CSE543 - Introduction to Computer and Network Security
Module: Operating System Security and Capabilities

Professor Trent Jaeger
OS Security

• So, you have built an operating system that enables user-space processes to access hardware resources
  ‣ Thru various abstractions: files, pages, devices, etc.
• And you have added a reference validation mechanism that enforces an access control model of your choice
  ‣ Satisfies the reference monitor concept
  ‣ Implements a mandatory protection system
• Now, you want your operating system to prevent the vulnerabilities we discussed previously
  ‣ How do you do that?
OS Security

• So, you have built an operating system that enables user-space processes to access hardware resources
  ‣ Thru various abstractions: files, pages, devices, etc.
• And you have added a reference validation mechanism that enforces an access control model of your choice
  ‣ Satisfies the reference monitor concept
  ‣ Implements a mandatory protection system
• Now, you want your operating system to prevent the vulnerabilities we discussed previously
  ‣ How do you do that?
Prevent Buffer Overflows

• What can access control do about buffer overflows?
Prevent Buffer Overflows

• You could **protect** the process from **untrusted inputs**

• How would you use **Biba integrity**?
  ‣ What function would you give up?

• How would you use **LOMAC**?
  ‣ What function would you give up?

• How would you use **Windows Vista**?
  ‣ Would this work?

• Any other ideas...
Prevent * Overflows

• Could you do anything different for other types of attacks based on malicious input?
  ‣ Heap, integer overflows
  ‣ Malicious names
  ‣ Confused deputy (various)

• Also, consider secrecy attacks like Trojan horses
• Exercise for the student...
Prevent Buffer Overflows

- What if you had to allow the process to read untrusted input? E.g., client requests
  - Can access control still help?
Untrusted Programs

• You could **confine** the process **from propagating attacks** should it become untrusted
Sandboxing

• An execution environment for programs that contains a limited set of rights
  ▸ A subset of your permissions (meet secrecy and integrity goals)
  ▸ Cannot be changed by the running program (mandatory)
UNIX Chroot

• Create a domain in which a process is confined
  ‣ Process can only read/write within file system subtree
  ‣ Applies to all descendant processes
  ‣ Can carry file descriptors in ‘chroot jail’

SYNOPSIS
chroot [-u -user] [-g -group] [-G -group,group,...] newroot [command]
Chroot Vulnerability

• Unfortunately, chroot can trick its own system
  ‣ define a passwd file at <newroot>/etc/passwd
  ‣ run su
    • su thinks that this is the real passwd file
  ‣ gives root access
    • Use mknod to create device file to access physical memory

• Setup requires great care
  ‣ Never run chroot process as root
  ‣ Must not be able to get root privileges
  ‣ No control by chrooted process (user) of contents in jail
  ‣ Be careful about descriptors, open sockets, IPC that may be available
Limit Impact of Attacks

- You could **confine** the process **from propagating attacks** should it be untrusted
- How would you use **Biba integrity**?
  - What function would you give up?
- How would you use **LOMAC**?
  - What function would you give up?
- How would you use **Windows Vista**?
  - Would this work?
- What if you had to **allow** the process to modify the **kernel**? E.g., client requests to privileged service
  - Can **access control** still help?
Protect and Confine

• Access control plays two roles
  ‣ Protect the process from others (untrusted)
  ‣ Conﬁne the process from others (privileged)

• Plays **both roles** all the time
Principle of Least Privilege

A system should only provide those rights needed to perform the processes function and no more.

- **Implication 1**: you want to reduce the protection domain to the smallest possible set of objects
- **Implication 2**: you want to assign the minimal set of rights to each object
- **Caveat**: of course, you need to provide enough rights and a large enough protection domain to get the job done.
Least Privilege

- Limit permissions to those required and no more
- Consider three processes for user J
  - Restrict privilege of the process J₁ to prevent leaks

<table>
<thead>
<tr>
<th></th>
<th>O₁</th>
<th>O₂</th>
<th>O₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>J₁</td>
<td>R</td>
<td>RW</td>
<td></td>
</tr>
<tr>
<td>J₂</td>
<td>-</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>J₃</td>
<td>-</td>
<td>R</td>
<td>RW</td>
</tr>
</tbody>
</table>
Enforce Least Privilege

• How would you build a protection system that would enable enforcement of least privilege using mandatory access control?
# Mandatory Protection System

**Labeling State**

<table>
<thead>
<tr>
<th>File: newfile</th>
<th>File: acct</th>
</tr>
</thead>
<tbody>
<tr>
<td>secret</td>
<td>secret</td>
</tr>
<tr>
<td>unclassified</td>
<td>unclassified</td>
</tr>
<tr>
<td>trusted</td>
<td>trusted</td>
</tr>
<tr>
<td>untrusted</td>
<td>untrusted</td>
</tr>
</tbody>
</table>

**Process:**
- newproc
- other

**Transition State**

**Protection State**

<table>
<thead>
<tr>
<th></th>
<th>secret</th>
<th>unclassified</th>
<th>trusted</th>
<th>untrusted</th>
</tr>
</thead>
<tbody>
<tr>
<td>secret</td>
<td>read</td>
<td>read</td>
<td>read</td>
<td></td>
</tr>
<tr>
<td>unclassified</td>
<td>read</td>
<td>read</td>
<td>read</td>
<td></td>
</tr>
<tr>
<td>trusted</td>
<td>write</td>
<td>read</td>
<td>write</td>
<td></td>
</tr>
<tr>
<td>untrusted</td>
<td>read</td>
<td>read</td>
<td>read</td>
<td></td>
</tr>
</tbody>
</table>
Mandatory Protection System

• Protection State
  ‣ Fixed set of labels for subjects and objects
  ‣ Fixed set of operations
  ‣ What happens when a new file is created?

• Labeling State
  ‣ Associates subjects and objects with labels
  ‣ All subjects and objects are labeled at all times
  ‣ What happens when you want to change permissions?

• Transition State
  ‣ Associate condition with label change
SELinux

- Protection State
  - SELinux requires explicit rules for protection states
  - *allow subject_label object_label operations*
  - Usual access control rule

- Labeling State
  - SELinux requires explicit rules for labeling states
  - *type_transition source_label target_label default_label*
  - Source creates object at target, relabel target as default

- Transition State
  - SELinux uses *type_transition* for changing labels
  - For processes, relabel source as default
SELinux for Privilege Escalation

• For user to run passwd program
  ‣ Only passwd should have permission to modify /etc/shadow

• Need permission to execute the passwd program
  ‣ *allow user_t passwd_exec_t:file execute* (user can exec /usr/bin/passwd)
  ‣ *allow user_t passwd_t:process transition* (user gets passwd perms)

• Must transition to passwd_t from user_t
  ‣ *allow passwd_t passwd_exec_t:file entrypoint* (run w/ passwd perms)
  ‣ *type_transition user_t passwd_exec_t:process passwd_t*

• Passwd can the perform the operation
  ‣ *allow passwd_t shadow_t:file {read write}* (can edit passwd file)
Namespace Problems

• What if victim is susceptible to name resolution attacks?
  ‣ Link traversal
  ‣ File squat
  ‣ Directory traversal
  ‣ Untrusted search path
  ‣ Etc.
• Can access control help with those?
Attacks on Name Resolution

- Improper Binding Attack
  - Adversary controls bindings to redirect victim to a resource not under adversary’s control (confused deputy)
  - Victim expects low integrity/secrecy, gets high instead
    - E.g., Link traversal attacks, including TOCTTOU races

V<sub>root</sub> open("/var/mail/root")

A<sub>mail</sub>
Access Control Is Insufficient

- Webserver vulnerable to directory traversal
- Therefore, namespace resolution enforcement needs additional context than traditional access control
  - In this case, interface in the webserver making the call

![Diagram]

- Webserver
- Access Control: OK
- Password File
- Access Control: OK
- Web Pages

- Name 1
- Name 2
- Passwd File
- Web Pages
Action-specific Permissions

• Design the permissions of a process specific to its use

• How do we change the permissions of a process in an ACL system?
Changing Permissions

• Imagine a **multi-client server** (e.g., web server)
  ‣ Clients have a different set of objects that they can access

• In an ACL system, the **server always has access to all the objects**
  ‣ What happens if a client tricks the server into accessing into another client’s objects?
  ‣ Shouldn’t the server only have access to that client’s objects for its requests?

• Leads to confused deputy...
Capabilities

• A capability is the tuple (object, rights)

• A capability system implements access control by checking if the process has an appropriate capability
  ‣ Simple, right?
  ‣ This is a little like a ticket in the Kerberos system

• Q: Does this eliminate the need for authentication?
Capabilities

• A: Well, yes and no …
• Capabilities remove the overhead of managing per object rights, but add the overhead of managing capabilities
• Moreover, to get any real security, they have to be unforgeable
  ‣ Hardware tags (to protect capabilities)
  ‣ Protected address space/registers
  ‣ Language based techniques
    • Enforce access restrictions on caps.
  ‣ Cryptography
    • Make them unforgeable
User space capability?

• Well, what are the requirements?
  ‣ Authenticity/integrity - do not want malicious process to forge capabilities

• Start with the data itself: [object, rights]
  ‣ Object is typically encoded with identifier, or by some other tag (capabilities are sometimes known as tags)
  ‣ Rights are often fixed (read, modify, write, execute, etc.)

• Now, do what you with any other data (assume the kernel has a secret key $k$)

$$\text{E}(k, [O_i, r_1, r_2, \ldots r_n])$$

• What’s wrong with this construction (I got it from the website of one of the experts in the area)?
The right construction

- Encryption does not provide authenticity/integrity, it provides confidentiality

\[ [O_i, r_1, r_2, \ldots r_n], \text{HMAC}(k, [O_i, r_1, r_2, \ldots r_n]) \]

- So how would you attack the preceding construction?
### Real OS Capabilities

- The OS kernel manages capabilities in the process table, out of reach of the process.
- Capabilities added by user requests (that comply with policy).

#### Process Table

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process Z</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### C List

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>RX A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RW B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>W E</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- A
- B
- C
- D
A (fictional) Capability Example

• We use the “ls -lt” command to view the contents of our home directory in a OS implementing capabilities:
  ‣ Initially, our shell process has RWX capabilities for our home directory, and RX capabilities for all the directories to the root.
  ‣ The “ls -lt” command is exec()ed, and the shell delegates the directory permissions by giving “ls” the capabilities
    • Note that the capabilities are _not_ tied to any subject
  ‣ The “ls -lt” process exercises the rights to read the directories structure all the way down to the local
  ‣ Of course, the “ls -lt” process now need to obtain read rights to the files (to get their specific meta-information), and obtains them by appealing to the security manager (in kernel) -- the request fulfills the policy, and they are added and exercised
    • The “ls -lt” uses access rights given to the terminal to write output
• Note: there are many ways that the policy can be implemented, rights handed off, etc. We will talk about a couple in the following discussions.
Procedure-Level Protection Domains

- **HYDRA**
  - Each procedure defines a new protection domain

- **Procedure**
  - Code
  - Data
  - Capabilities to other objects
    - Caller-independent
    - Caller-dependent templates

- **Local Name Space**
  - Capabilities are bound here
  - Record of a procedure invocation (procedure instance)

- **Process**
  - Stack of LNSs
• Q: Which object defines the *protection domain*?
Implications of Fine-Grained Protection

• Programmer
  ‣ Must define *templates* for procedure
  ‣ Connect the procedure rights together
• Programmer is *responsible* for
  ‣ Functionality
  ‣ And security
  ‣ At the same time

• *Proven difficult to convince programmers to do this*
Capabilities and *-Property

• Earl Boebert found the following problem:
• In a pure capability model capabilities are data too
  ‣ E.g., pass to a procedure as in Hydra
  ‣ Programs can assert them
• So, consider a high secrecy Trojan horse program
  ‣ Can it read a low secrecy object?
  ‣ What if that object has a capability inside it?
• Then, it can read the capability to read-write low secrecy objects
  ‣ Use this capability in an operation to violate the *-property
• What would you do to prevent this?
Capability Systems

- **Solutions** to this problem have been produced
  - Compare capability to MLS before use
  - Compare capability to MLS before propagation

- **Not really holding up** use of capability systems
  - Security people like capability systems
  - In general, prevents all kinds of confused deputy attacks
    - Think about why...

- **Problem is that** programmers won’t use capability systems
  - And programmers often make mistakes programming simple security code, so ...
**Take Away**

- Lots of *vulnerabilities* out there
  - OS security mechanisms should reduce/eliminate vulnerabilities
- **Access control** limits process access to system resources
  - Including those used for processes to communicate
- However, access control treats the process as a **black box** - why?
  - Protects and confines processes
- So, is limited in protection unless programmers help
  - Capability systems
  - Be careful about *complexity* and *user-management*