CS348
Access Control

Min Suk Kang & **Kihong Heo**

KAIST

Operating System Security

- The main goal of OS: resource sharing
  - A lot of requests from multiple users 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

How does the system determine what users have permission to do?

- Method to determine who has access to various information
  - Defense against attacks in the first place!
- Access control is everywhere: not only OS
  - E.g., hardware, databases, network etc

Access Control Failure

Reference Monitor

- Check if a subject can perform an operation on an object
- Three key properties:
  - Complete mediation: must always be called
  - Tamper-proof: cannot be altered
  - Small: easy to verify

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!
  - E.g., SELinux, military, etc

Access Control Matrix

- Relationship among subjects, objects and permissions

Object

Subject

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

Permission, Access Control Entry (ACE)

Why so large spare matrix?

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
prof	r
ta	r
student1	r
student2	r

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

	/home/prof/exam/
root	rwx
prof	rwx
ta	r

	/home/admin/
root	rwx

Example

student1.txt	
autograder	
student1	

student2.txt	
autograder	
student2	

grade.txt
autograder

Example

	student1.txt
autograder	rw
student1	r

	student2.txt
autograder	rw
student2	r

	grade.txt
autograder	rw

The Confused Deputy Problem

	student1.txt
autograder	rw
student1	r

	student2.txt
autograder	rw
student2	r

	grade.txt
autograder	rw

No violation!

Real World Examples

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.

Analysis

- Who is to blame?
- The deputy (autograder) got confused
  - Runs with authority from two masters at a time: prof and student2
  - For two different objects at a time: grade.txt and student2.txt
- No way to tell which master has authority of which resource

	grade.txt
autograder	rw

Ambient Authority

- Authority that is automatically exercised but not selected by its user
- Object is designated by its name only
- The dominant form of access control (e.g., access control list)
  - Easy to use: no need to specify authorities all the time
  - E.g., when you run a process, it implicitly inherits all the authorities from you
- Violate the principle of least privilege

*M. Miller, K. Yee, J. Shapiro. *Capability Myths Demolished*. 2003

Capability-based Approach

	student1.txt	student2.txt	grade.txt
autograder	rw	rw	rw

- Subject-centered approach
- Capability: a pair of an object and a set of privileges
  - E.g., fd = fopen(“/etc/passwd”, “r”);
- Mostly used for more secure operating systems (E.g., KeyKOS)
  - Also partly used in main stream OS such as Linux
- Resource request based on capability

Example

	student1.txt	student2.txt	grade.txt
autograder	rw	rw	rw

	student1.txt
student1	rw

	student2.txt
student2	rw

ACL vs Capabilities

- ACL: “for each object, which subjects have permissions?”
  - More efficient when #object >> #subject
  - Ambient authority problem
- Capabilities: “for each subject, which objects are allowed to be accessed?”
  - More secure by fine-grained access control
  - No ambient authority
- Then, capability-based approach is secure enough?

Trojan Horse

- A type of malware disguised as legitimate software
- Example:

```
void autograder(…) {
  …
+  if (user == “prof”) {
+    solution = read(“solution.txt”);
+    write(“student2.txt”, solution);
+  }
}
```

Example

	student2.txt	solution.txt
prof	rw	rw
autograder	rw	rw

	student2.txt
student2	r

Exploitation

- Why this happen?
  - Malware or buggy SW
  - Attacker's intention with the user's privileges
  - No way to tell the difference between the user and a Trojan
- How to avoid this?
  - More restrictive access control (later this lecture)
  - Program analysis (later this semester)

> *"To what extent should one trust a statement that a program is free of Trojan horses? Perhaps it is more important to trust the people who wrote the software."*
>
> - Ken Thompson, *Reflections on Trusting Trust.* Turing Award Lecture. 1984

Lessons from DAC

- Access control: not an easy problem
  - Balance between usability and security
- Well-known issues
  - Confused deputy (ambient authority): protected by capability-based systems
  - Trojan horse: do not trust programs from unknown sources

Trojan Horse Revisited

- How to prevent this Trojan horse?
- Idea: different security level
  - “prof” is a higher security level subject than “student2”
  - “solution.txt” is a higher security level object than “student2.txt”

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
  - E.g., SELinux, military, etc

Mandatory Access Control

- 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

- 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 objects's class

Security Classification

- Two components: security level (sensitivity) and compartment (category)
- Security level: a total ordered (small) set
  - Access allowed if subject's level is **higher** than object's 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}

Classification Lattice

- The combination of security level S and compartment C forms a lattice SC
  - E.g., Sec 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: $\forall x, y, z \in SC \,.\, x \sqsubseteq y \wedge y \sqsubseteq z \implies x \sqsubseteq z$
  - Antisymmetry: $\forall x, y \in SC \,.\, x \sqsubseteq y \wedge y \sqsubseteq x \implies x = y$
  - Order: $\langle S_1, C_1 \rangle \sqsubseteq \langle S_2, C_2 \rangle \iff S_1 \leq S_2 \wedge C_1 \subseteq C_2$

Example

- $S = \{\mathsf{TS}, \mathsf{S}\}$, $C = \{\mathsf{Army}, \mathsf{Nuclear}\}$
- SC :

The Bell-LaPadula Model (BLP)

- The first mathematical model of a multilevel secure system in 1973
- Main concern: prevent information leak (confidentiality)
  - from higher security class to lower/incomparable class
  - E.g., Trojan horse

- Idea:
  - Lower class subjects **cannot read higher** class objects (NO READ UP)
  - Higher class subjects **cannot write lower** class objects (NO WRITE DOWN)

Properties

- Let $\lambda : (S \cup O) \rightarrow SC$ (classification of a given subject or object)
  - S: a set of subjects, O: a set of objects, SC : the classification
- Simple property (NO READ UP):
  A subject s can read object o only if $\lambda(o) \sqsubseteq \lambda(s)$
- $*$-property (NO WRITE DOWN):
  A subject s can write object o only if $\lambda(s) \sqsubseteq \lambda(o)$

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)

Properties

- Let $\lambda : (S \cup O) \rightarrow SC$
  - S: a set of subjects, O: a set of objects, SC : the classification
- Simple property (NO WRITE UP):
  A subject s can write object o only if $\lambda(o) \sqsubseteq \lambda(s)$
- $*$-property (NO READ DOWN):
  A subject s can read object o only if $\lambda(s) \sqsubseteq \lambda(o)$

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

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 Contol List

- 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

setid – $\underbrace{\texttt{rwx}}_{\text{owner}}$ $\underbrace{\texttt{rwx}}_{\text{group}}$ $\underbrace{\texttt{rwx}}_{\text{other}}$

User ID

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

Example

// RUID 25
…
exec(…);
// Owner 18
// SetUID
…
…
i = getruid()
setuid(i);
…
…
RUID: 25
EUID: 18
RUID: 25
EUID: 25
read/write
read/write
File 1
Owner: 18
-rw-r-r-
File 2
Owner: 25
-rw-r-r-

SetUID Programs

- Very widely used: passwd, mount, sudo, …
- Should be very careful
- What happens if abused?
- How can we prevent such threats?

Summary

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