2550 Intro to cybersecurity

L6: Authorization

Test

Ran Cohen/abhi shelat

Thanks Christo for slides!

Authentication:

Verification of an identity claim by a subject on behalf of a principal.

Authorization

After Authenticating a subject, what next?

Decisions are made about which objects the subject can access

Access Control

- Policy specifying how entities can interact with resources
  - i.e., Who can access what?
  - Requires authentication and authorization
- Access control primitives

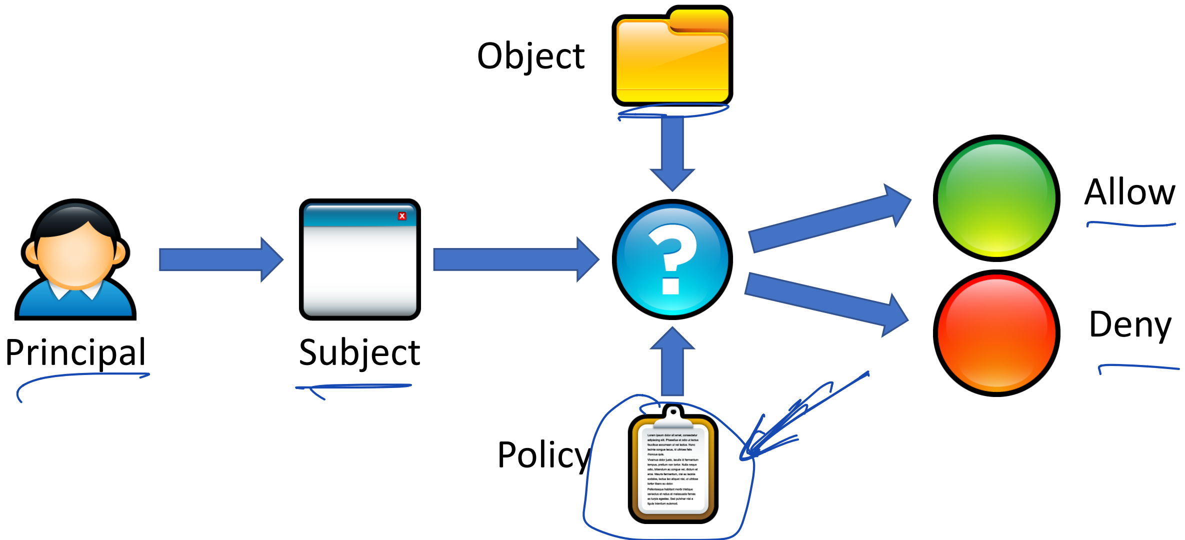
Principal User of a system

Subject Entity that acts on behalf of principals — Software program

Object Resource acted upon by subjects — Files, Sockets, Devices, OS APIs

Access Control Check

- Given an access request from a subject, on behalf of a principal, for an object, return an access control decision based on the policy



Access Control Models

- Discretionary Access Control (DAC)
  - The kind of access control you are familiar with
  - Access rights propagate and may be changed at subject's discretion

Access Control Models

- Discretionary Access Control (DAC)
  - The kind of access control you are familiar with
  - Access rights propagate and may be changed at subject's discretion
- Mandatory Access Control (MAC)
  - Access of subjects to objects is based on a system-wide policy
  - Denies users full control over resources they create

to defend against failures of operation.

Discretionary Access Control

Access Control Matrices

Access Control Lists

Unix Access Control

Discretionary Access Control

- According to Trusted Computer System Evaluation Criteria (TCSEC)

"A means of restricting access to objects based on the identity and need-to-know of users and/or groups to which they belong.

Controls are discretionary in the sense that a subject with a certain access permission is capable of passing that permission (directly or indirectly) to any other subject."

Access Control Matrices

Given subjects $s_i \in S$, objects $o_j \in O$, rights {**R**ead, **W**rite, e**X**ecute},

RWX
read
write
execute.

objects

subjects

	o_1	o_2	o_3
s_1	RW	RX	
s_2	R	RWX	RW
s_3		RWX	

- Introduced by Lampson in 1971
- Static description of protection state
- Abstract model of concrete systems

Access Control Lists (ACL)

(columns of the matrix)

- Each object has an associated list of subject→operation pairs
- Authorization verified for each request by checking list of tuples
- Used pervasively in filesystems and networks
  - "Users a, b, and c and read file x."
  - "Hosts a and b can listen on port x."

columns.

objects

	O1	O2	O3
S1	RW	RX	
S2	R	RWX	RW
S3		RWX	

Access Control List (ACL)

- Each object has an associated list of subject→operation pairs
- Authorization verified for each request by checking list of tuples
- Used pervasively in filesystems and networks
  - "Users a, b, and c and read file x."
  - "Hosts a and b can listen on port x."

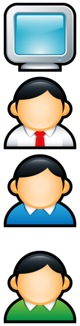
ACL for o_2

	o_1	o_2	o_3
s_1	RW	RX	
s_2	R	RWX	RW
s_3		RWX	

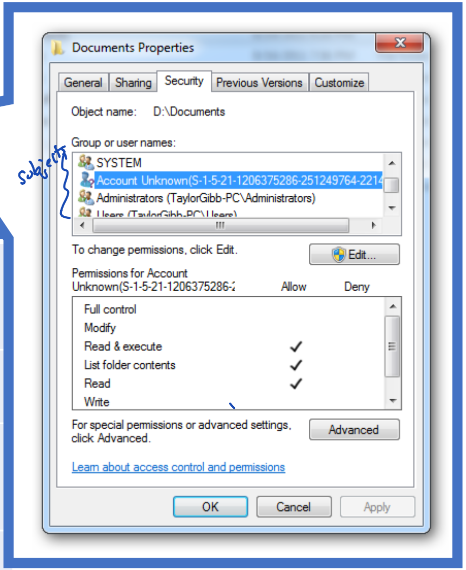
Windows ACLs

	D:\Music	***D:\Images***	***D:\Documents***
System	RWX	RWX	RWX
Administrators	RW	RW	RW
Users:Bob	RWX	RW	
Users:Alice		RW	R

Windows ACLs



	D:\Music	D:\Images	D:\Documents
System	RWX	RWX	RWX
Administrators	RW	RW	RW
Users:Bob	RWX	RW	
Users:Alice		RW	R



ACL Review

The Good

- Very flexible
  - Can express any possible access control matrix
  - Any principal can be configured to have any rights on any object

The Bad

complicated

tedious.

ACL Review

The Good

- Very flexible
  - Can express any possible access control matrix
  - Any principal can be configured to have any rights on any object

The Bad

- Complicated to manage
  - Every object can have wildly different policies
  - Infinite permutations of subjects, objects, and rights

Unix-style Permissions

- Based around the concept of owners and groups
  - All objects have an owner and a group
  - Permissions assigned to owner, group, and everyone else
- Authorization verified for each request by mapping the subject to owner, group, or other and checking the associated permissions

Unix Permissions

```
cbw@DESKTOP:~$ ls -l
drwxrwxrwx 0 cbw  cbw      512 Jan 29 22:46 my_dir
-rw-rw-rw- 1 cbw  cbw       17 Jan 29 22:46 my_file
-rwxrwxrwx 1 cbw  faculty 313 Jan 29 22:47 my_program.py
-rw------- 1 root root     896 Jan 29 22:47 sensitive_data.csv
```

d → Directory

r → Read   w → Write   x → eXecute

Unix Permissions

```
cbw@DESKTOP:~$ ls -l
drwxrwxrwx 0 cbw  cbw     512 Jan 29 22:46 my_dir
-rw-rw-rw- 1 cbw  cbw      17 Jan 29 22:46 my_file
-rwxrwxrwx 1 cbw  faculty 313 Jan 29 22:47 my_program.py
-rw------- 1 root root    896 Jan 29 22:47 sensitive_data.csv
```

Owner

d → Directory

r → Read w → Write x → eXecute

Unix Permissions

```
cbw@DESKTOP:~$ ls -l
drwxrwxrwx 0 cbw  cbw      512 Jan 29 22:46 my_dir
-rw-rw-rw- 1 cbw  cbw       17 Jan 29 22:46 my_file
-rwxrwxrwx 1 cbw  faculty 313 Jan 29 22:47 my_program.py
-rw------- 1 root root     896 Jan 29 22:47 sensitive_data.csv
```

Owner

Owner

d → Directory   **r → Read**   **w → Write**   **x → eXecute**

Unix Permissions

```
cbw@DESKTOP:~$ ls -l
drwxrwxrwx 0 cbw  cbw      512 Jan 29 22:46 my_dir
-rw-rw-rw- 1 cbw  cbw       17 Jan 29 22:46 my_file
-rwxrwxrwx 1 cbw  faculty 313 Jan 29 22:47 my_program.py
-rw------- 1 root root     896 Jan 29 22:47 sensitive_data.csv
```

Owner group others

Owner Group

— : nothing

d → Directory **r → Read** **w → Write** **x → eXecute**

Unix Permissions

```
cbw@DESKTOP:~$ ls -l
drwxrwxrwx 0 cbw  cbw      512 Jan 29 22:46 my_dir
-rw-rw-rw- 1 cbw  cbw       17 Jan 29 22:46 my_file
-rwxrwxrwx 1 cbw  faculty 313 Jan 29 22:47 my_program.py
-rw------- 1 root root     896 Jan 29 22:47 sensitive_data.csv
```

Owner Group

Owner Group

d → Directory

r → Read **w → Write** **x → eXecute**

Unix Permissions

```
cbw@DESKTOP:~$ ls -l
drwxrwxrwx 0 cbw  cbw     512 Jan 29 22:46 my_dir
-rw-rw-rw- 1 cbw  cbw      17 Jan 29 22:46 my_file
-rwxrwxrwx 1 cbw  faculty 313 Jan 29 22:47 my_program.py
-rw------- 1 root root    896 Jan 29 22:47 sensitive_data.csv
```

Owner Group Other

Owner Group

d → Directory

r → Read **w → Write** **x → eXecute**

Unix Permissions

Directory

```
cbw@DESKTOP:~$ ls -l
drwxrwxrwx 0 cbw  cbw      512 Jan 29 22:46 my_dir
-rw-rw-rw- 1 cbw  cbw       17 Jan 29 22:46 my_file
-rwxrwxrwx 1 cbw  faculty 313 Jan 29 22:47 my_program.py
-rw------- 1 root root     896 Jan 29 22:47 sensitive_data.csv
```

Owner Group Other

Owner Group

d → Directory

r → Read **w → Write** **x → eXecute**

Unix Permissions

Directory

Permission to list the contents of a directory

```
cbw@DESKTOP $ ls -l
drwxrwxrwx 0 cbw  cbw      512 Jan 29 22:46 my_dir
-rw-rw-rw- 1 cbw  cbw       17 Jan 29 22:46 my_file
-rwxrwxrwx 1 cbw  faculty  313 Jan 29 22:47 my_program.py
-rw------- 1 root root     896 Jan 29 22:47 sensitive_data.csv
```

Owner Group Other

Owner Group

d → Directory

r → Read **w → Write** **x → eXecute**

Setting Permissions

+ → add permissions
- → remove permissions

chmod [who]<+/-><permissions> <file1> [file2] …

(omitted) → user, group, and other
a → user, group, and other
u → user
g → group
o → other

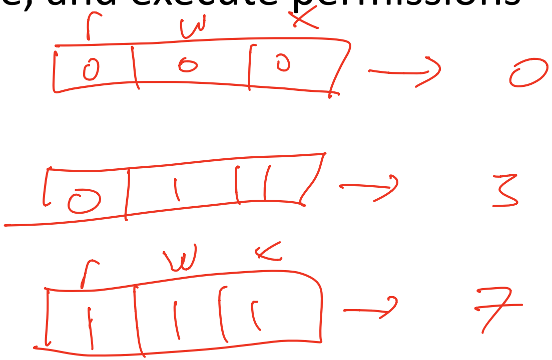
r → Read
w → Write
x → eXecute

```
cbw@DESKTOP:~$ ls -l
drwxrwxrwx 0 cbw   cbw     512 Jan 29 22:46 my_dir
-rw-rw-rw- 1 cbw   cbw      17 Jan 29 22:46 my_file
-rwxrwxrwx 1 cbw   faculty 313 Jan 29 22:47 my_program.py
cbw@DESKTOP:~$ chmod ugo-rwx my_dir
cbw@DESKTOP:~$ chmod go-rwx my_program.py
cbw@DESKTOP:~$ chmod u-rw my_program.py
cbw@DESKTOP:~$ chmod +x my_file
cbw@DESKTOP:~$ ls -l
d--------- 0 cbw   cbw     512 Jan 29 22:46 my_dir
-rwxrwxrwx 1 cbw   cbw      17 Jan 29 22:46 my_file
---x------ 1 cbw   faculty 313 Jan 29 22:47 my_program.py
```

Alternate Form of Setting Permissions

chmod ### <file1> [file2] …

- #s correspond to owner, group, and other
- Each value encodes read, write, and execute permissions
  - 1 → execute
  - 2 → write
  - 4 → read



Alternate Form of Setting Permissions

chmod ### <file1> [file2] …

- #s correspond to owner, group, and other
- Each value encodes read, write, and execute permissions
  - 1 → execute
  - 2 → write
  - 4 → read
- What if you want to set something as read, write, and execute?

Alternate Form of Setting Permissions

chmod ### <file1> [file2] …

- #s correspond to owner, group, and other
- Each value encodes read, write, and execute permissions
  - 1 → execute
  - 2 → write
  - 4 → read
- What if you want to set something as read, write, and execute?
  - 1 + 2 + 4 = 7

```
cbw@DESKTOP:~$ ls -l
drwxrwxrwx 0 cbw   cbw      512 Jan 29 22:46 my_dir
-rw-rw-rw- 1 cbw   cbw       17 Jan 29 22:46 my_file
-rwxrwxrwx 1 cbw   faculty  313 Jan 29 22:47 my_program.py
cbw@DESKTOP:~$ chmod 000 my_dir
cbw@DESKTOP:~$ chmod 100 my_program.py
cbw@DESKTOP:~$ chmod 777 my_file
cbw@DESKTOP:~$ ls -l
d--------- 0 cbw   cbw      512 Jan 29 22:46 my_dir
-rwxrwxrwx 1 cbw   cbw       17 Jan 29 22:46 my_file
---x------ 1 cbw   faculty  313 Jan 29 22:47 my_program.py
```

Who May Change Permissions?

```
cbw@DESKTOP:~$ groups
cbw faculty
cbw@DESKTOP:~$ ls -l
-rw-rw-rw- 1 cbw  cbw      17 Jan 29 22:46 my_file
-rw-rw-rw- 1 cbw  faculty  17 Jan 29 22:46 my_other_file
-rw------- 1 root root    896 Jan 29 22:47 sensitive_data.csv
-rwxrwx--- 1 root faculty 313 Jan 29 22:47 program.py
```

- Which files is user *cbw* permitted to *chmod*? all files owned by cbw

Who May Change Permissions?

```
cbw@DESKTOP:~$ groups
cbw faculty
cbw@DESKTOP:~$ ls -l
-rw-rw-rw- 1 cbw  cbw      17 Jan 29 22:46 my_file
-rw-rw-rw- 1 cbw  faculty  17 Jan 29 22:46 my_other_file
-rw------- 1 root root    896 Jan 29 22:47 sensitive_data.csv
-rwxrwx--- 1 root faculty 313 Jan 29 22:47 program.py
```

- Which files is user *cbw* permitted to *chmod*?
  - Only owners can *chmod* files
  - *cbw* can *chmod my_file* and *my_other_file*
  - Group membership doesn't grant *chmod* ability (cannot *chmod program.py*)

Setting Ownership

- Unix uses discretionary access control
  - New objects are owned by the subject that created them
- How can you modify the owner or group of an object?

chown <owner>:<group> <file1> [file2] …

Who May Change Ownership?

```
cbw@DESKTOP:~$ groups
cbw faculty
cbw@DESKTOP:~$ ls -l
-rw-rw-rw- 1 cbw   cbw      17 Jan 29 22:46 my_file
-rw-rw-rw- 1 cbw   faculty  17 Jan 29 22:46 my_other_file
-rw------- 1 root  root    896 Jan 29 22:47 sensitive_data.csv
-rwxrwx--- 1 root  faculty 313 Jan 29 22:47 program.py
```

- Which operations are permitted?

chown cbw:faculty my_file

chown root:root my_other_file — FAIL. cbw not in root group

chown cbw:cbw sensitive_date.csv — FAIL cant hijack file

chown cbw:faculty program.py — FAIL

Who May Change Ownership?

```
cbw@DESKTOP:~$ groups
cbw faculty
cbw@DESKTOP:~$ ls -l
-rw-rw-rw- 1 cbw  cbw      17 Jan 29 22:46 my_file
-rw-rw-rw- 1 cbw  faculty  17 Jan 29 22:46 my_other_file
-rw------- 1 root root    896 Jan 29 22:47 sensitive_data.csv
-rwxrwx--- 1 root faculty 313 Jan 29 22:47 program.py
```

- Which operations are permitted?

chown cbw:faculty my_file	Yes, cbw belongs to the faculty group
chown root:root my_other_file	No, only root many change file owners!
chown cbw:cbw sensitive_date.csv	No, only root many change file owners!
chown cbw:faculty program.py	No, only root many change file owners!

Unix Access Control Exercise (1)

- What Unix group and permission assignments satisfy this access control matrix?

Desired Permissions

	file1	file2
user1	r--	rwx
user2	r--	rw-
user3	r--	rw-
user4	rwx	rw-

Unix Access Control Exercise (1)

- What Unix group and permission assignments satisfy this access control matrix?

Desired Permissions

	file1	file2
user1	r--	rwx
user2	r--	rw-
user3	r--	rw-
user4	rwx	rw-

User	Groups
user1	user1
user2	user2
user3	user3
user4	user4

```
~$ ls -l
-rwxr--r-- 1 user4  user4  0 file1
-rwxrw-rw- 1 user1  user1  0 file2
```

Unix Access Control Exercise (2)

- What Unix group and permission assignments satisfy this access control matrix?

Desired Permissions

	file1	file2
user1	r--	--x
user2	r-x	rwx
user3	r-x	r--
user4	rwx	r--

group1: user2 user3

group2: user3 user4

f1 u4 g1 rwx r-x r--

f2 u2 g2 rwx r-- --x

Unix Access Control Exercise (2)

- What Unix group and permission assignments satisfy this access control matrix?

Desired Permissions

	file1	file2
user1	r--	--x
user2	r-x	rwx
user3	r-x	r--
user4	rwx	r--

User	Groups
user1	user1
user2	user2, group1
user3	user3, group1, group2
user4	user4, group2

```
~$ ls -l
-rwxr-xr-- 1 user4  group1  0 file1
-rwxr----x 1 user2  group2  0 file2
```

Unix Access Control Exercise (3)

- What Unix group and permission assignments satisfy this access control matrix?

not possible.

Desired Permissions

	file 1	file 2
user 1	---	rw-
user 2	r--	r--
user 3	rwx	rwx
user 4	rwx	---

4 access patterns, can not be expressed in unix permissions.

Unix Access Control Exercise (3)

- What Unix group and permission assignments satisfy this access control matrix?

Desired Permissions

	file 1	file 2
user 1	---	rw-
user 2	r--	r--
user 3	rwx	rwx
user 4	rwx	---

- Trick question! This matrix **cannot** be represented

Unix Access Control Exercise (3)

- What Unix group and permission assignments satisfy this access control matrix?

Desired Permissions

	file 1	file 2
user 1	---	rw-
user 2	r--	r--
user 3	rwx	rwx
user 4	rwx	---

- Trick question! This matrix **cannot** be represented
- *file2*: four distinct privilege levels
  - Maximum of three levels (user, group, other)

Unix Access Control Exercise (3)

- What Unix group and permission assignments satisfy this access control matrix?

Desired Permissions

	file 1	file 2
user 1	---	rw-
user 2	r--	r--
user 3	rwx	rwx
user 4	rwx	---

- Trick question! This matrix **cannot** be represented
- *file2*: four distinct privilege levels
  - Maximum of three levels (user, group, other)
- *file1*: two users have high privileges
  - If *user3* and *user4* are in a group, how to give *user2* read and *user1* nothing?
  - If *user1* or *user2* are owner, they can grant themselves write and execute permissions :(

Unix Access Control Review

The Good

- Very simple model
  - Owners, groups, and other
  - Read, write, execute
- Relatively simple to manage and understand

The Bad

limited

Unix Access Control Review

The Good

- Very simple model
  - Owners, groups, and other
  - Read, write, execute
- Relatively simple to manage and understand

The Bad

- Not all policies can be encoded!
  - Contrast to ACL

Unix Access Control Review

The Good

- Very simple model
  - Owners, groups, and other
  - Read, write, execute
- Relatively simple to manage and understand

The Bad

- Not all policies can be encoded!
  - Contrast to ACL
- Not quite as simple as it seems
  - setuid

Problems with Principals

- setuid
- The Confused Deputy Problem
- Capability-based Access Control (rows)

From Principals to Subjects

- Thus far, we have focused on principals
  - What user created/owns an object?
  - What groups does a user belong to?
- What about subjects?
  - When you run a program, what permissions does it have?
  - Who is the "owner" of a running program?

ps

Process Owners

```
cbw@DESKTOP:~$ ls -l
-rwxr-xr-x 1 cbw cbw 313 Jan 29 22:47 my_program.py
cbw@DESKTOP:~$ ./my_program.py
…
```

Process Owners

```
cbw@DESKTOP:~$ ls -l
-rwxr-xr-x 1 cbw cbw 313 Jan 29 22:47 my_program.py
cbw@DESKTOP:~$ ./my_program.py
…
```

Who is the owner of this process?

Process Owners

```
cbw@DESKTOP:~$ ls -l
-rwxr-xr-x 1 cbw cbw 313 Jan 29 22:47 my_program.py
cbw@DESKTOP:~$ ./my_program.py
…
```

Who is the owner of this process?

```
cbw@DESKTOP:~$ ps aux | grep my_program.py
cbw          tty1      S      01:06    0:00 python ./my_program.py
```

Process Owners

```
cbw@DESKTOP:~$ ls -l
-rwxr-xr-x 1 cbw cbw 313 Jan 29 22:47 my_program.py
cbw@DESKTOP:~$ ./my_program.py
…
```

cbw is the owner. Why?

Who is the owner of this process?

```
cbw@DESKTOP:~$ ps aux | grep my_program.py
cbw          tty1      S      01:06    0:00 python ./my_program.py
```

Process Owners

```
cbw@DESKTOP:~$ ls -l /bin/ls*
-rwxr-xr-x 1 root root 110080 Mar 10  2016 /bin/ls
-rwxr-xr-x 1 root root  44688 Nov 23  2016 /bin/lsblk
cbw@DESKTOP:~$ ls
…
```

Process Owners

```
cbw@DESKTOP:~$ ls -l /bin/ls*
-rwxr-xr-x 1 root root 110080 Mar 10  2016 /bin/ls
-rwxr-xr-x 1 root root  44688 Nov 23  2016 /bin/lsblk
cbw@DESKTOP:~$ ls
…
```

Who is the owner of this process?

Process Owners

```
cbw@DESKTOP:~$ ls -l /bin/ls*
-rwxr-xr-x 1 root root 110080 Mar 10  2016 /bin/ls
-rwxr-xr-x 1 root root  44688 Nov 23  2016 /bin/lsblk
cbw@DESKTOP:~$ ls
…
```

Who is the owner of this process?

```
cbw@DESKTOP:~$ ps aux | grep ls
cbw          tty1     S     01:06    0:00 /bin/ls
```

Process Owners

```
cbw@DESKTOP:~$ ls -l /bin/ls*
-rwxr-xr-x 1 root root 110080 Mar 10  2016 /bin/ls
-rwxr-xr-x 1 root root  44688 Nov 23  2016 /bin/lsblk
cbw@DESKTOP:~$ ls
…
```

Who is the owner of this process?

cbw is the owner. Why?

```
cbw@DESKTOP:~$ ps aux | grep ls
cbw          tty1      S     01:06    0:00 /bin/ls
```

Process Owners

```
cbw@DESKTOP:~$ ls -l /bin/ls*
-rwxr-xr-x 1 root root 110080 Mar 10  2016 /bin/ls
-rwxr-xr-x 1 root root  44688 Nov 23  2016 /bin/lsblk
cbw@DESKTOP:~$ ls
…
```

Who is the owner of this process?

cbw is the owner. Why?

```
cbw@DESKTOP:~$ ps aux | grep ls
cbw          tty1      S     01:06    0:00 /bin/ls
```

Subject Ownership

Subject Ownership

- Under normal circumstances, subjects are owned by the principal that executes them
  - **File ownership is irrelevant**
- Why is this important for security?
  - A principal that is able to execute a file owned by root should not be granted root privileges

Subject Ownership

- Under normal circumstances, subjects are owned by the principal that executes them
  - **File ownership is irrelevant**
- Why is this important for security?
  - A principal that is able to execute a file owned by root should not be granted root privileges

```
cbw@DESKTOP:~$ ls -l /bin/bash
-rwxr-xr-x 1 root root 110080 Mar 10  2016 /bin/bash
```

Corner Cases

```
cbw@DESKTOP:~$ passwd
Changing password for cbw.
(current) UNIX password:
```

Corner Cases

```
cbw@DESKTOP:~$ passwd
Changing password for cbw.
(current) UNIX password:
```

- Consider the *passwd* program
  - All users must be able to execute it (to set and change their passwords)
  - Must have write access to */etc/shadow* (file where password hashes are stored)
- Problem: */etc/shadow* is only writable by root user

```
cbw@DESKTOP:~$ ls -l /etc/shadow
-rw-r----- 1 root shadow 922 Jan  8 14:56 /etc/shadow
```

setuid

x, s

```
cbw@DESKTOP:~$ ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 47032 May 16  2017 /usr/bin/passwd
cbw@DESKTOP:~$ passwd
Changing password for cbw.
(current) UNIX password:
```

setuid

```
cbw@DESKTOP:~$ ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 47032 May 16  2017 /usr/bin/passwd
cbw@DESKTOP:~$ passwd
Changing password for cbw.
(current) UNIX password:
```

setuid

- Objects may have the *setuid* permission
  - Program may execute as the **file owner**, rather than **executing principal**

```
cbw@DESKTOP:~$ ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 47032 May 16  2017 /usr/bin/passwd
cbw@DESKTOP:~$ passwd
Changing password for cbw.
(current) UNIX password:
```

setuid

- Objects may have the *setuid* permission
  - Program may execute as the **file owner**, rather than **executing principal**

```
cbw@DESKTOP:~$ ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 47032 May 16  2017 /usr/bin/passwd
cbw@DESKTOP:~$ passwd
Changing password for cbw.
(current) UNIX password:
```

```
cbw@DESKTOP:~$ ps aux | grep passwd
root        tty1     S     01:06    0:00 python ./my_program.py
```

setuid

- Objects may have the *setuid* permission
  - Program may execute as the **file owner**, rather than **executing principal**

```
cbw@DESKTOP:~$ ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 47032 May 16  2017 /usr/bin/passwd
cbw@DESKTOP:~$ passwd
Changing password for cbw.
(current) UNIX password:
```

```
cbw@DESKTOP:~$ ps aux | grep passwd
root         tty1     S     01:06    0:00 python ./my_program.py
```

chmod Revisited

- How to add *setuid* to an object?

chmod u+s <file1> [file2] …

chmod 2### <file1> [file2] …

chmod Revisited

- How to add *setuid* to an object?

chmod u+s <file1> [file2] …

chmod 2### <file1> [file2] …

- **WARNING: NEVER SET A SCRIPT AS SETUID**
  - Only set *setuid* on compiled binary programs
  - Scripts with *setuid* lead to Time of Check Time of Use (TOCTOU) vulnerabilities

Another setuid Example

- Consider an example *turnin* program

/cs2550/turnin <project #> <in_file> <out_file>

1. Copies *<in_file>* to *<out_file>*
2. Grades the assignment
3. Writes the grade to */cs2550/<project#>/grades*

Another setuid Example

- Consider an example *turnin* program

  /cs2550/turnin <project #> <in_file> <out_file>

1. Copies *<in_file>* to *<out_file>*
2. Grades the assignment
3. Writes the grade to */cs2550/<project#>/grades*

- Challenge: students cannot have write access to project directories or grade files
  - *turnin* program must be *setuid*

```
alice@login:~$ /cs2550/turnin project1 pwcrack.py /cs2550/project1/pwcrack.py
Thank you for turning in project 1.
```

```
alice@login:~$ /cs2550/turnin project1 pwcrack.py /cs2550/project1/
pwcrack.py
Thank you for turning in project 1.
alice@login:~$ ls –l /cs2550/
drwx--x--x 0 cbw  faculty     512 Jan 29 22:46 project1
-rwsr-xr-x 1 cbw  faculty      17 Jan 29 22:46 turnin
```

```
alice@login:~$ /cs2550/turnin project1 pwcrack.py /cs2550/project1/
pwcrack.py
Thank you for turning in project 1.
alice@login:~$ ls —l /cs2550/
drwx--x--x 0 cbw  faculty    512 Jan 29 22:46 project1
-rwsr-xr-x 1 cbw  faculty     17 Jan 29 22:46 turnin
alice@login:~$ ls —l /cs2550/project1/
-r-x------ 0 cbw  faculty    512 Jan 29 22:46 pwcrack.py
-rw------- 1 cbw  faculty     17 Jan 29 22:46 grades
```

Ambient Authority

Ambient Authority

- Ambient authority
  - A subject's permissions are automatically exercised
  - No need to select specific permissions
- Systems that use ACLs or Unix-style permissions grant ambient authority
  - A subject automatically gains all permissions of the principal
  - A setuid subject also gains permissions of the file owner
- Ambient authority is a security vulnerability

The Confused Deputy Problem

```
mallory@login:~$ /cs2550/turnin project1 best_grade.txt /cs2550/project1/grades
Thank you for turning in project 1.
alice@login:~$ ls –l /cs2550/project1/
```

The Confused Deputy Problem

```
mallory@login:~$ /cs2550/turnin project1 best_grade.txt /cs2550/project1/grades
Thank you for turning in project 1.
alice@login:~$ ls –l /cs2550/project1/
-rw------- 1 cbw  faculty      17 Jan 29 22:46 grades
```

The Confused Deputy Problem

```
mallory@login:~$ /cs2550/turnin project1 best_grade.txt /cs2550/project1/grades
Thank you for turning in project 1.
alice@login:~$ ls –l /cs2550/project1/
-rw------- 1 cbw  faculty      17 Jan 29 22:46 grades
```

- The *turnin* program is a confused deputy
  - It is the deputy of two principals: *mallory* and *cbw*
  - *mallory* cannot directly access */cs2550/project1/grades*
  - However, *cbw* can access */cs2550/project1/grades*

The Confused Deputy Problem

```
mallory@login:~$ /cs2550/turnin project1 best_grade.txt /cs2550/project1/grades
Thank you for turning in project 1.
alice@login:~$ ls –l /cs2550/project1/
-rw------- 1 cbw  faculty      17 Jan 29 22:46 grades
```

- The *turnin* program is a confused deputy
  - It is the deputy of two principals: *mallory* and *cbw*
  - *mallory* cannot directly access */cs2550/project1/grades*
  - However, *cbw* can access */cs2550/project1/grades*
- Key problem: the subject cannot tell which principal it is serving when it performs a write

Preventing Confused Deputies

- ACL and Unix-style systems are fundamentally vulnerable to confused deputies
  - Cannot prevent misuse of ambient authority
- Solution: move to capability-based access control system

Capabilities

ACLs

- Encode columns of an access control matrix

ACL for o_2

	o_1	o_2	o_3
s_1	RW	RX	
s_2	R	RWX	RW
s_3		RWX	

Capabilities

Capabilities

ACLs

- Encode columns of an access control matrix

ACL for o_2

	o_1	o_2	o_3
s_1	RW	RX	
s_2	R	RWX	RW
s_3		RWX	

Capabilities

- Encode rows of an access control matrix

	o_1	o_2	o_3
s_1	RW	RX	
s_2	R	RWX	RW
s_3		RWX	

Capabilities for s_1

Capability-based Access Control

- Principals and subjects have capabilities which:
  - Give them access to objects
    - Files, keys, devices, etc.
  - Are transferable and unforgeable tokens of authority
    - Can be passed from principal to subject, and subject to subject
    - Similar to file descriptors
- Why do capabilities solve the confused deputy problem?
  - When attempting to access an object, a capability must be selected
  - Selecting a capability inherently also selects a master

Confused Deputy Revisited

```
mallory@login:~$ /cs2550/turnin project1 best_grade.txt /
cs2550/project1/grades
```

Confused Deputy Revisited

Principal	…	/home/mallory/*	/cs2550/project1/grades	…
mallory	…	RWX	---	…

```
mallory@login:~$ /cs2550/turnin project1 best_grade.txt /
cs2550/project1/grades
```

Confused Deputy Revisited

Principal	…	/home/mallory/*	/cs2550/project1/grades	…
mallory	…	RWX	---	…

```
mallory@login:~$ /cs2550/turnin project1 best_grade.txt /
cs2550/project1/grades
```

Allow

Deny

Confused Deputy Revisited

Principal	…	/home/mallory/*	/cs2550/project1/grades	…
mallory	…	RWX	---	…

Allow

```
mallory@login:~$ /cs2550/turnin project1 best_grade.txt /cs2550/project1/grades
ERROR: Permission denied to /cs2550/project1/grades
```

Deny

Confused Deputy Revisited

Principal	…	/home/mallory/*	/cs2550/project1/grades	…
mallory	…	RWX	---	…

```
mallory@login:~$ /cs2550/turnin project1 best_grade.txt /cs2550/project1/grades
ERROR: Permission denied to /cs2550/project1/grades
```

Allow

Deny

- Principal must pass capabilities to objects at invocation time
  - *mallory* has permission to access best_grade.txt
  - *mallory* does not have permission to access */cs2550/project1/grades*

Confused Deputy Revisited

Principal	...	/home/mallory/*	/cs2550/project1/grades	...
mallory	...	RWX	---	...

Allow

```
mallory@login:~$ /cs2550/turnin project1 best_grade.txt /
cs2550/project1/grades
ERROR: Permission denied to /cs2550/project1/grades
```

Deny

- Principal must pass capabilities to objects at invocation time
  - *mallory* has permission to access best_grade.txt
  - *mallory* does not have permission to access */cs2550/project1/grades*
- No ambient authority in a capability-based access control system
  - Principal cannot pass a capability it doesn't have

Capabilities vs. ACLs

- Consider two security mechanisms for bank accounts

1. Identity-based
   - Each account has multiple authorized owners
   - To authenticate, show a valid ID at the bank
   - Once authenticated, you may access all authorized accounts
2. Token-based
   - When opening an account, you are given a unique hardware key
   - To access an account, you must possess the corresponding key
   - Keys may be passed from person to person

Capabilities vs. ACLs

- Consider two security mechanisms for bank accounts

1. Identity-based
   - Each account has multiple authorized owners
   - To authenticate, show a valid ID at the bank
   - Once authenticated, you may access all authorized accounts

- ACL system
- Ambient authority to access all authorized accounts

2. Token-based
   - When opening an account, you are given a unique hardware key
   - To access an account, you must possess the corresponding key
   - Keys may be passed from person to person

Capabilities vs. ACLs

- Consider two security mechanisms for bank accounts

1. Identity-based
   - Each account has multiple authorized owners
   - To authenticate, show a valid ID at the bank
   - Once authenticated, you may access all authorized accounts

> - ACL system
> - Ambient authority to access all authorized accounts

2. Token-based
   - When opening an account, you are given a unique hardware key
   - To access an account, you must possess the corresponding key
   - Keys may be passed from person to person

> - Capability system
> - No ambient authority

Capabilities IRL

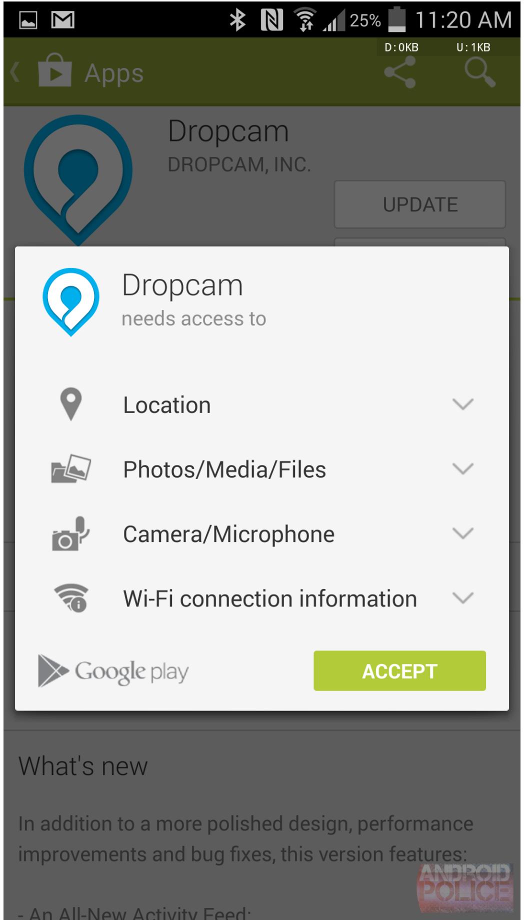
- From a security perspective, capability systems are more secure than ACL and Unix-style systems
- … and yet, most major operating systems use the latter
- Why?
  - Easier for users
    - ACLs are good for user-level sharing, intuitive
    - Capabilities are good for process-level sharing, not untuitive
  - Easier for developers
    - Processes are tightly coupled in capability systems
    - Must carefully manage passing capabilities around
    - In contrast, ambient authority makes programming easy, but insecure

Small Steps Towards Capabilities

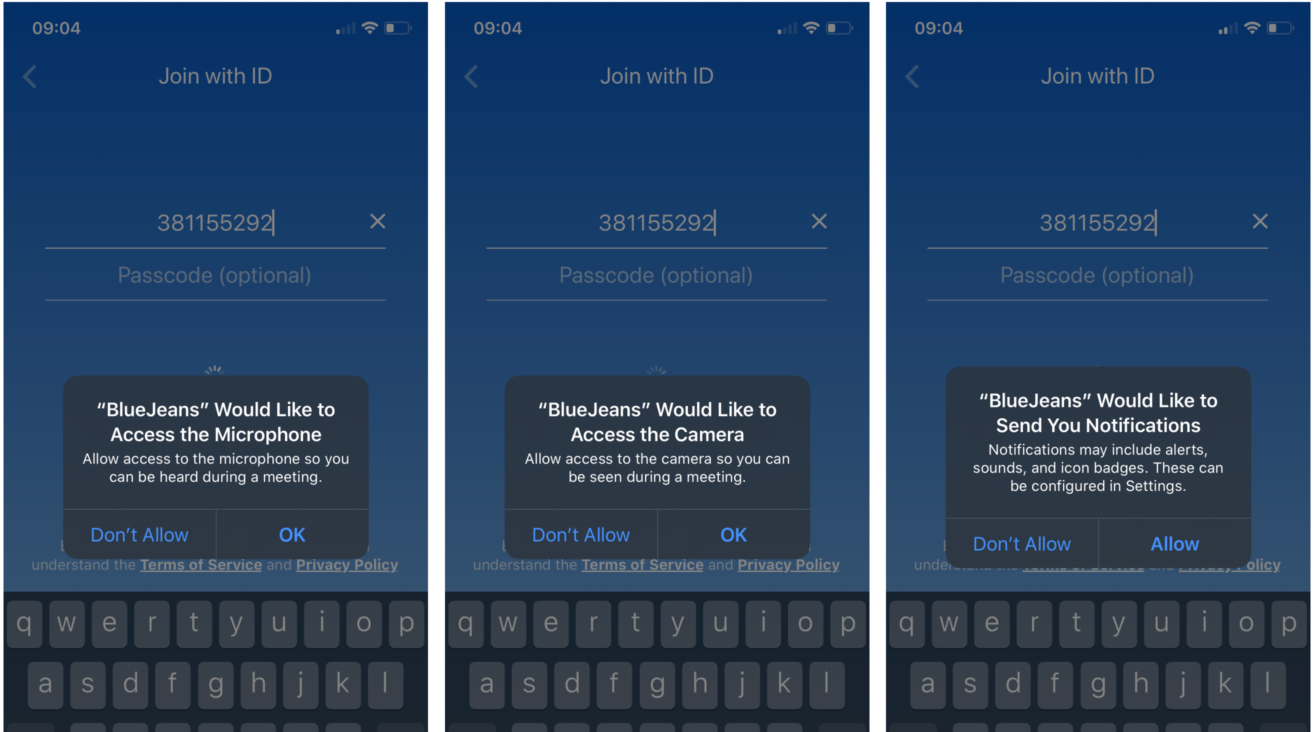
- Some limited examples of capability systems exist
  - Android/iOS app permissions
  - POSIX capabilities
  - SELinux

Android/iOS Capabilities

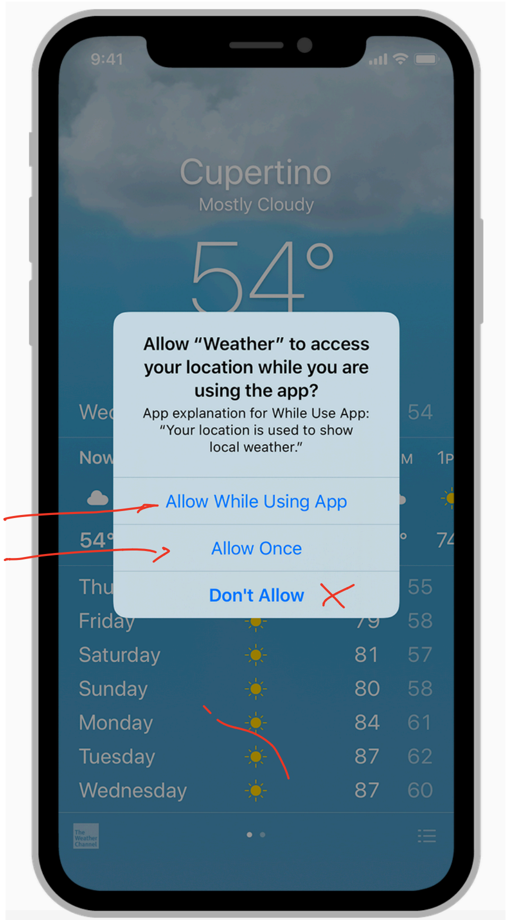
- Android and iOS support (relatively) fine grained capabilities for apps
  - User must grant permissions to apps at install time
  - May only access sensitive APIs with user consent
- Apps can "borrow" capabilities from each other by exporting *intents*
  - Example: an app without camera access can ask the camera app to return a photo



Android/IOS just-in-time capability



Per-event capability



Se-linux

POSIX Capabilities

- Traditional Unix systems had two types of processes
  - Privileged, i.e. root processes
    - Bypass all security and access control checks
  - Unprivileged, i.e. everything else
    - Subject to access controls
- Modern Unix/Linux systems offer some finer grained capabilities
  - Specified processes may be granted a subset of root privileges
  - CAP_CHOWN: make arbitrary changes to file owners and groups
  - CAP_KILL: kill arbitrary processes
  - CAP_SYS_TIME: change the system clock

Keeping Secrets?

- Suppose we have secret data that only certain users should access
- Is DAC enough to prevent leaks?

```
charlie@DESKTOP:~$ groups
charlie topsecret
```

Keeping Secrets?

- Suppose we have secret data that only certain users should access
- Is DAC enough to prevent leaks?

```
charlie@DESKTOP:~$ groups
charlie topsecret
charlie@DESKTOP:~$ ls –la /top-secret-intel/
drwxr-xr-x 0 root root       512 Jan  8 14:55 .
drwxr-xr-x 0 root root       512 Oct 11 19:58 ..
-rw-r----- 1 root topsecret 896 Jan 29 22:47 northkorea.pdf
```

Keeping Secrets?

- Suppose we have secret data that only certain users should access
- Is DAC enough to prevent leaks?

```
charlie@DESKTOP:~$ groups
charlie topsecret
charlie@DESKTOP:~$ ls –la /top-secret-intel/
drwxr-xr-x 0 root root       512 Jan  8 14:55 .
drwxr-xr-x 0 root root       512 Oct 11 19:58 ..
-rw-r----- 1 root topsecret 896 Jan 29 22:47 northkorea.pdf
```

Keeping Secrets?

- Suppose we have secret data that only certain users should access
- Is DAC enough to prevent leaks?

```
charlie@DESKTOP:~$ groups
charlie topsecret
charlie@DESKTOP:~$ ls –la /top-secret-intel/
drwxr-xr-x 0 root root      512 Jan  8 14:55 .
drwxr-xr-x 0 root root      512 Oct 11 19:58 ..
-rw-r----- 1 root topsecret 896 Jan 29 22:47 northkorea.pdf
charlie@DESKTOP:~$ groups mallory
mallory secret
```

Keeping Secrets?

- Suppose we have secret data that only certain users should access
- Is DAC enough to prevent leaks?

```
charlie@DESKTOP:~$ groups
charlie topsecret
charlie@DESKTOP:~$ ls –la /top-secret-intel/
drwxr-xr-x 0 root root      512 Jan  8 14:55 .
drwxr-xr-x 0 root root      512 Oct 11 19:58 ..
-rw-r----- 1 root topsecret 896 Jan 29 22:47 northkorea.pdf
charlie@DESKTOP:~$ groups mallory
mallory secret
```

Keeping Secrets?

- Suppose we have secret data that only certain users should access
- Is DAC enough to prevent leaks?

```
charlie@DESKTOP:~$ groups
charlie topsecret
charlie@DESKTOP:~$ ls –la /top-secret-intel/
drwxr-xr-x 0 root root      512 Jan  8 14:55 .
drwxr-xr-x 0 root root      512 Oct 11 19:58 ..
-rw-r----- 1 root topsecret 896 Jan 29 22:47 northkorea.pdf
charlie@DESKTOP:~$ groups mallory
mallory secret
charlie@DESKTOP:~$ ls –la /home/mallory
drwxrwxrwx 0 mallory mallory   512 Jan  8 14:55 .
drwxr-xr-x 0 root    root      512 Oct 11 19:58 ..
```

Keeping Secrets?

- Suppose we have secret data that only certain users should access
- Is DAC enough to prevent leaks?

```
charlie@DESKTOP:~$ groups
charlie topsecret
charlie@DESKTOP:~$ ls –la /top-secret-intel/
drwxr-xr-x 0 root root      512 Jan  8 14:55 .
drwxr-xr-x 0 root root      512 Oct 11 19:58 ..
-rw-r----- 1 root topsecret 896 Jan 29 22:47 northkorea.pdf
charlie@DESKTOP:~$ groups mallory
mallory secret
charlie@DESKTOP:~$ ls –la /home/mallory
drwxrwxrwx 0 mallory mallory   512 Jan  8 14:55 .
drwxr-xr-x 0 root    root      512 Oct 11 19:58 ..
```

Keeping Secrets?

- Suppose we have secret data that only certain users should access
- Is DAC enough to prevent leaks?

```
charlie@DESKTOP:~$ groups
charlie topsecret
charlie@DESKTOP:~$ ls –la /top-secret-intel/
drwxr-xr-x 0 root root      512 Jan  8 14:55 .
drwxr-xr-x 0 root root      512 Oct 11 19:58 ..
-rw-r----- 1 root topsecret 896 Jan 29 22:47 northkorea.pdf
charlie@DESKTOP:~$ groups mallory
mallory secret
charlie@DESKTOP:~$ ls –la /home/mallory
drwxrwxrwx 0 mallory mallory   512 Jan  8 14:55 .
drwxr-xr-x 0 root    root      512 Oct 11 19:58 ..
charlie@DESKTOP:~$ cp /top-secret-intel/northkorea.pdf /home/mallory
charlie@DESKTOP:~$ ls –l /home/mallory
-rw-r----- 1 charlie charlie 896 Jan 29 22:47 northkorea.pdf
charlie@DESKTOP:~$ chmod ugo+rw /home/mallory/northkorea.pdf
```

Keeping Secrets?

- Suppose we have secret data that only certain users should access
- Is DAC enough to prevent leaks?

```
charlie@DESKTOP:~$ groups
charlie topsecret
charlie@DESKTOP:~$ ls —la /top-secret-intel/
drwxr-xr-x 0 root root      512 Jan  8 14:55 .
drwxr-xr-x 0 root root      512 Oct 11 19:58 ..
-rw-r----- 1 root topsecret 896 Jan 29 22:47 northkorea.pdf
charlie@DESKTOP:~$ groups mallory
mallory secret
charlie@DESKTOP:~$ ls —la /home/mallory
drwxrwxrwx 0 mallory mallory   512 Jan  8 14:55 .
drwxr-xr-x 0 root    root      512 Oct 11 19:58 ..
charlie@DESKTOP:~$ cp /top-secret-intel/northkorea.pdf /home/mallory
charlie@DESKTOP:~$ ls —l /home/mallory
-rw-r----- 1 charlie charlie 896 Jan 29 22:47 northkorea.pdf
charlie@DESKTOP:~$ chmod ugo+rw /home/mallory/northkorea.pdf
```

Keeping Secrets?

- Suppose we have secret data that only certain users should access
- Is DAC enough to prevent leaks?

```
charlie@DESKTOP:~$ groups
charlie topsecret
charlie@DESKTOP:~$ ls –la /top-secret-intel/
drwxr-xr-x 0 root root      512 Jan  8 14:55 .
drwxr-xr-x 0 root root      512 Oct 11 19:58 ..
-rw-r----- 1 root topsecret 896 Jan 29 22:47 northkorea.pdf
charlie@DESKTOP:~$ groups mallory
mallory secret
charlie@DESKTOP:~$ ls –la /home/mallory
drwxrwxrwx 0 mallory mallory   512 Jan  8 14:55 .
drwxr-xr-x 0 root    root      512 Oct 11 19:58 ..
charlie@DESKTOP:~$ cp /top-secret-intel/northkorea.pdf /home/mallory
charlie@DESKTOP:~$ ls –l /home/mallory
-rw-r----- 1 charlie charlie 896 Jan 29 22:47 northkorea.pdf
charlie@DESKTOP:~$ chmod ugo+rw /home/mallory/northkorea.pdf
```

Failure of DAC

- DAC cannot prevent the leaking of secrets

User A

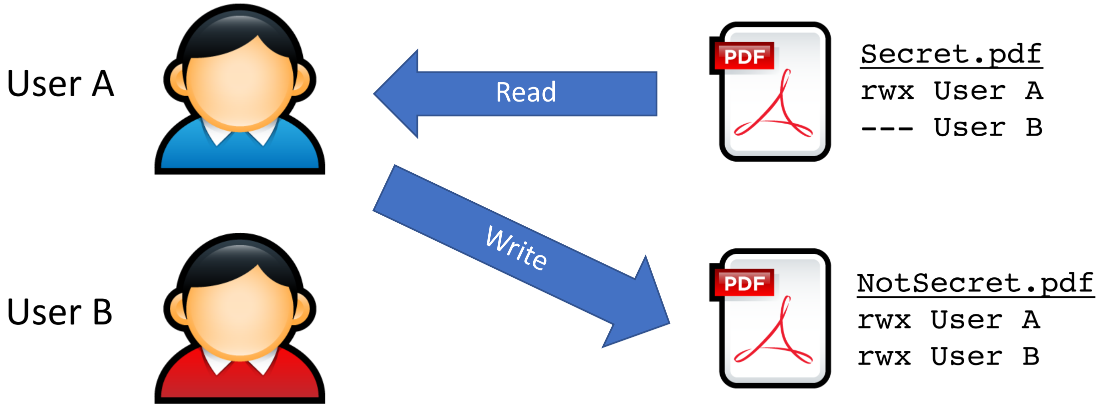
User B

Secret.pdf
rwx User A
--- User B

NotSecret.pdf
rwx User A
rwx User B

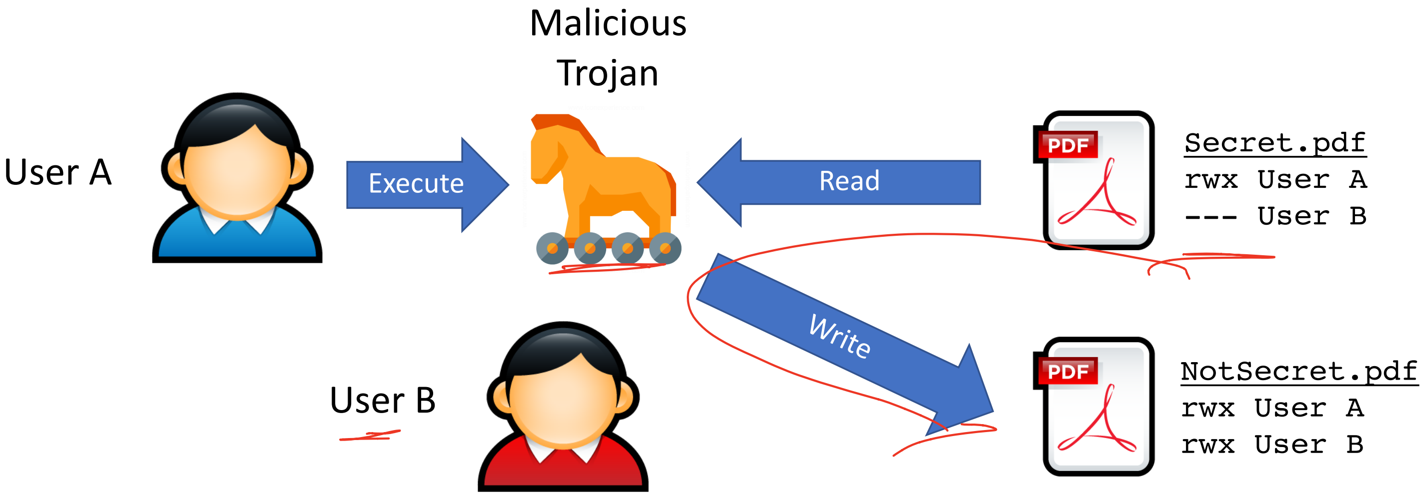
Failure of DAC

- DAC cannot prevent the leaking of secrets



Failure of DAC

- DAC cannot prevent the leaking of secrets



Mandatory Access Control

Mandatory Access Control Goals

- Restrict the access of subjects to objects based on a system-wide policy

Bell-Lapadula (1973)

"No read up, no write down"

System Model: Abstract machine that keeps track of the "system state"

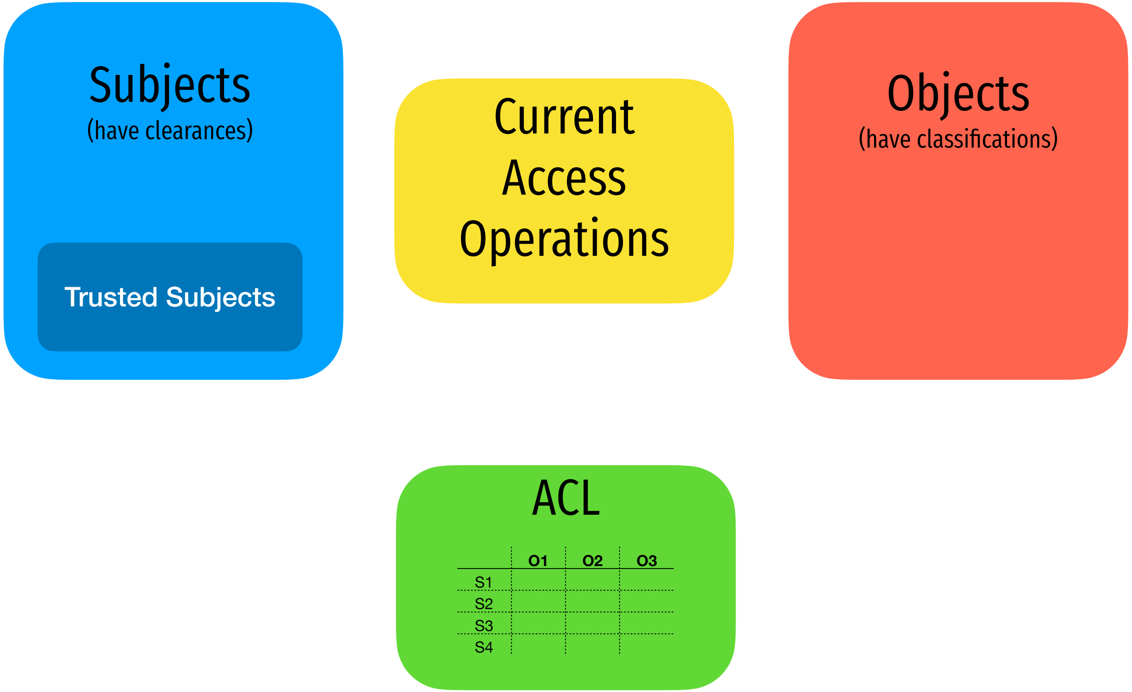
Security Policy: Rules govern changes in the state

BLP System Model

Clearances:

Classifications:

BLP System State



Elements of the Bell-LaPadula Model

Subjects

$L_m(s)$: maximum level

$L_c(s)$: current level

Top Secret

Secret

Confidential

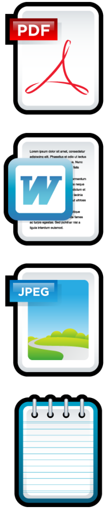
Discretionary Access Control Matrix

Defined by the administrator

	o_1	o_2	o_3
s_1	RW	RX	
s_2	R	RWX	RW
s_3		RWX	

Objects

$L(o)$: level



Top Secret

Secret

Confidential

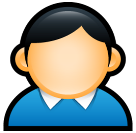
Unclassified

Simplified Bell-LaPadula Example

- Assume $L_m(s) = L_c(s)$ is always true

Top Secret


Secret

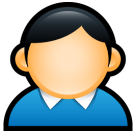
Confidential


Confidential


Unclassified

Simplified Bell-LaPadula Example

- Assume $L_m(s) = L_c(s)$ is always true
- ★-property
  - s can read o iff $L(s) >= L(o)$ (**no read up**)
  - s can write o iff $L(s) <= L(o)$ (**no write down**)

Confidential 

Top Secret


Secret

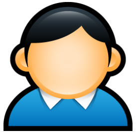
Confidential

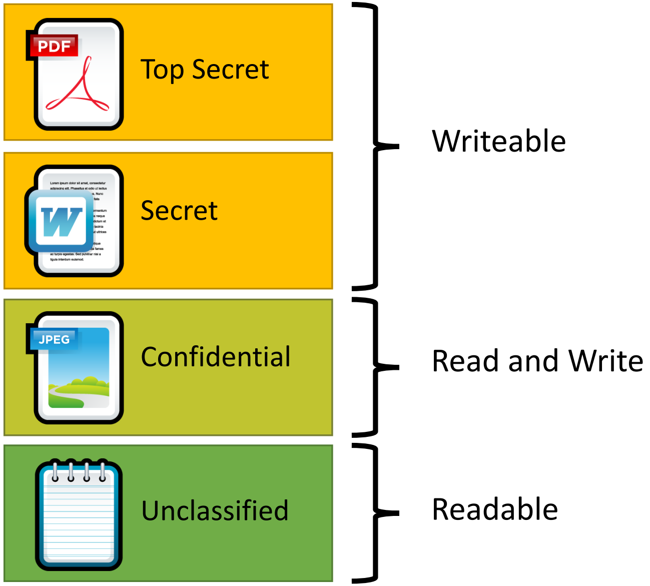

Unclassified

Simplified Bell-LaPadula Example

- Assume $L_m(s) = L_c(s)$ is always true
- ★-property
  - *s* can read *o* iff $L(s) >= L(o)$ (**no read up**)
  - *s* can write *o* iff $L(s) <= L(o)$ (**no write down**)

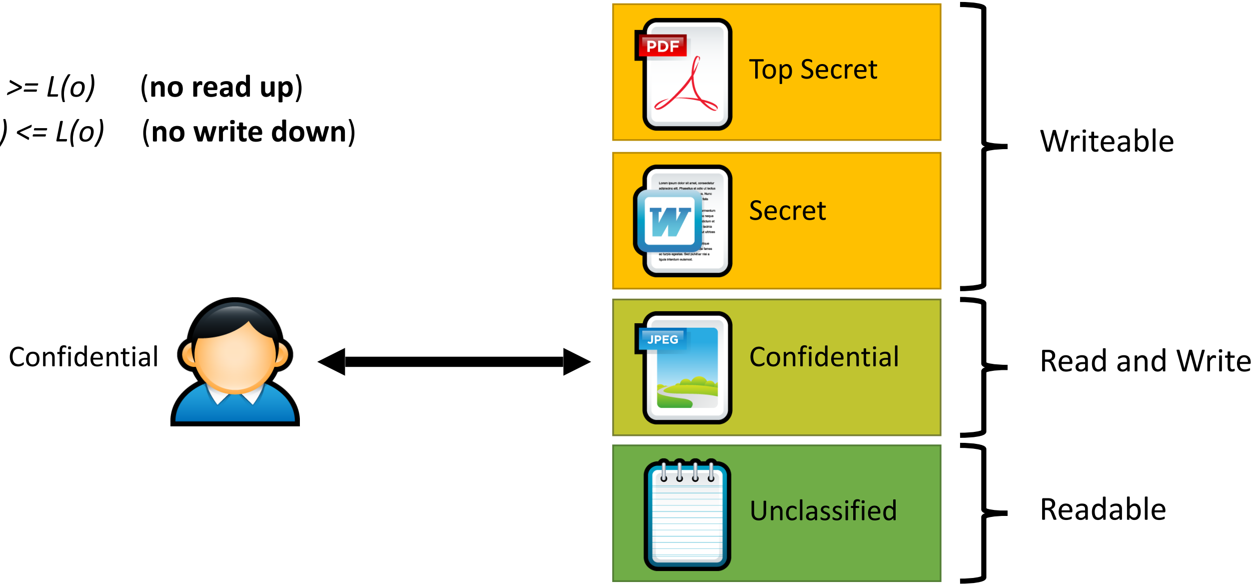
Confidential





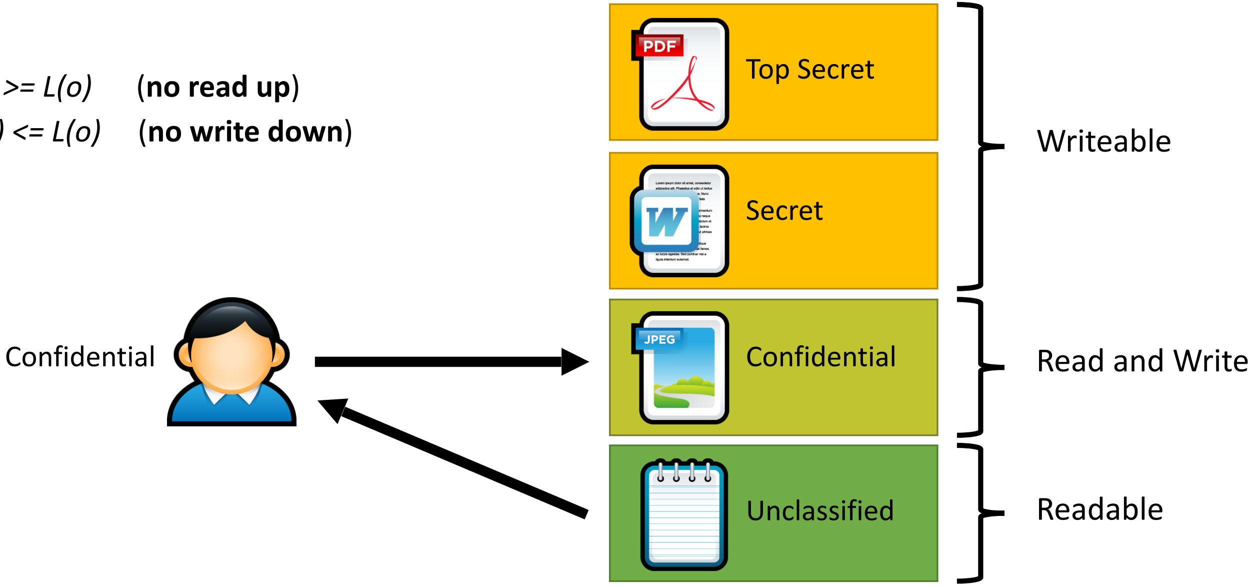
Simplified Bell-LaPadula Example

- Assume $L_m(s) = L_c(s)$ is always true
- ★-property
  - *s* can read *o* iff $L(s) >= L(o)$ (**no read up**)
  - *s* can write *o* iff $L(s) <= L(o)$ (**no write down**)



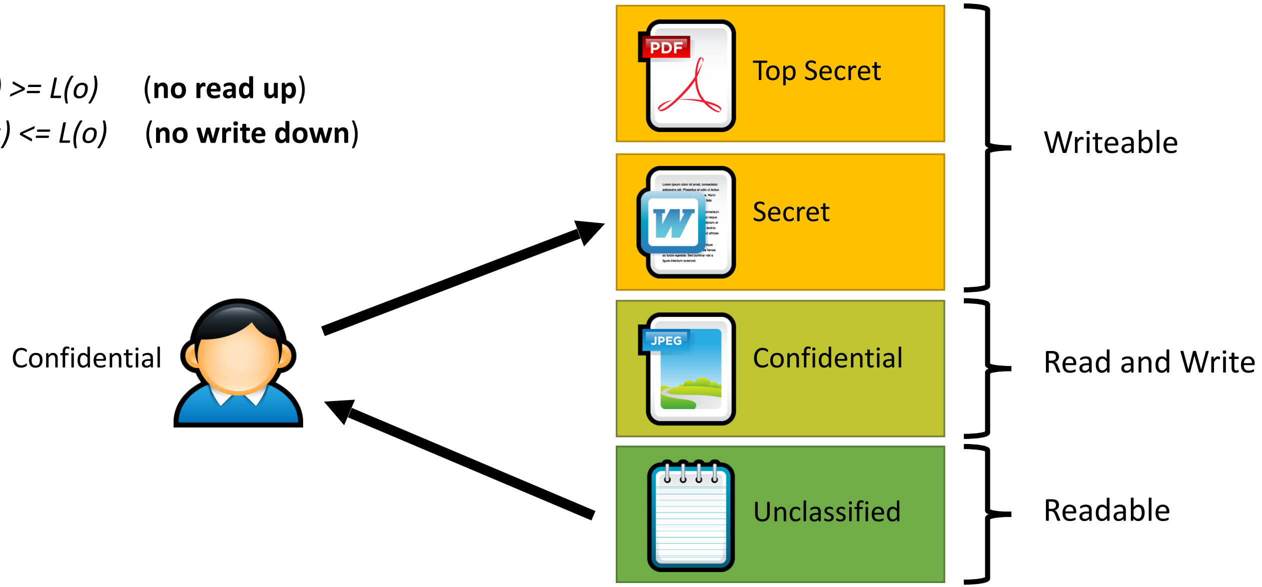
Simplified Bell-LaPadula Example

- Assume $L_m(s) = L_c(s)$ is always true
- ★-property
  - *s* can read *o* iff $L(s) >= L(o)$ (**no read up**)
  - *s* can write *o* iff $L(s) <= L(o)$ (**no write down**)



Simplified Bell-LaPadula Example

- Assume $L_m(s) = L_c(s)$ is always true
- ★-property
  - *s* can read *o* iff $L(s) >= L(o)$ (**no read up**)
  - *s* can write *o* iff $L(s) <= L(o)$ (**no write down**)



BLP Idea

A computer system is in a state, and undergoes state transitions whenever an operation occurs..

System is secure if all transitions satisfy 3 properties:

Simple:

Star:

Discretionary:

BLP Idea

A computer system is in a state, and undergoes state transitions whenever an operation occurs..

System is secure if all transitions satisfy 3 properties:

Simple: S can read O if S has higher clearance

Star: S can write O if S has lower clearance.

Discretionary: Every access allowed by ACL.

Users are trusted

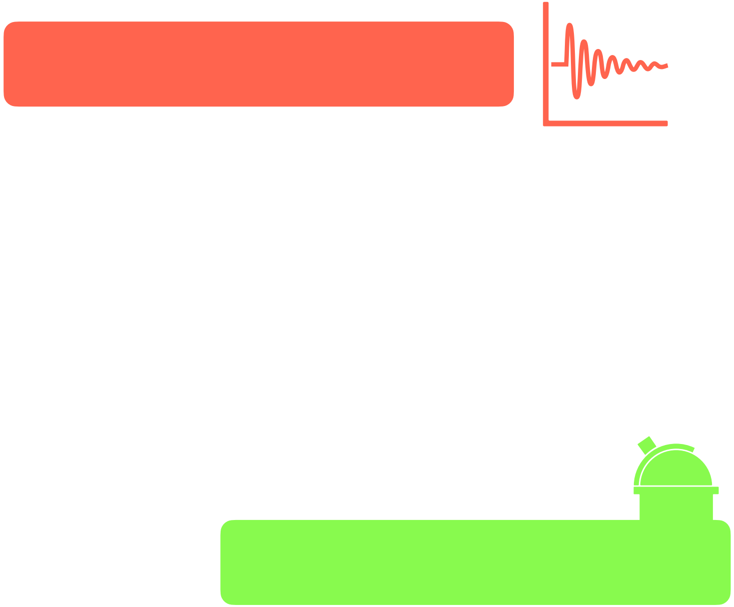
Subjects are not trusted. (Malware)

Not Enough

<u>TopSecret.pdf</u>
rwx User A
--- User B

<u>NotSecret.pdf</u>
rwx User A
rwx User B

Not Enough: Covert channels

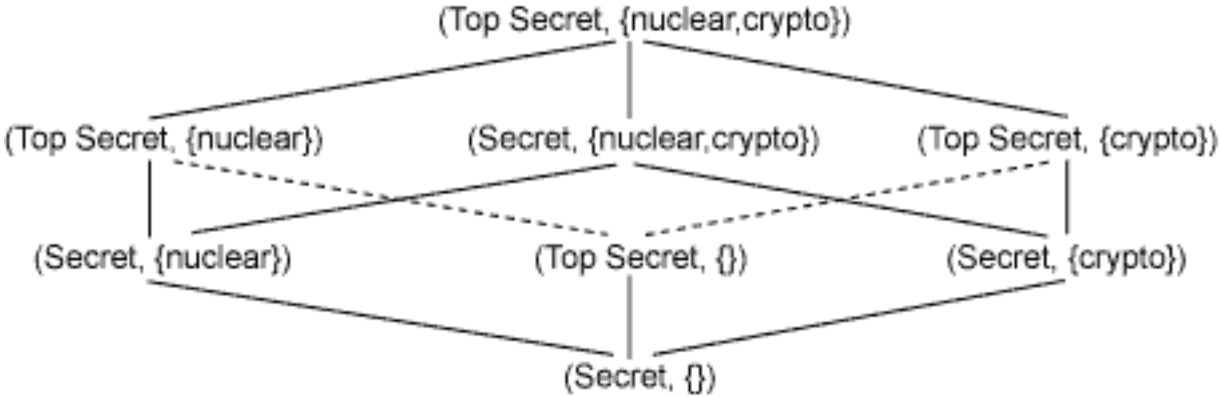


Security Lattice

Compartments:

Ordering between (Level, Compartment)

Lattice



(Top Secret, {nuclear,crypto})
(Top Secret, {nuclear})
(Secret, {nuclear,crypto})
(Top Secret, {crypto})
(Secret, {nuclear})
(Top Secret, {})
(Secret, {crypto})
(Secret, {})

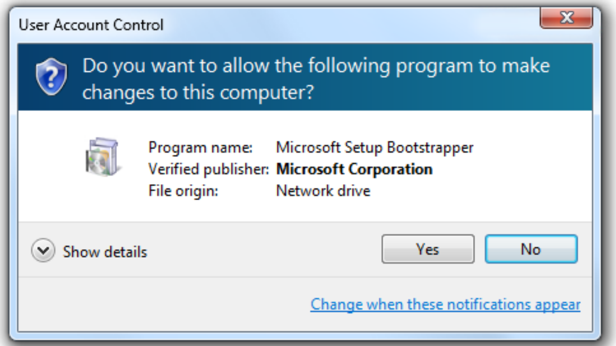
Need-to-Know policy

Integrity Protection in Practice

- Mandatory Integrity Control in Windows
  - Since Vista
  - Four integrity levels: Low, Medium, High, System
  - Each process assigned a level
    - Processes started by normal users are Medium
    - Elevated processes have High
  - Some processes intentionally run as Low
    - Internet Explorer in protected mode
  - Ring policy
    - Reading and writing do not change integrity level

Integrity Protection in Practice

- Mandatory Integrity Control in Windows
  - Since Vista
  - Four integrity levels: Low, Medium, High, System
  - Each process assigned a level
    - Processes started by normal users are Medium
    - Elevated processes have High
  - Some processes intentionally run as Low
    - Internet Explorer in protected mode
  - Ring policy
    - Reading and writing do not change integrity level



Hybrid

SELinux, TrustedBSD: MAC + DAC system

Confidentiality? What else?

Biba Integrity Policy

Biba Integrity Model

- Proposed in 1975
- Like Bell-LaPadula, security model with provable properties based on a state transition model
  - Each subject has an integrity level
  - Each object has an integrity level
  - Integrity levels are totally ordered (high → low)
- Integrity levels in Biba are not the same as security levels in Bell-LaPadula
  - Some high integrity data does not need confidentiality
  - Examples: stock prices, official statements from the president

Possible Mandatory Policies in Biba

1. Strict integrity
   - *s* can read *o* iif *i(s) <= i(o)* (**no read down**)
   - *s* can write *o* iff *i(s) >= i(o)* (**no write up**)

Possible Mandatory Policies in Biba

1. Strict integrity
   - *s* can read *o* iif *i(s) <= i(o)* (**no read down**)
   - *s* can write *o* iff *i(s) >= i(o)* (**no write up**)
2. Subject low-water mark
   - *s* can always read *o*; afterward *i(s) = min(i(s), i(o))* (**subject tainting**)
   - *s* can write *o* iff *i(s) >= i(o)* (**no write up**)

Possible Mandatory Policies in Biba

1. Strict integrity
   - *s* can read *o* iif *i(s) <= i(o)* (**no read down**)
   - *s* can write *o* iff *i(s) >= i(o)* (**no write up**)
2. Subject low-water mark
   - *s* can always read *o*; afterward *i(s) = min(i(s), i(o))* (**subject tainting**)
   - *s* can write *o* iff *i(s) >= i(o)* (**no write up**)
3. Object low-water mark
   - *s* can read *o* iif *i(s) <= i(o)* (**no read down**)
   - *s* can always write *o*; afterward *o(s) = min(i(s), i(o))* (**object tainting**)

Possible Mandatory Policies in Biba

1. Strict integrity
   - *s* can read *o* iif *i(s) <= i(o)* (**no read down**)
   - *s* can write *o* iff *i(s) >= i(o)* (**no write up**)
2. Subject low-water mark
   - *s* can always read *o*; afterward *i(s) = min(i(s), i(o))* (**subject tainting**)
   - *s* can write *o* iff *i(s) >= i(o)* (**no write up**)
3. Object low-water mark
   - *s* can read *o* iif *i(s) <= i(o)* (**no read down**)
   - *s* can always write *o*; afterward *o(s) = min(i(s), i(o))* (**object tainting**)
4. Low-water mark integrity audit
   - *s* can always read *o*; afterward *i(s) = min(i(s), i(o))* (**subject tainting**)
   - *s* can always write *o*; afterward *o(s) = min(i(s), i(o))* (**object tainting**)

Possible Mandatory Policies in Biba

1. Strict integrity
   - *s* can read *o* iif *i(s) <= i(o)* (**no read down**)
   - *s* can write *o* iff *i(s) >= i(o)* (**no write up**)
2. Subject low-water mark
   - *s* can always read *o*; afterward *i(s) = min(i(s), i(o))* (**subject tainting**)
   - *s* can write *o* iff *i(s) >= i(o)* (**no write up**)
3. Object low-water mark
   - *s* can read *o* iif *i(s) <= i(o)* (**no read down**)
   - *s* can always write *o*; afterward *o(s) = min(i(s), i(o))* (**object tainting**)
4. Low-water mark integrity audit
   - *s* can always read *o*; afterward *i(s) = min(i(s), i(o))* (**subject tainting**)
   - *s* can always write *o*; afterward *o(s) = min(i(s), i(o))* (**object tainting**)
5. Ring
   - *s* can read any object *o*
   - *s* can write *o* iff *i(s) >= i(o)* (**no write up**)

Biba Strict Integrity Example

- Strict integrity
  - *s* can read *o* iif *i(s) <= i(o)* **(no read down)**
  - *s* can write *o* iff *i(s) >= i(o)* **(no write up)**

High Integrity

Medium Integrity

Medium Integrity

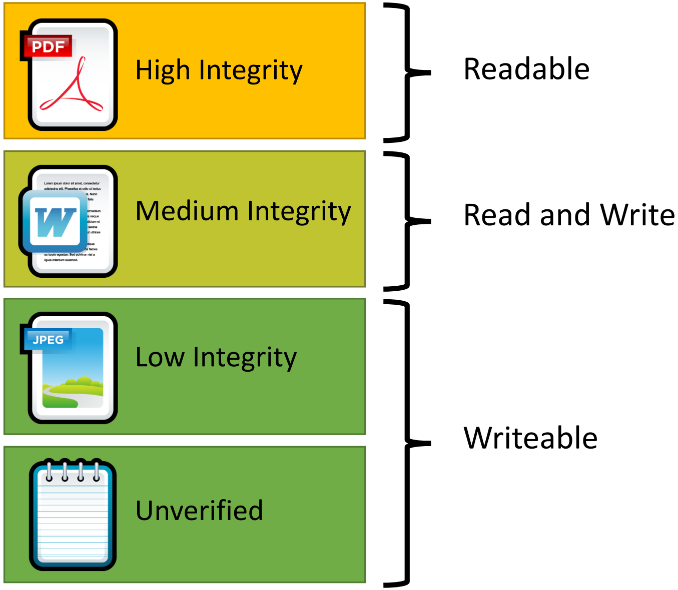
Low Integrity

Unverified

Biba Strict Integrity Example

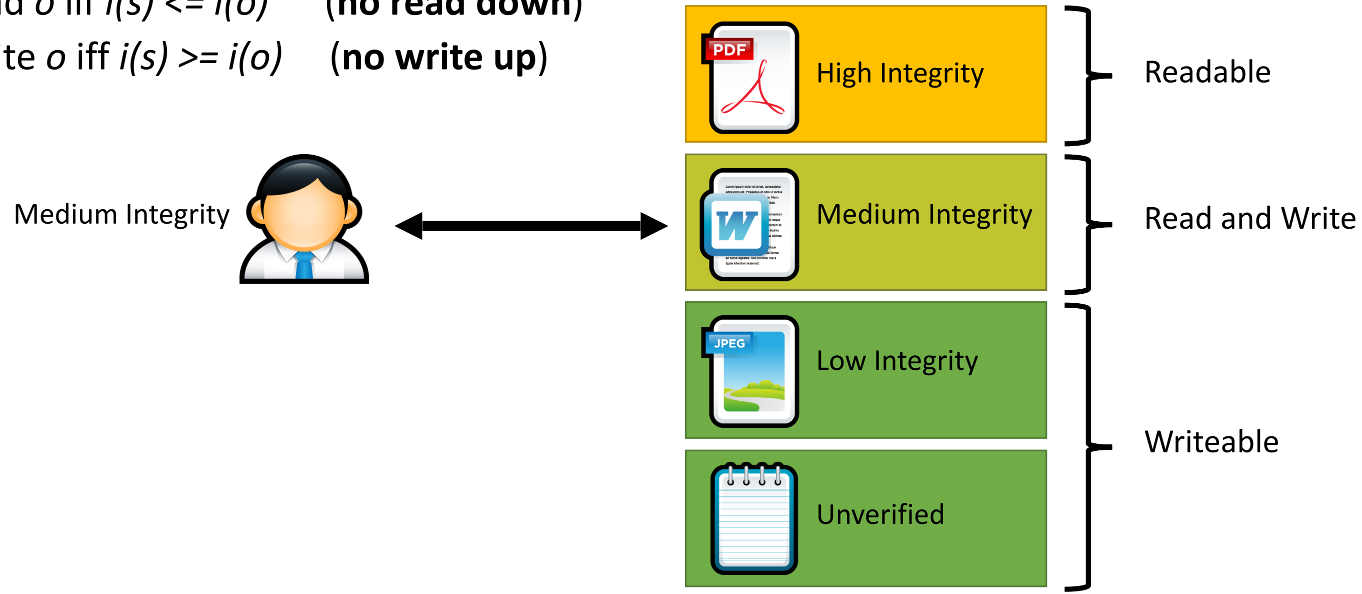
- Strict integrity
  - s can read o iif $i(s) <= i(o)$ **(no read down)**
  - s can write o iff $i(s) >= i(o)$ **(no write up)**

Medium Integrity



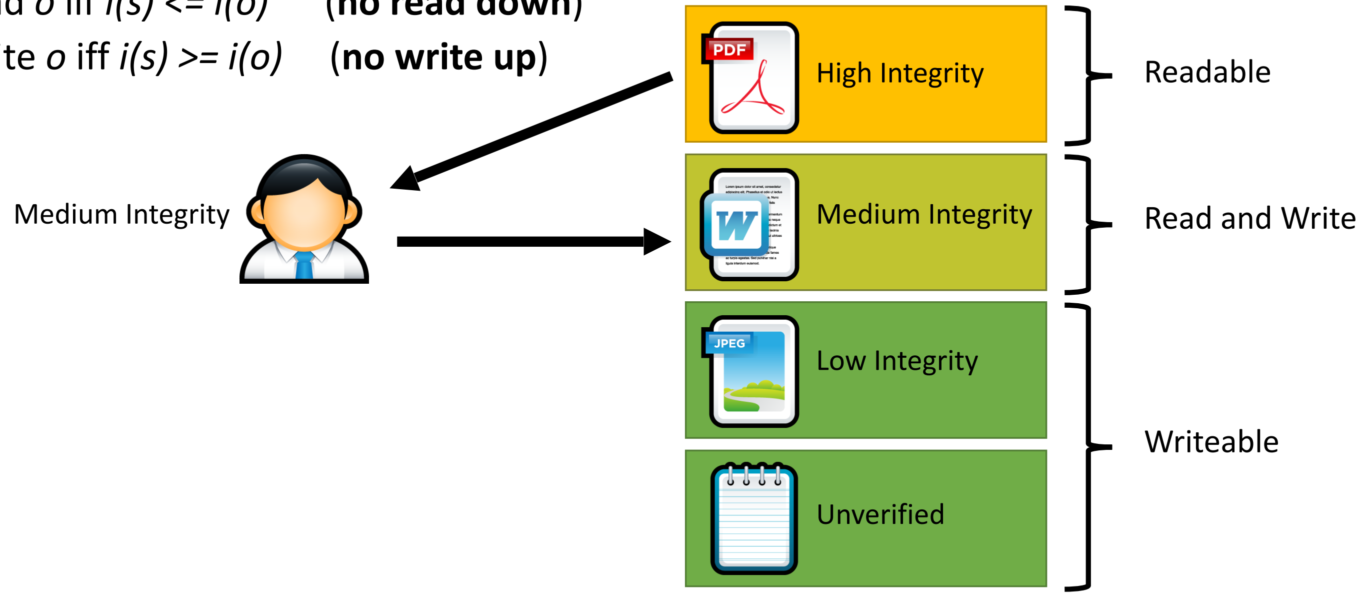
Biba Strict Integrity Example

- Strict integrity
  - *s* can read *o* iif *i(s)* <= *i(o)* **(no read down)**
  - *s* can write *o* iff *i(s)* >= *i(o)* **(no write up)**



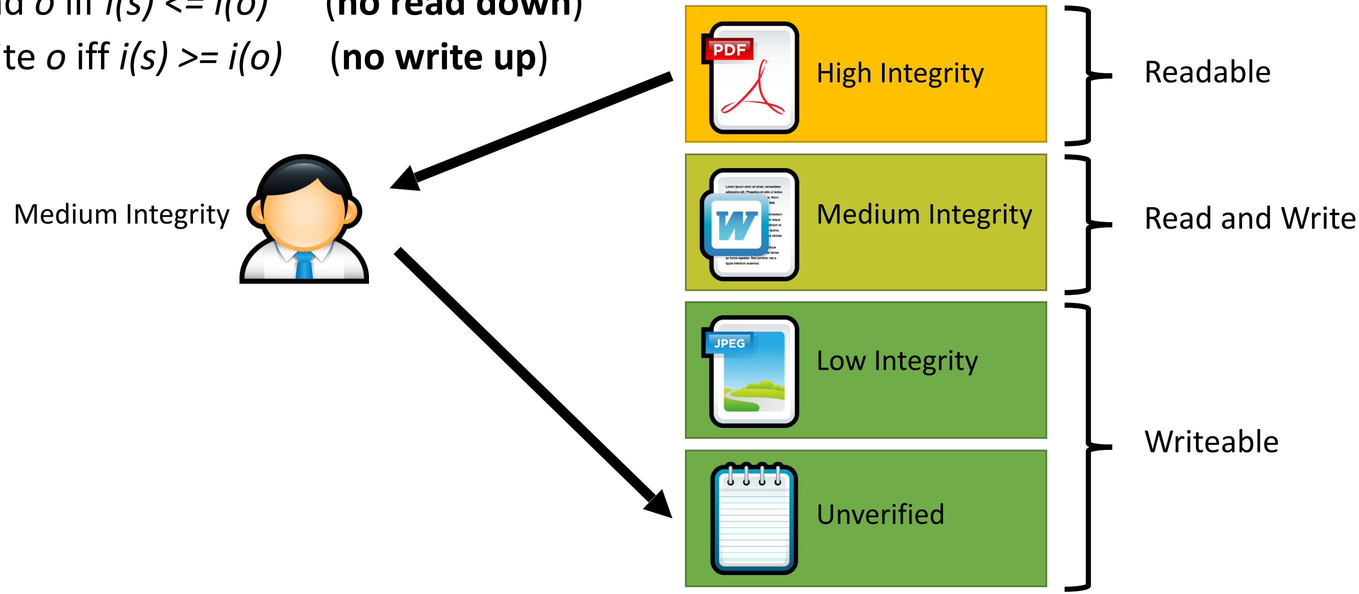
Biba Strict Integrity Example

- Strict integrity
  - *s* can read *o* iif *i(s)* <= *i(o)* **(no read down)**
  - *s* can write *o* iff *i(s)* >= *i(o)* **(no write up)**



Biba Strict Integrity Example

- Strict integrity
  - *s* can read *o* iif *i(s)* <= *i(o)* **(no read down)**
  - *s* can write *o* iff *i(s)* >= *i(o)* **(no write up)**



Practical Example of Biba Integrity

- Military chain of command
  - Generals may issue orders to majors and privates
  - Majors may issue orders to privates, but not generals
  - Privates may only take orders

Comparison

BPL

- Offers confidentiality
- “Read down, write up”
- Focuses on controlling reads
- Theoretically, no requirement that subjects be trusted
  - Even malicious programs can’t leak secrets they don’t know

Biba

Comparison

BPL

- Offers confidentiality
- “Read down, write up”
- Focuses on controlling reads
- Theoretically, no requirement that subjects be trusted
  - Even malicious programs can’t leak secrets they don’t know

Biba

- Offers integrity

Comparison

BPL

- Offers confidentiality
- “Read down, write up”
- Focuses on controlling reads
- Theoretically, no requirement that subjects be trusted
  - Even malicious programs can’t leak secrets they don’t know

Biba

- Offers integrity
- “Read up, write down”

Comparison

BPL

- Offers confidentiality
- "Read down, write up"
- Focuses on controlling reads
- Theoretically, no requirement that subjects be trusted
  - Even malicious programs can't leak secrets they don't know

Biba

- Offers integrity
- "Read up, write down"
- Focuses on controlling writes

Comparison

BPL

- Offers confidentiality
- “Read down, write up”
- Focuses on controlling reads
- Theoretically, no requirement that subjects be trusted
  - Even malicious programs can’t leak secrets they don’t know

Biba

- Offers integrity
- “Read up, write down”
- Focuses on controlling writes
- Subjects must be trusted
  - A malicious program can write bad information

Covert and Side Channels

Caveats of Bell-LaPadula

Caveats of Bell-LaPadula

- ★-property prevents **overt** leakage of information
  - Does not address covert channels

Caveats of Bell-LaPadula

- ★-property prevents **overt** leakage of information
  - Does not address covert channels
- What does this mean?

Covert Channels

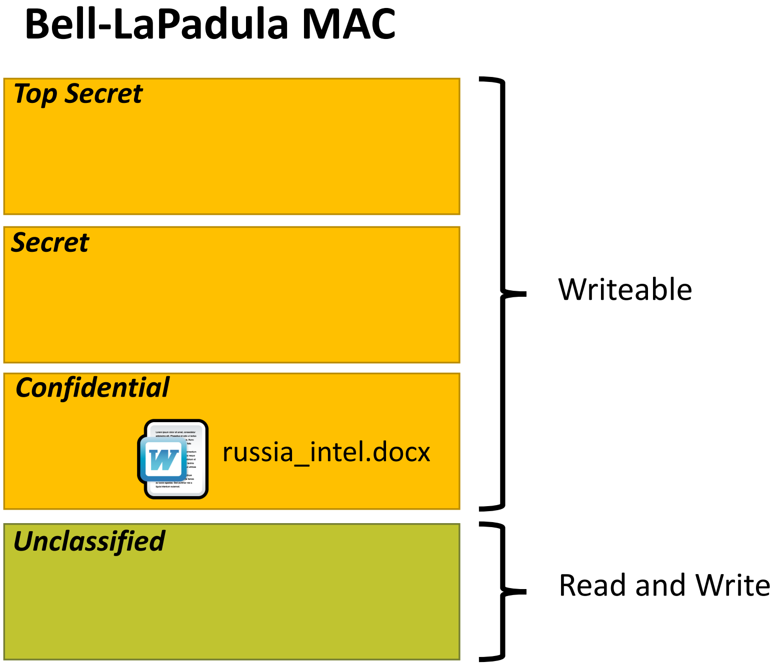
- Access control is defined over “legitimate” channels
  - Read/write an object
  - Send/receive a packet from the network
  - Read/write shared memory
- However, isolation in real systems is imperfect
  - Actions have observable side-effects

Covert Channels

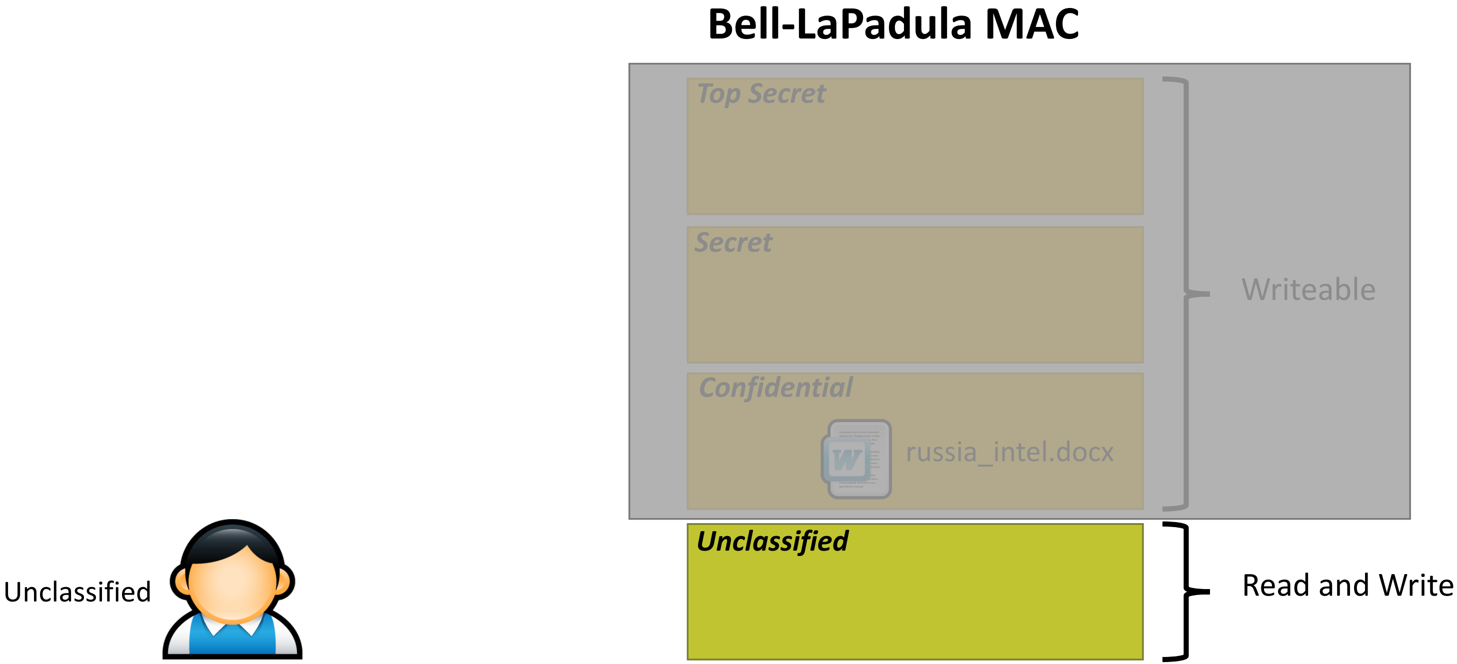
- Access control is defined over “legitimate” channels
  - Read/write an object
  - Send/receive a packet from the network
  - Read/write shared memory
- However, isolation in real systems is imperfect
  - Actions have observable side-effects
- External observations can create covert channels
  - Communication via unintentional channels
  - Examples:
    - Existence of file(s) or locks on file(s)
    - Measure the timing of events
    - CPU cache (e.g. Meltdown and Spectre)

Simple Example

Unclassified

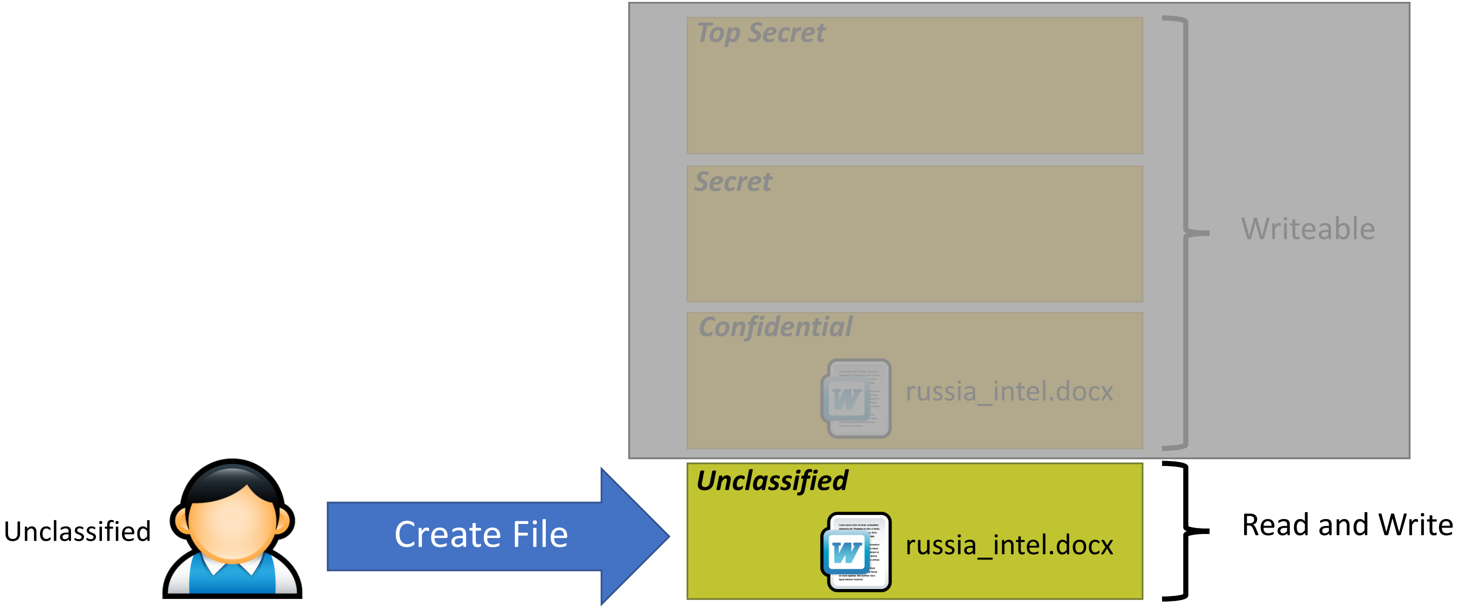


Simple Example



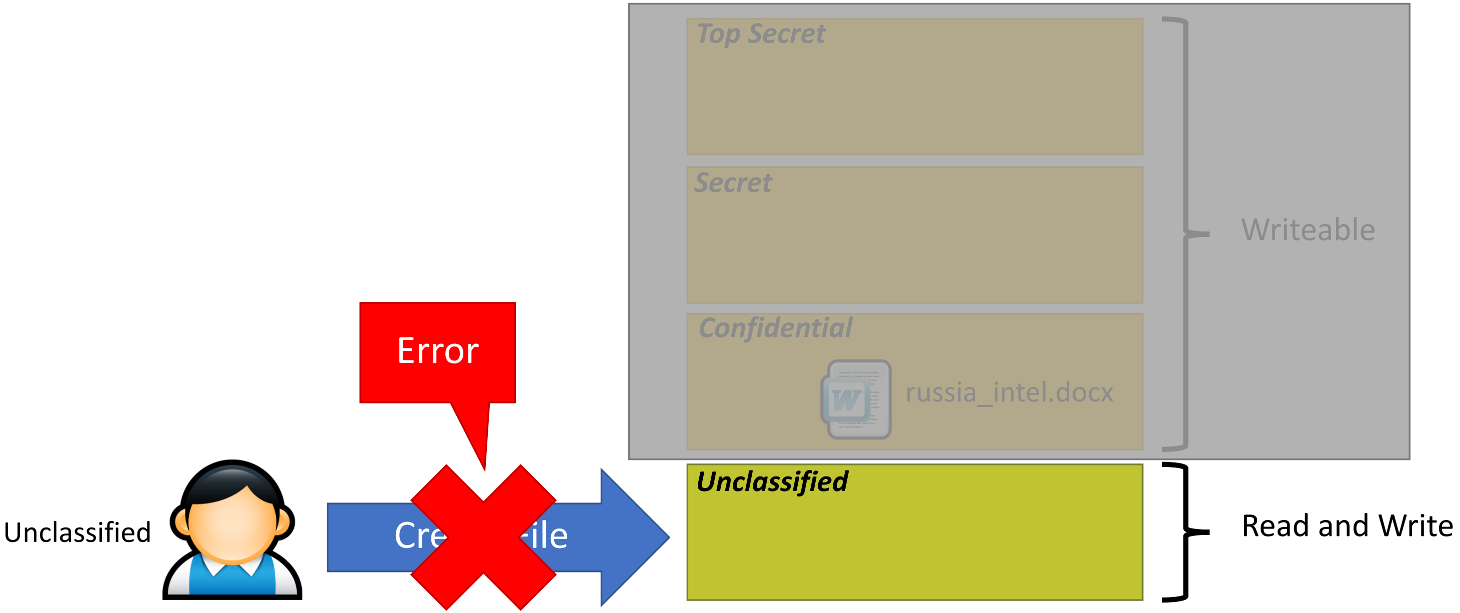
Simple Example

Bell-LaPadula MAC

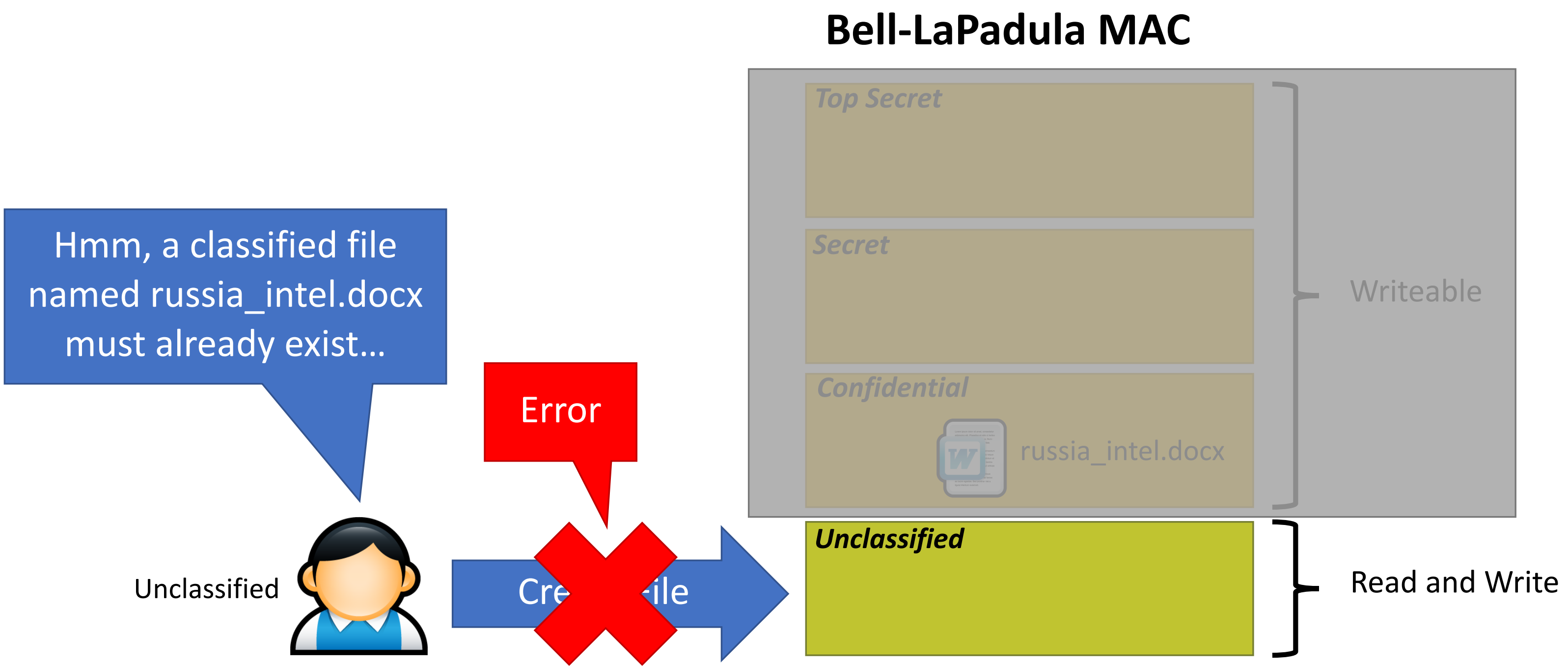


Simple Example

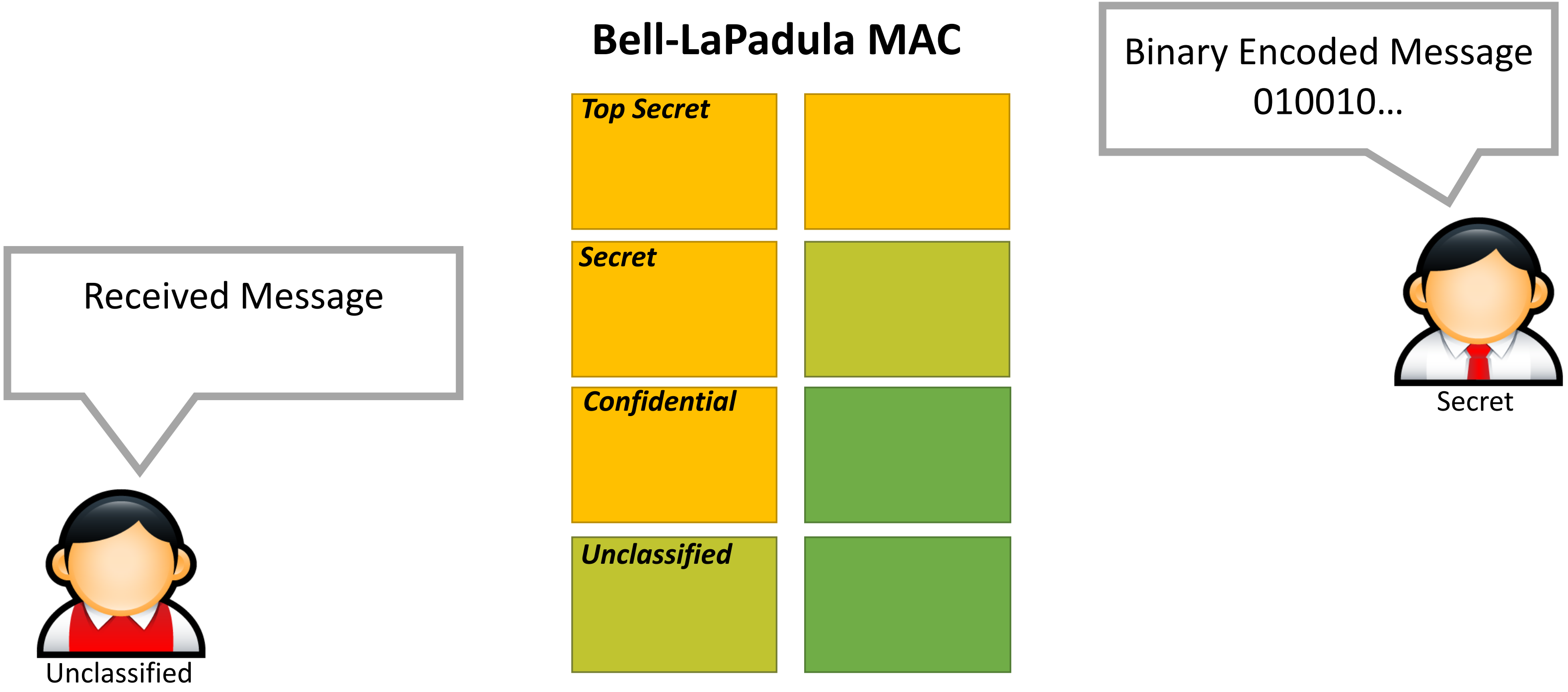
Bell-LaPadula MAC



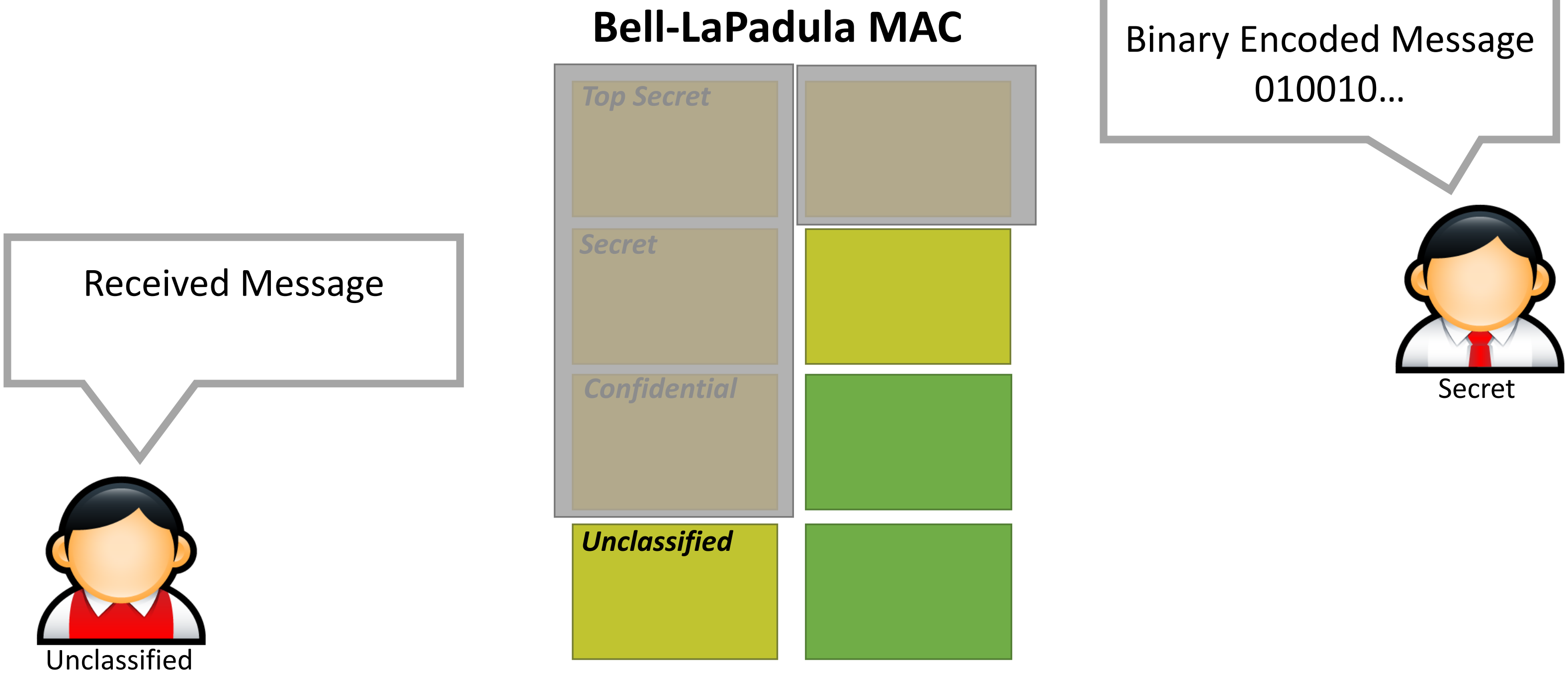
Simple Example



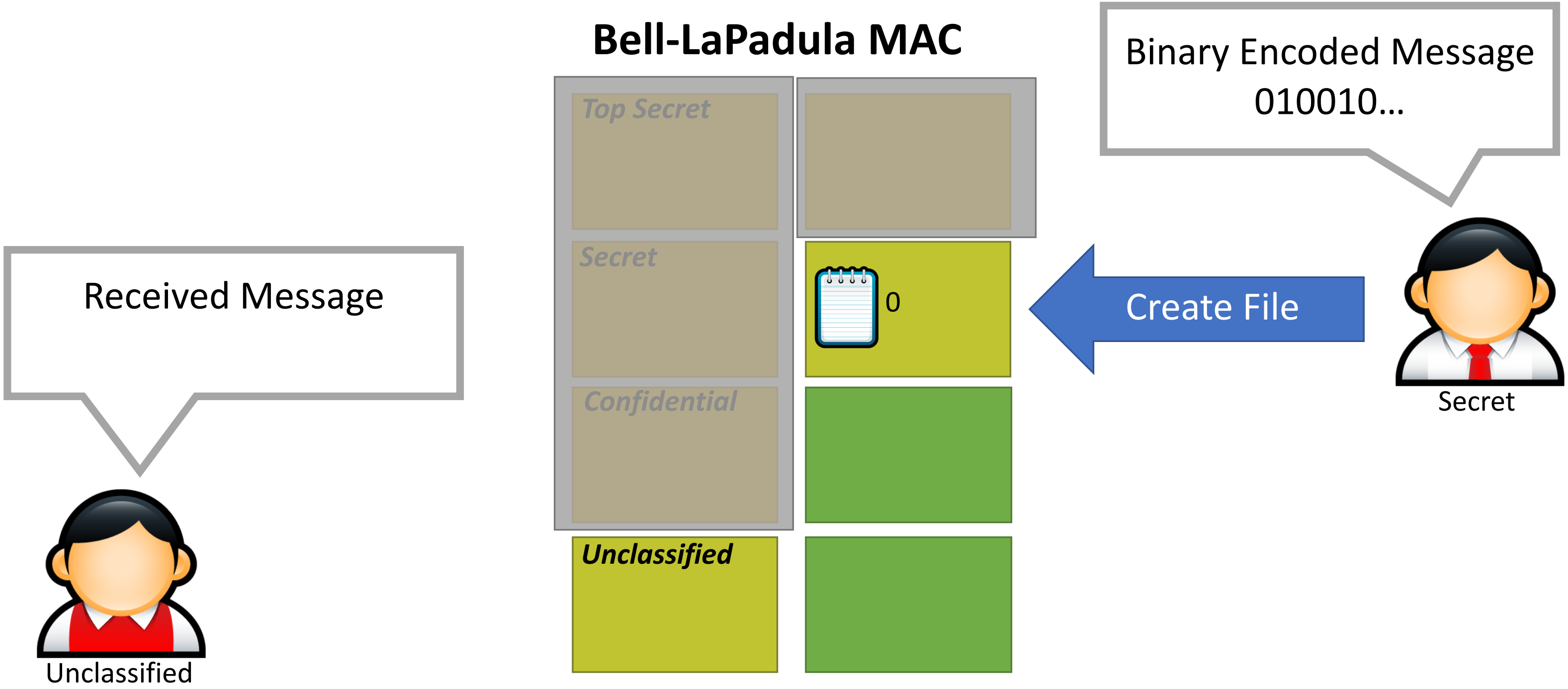
Exploiting a Covert Channel



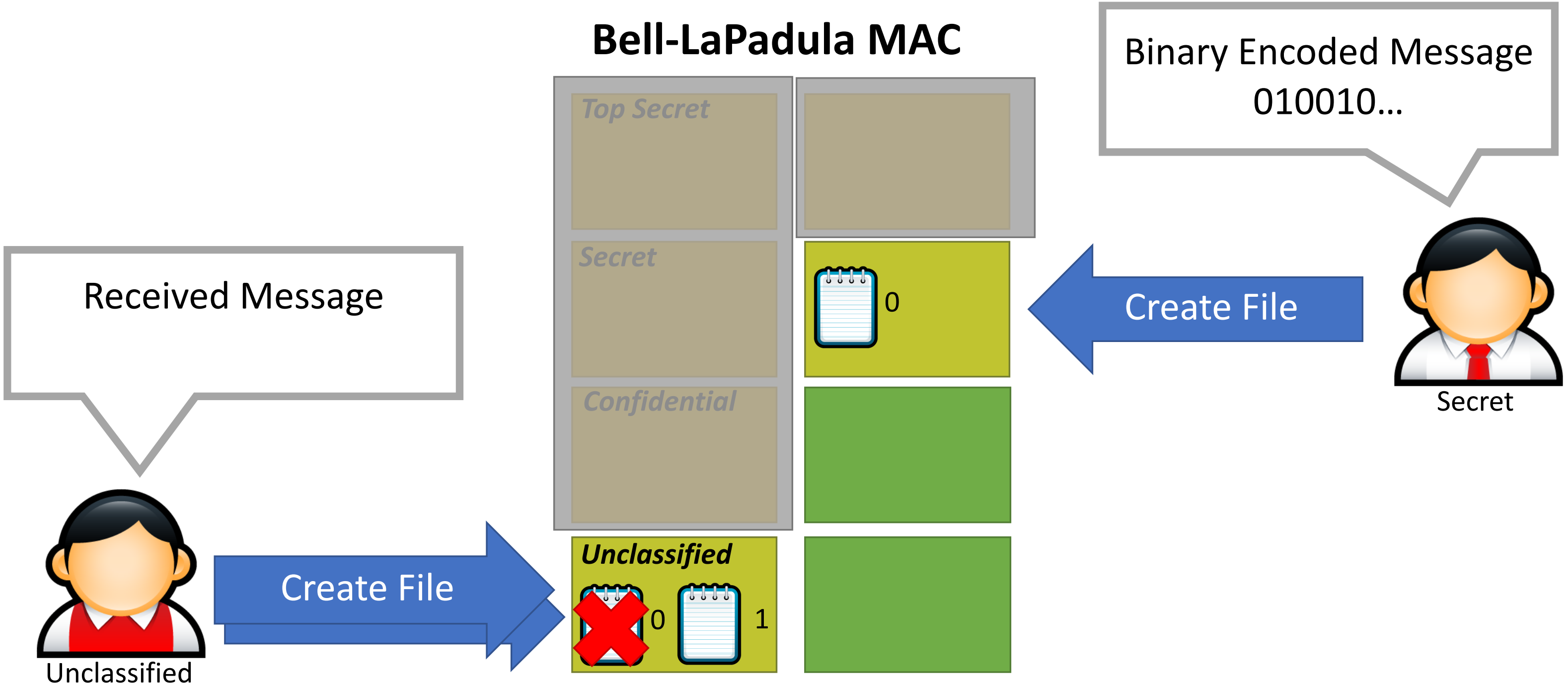
Exploiting a Covert Channel



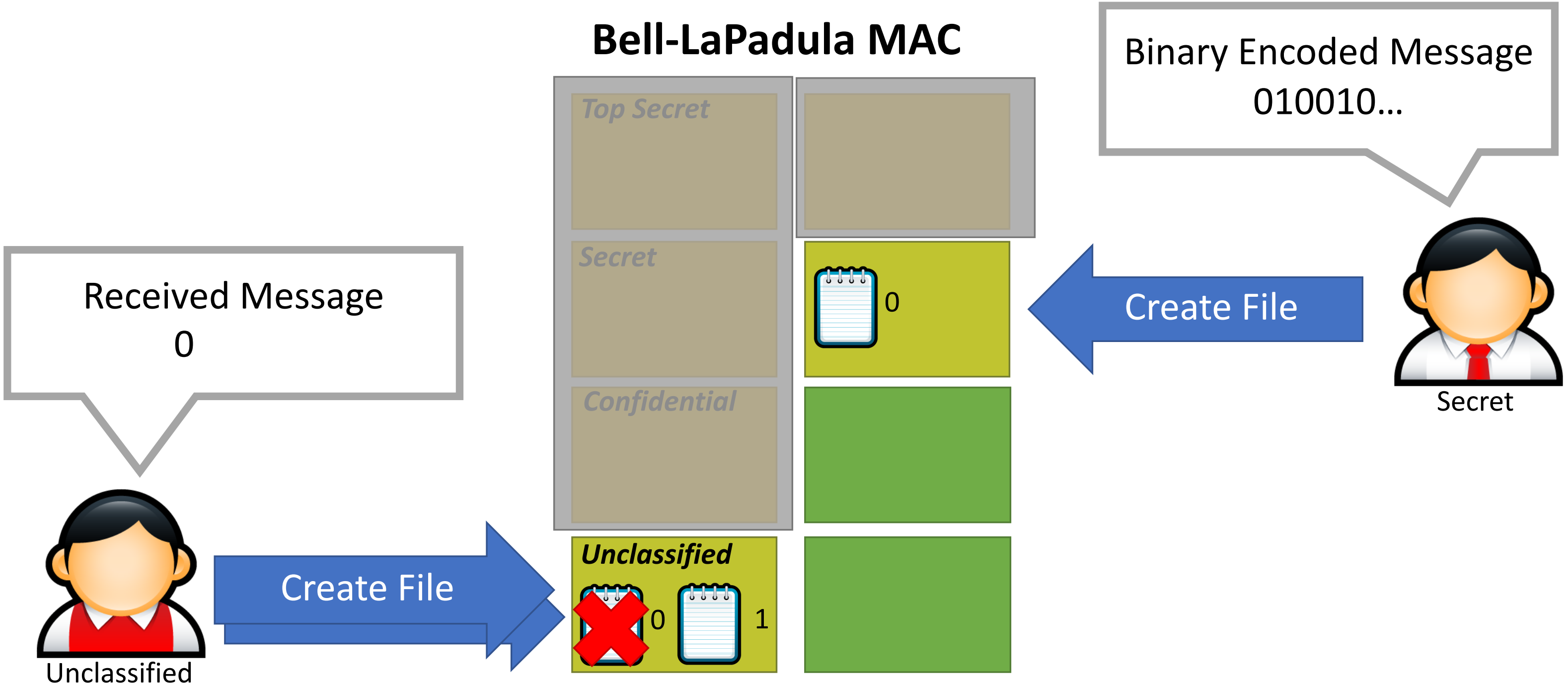
Exploiting a Covert Channel



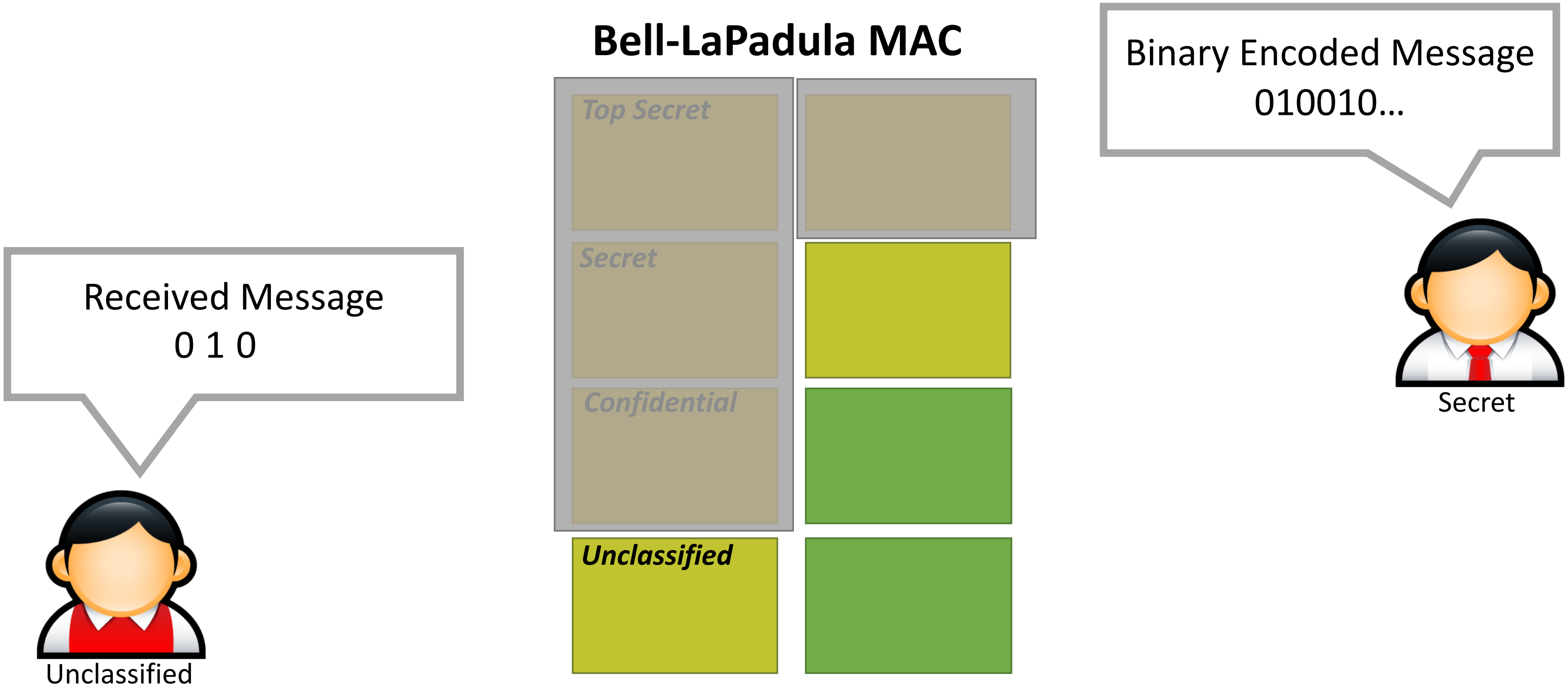
Exploiting a Covert Channel



Exploiting a Covert Channel



Exploiting a Covert Channel



Leveraging Covert Channels

- Covert channels are typically noisy
  - Based on precise timing of events
  - May result in encoding errors, i.e. errors in data transmission
  - Communication is probabilistic
- Information theory and coding theory can be applied to make covert channels more robust
  - Naïve approach: duplicate the data *n* times
  - Better approach: uses Forward Error Correction (FEC) coding
  - Zany approach: use Erasure Coding

Bell-LaPadula and Covert Channels

- Covert channels are not blocked by the ★-property
- It is very hard, perhaps impossible, to block all covert channels
  - May appear in program code
  - Or operating system code
  - Or in the hardware itself (e.g. CPU covert channels)

Bell-LaPadula and Covert Channels

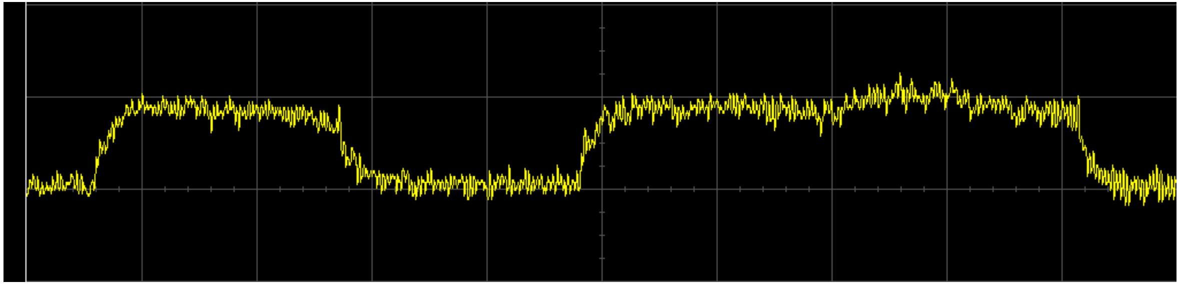
- Covert channels are not blocked by the ★-property
- It is very hard, perhaps impossible, to block all covert channels
  - May appear in program code
  - Or operating system code
  - Or in the hardware itself (e.g. CPU covert channels)
- Potential mitigations:
  - Limit the bandwidth of covert channels by enforcing rate limits
    - Warning: may negatively impact system performance
  - Intentionally make channels noisier by using randomness to introduce “chaff”
    - Warning: slows down attacks, but may not stop them
  - Use anomaly detection to identify subjects using a covert channel
    - Warning: may result in false positives
    - Warning: no guarantee this will detect all covert channels

Side Channel Attacks

- Side channels result from inadvertent information leakage
  - Timing – e.g., password recovery by timing keystrokes
  - Power – e.g., crypto key recovery by power fluctuations
  - RF emissions – e.g., video signal recovery from video cable EM leakage
- Virtually any shared resource can be used

Side Channel Attack Example

- Victim is decrypting RSA data
  - Key is not known to the attacker
  - Encryption process is not directly accessible to the attacker
- Attacker is logged on to the same machine as the victim
  - Secret key can be deciphered by observing the CPU voltage
  - Short peaks = no multiplication (0 bit), long peaks = multiplication (1 bit)



Real Side Channel Attacks

- CPU voltage attacks against RSA
- Keystroke timing attacks against SSH
- Timing and CPU cache attacks against AES
- RF radiation attacks against computer monitors!
  - Attacker can observe what is on your screen
- CPU cache attacks against process isolation
  - Meltdown and Spectre
  - Also leverage a covert channel ;)