

CSE467: Computer Security

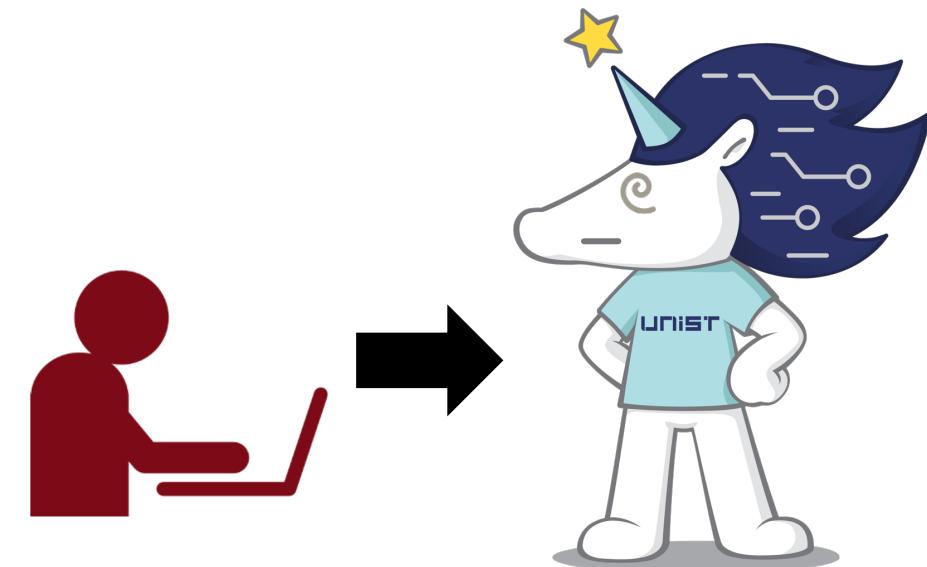
20. Access Control

Seongil Wi

Activity #1: SaveUNIST



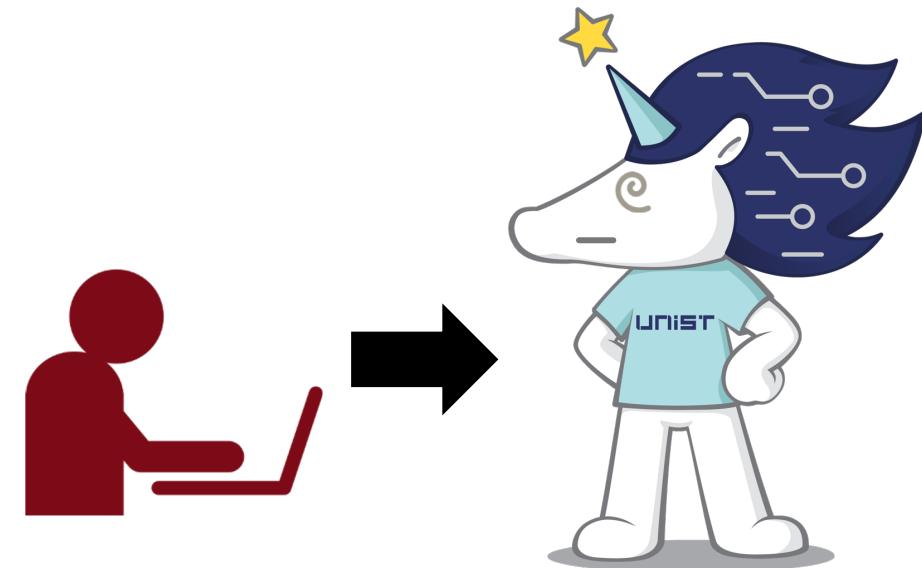
- If you find unknown security problems on campus, report them to me!
- Depending on the severity, bonus points will be given
 - E.g., A+ → 1 letter grade up → 10% score up → ... → 1 drink → ...
- **(IMPORTANT!) DO NOT** try anything illegal
 - If you cannot decide by yourself, discuss it with us first!



Activity #1: SaveUNIST

- Period (confirmed): **Nov 20 ~ Dec 8**
- Time: **9:00 ~ 18:00**
- Target: UNIST IST homepage (<https://ist.unist.ac.kr/>), use a VPN to access from outside

Activity should only
be done during this
period!



Activity #1: SaveUNIST

보안서약서

본인은 2023년 11월 6일 부터 2023년 11월 17일 까지 울산과학기술원 컴퓨터 공학과의 컴퓨터보안 과목에서 과제를 수행함에 있어, 다음과 같이 1)울산과기원의 정보보안 규정과 통제절차 및 2)상급기관 및 유관기관 규정의 보안사항을 준수할 것을 엄숙히 서약 합니다.

1. 나는 상기한 과제를 수행하며, 습득한 IT 정보와 보안관련 사항 등의 기관정보와 관련된 어떠한 정보도 외부 및 타인에게 일체 발설하거나 노출 또는 반출하지 않을 것을 서약한다.
 - 가. 네트워크/시스템/보안 등 IT인프라 관련 구성 일체
 - 나. 시스템 접근권한 정보 일체
 - 다. 기관 내에 있는 모든 개인정보 일체
 - 라. DB정보 일체
 - 마. 정보시스템 취약점 등의 정보 일체
 - 바. 기타 기관 정보보안관련 비밀, 대외비, 누출금지 대상 일체
2. 나는 위의 사항을 위반했거나 기밀을 누설한 경우, 아래의 관계 법규 및 규정에 따라 엄중한 처벌을 받을 것을 서약한다.
 - 가. 「형법」 및 「개인정보보호법」
 - 나. 「정보통신망 이용촉진 및 정보보호 등에 관한 법률」
 - 다. 「울산과학기술원 학칙」
 - 라. 「울산과학기술원 학생 징계 조례」
3. 서약자 연명부

*A pledge not to do
anything illegal*

구분 (idx)	소속 (department)	연락처 (email address)	성명 (name)	서명 (sign)	교수 서명 Prof's sign
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					

SaveUNIST – Instruction



- Submit your found via email:
 - TO: seongil.wi@unist.ac.kr
 - CC: dy3199@unist.ac.kr
 - Title: [SaveUNIST, ID, Name] Title of the vulnerability
 - Content:
 - Bug description
 - Attack step with exploit
 - Provide a screenshot
 - Describe the security impacts that may occur as a result of the attack

HW3 Problem 2



- I've seen some students perform brute force in a manual way.
- Please leverage a script (refer to appendix)
 - Save your time!

Check Duplicate

Sample Username: guest

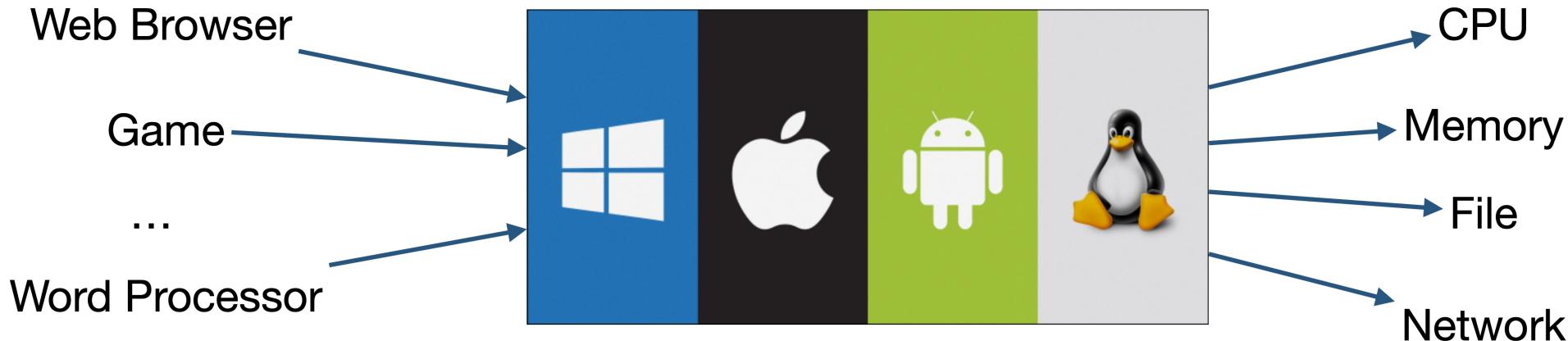
0 user has the name **sadfds**

Username

Check!

Operating System Security

- One of the main goal of OS: resource sharing
 - A lot of requests from multiple users/programs for resources

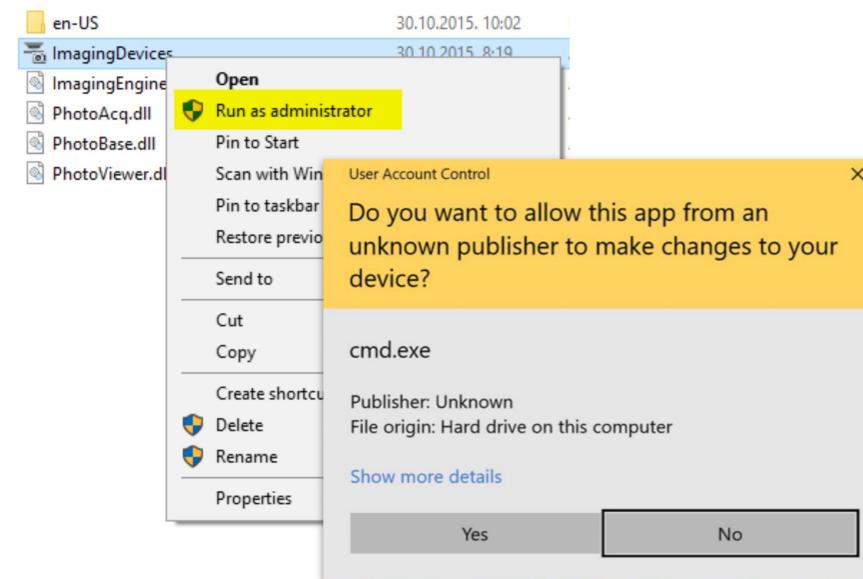
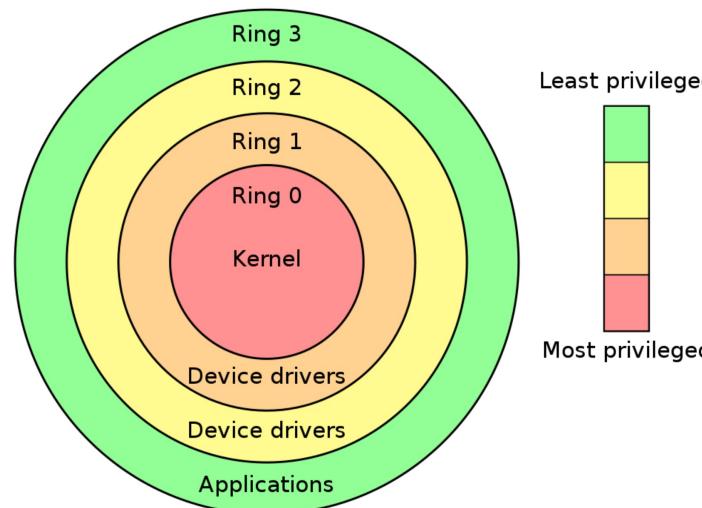


- Important question: how to **securely** share resources?
 - What if someone else abruptly read/write my data/code in memory?
 - What if someone else intentionally change my password?

Principle of Least Privilege

*“Every program and every privileged user of the system should operate using the **least amount of privilege** necessary to complete the job”*

- Jerome Saltzer, *Protection and the Control of Information Sharing in Multics*, CACM, 1973



Access Control



- **Rules** and **policies** that limit access to confidential information
 - Determine what users have permission to do
 - Permission is determined by identity (e.g., name, serial) or role (e.g., professor, TA, student)
-
- Defense against attacks in the first place!
 - Access control is every where: not only OS
 - E.g., hardware, databases, network, etc.



Question!

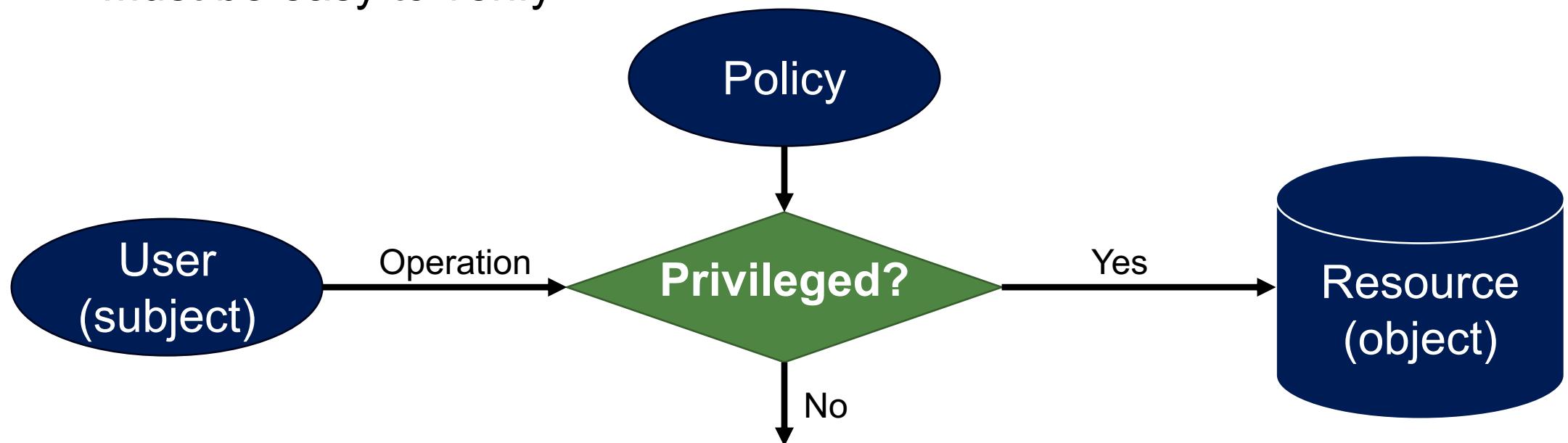


- What topics we covered in class are related to access control?

Access Control Method: Reference Monitor

12

- Check if a **subject (user)** can perform an operation on an **object (resource)**
- Three key properties:
 - Must be always invoked
 - Must be tamper-proof
 - Must be easy to verify



Access Control Policy

13

How does one grant the right level of permission to an individual?

- **Discretionary access control (DAC)**
 - All objects have owners
 - What permission to grant others? **Owners** can decide
 - E.g., Unix, Windows, etc.
- **Mandatory access control (MAC)**
 - What permission to grant others? **Only the admin** can decide
 - Users cannot change policy themselves
 - E.g., SELinux, military, etc.

Access Control Policy

14

How does one grant the right level of permission to an individual?

- **Discretionary access control (DAC)**
 - All objects have owners
 - What permission to grant others? **Owners** can decide
 - E.g., Unix, Windows, etc.
- **Mandatory access control (MAC)**
 - What permission to grant others? **Only the admin** can decide
 - Users cannot change policy themselves
 - E.g., SELinux, military, etc.

Access Control Matrix



- Relationship among **subjects** (users), **objects** (resources), and permissions

	/etc/passwd	/usr/bin/	/home/prof/exam/	/home/admin/
root	rw	rwx	rwx	rwx
professor	r	rx	rwx	-
ta	r	rx	r	-
student1	r	rx	-	-
student2	r	rx	-	-

Access Control Matrix

- Relationship among **subjects** (users), **objects** (resources), and permissions

Object

	<i>/etc/passwd</i>	<i>/usr/bin/</i>	<i>/home/prof/exam/</i>	<i>/home/admin/</i>
root	r w	r w x	r w x	r w x
professor	r	r x	r w x	-
ta	r	r x	r	-
student1	r	r x	-	-
student2	r	r x	-	-

Subject

**Permission,
Access Control Entry (ACE)**

Problems?

Problem #1: Matrix Size



- Relationship among **subjects** (users), **objects** (resources), and permissions

	/etc/passwd	/usr/bin/	/home/prof/exam/	/home/admin/→
root	rw	rwx	rwx	rwx
professor	r	rx	rwx	-
ta				-
student1				-
student2				-

As the number of subjects and objects increases, a large size of memory is required



Problems?

Problem #2: Useless Space

- Relationship among **subjects** (users), **objects** (resources), and permissions

	/etc/passwd	/usr/bin/	/home/prof/exam/	/home/admin/
root	rw	rwx	rwx	rwx
professor	r	rx	rwx	-
ta	r	rx	r	-
student1	r	rx	-	-
student2	r	rx	-	-

Memory space
is wasted

Solution

- Relationship among **subjects** (users), **objects** (resources), and permissions

	/etc/passwd	/usr/bin/	/home/prof/exam/	/home/admin/
root	rw	rwx	rwx	rwx
professor	r	rx	rwx	-
ta	r	rx		-
student1	r	rx		
student2	r	rx		

We need to
manage metrics by
row or column

Two Approaches for DAC

20

- Relationship among **subjects** (users), **objects** (resources), and permissions

	/etc/passwd	/usr/bin/	/home/prof/exam/	/home/admin/
root	rw	rwx	rwx	rwx
professor	r	rx	rwx	-
ta	r	rx	r	-
student1	r	rx	-	-
student2	r	rx	-	-

 Object-oriented approach: Access Control List (ACL)
Subject-oriented approach: Capability

Access Control List (ACL)

- Object-centered approach
- Widely used in many operating systems as a basic access control method
 - E.g., Unix file system
- Approve requests if the subject has privilege to perform the operation on the object

	/etc/passwd
root	rw
professor	r
ta	r
student1	r
student2	r

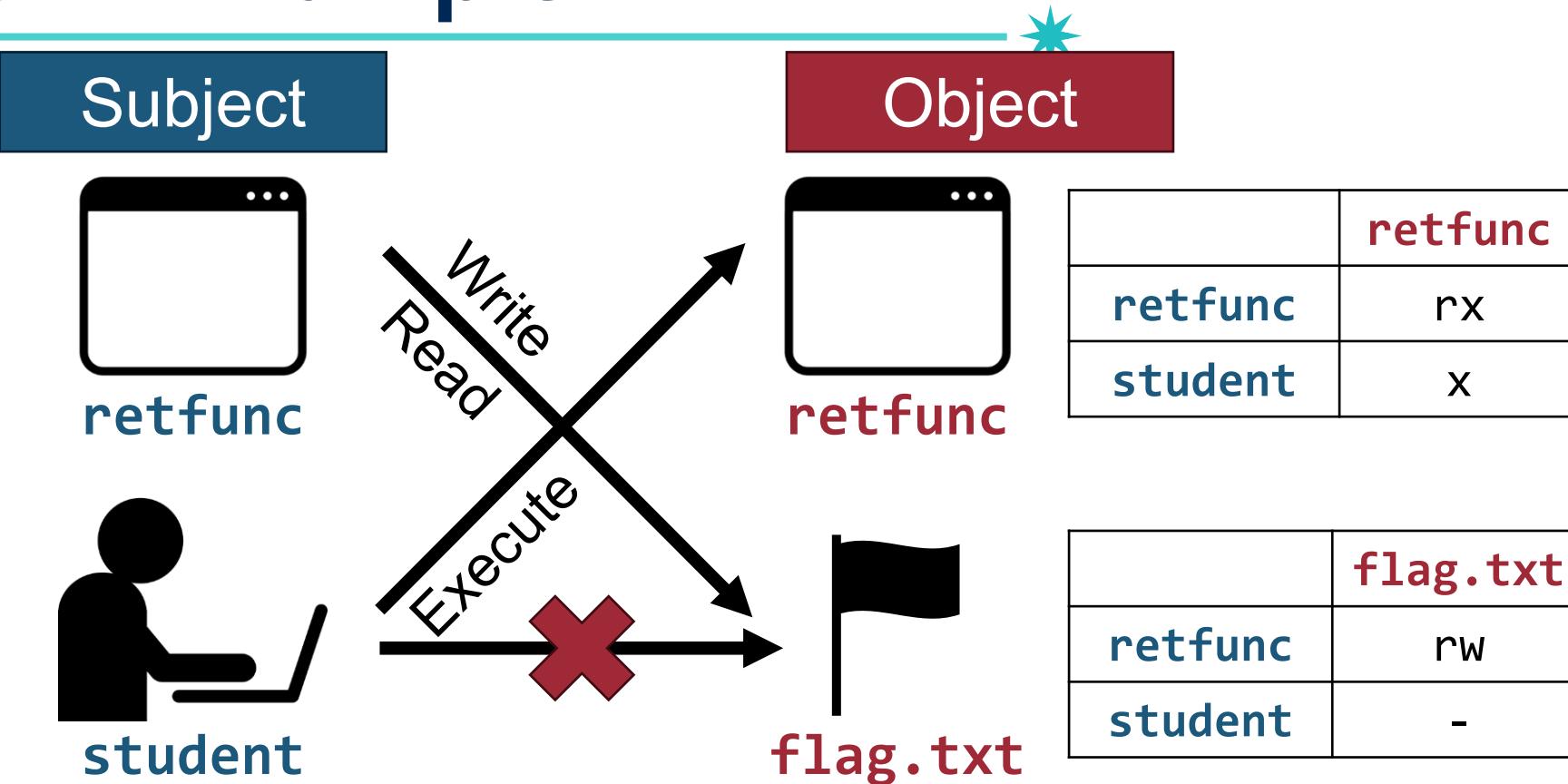
	/usr/bin/
root	rwx
professor	rx
ta	rx
student1	rx
student2	rx

	/home/prof/exam/
root	rwx
professor	rwx
ta	rwx
student1	rwx
student2	rwx

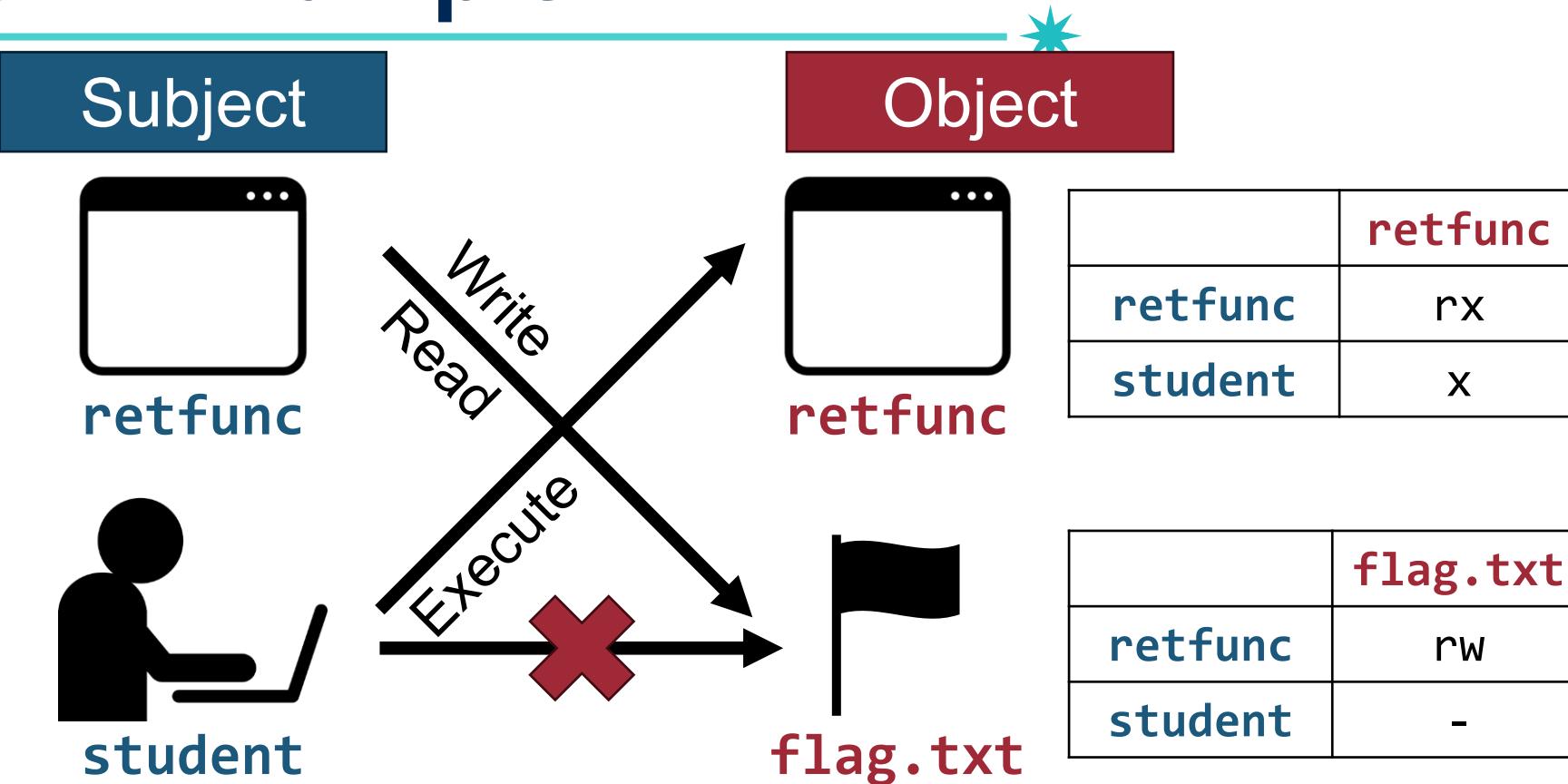
	/home/admin/
root	rwx

ACL – Example

22

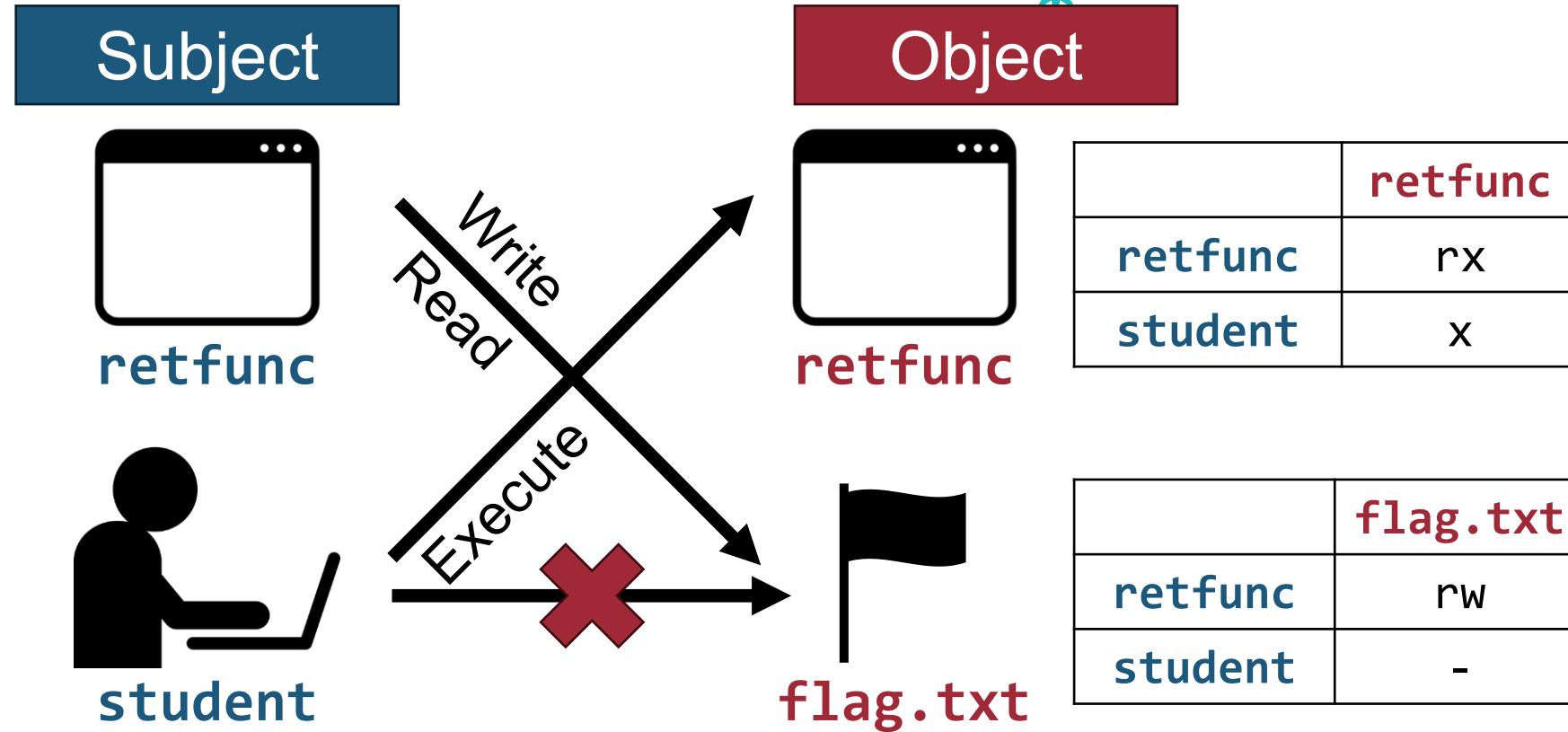


ACL – Example

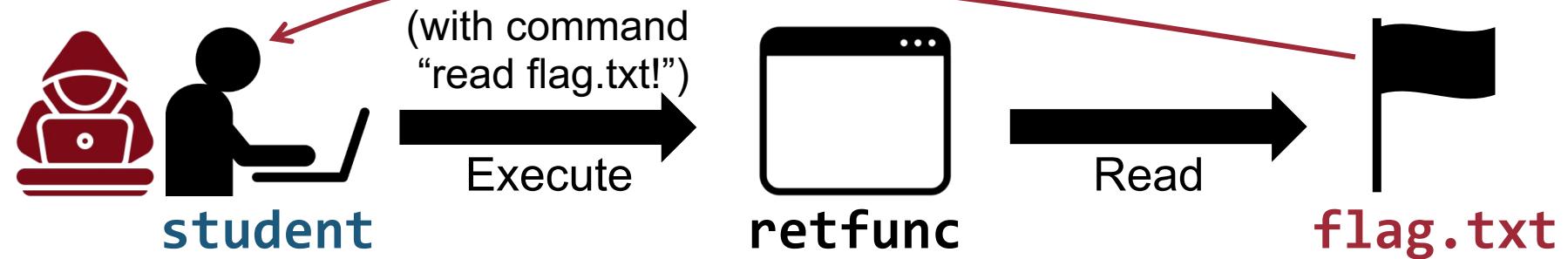


Problem?

ACL Limitation: Confused Deputy Problem²⁴

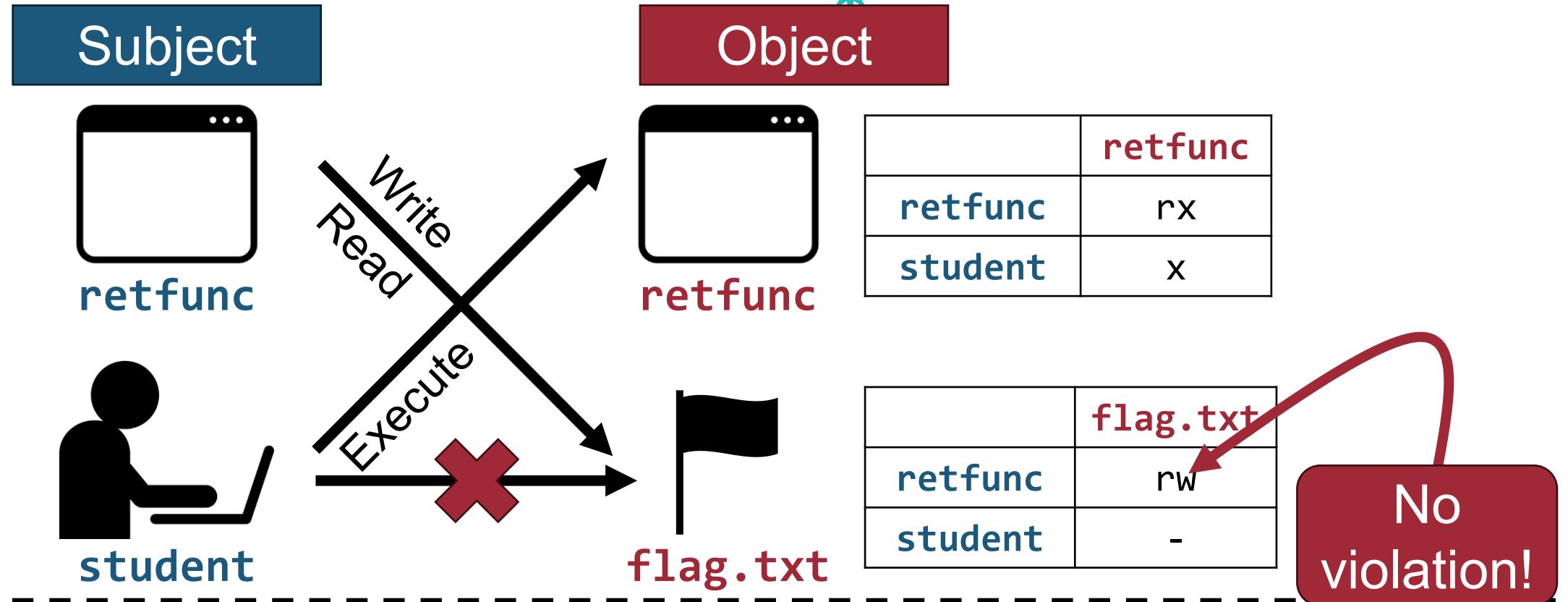


Confused
Deputy
Problem

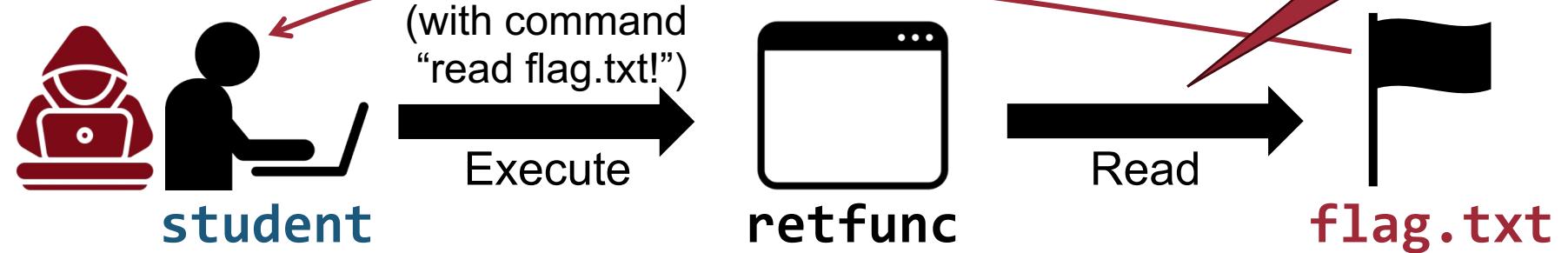


ACL Limitation: Confused Deputy Problem

25



Confused
Deputy
Problem



Confused Deputy Problem: Real-World Example

26

Confused deputy: How did the vulnerability affect Slack?

A major SAML vulnerability was found in Slack that granted expired login credentials permission into the system. Matt Pascucci explains how this 'confused deputy' problem was handled.

Matthew Pascucci



Security researchers found a major SAML vulnerability in Slack's implementation that led to what's called a **confused deputy** issue. How does the SAML vulnerability work, and what is the confused deputy problem?



AOSP > Secure > Bulletins



Android Security Bulletin—January 2021

Published January 4, 2021 / Updated January 7, 2021

The Android Security Bulletin contains details of security vulnerabilities affecting Android devices. Security patch levels of 2021-01-05 or later address all of these issues. To learn how to check a device's security patch level, see [Check and update your Android version](#).

Android partners are notified of all issues at least a month before publication. Source code patches for these issues have been released to the Android Open Source Project (AOSP) repository and linked from this bulletin. This bulletin also includes links to patches outside of AOSP.

The most severe of these issues is a critical security vulnerability in the System component that could enable a remote attacker using a specially crafted transmission to execute arbitrary code within the context of a privileged process. The [severity assessment](#) is based on the effect that exploiting the vulnerability would possibly have on an affected device, assuming the platform and service mitigations are turned off for development purposes or if successfully bypassed.



SektionEins About SektionEins Services News Contact



OS X 10.10 DYLD_PRINT_TO_FILE Local Privilege Escalation Vulnerability

Stefan Esser — 2015-07-07 17:30

The DYLD_PRINT_TO_FILE environment variable can be used for local privilege escalation in OS X Yosemite.

Introduction

With the release of OS X 10.10 Apple added some new features to the dynamic linker dyld. One of these features is the new environment variable `DYLD_PRINT_TO_FILE` that enables error logging to an arbitrary file.

`DYLD_PRINT_TO_FILE`

This is a path to a (writable) file. Normally, the dynamic linker writes all logging output (triggered by `DYLD_PRINT_*` settings) to file descriptor 2 (which is usually stderr). But this setting causes the dynamic linker to write logging output to the specified file.

When this variable was added the usual safeguards that are required when adding support for new environment variables to the dynamic linker have not been used. Therefore it is possible to use this new feature even with SUID root binaries. This is dangerous, because it allows to open or create arbitrary files owned by the root user anywhere in the file system. Furthermore the opened log file is never closed and therefore its file descriptor is leaked into processes spawned by SUID binaries. This means child processes of SUID root processes can write to arbitrary files owned by the root user anywhere in the filesystem. This allows for easy privilege escalation in OS X 10.10.x.

Confused Deputy Problem: Root Cause

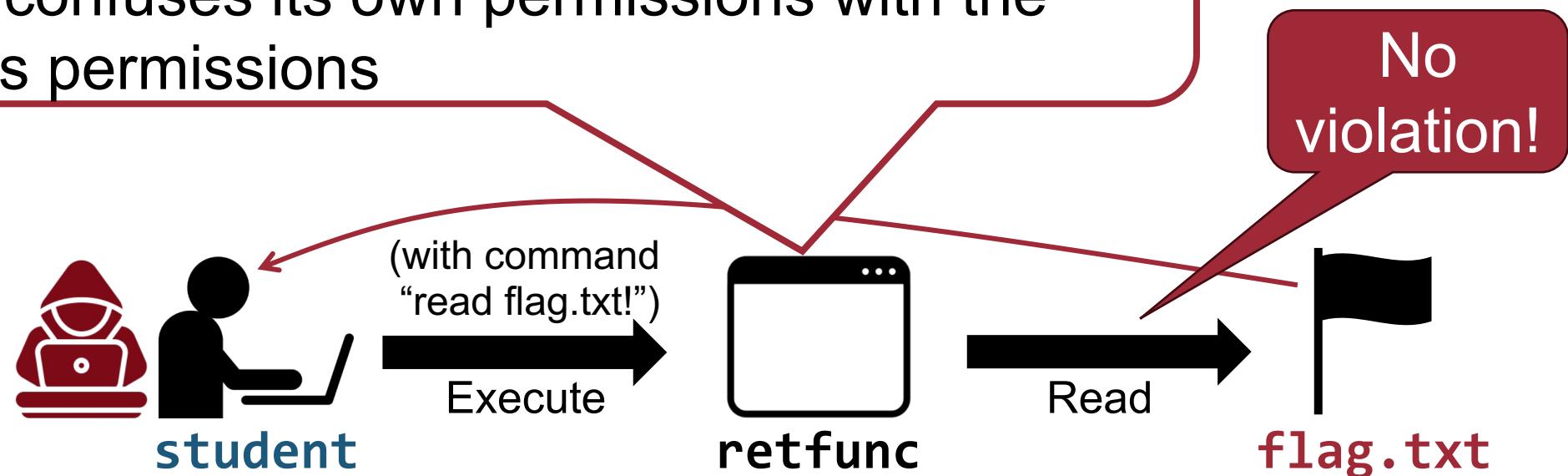
27

- A subject that is tricked by another subject (with fewer privileges) into misusing its authority
- It is a specific type of privilege escalation

Root cause: the deputy (`retfunc`) got confused

- **student** is not permitted to read the **flag.txt** file
- **retfunc** confuses its own permissions with the **student's** permissions

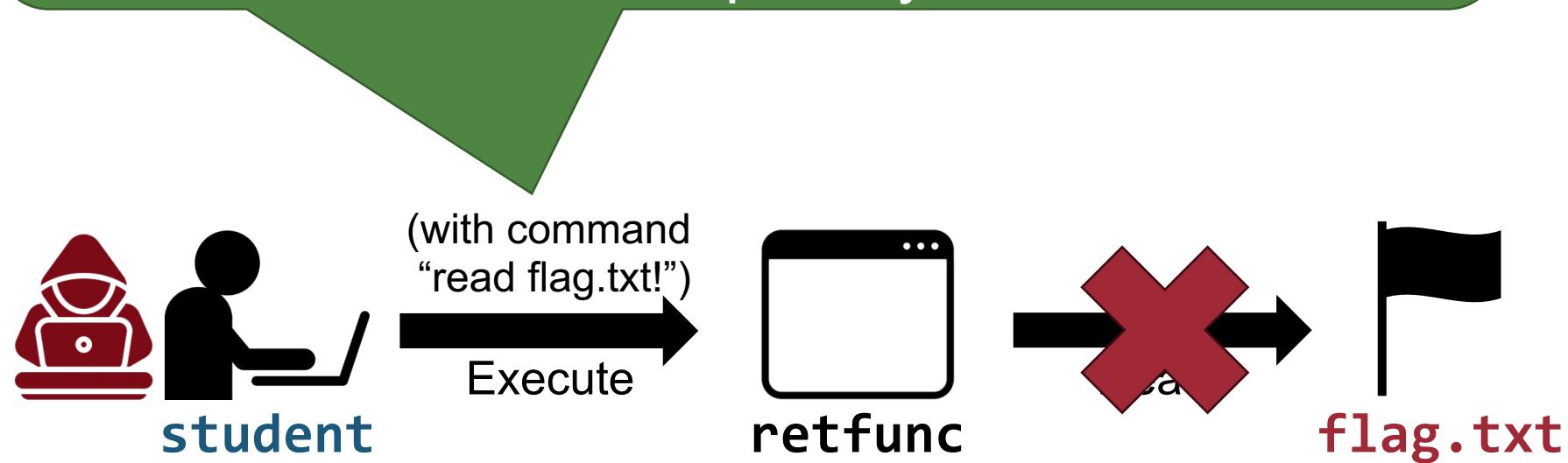
Confused
Deputy
Problem



Simple Solution

28

Delegate authority:
Student should pass both the object (flag.txt)
it wants to read, and a permission token (-)
⇒ Capability



Two Approaches for DAC

- Relationship among **subjects** (users), **objects** (resources), and permissions

	/etc/passwd	/usr/bin/	/home/prof/exam/	/home/admin/
root	rw	rwx	rwx	rwx
professor	r	rx	rwx	-
ta	r	rx	r	-
student1	r	rx	-	-
student2	r	rx	-	-

 Object-oriented approach: Access Control List (ACL)
Subject-oriented approach: Capability

Capability

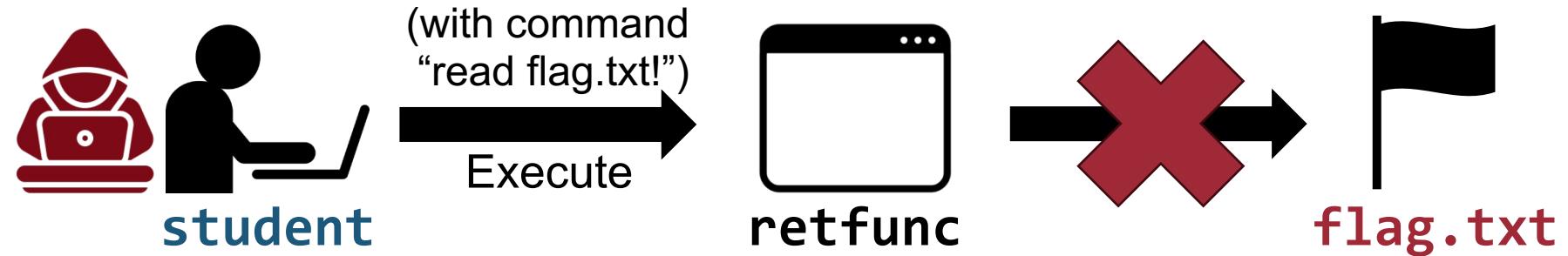
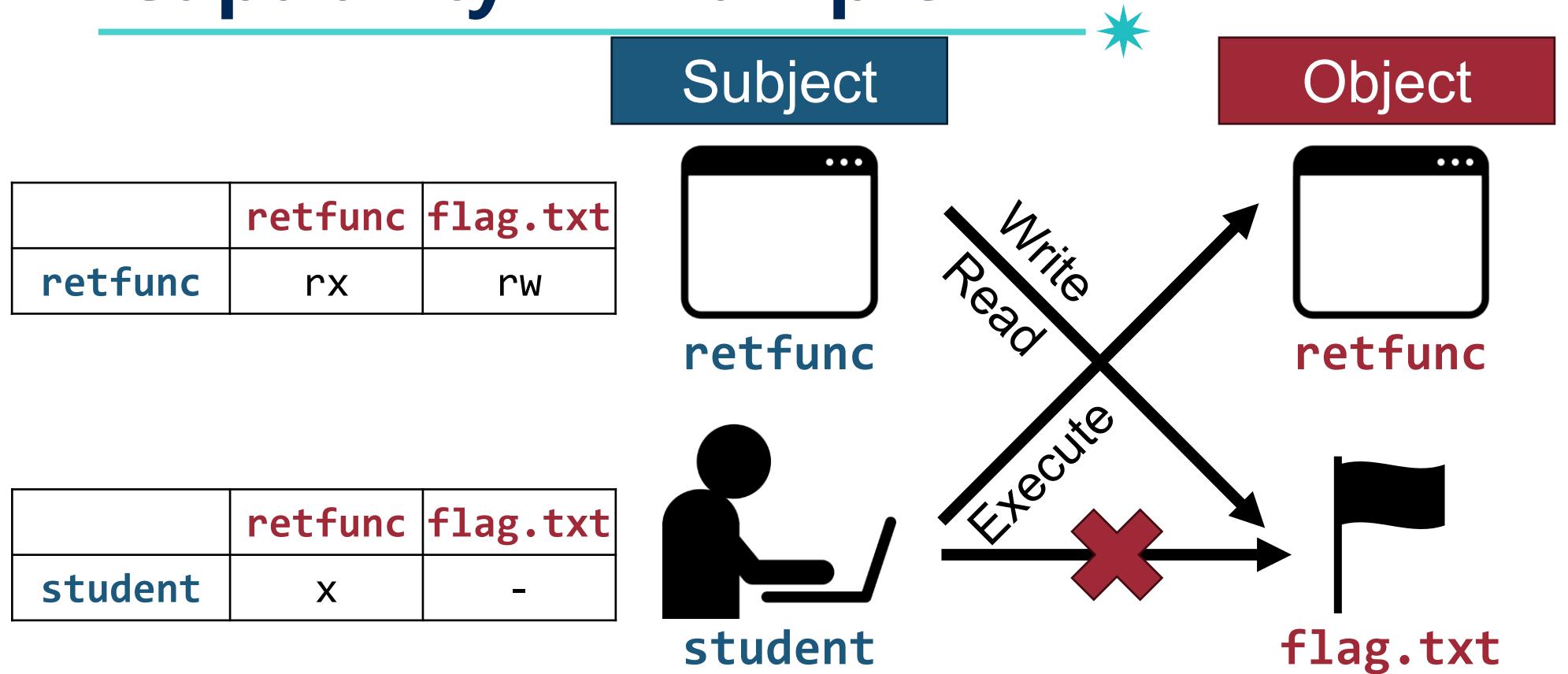


- Subject-centered approach
- **Capability**: a pair of an **object** and a set of privileges
- A deputy (**retfunc**) could ask its requester (**subject**) to provide a capability and use it to request resources
- Mostly used for more secure operating systems (E.g., KeyKOS)
 - Also partly used in main stream OS such as Linux

	retfunc	flag.txt
student	x	-
retfunc	rx	rw

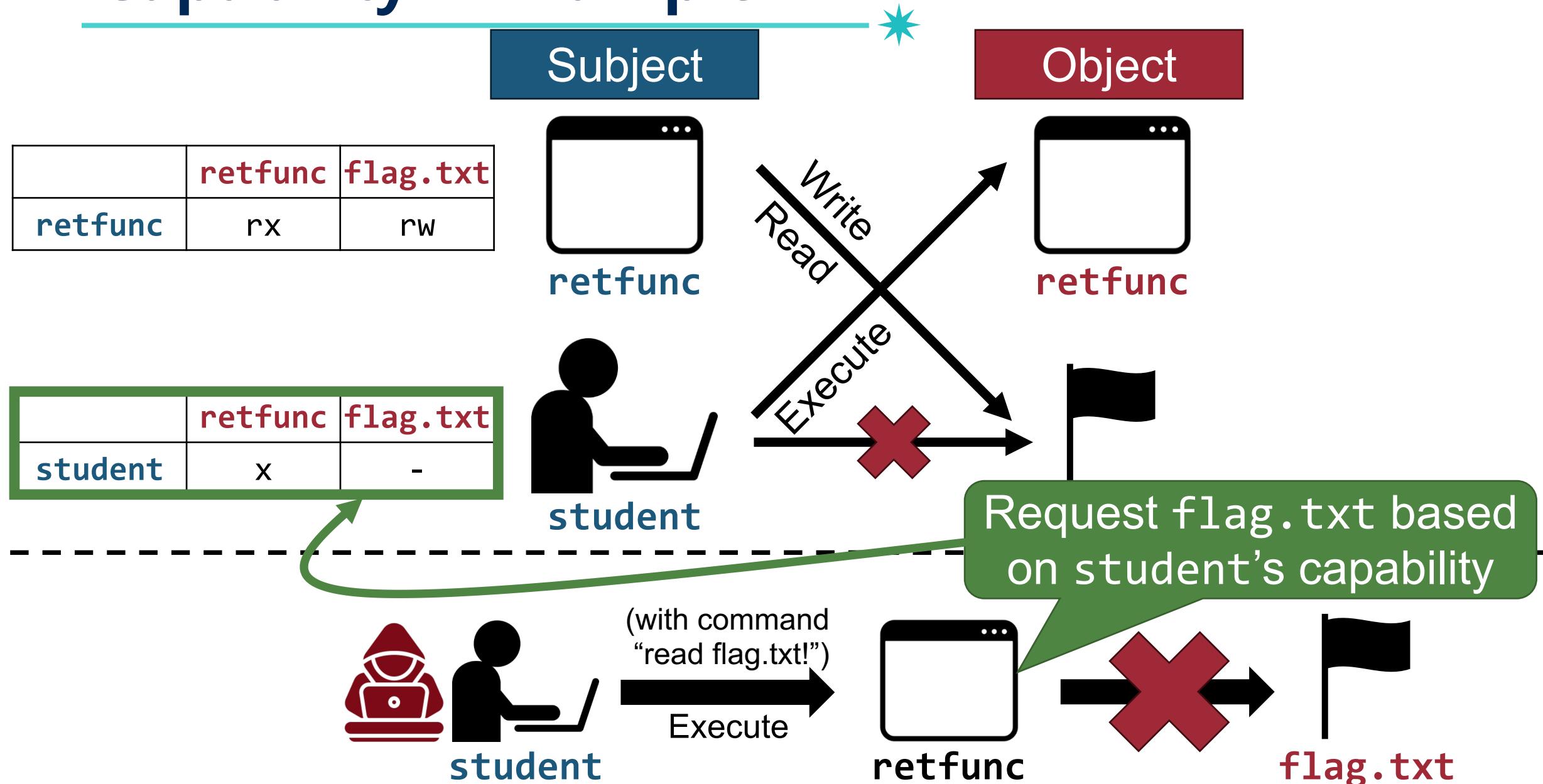
Capability – Example

31



Capability – Example

32



ACL vs. Capabilities



- ACL: “for each object, which subjects have permissions?”
 - More efficient to implement
 - Confused deputy problem
- Capabilities: “for each subject, which objects are allowed to be accessed?”
 - More secure by fine-grained access control
 - Easier to avoid the confused deputy (More secure)
 - More difficult to implement
- Then, capability-based approach is secure enough?

Trojan Horse

- A type of malware that disguises itself as legitimate code



Dear Professor,
I found and fixed a functional bug in the **retfunc**. Here is the patched one

```
void retfunc(...) {  
    ...  
    + if (user == "professor") {  
    +     flag = read("flag.txt");  
    +     write("/tmp/myown.txt", flag);  
    + }  
}
```



Thanks a lot! I will give you bonus points!

Trojan Horse

35

	retfunc	flag.txt	/tmp/myown.txt
professor	x	rw	rw

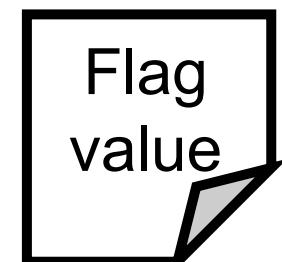
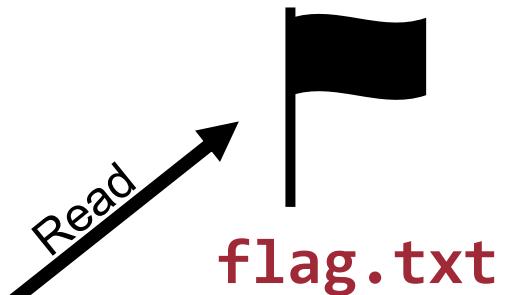


(with command
"read flag.txt!")

	retfunc	flag.txt	/tmp/myown.txt
student	x	-	rw



```
if (user == "professor") {  
+   flag = read("flag.txt");  
+   write("/tmp/myown.txt", flag);  
+ }
```



/tmp/myown.txt

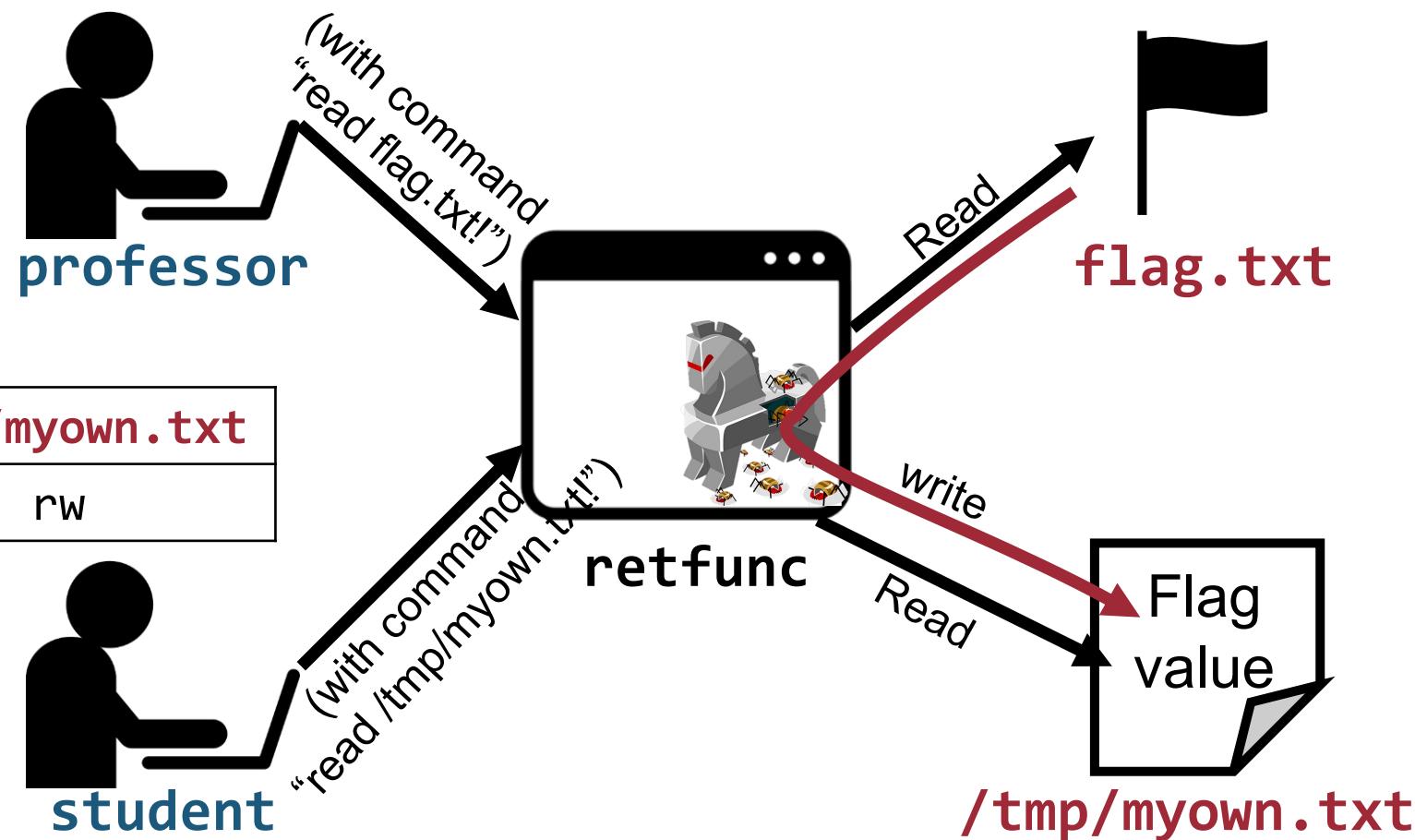
Trojan Horse

36

	retfunc	flag.txt	/tmp/myown.txt
professor	x	rw	rw

```
if (user == "professor") {  
+   flag = read("flag.txt");  
+   write("/tmp/myown.txt", flag);  
+ }
```

	retfunc	flag.txt	/tmp/myown.txt
student	x	-	rw

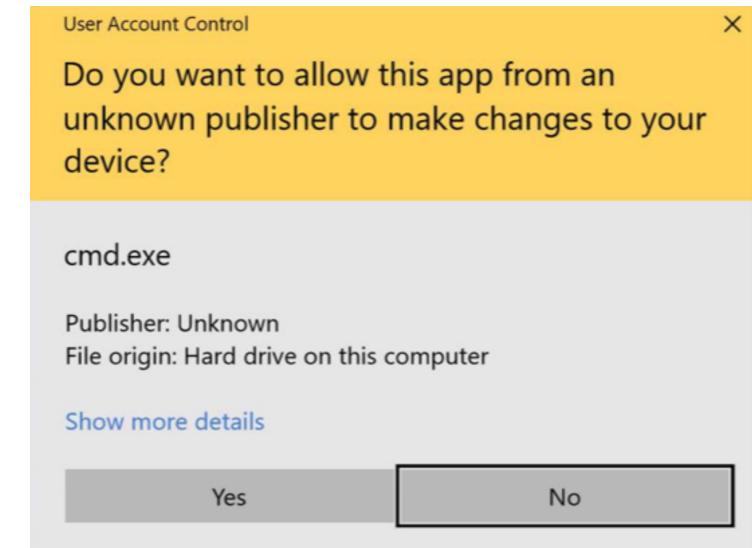


Exploitation

37

- Why this happen?
 - Malware or buggy SW
 - Attacker's intention with the user's privileges
 - No way to tell the difference between the legitimate software and a Trojan

- How to avoid this?
 - More restrictive access control (later this lecture)
 - Program analysis (later this semester)



Lessons from DAC

38



- Access control: not an easy problem
 - Balance between usability and security

- Well-known issues
 - Confused deputy: protected by capability-based systems
 - Trojan horse: do not trust programs from unknown sources

Real World Example: Unix



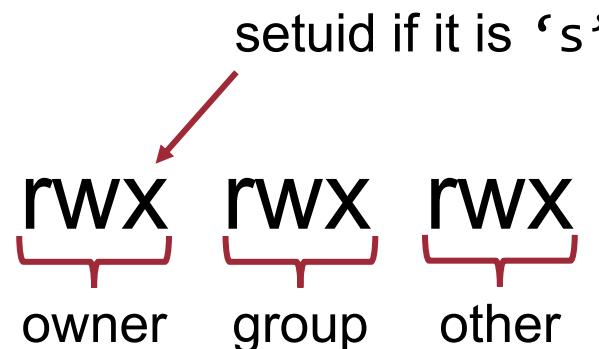
- Each process has a user id
 - Inherit from the parent process
 - Special root id: all access allowed
- Each **file** has an access control list (ACL)
 - Grants permission to users
 - Owner, group, other

	File1	File2	File3	...
User1	rw	rwx	rwx	rwx
User2	r	rx	rwx	-
User3	r	rx	r	-
...	r	rx	-	-

Unix File Access Control List

40

- Each file has an owner and group
- Permissions set by owners
 - {read, write, execute} x {owner, group, other}
- Only owners and root can change permissions
- Q: How can we run processes on behalf of someone else?
 - E.g., password change



User ID



- Each process has three IDs in Unix
- Real user ID (RUID)
 - Same as the user ID of the parent
 - To determine which user started the process
- Effective user ID (EUID)
 - From “Set User ID” bit on the file being executed or system calls
 - To determine the permissions for the process
- Saved user ID (SUID)
 - To restore previous EUID

```
vagrant@cse467:~/hw2$ sudo chmod 4550 retfunc
vagrant@cse467:~/hw2$ ls -al
-r--r---- 1 flag      flag          5 Nov 16 06:10 flag.txt
-r-sr-x-- 1 flag      retfunc     15480 Nov 16 06:09 retfunc
```

User ID



- Each process has three IDs in Unix
- Real user ID (RUID)
 - Same as the user ID of the parent
 - To determine which user started the process
- Effective user ID (EUID)
 - From “Set User ID” bit on the file being executed by system calls
 - To determine the permissions for the program
- Saved user ID (SUID)
 - To restore previous EUID

If you execute retfunc,
the EUID of the process is “flag”

```
vagrant@controller:~/hw2$ sudo chmod 4550 retfunc
vagrant@controller:~/hw2$ ls -al
-r--r----- 1 flag      flag          5 Nov 16 06:10 flag.txt
-r-sr-x--- 1 flag      retfunc     15480 Nov 16 06:09 retfunc
```

Example – HW2

```
*-r--r---- 1 flag      flag          5 Nov 16 06:10 flag.txt  
-r-xr-x-- 1 flag      retshell    15480 Nov 16 06:09 retshell
```

- **Goal:** Execute the shell by logging in with the flag account (RUID = flag)
- Step 1. ssh login with retfunc account (Your RUID = retshell)
- Step 2. execute retshell!
 - process's RUID = retshell
 - process's EUID = flag
- Step 3. Buffer overflow! execute setreuid(geteuid(), geteuid())
 - process's RUID = flag
 - process's EUID = flag
- Step 4. Buffer overflow! execute /bin/bash
- Step 5. spawn shell with flag account (RUID = flag)

Change both real user ID and effective user ID into effective user ID

Access Control Policy



How does one grant the right level of permission to an individual?

- **Discretionary access control (DAC)**
 - All objects have owners
 - What permission to grant others? **Owners** can decide
 - E.g., Unix, Windows, etc.
- **Mandatory access control (MAC)**
 - What permission to grant others? **Only the admin** can decide
 - Users cannot change policy themselves
 - E.g., SELinux, military, etc.

Mandatory Access Control (MAC)

45

- The system assigns both **subjects** and **objects** special security attributes
 - E.g., top secret, unclassified
- Privileges cannot be changed by users but by **system administrators**
- More restrictive and secure than discretionary access control
- Mostly used in security-critical systems (e.g., military)



Multilevel Security (MLS)



- Most common form of mandatory access control
- Developed by the US Department of Defense
 - All **information (objects)** possesses a classification
 - Every **person (subject)** posses a classification (or clearance)
- Access is allowed if the **person's** class is *higher than* the **information'** class

Security Classification

47

- Two components: **security level** (sensitivity) and **compartment** (category)
- **Security level**: a total ordered (small) set
 - Access is allowed if the **subject**'s level is *higher than* the **objects**' level
 - e.g., UNCLASSIFIED < CONFIDENTIAL < SECRET < TOP SECRET
- **Compartment**: a set of categories
 - Access allowed if **subject**'s compartments *includes* **object**'s compartments
 - E.g., {MILITARY, ECONOMY, ENVIRONMENT}

TOP SECRET

SECRET

CLASSIFIED

CONFIDENTIAL

Classification Lattice



- The combination of **security level** and **compartment** forms a *lattice*
 - E.g., Security level = {TOP SECRET, SECRET},
Compartment = {Army, Nuclear}
- Lattice (SC, \sqsubseteq) : a **partially ordered set** that satisfies the followings
 - Reflexivity: $\forall x \in SC. x \sqsubseteq x$
 - Transitivity: $\forall x, y, z \in SC. x \sqsubseteq y \wedge y \sqsubseteq z \Rightarrow x \sqsubseteq z$
 - Anti-symmetry: $\forall x, y \in SC. x \sqsubseteq y \wedge y \sqsubseteq x \Rightarrow x = y$
 - Order: $\langle S1, C1 \rangle \sqsubseteq \langle S2, C2 \rangle \Leftrightarrow S1 \leq S2 \wedge C1 \subseteq C2$

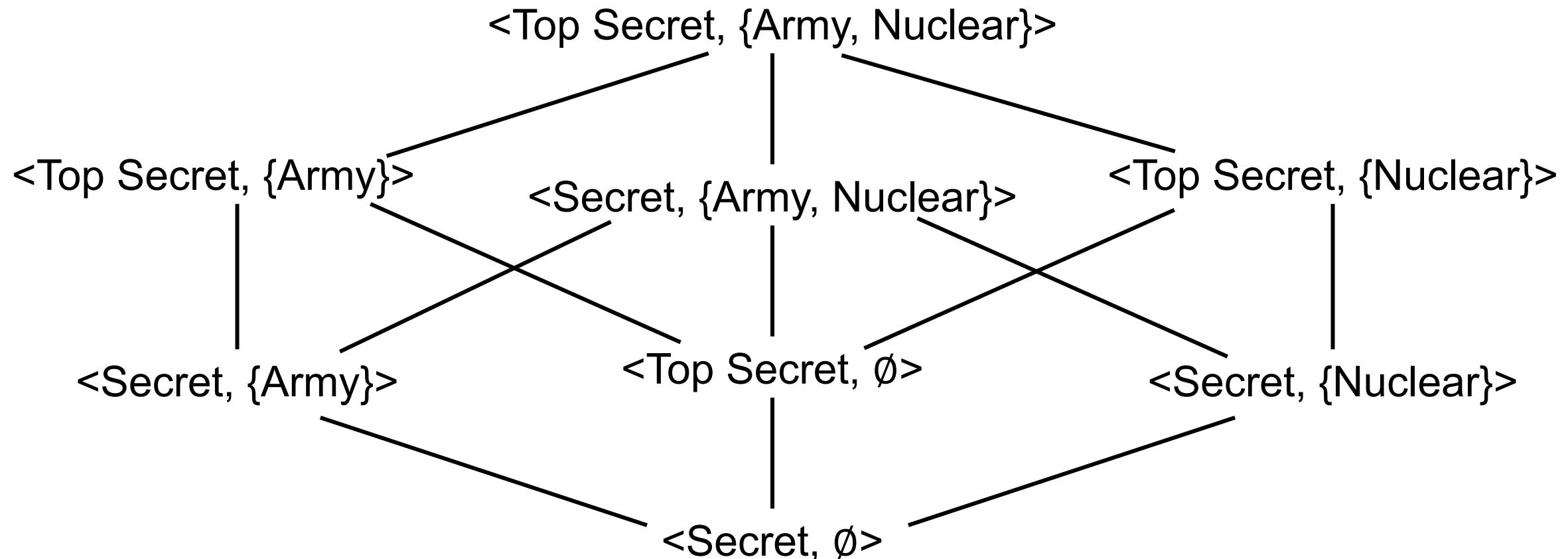
Classification Lattice – Example

- The combination of **security level** and **compartment** forms a *lattice*
 - E.g., Security level = {TOP SECRET, SECRET},
Compartment = {Army, Nuclear}
 - All possible set:
 - Security level: TOP SECRET or SECRET
 - Compartment: \emptyset , {Army}, {Nuclear}, {Army, Nuclear}

Classification Lattice – Example

50

- The combination of **security level** and **compartment** forms a *lattice*
 - E.g., Security level = {TOP SECRET, SECRET},
Compartment = {Army, Nuclear}

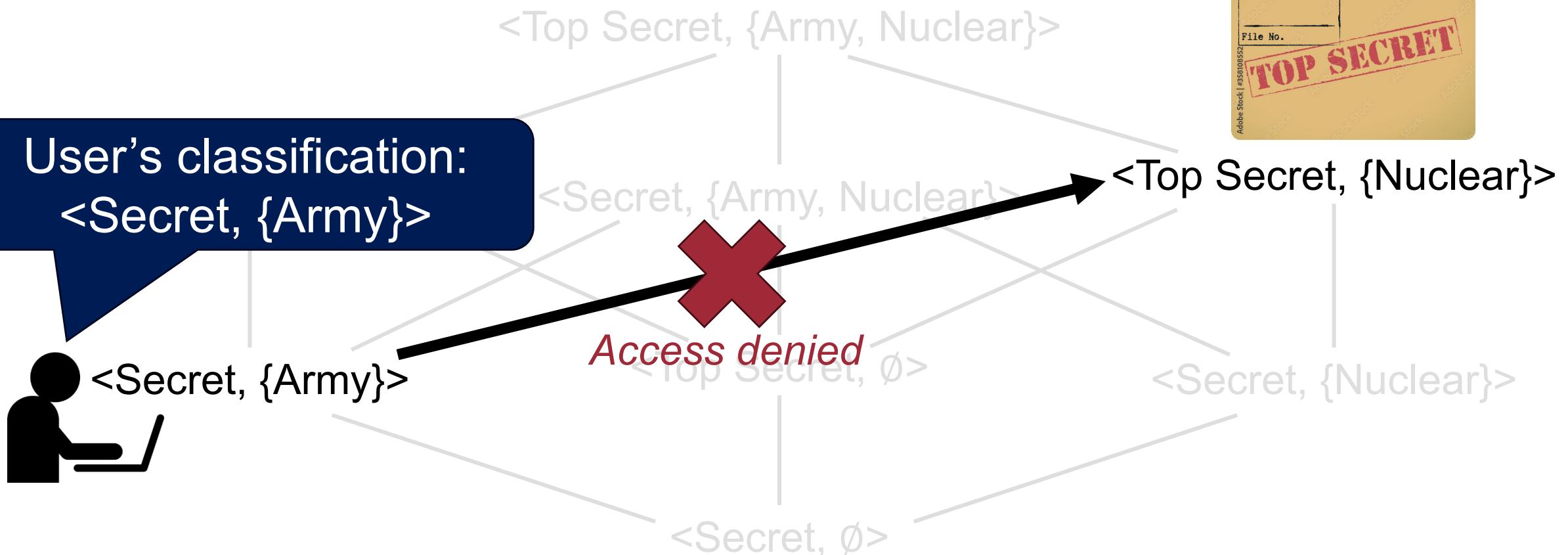


Classification Lattice – Example

51

- The combination of **security level** and **compartment**
 - E.g., Security level = {TOP SECRET, SECRET, ...}, Compartment = {Army, Nuclear}

File's classification:
<Top Secret, {Nuclear}>

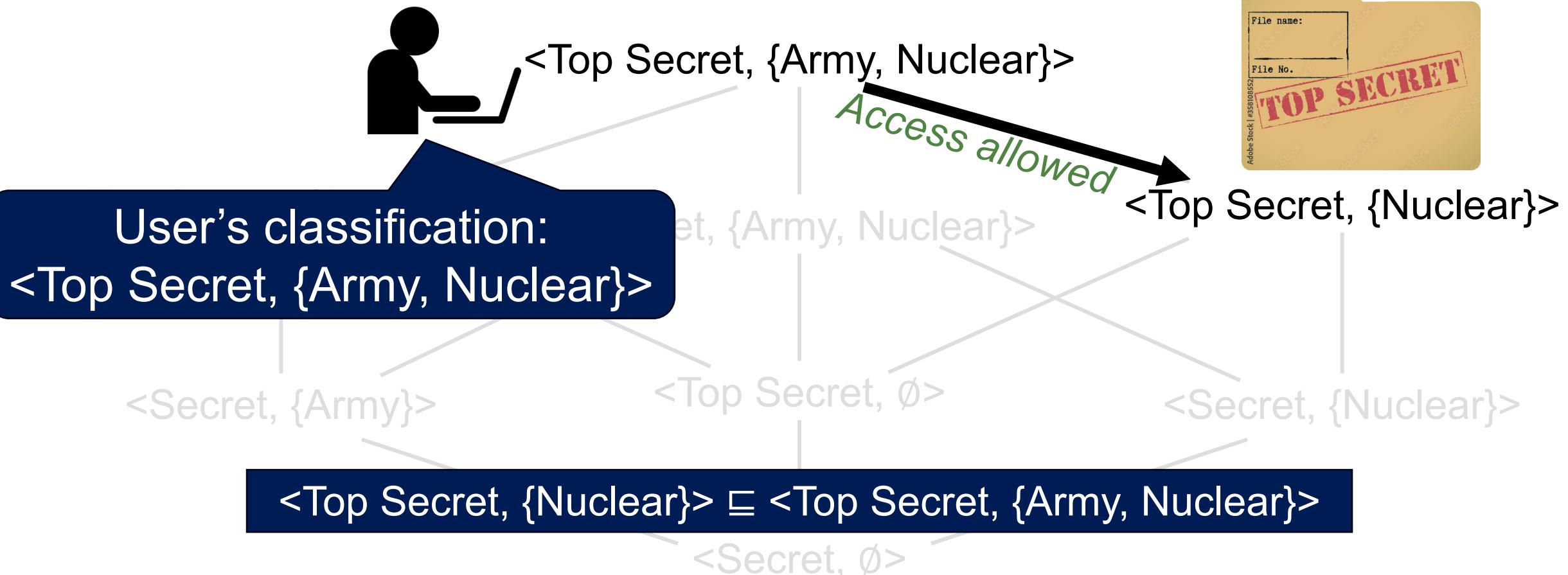


Classification Lattice – Example

52

- The combination of **security level** and **compartment**
 - E.g., Security level = {TOP SECRET, SECRET, ...}, Compartment = {Army, Nuclear}

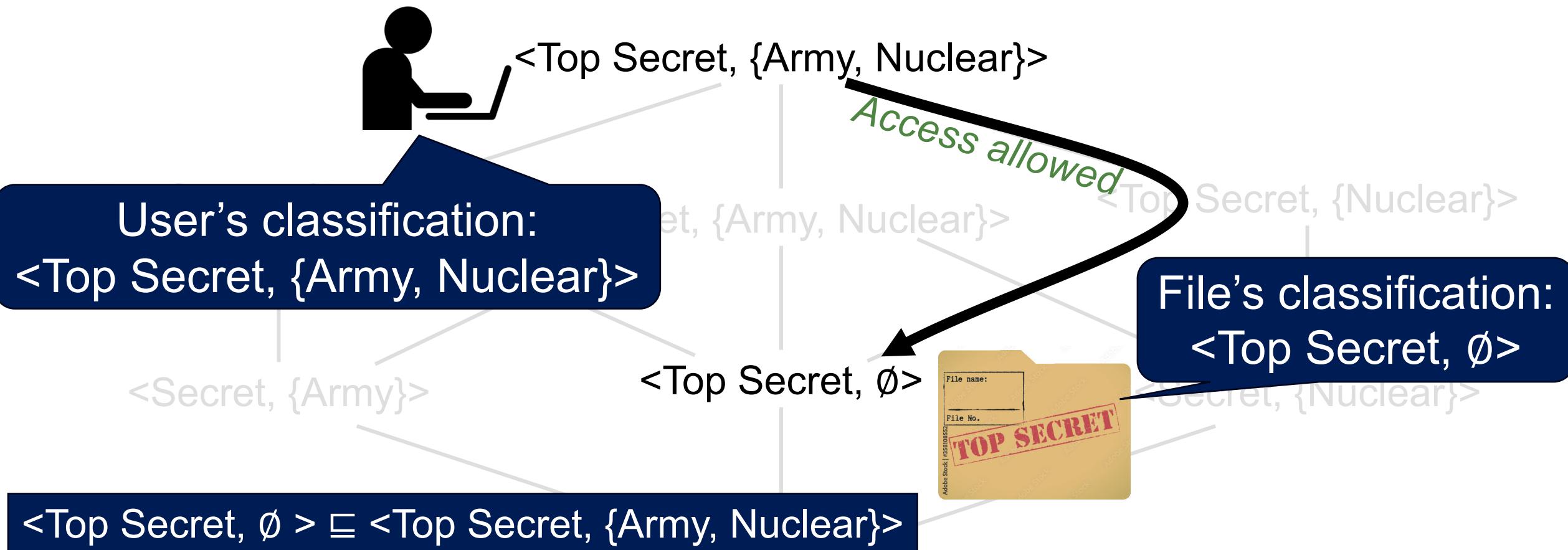
File's classification:
<Top Secret, {Nuclear}>



Classification Lattice – Example

53

- The combination of **security level** and **compartment** forms a *lattice*
 - E.g., Security level = {TOP SECRET, SECRET},
Compartment = {Army, Nuclear}



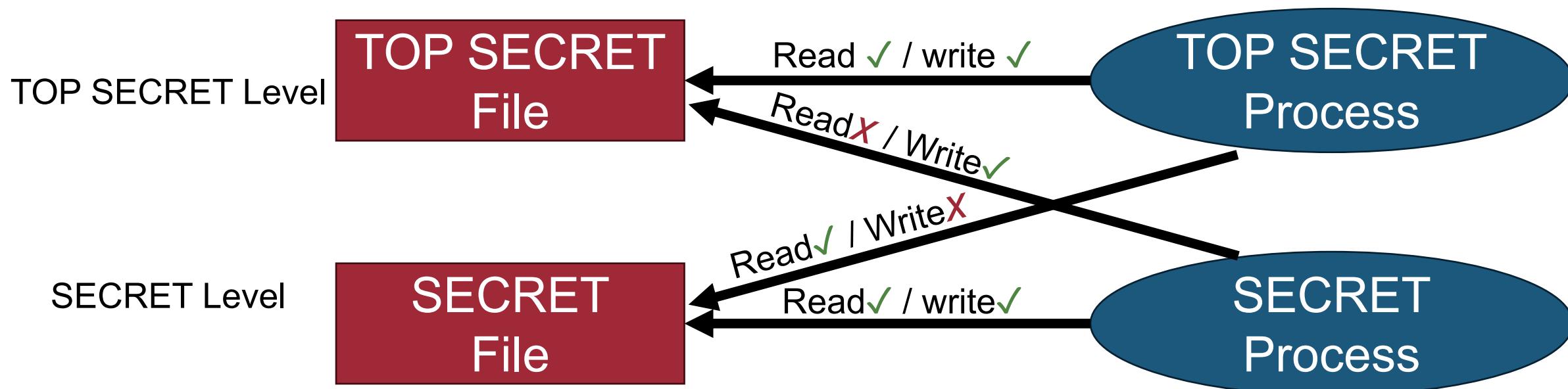
The Bell-LaPadula Model (BLP)

- The first mathematical model of a multilevel secure system in 1973
- Main concern: prevent information leak (confidentiality)
 - Ensures that information do not flow from higher security class to lower/incomparable class
- Idea:
 - Lower class subjects **cannot read higher** class objects (NO READ UP)
 - Higher class subjects **cannot write lower** class objects (NO WRITE DOWN)

The BLP Model: Properties

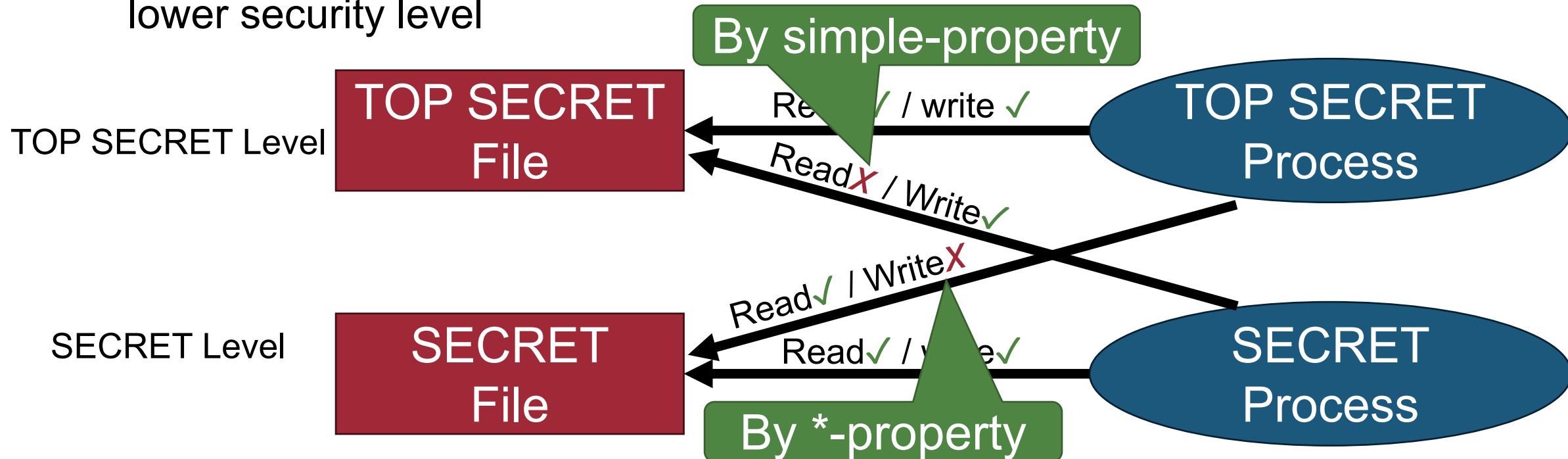
55

- Simple property (NO READ UP):
 - a **subject** at a given security level may not read an **object** at a higher security level
- *-property (NO WRITE DOWN)
 - a **subject** at a given security level may not write to any **object** at a lower security level



The BLP Model: Properties

- Simple property (NO READ UP):
 - a **subject** at a given security level may not read an **object** at a higher security level
- *-property (NO WRITE DOWN)
 - a **subject** at a given security level may not write to any **object** at a lower security level



Pros and Cons of BLP



- Pros: confidentiality
- When is BLP useful?
 - NO READ UP, NO WRITE DOWN = READ DOWN, WRITE UP
 - Reporting system (e.g., government, military, school)
- Cons: integrity
 - Attackers cannot read secret information, but what about write?
 - E.g., fake announcement

The Biba Model

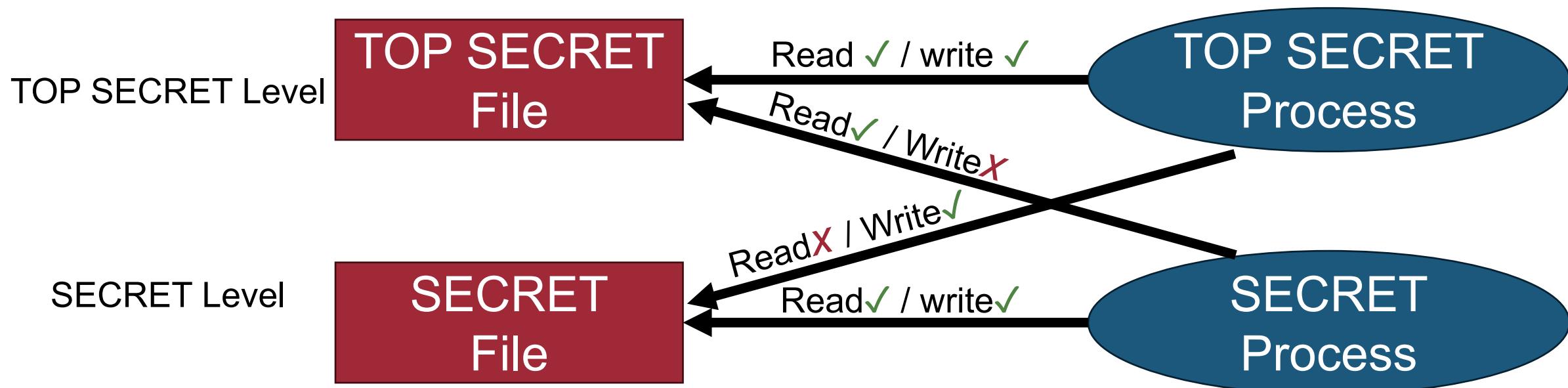


- Dual of BLP
- Main concern: prevent unauthorized change (integrity)
 - from **lower class** to **higher class**
- Idea:
 - Lower class subjects **cannot write higher** class objects (NO WRITE UP)
 - Higher class subjects **cannot read lower** class objects (NO READ DOWN)

The Biba Model: Properties

59

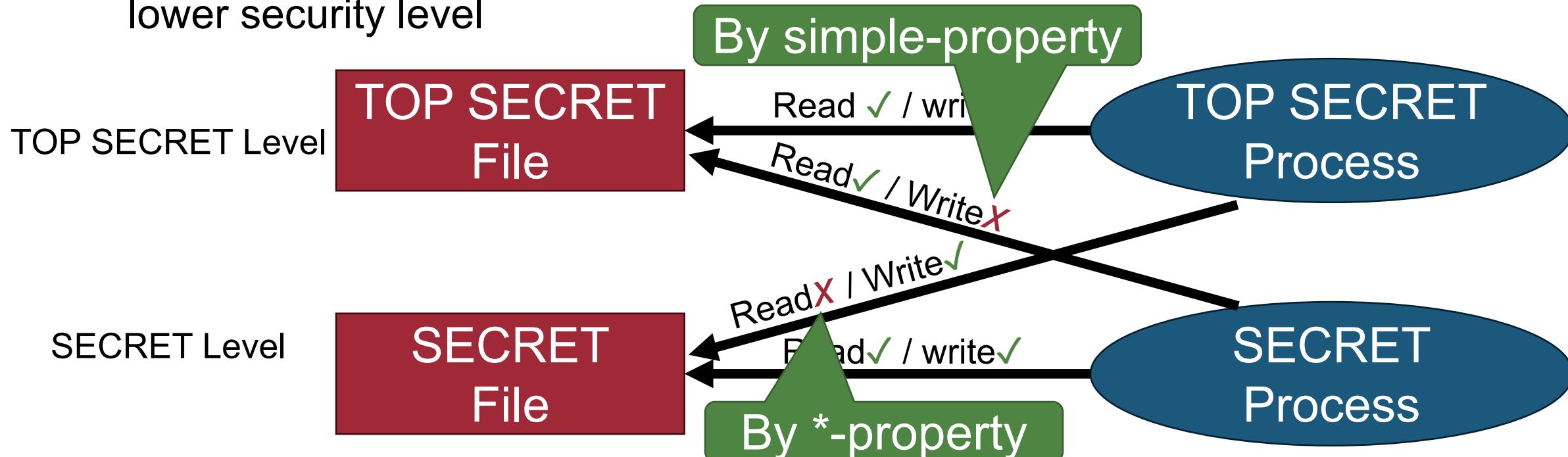
- Simple property (NO WRITE UP):
 - a **subject** at a given security level may not write an **object** at a higher security level
- *-property (NO READ DOWN)
 - a **subject** at a given security level may not read to any **object** at a lower security level



The Biba Model: Properties

60

- Simple property (NO WRITE UP):
 - a **subject** at a given security level may not write an **object** at a higher security level
- *-property (NO READ DOWN)
 - a **subject** at a given security level may not read to any **object** at a lower security level



Pros and Cons of Biba

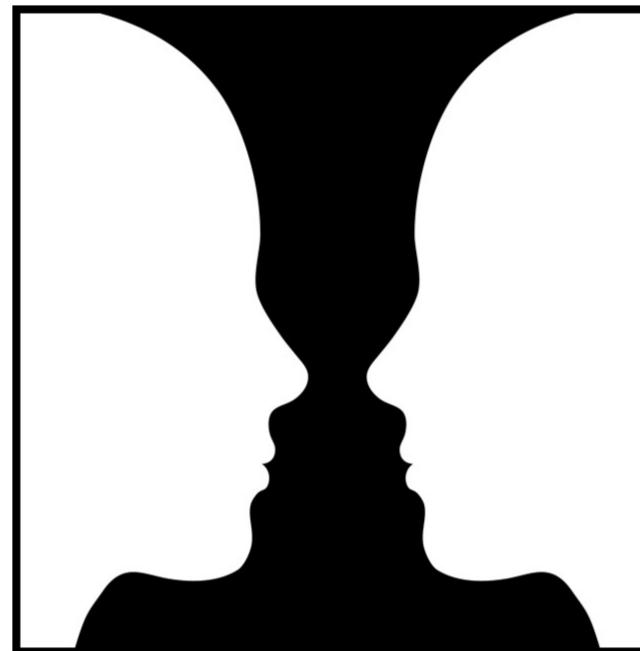


- Pros: integrity
- When is BLP useful?
 - NO WRITE UP, NO READ DOWN = WRITE DOWN, READ UP
 - Notification system (i.e., important notice won't be compromised)
- Cons: confidentiality

BLP vs. Biba



- BLP and Biba are mutually exclusive in the same classification
- But, confidentiality and integrity can coexist in the same system
 - In their own independent classifications



Summary

63

- Access control: determine what users have permission to do
- Discretionary access control (DAC): owner's discretionary
 - ACL-based: more **usable** but the **confused deputy problem** can occur
 - Capability-based: more **secure** but still **Trojan** can occur
- Mandatory Access Control (MAC): admin's policy
 - BLP (**confidentiality**) vs Biba (**integrity**)
- Which (combination of) models to choose? depending on the systems and goals