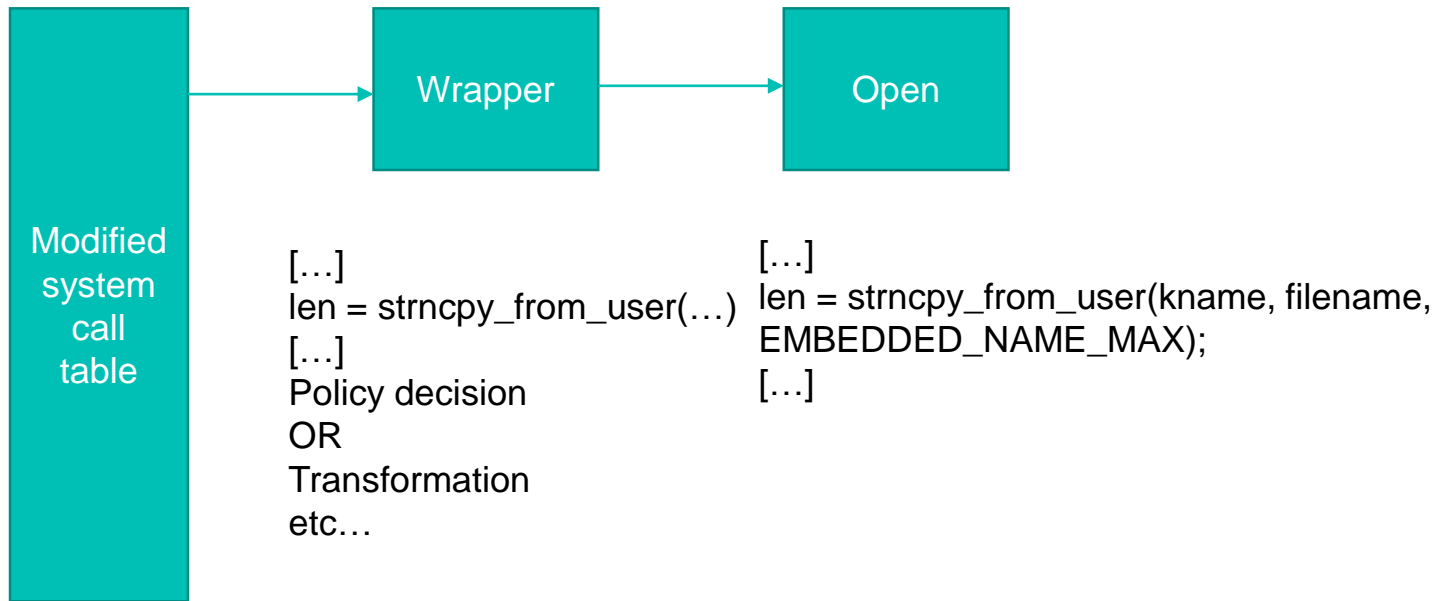


e.g. LOMAC

Example 2: implementing OS reference monitor



Example 2: implementing OS reference monitor



Problem?



Example 2: implementing OS reference monitor

- Wrapper and syscall work on two different copy of the buffer!
- User space controlled buffer
 - Can be controlled by an attacker
- The value checked to enforce policy!=value seen by syscall

Example 2: implementing OS reference monitor

- Wrapper and syscall work on two different copy of the buffer!
- User space controlled buffer
 - Can be controlled by an attacker
- The value checked to enforce policy!=value seen by syscall
- Check Robert Watson's paper for "real" exploit to syscall wrapper

Example 2: implementing OS reference monitor

- Wrapper and syscall work on two different copy of the buffer!
- User space controlled buffer
 - Can be controlled by an attacker
- The value checked to enforce policy!=value seen by syscall
- Check Robert Watson's paper for "real" exploit to syscall wrapper
- Check LSM paper for how to properly implement a reference monitor!

Example 2: implementing OS reference monitor

- Wrapper and syscall work on two different copy of the buffer!
- User space controlled buffer
 - Can be controlled by an attacker
- The value checked to enforce policy!=value seen by syscall

Homework/exam question:
Explain why system call wrapper
should not be used to implement
access control.

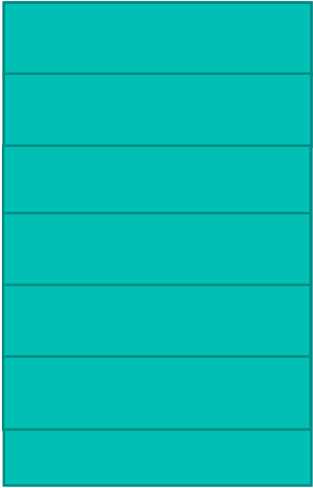
Example 3: dirty cow



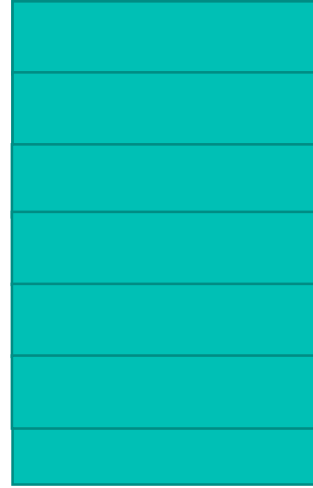
Example 3: dirty cow

- CVE-2016-5195
- Linux vulnerability that could be exploited to gain root access
- Concurrency issue relating to how memory is managed

Example 3: dirty cow

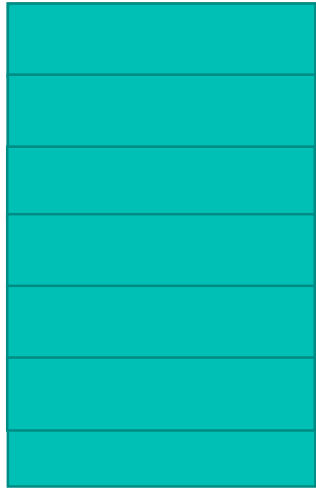


Process memory



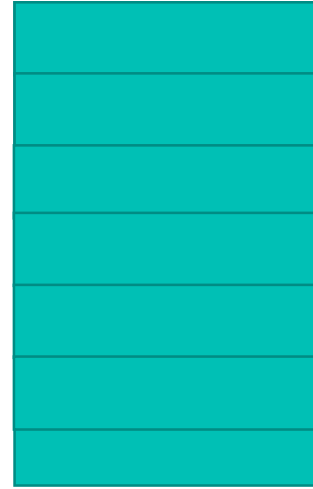
Physical memory

Example 3: dirty cow



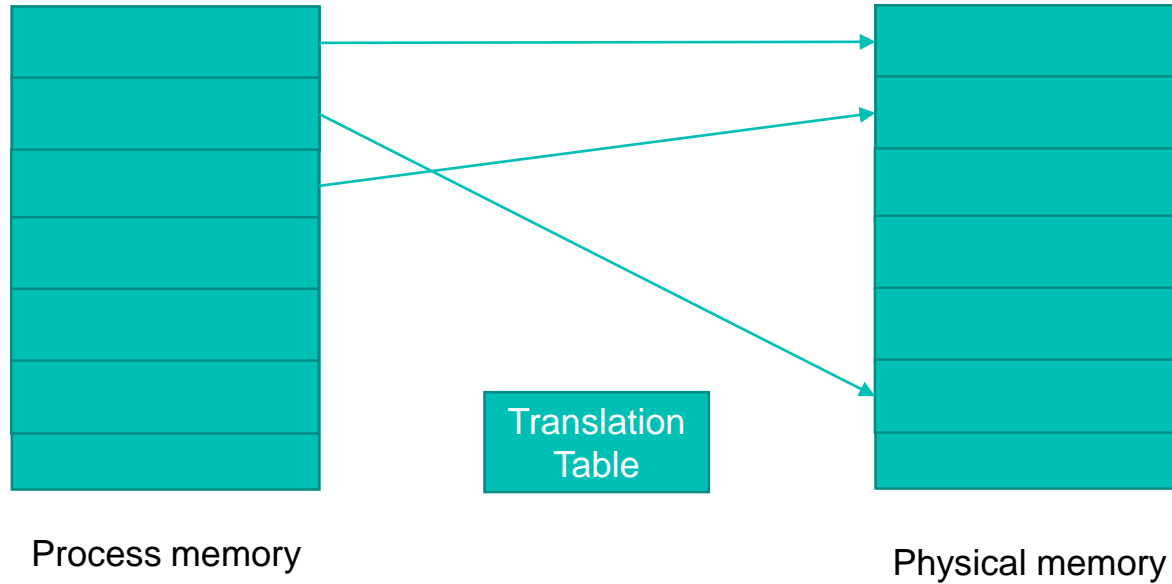
Process memory

Translation
Table

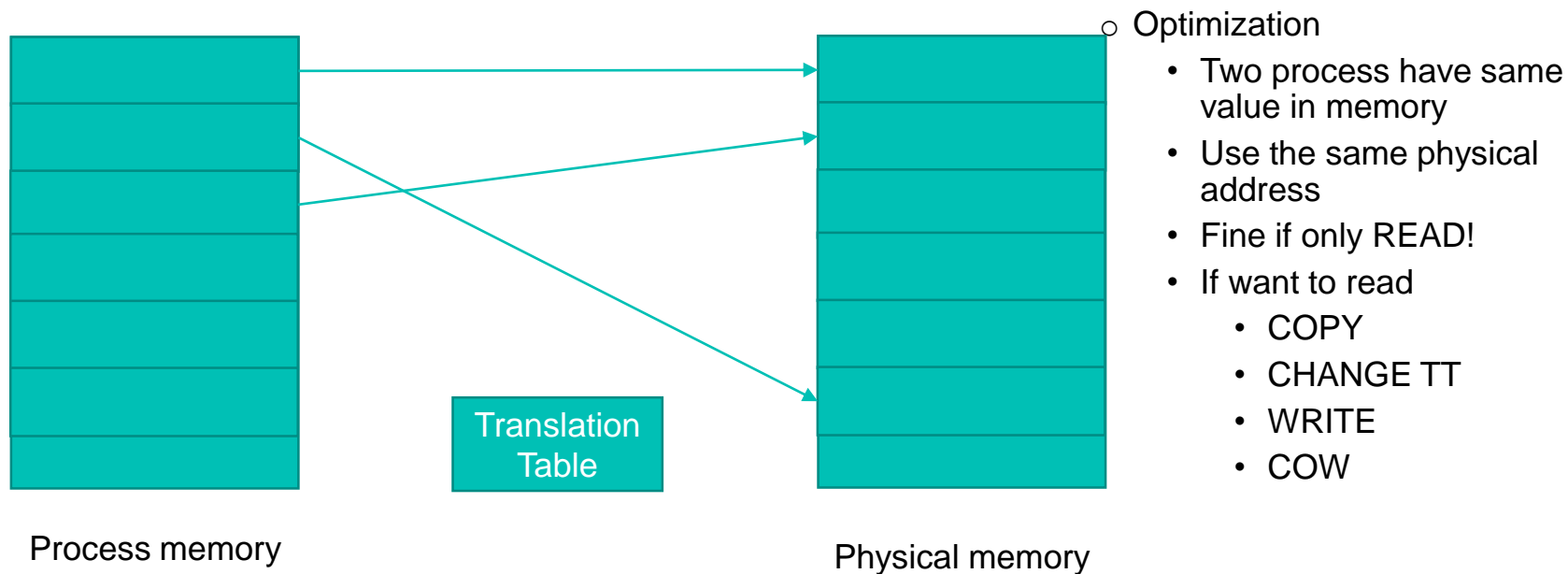


Physical memory

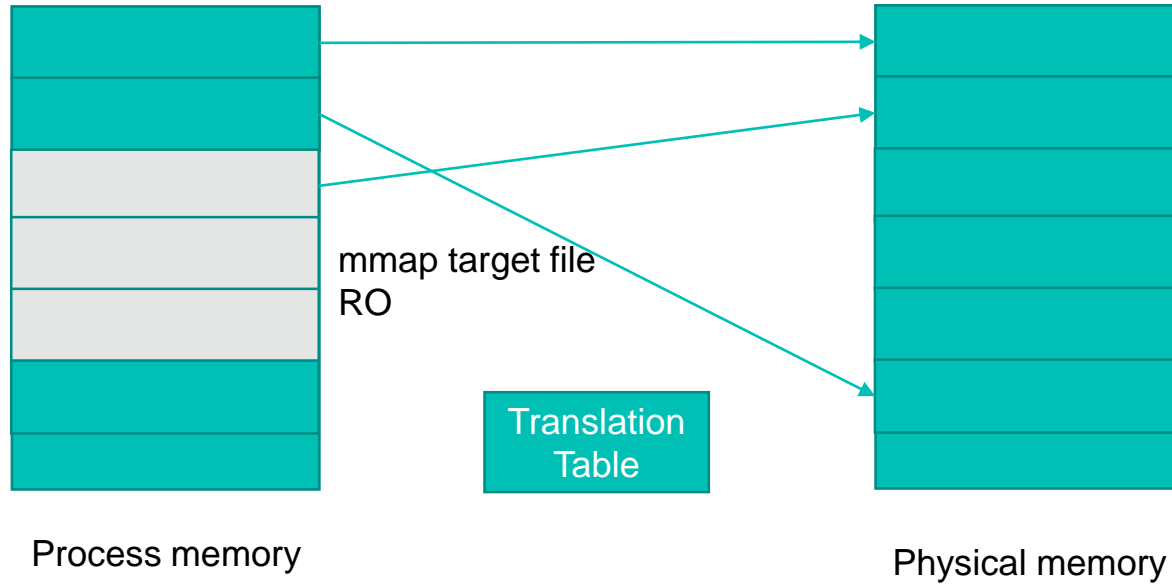
Example 3: dirty cow



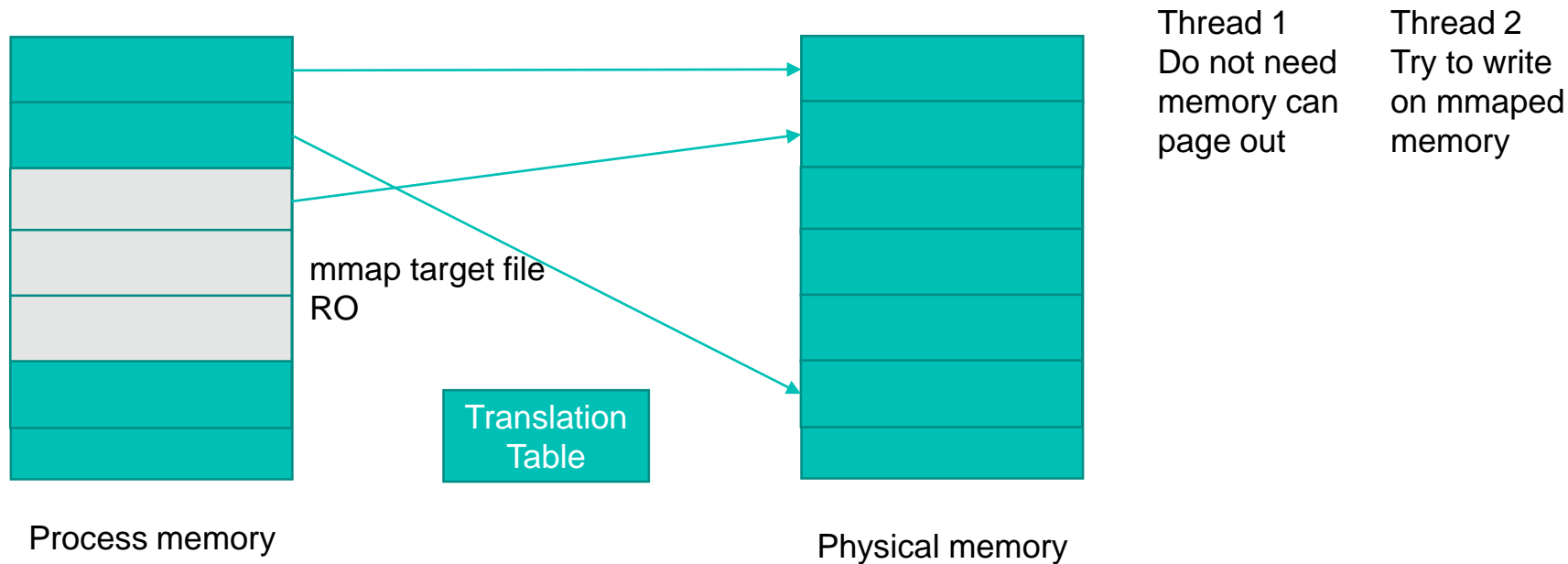
Example 3: dirty cow



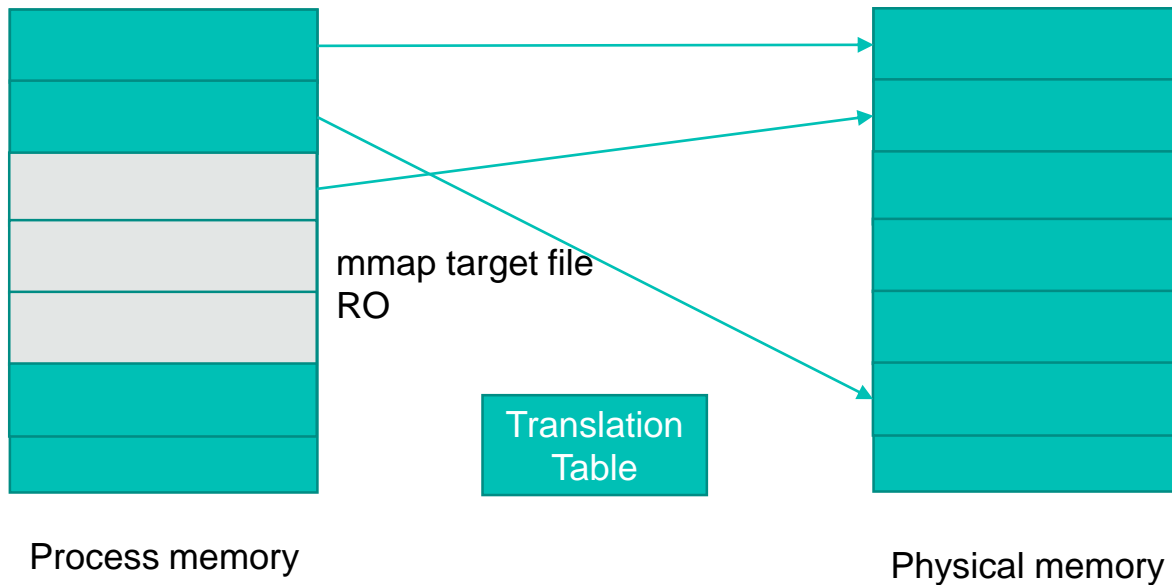
Example 3: dirty cow



Example 3: dirty cow



Example 3: dirty cow

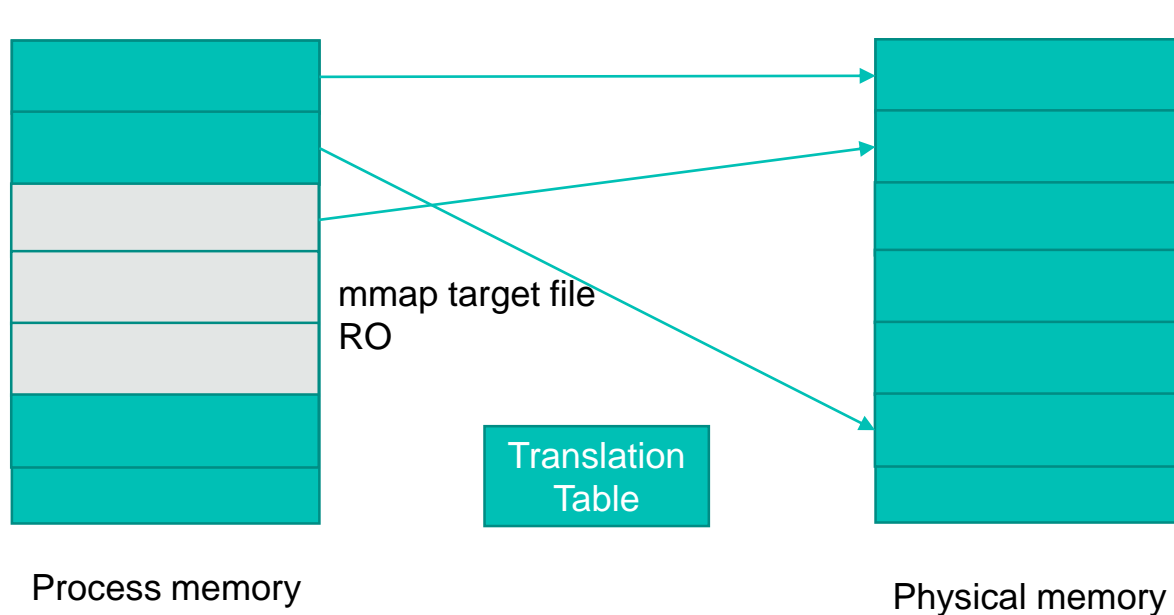


Thread 1
Do not need
memory can
page out

Thread 2
Try to write
on mmaped
memory

Normally fine!

Example 3: dirty cow



Thread 1
Do not need
memory can
page out

Thread 2
Try to write
on mmaped
memory

Sometimes race condition
kernel see change to mmaped file
Copy changes to underlying file
As it is paging out...
... but user did not have acces!
Can do this on...
passwd for example

Example 3: dirty cow

- <https://github.com/dirtycow/dirtycow.github.io/blob/master/dirtycow.c>

Example 3: dirty cow

Homework/exam question:
Explain the dirty cow
vulnerability.

- <https://github.com/dirtycow/dirtycow.github.io/blob/master/dirtycow.c>

Plan

Homework/exam question:
Give an example of race
condition vulnerability.

- Race condition
- Example of race condition vulnerabilities
 - access system call
 - How (not to) implement an OS reference monitor
 - dirty cow

Thank you, questions?

Office MVB 3.26

bristol.ac.uk

