Outline for Lecture 3

1. Wrap up “What is a process?”

2. Abstract approaches to access control (5.2)

3. UNIX notions of users, ownership, and permissions (5.1, 5.3)

4. suid Permissions
Back to our diagram…

Questions, though:
- What distinguishes the kernel from not-kernel?
- What *is* a process?

The CPL!
What is a process?

- **One Answer**: A data structure in “kernel memory”, including
  - MMU configuration
  - Register values
  - Kernel can load these values up, set CPL=3, and turn over control “to the process” (i.e. set EIP)
  - If kernel regains control, it can save these values to swap process out
Handling Memory for a Process

- Kernel creates a “virtual address space” for each process.
- Same virtual addresses (e.g. starting near 0) can be used by every process! They get translated to different physical addresses.
- Kernel can also mark some virtual address ranges (called segments) as “read only” or “do not execute” (EIP not allowed to point there).
- Violations are SEGFAUL Ts: MMU will take over in this case
• Kernel can also map same memory into several processes’ virtual address space
• Ex: Code for `malloc` is not copied for every process.
Handling Memory for a Process (cont.)

- Kernel configures MMU to translate addresses for proc1:
  - Read/Write/Execute to memory specific to proc1
  - Read/Execute access to libc
  - Possibly other special “segments”
- No access to memory to Kernel or proc2 memory!
  - They’re not even mapped; MMU will never allows access!
A process (i.e. code running with CPL=3) often needs to do privileged actions that the CPU won’t allow directly
  
  e.g. access device, write output, spawn new process, …

System calls allow this. They work roughly as follows:
  
  Process sets up arguments in pre-defined registers
  
  Then process executes instruction `int 0x80`
  
  CPU will set CPL=0 and jump to kernel handler
Outline for Lecture 3

1. Wrap up “What is a process?”

2. Abstract approaches to access control (5.2)

3. UNIX notions of users, ownership, and permissions (5.1, 5.3)

4. suid Permissions
So we have a secure kernel... What now?

1. Maybe all processes should not be “created equal”?  
   - e.g. Should one process be able to kill another?

2. Enable different people to use same machine?  
   - e.g. Need to enable confidential storage of files, sharing network, ...

3. System calls allow for safe entry into kernel, but only make sense for low-level stuff.  
   - We need a higher level to “do privileged stuff” like “change my password”.

All of this will be supported by an “access control” system.
Step 1: Give a crisp definition of a policy to be enforced.

1. Define a sets of subjects, objects, and verbs.

2. A policy consists of a yes/no answer for every combination of subject/object/verb.

Guiding philosophy: Utter simplicity.
The Access Control Matrix

- Entry in matrix is list of allowed verbs
- The matrix is not usually actually stored; It is an abstract idea.
Enforcing Policy: Reference Monitors

**Requirements:**

1. Tamper-proof.

2. Always invoked (not circumventable).

3. Verifiable; Simple enough to test thoroughly.

4. (Usually) Logs all requests.
Example Reference Monitor: The MMU

Registers

CPU

MMU

Memory

READ addr

Satisfies requirements?

<max>
Implementing Reference Monitors: ACLs

- ACL = “access control list”
- Logically, ACL is just a column of matrix
- Usually stored with object
- Can quickly answer question: “Who can access this object?”

Examples:
1. VIP list at event
2. This class on Canvas

More?
Implementing Reference Monitors: Capabilities

- “Capability” (of a subject) is a row of matrix
- Usually stored with subject
- Can quickly answer question: “What can this subject access?”

Examples:
1. Movie ticket
2. Physical key to door lock

More?
Files Descriptors in UNIX: ACL or Capability?

open(/bar/biz)

OS Kernel

Memory

process:
state=...
usage=...
openfiles=
1:stdin
2:stdout
3:/foo

Disk
Files Descriptors in UNIX: ACL or Capability?

open(/bar/biz) 4

OS Kernel

Disk

Process

Memory

process:
state=...
usage=...
openfiles= 1:stdin
2:stdout
3:/foo
4:/bar/biz

process:
state=...
usage=...
openfiles= 1:stdin
2:stdout
Files Descriptors in UNIX: ACL or Capability?

write(4, data)

OS Kernel

Disk

Process

OK

Process:
state=...
usage=...
openfiles=
1: stdin
2: stdout
3: /foo
4: /bar/biz
Reference monitor properties?

write(4, data)

OK

Disk

OS Kernel

Memory

process:
state=...
usage=...
openfiles=
1: stdin
2: stdout
3: /foo
4: /bar/biz

process:
state=...
usage=...
openfiles=
1: stdin
2: stdout
Outline for Lecture 3

1. Wrap up “What is a process?”

2. Abstract approaches to access control (5.2)

3. **UNIX notions of users, ownership, and permissions (5.1)**

4. suid Permissions
What is “UNIX”? Why should we study it?

- Initially an OS developed in the 1970s by AT&T Bell Labs.
- A riff on “Multics”. UNIX was meant to be simpler and leaner.
- Philosophy of small programs with simple communication mechanisms
- Licensed to vendors who developed their own versions. “BSD” = “Berkeley Software Distribution” may be most famous of those.
- Linux also later derived from UNIX. MacOS based on UNIX since 2000.

Why study UNIX?

1. Simple, even beautiful security design.
2. Looking at something concrete is enlightening.
3. You will almost certainly use it.
Subjects, Objects, and Verbs in UNIX (incomplete lists)

**Subjects:**
1. Users, identified by numbers called UIDs
2. Processes, identified by numbers called PIDs

**Objects:**
1. Files
2. Directories
3. Memory segments
4. Access control information (!)
5. Processes (!)
6. Users (!)

**Verbs (listed by object):**
1. For files and memory: Read, Write, Execute
2. For processes: Kill, debug
3. For users: Delete user, Change groups
Users, Groups, UIDs/GIDs and File Ownership

- A “user” is a sort of avatar that may or may not correspond to a person.
- Each user is identified by a number called UID that is fixed and unique.
- Each user may belong to 1 or more “groups”, each identified by number called GID.

```
inode:
mode=1010100...
uid=davidcash
gid=cs232
ctime=...
```

All files are owned by one user and one group.

- Changed with commands `chown` and `chgrp`. 
File Permissions

• Three bits for each of user, group, and other/all.
• Indicate read/write/execute permission respectively.

To check access:
1. If user is owner, then use owner perms.
2. If user is not owner but in group, user group perms.
3. Otherwise use “other” perms.

• Exception: Superuser (“root”) with UID=0 may bypass permissions.
The Root User

- “root” is the name for the administrator account
- UID = 0
- Can open/modify any file, kill any process, etc
- Rarely used as a log-in; Root’s powers are typically accessed via `sudo`
- Why not? (Which design principle(s) does this follow?)
Process Ownership and Permissions

- Every process has an owner; That process runs with permissions of the owner.
- `fork()` creates child process with same owner

Actually…. a process has three UIDs associated with it:

1. Real UID
2. Effective UID
3. Saved UID

- Why? To allow for fine-grained control over privileges via `setuid()` syscall.
- Implement least-privilege (P6) and isolated compartments (P5) in applications
Example: Web Servers

- Due to design of Linux, a web server must be run as **root** (!)
- Apache/NGINX written in C, a language in which vulnerabilities are common (next week!)

### Vulnerability Trends Over Time

<table>
<thead>
<tr>
<th>Year</th>
<th># of Vulnerabilities</th>
<th>DoS</th>
<th>Code Execution</th>
<th>Overflow</th>
<th>Memory Corruption</th>
<th>Sql Injection</th>
<th>XSS</th>
<th>Directory Traversal</th>
<th>Http Response Splitting</th>
<th>Bypass something</th>
<th>Gain Information</th>
<th>Gain Privileges</th>
<th>CSRF</th>
<th>File Inclusion</th>
<th># of exploits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1999</td>
<td>8</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td>7</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2001</td>
<td>12</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2002</td>
<td>20</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2003</td>
<td>16</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2004</td>
<td>20</td>
<td>8</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2005</td>
<td>10</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td>4</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td>17</td>
<td>5</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2008</td>
<td>12</td>
<td>2</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2009</td>
<td>8</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2010</td>
<td>9</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2011</td>
<td>12</td>
<td>8</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2012</td>
<td>8</td>
<td>4</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2013</td>
<td>5</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2014</td>
<td>11</td>
<td>9</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2015</td>
<td>4</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2016</td>
<td>4</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2017</td>
<td>11</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2018</td>
<td>13</td>
<td>3</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2019</td>
<td>14</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>225</td>
<td>79</td>
<td>25</td>
<td>21</td>
<td>1</td>
<td>22</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>20</td>
<td>14</td>
<td>6</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

| % Of All | 35.1 | 11.1 | 9.3 | 0.4 | 0.0 | 9.8 | 1.8 | 1.3 | 8.9 | 6.2 | 2.7 | 0.4 | 0.0 |

Apache » HTTP Server: Vulnerability Statistics
Example: Web Servers

- Due to design of Linux, a web server must be run as root (!)
- Apache/NGINX written in C, a language in which vulnerabilities are common (next week!)

**Vulnerability Details: CVE-2004-0492**

Heap-based buffer overflow in `proxy_util.c` for mod_proxy in Apache 1.3.25 to 1.3.31 allows remote attackers to cause a denial of service (process crash) and possibly execute arbitrary code via a negative Content-Length HTTP header field, which causes a large amount of data to be copied.

Publish Date: 2004-08-06 Last Update Date: 2017-10-10

- **CVSS Scores & Vulnerability Types**
  - **CVSS Score**: 10.0
  - **Confidentiality Impact**: Complete (There is total information disclosure, resulting in all system files being revealed.)
  - **Integrity Impact**: Complete (There is a total compromise of system integrity. There is a complete loss of system protection, resulting in the entire system being compromised.)
  - **Availability Impact**: Complete (There is a total shutdown of the affected resource. The attacker can render the resource completely unavailable.)
  - **Access Complexity**: Low (Specialized access conditions or extenuating circumstances do not exist. Very little knowledge or skill is required to exploit.)
  - **Authentication**: Not required (Authentication is not required to exploit the vulnerability.)
  - **Gained Access**: Admin
  - **Vulnerability Type(s)**: Denial Of Service; Execute Code; Overflow
  - **CWE ID**: CWE id is not defined for this vulnerability

- **Vendor Statements**
  Source: Apache
Example: Web Servers

- Due to design of Linux, a web server must be run as `root` (!)
- Apache/NGINX written in C, a language in which vulnerabilities are common (next week!)

---

### Nginx » Nginx: Vulnerability Statistics

**Vulnerabilities (26)**  |  **CVSS Scores Report**  |  **Browse all versions**  |  **Possible matches for this product**  |  **Related Metasploit Modules**
---|---|---|---|---
**Related OVAL Definitions**:  |  **Vulnerabilities (1)**  |  **Patches (2)**  |  **Inventory Definitions (0)**  |  **Compliance Definitions (0)**
**Vulnerability Feeds & Widgets**

#### Vulnerability Trends Over Time

<table>
<thead>
<tr>
<th>Year</th>
<th># of Vulnerabilities</th>
<th>DoS</th>
<th>Code Execution</th>
<th>Overflow</th>
<th>Memory Corruption</th>
<th>Sql Injection</th>
<th>XSS</th>
<th>Directory Traversal</th>
<th>Http Response Splitting</th>
<th>Bypass something</th>
<th>Gain Information</th>
<th>Gain Privileges</th>
<th>CSRF</th>
<th>File Inclusion</th>
<th># of exploits</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>2010</td>
<td>2</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2011</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>2012</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2013</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>2014</td>
<td>4</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>2016</td>
<td>5</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2017</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2018</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>Total</td>
<td>26</td>
<td>10</td>
<td>5</td>
<td>8</td>
<td>1</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>% Of All</td>
<td>38.5</td>
<td>19.2</td>
<td>30.8</td>
<td>3.8</td>
<td>0.0</td>
<td>0.0</td>
<td>7.7</td>
<td>0.0</td>
<td>7.7</td>
<td>19.2</td>
<td>3.8</td>
<td>0.0</td>
<td>0.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example: Dropping Privileges in OpenSSH Server

Connection request

<Authentication>

<shell session>

Demo
setuid() details are complicated

(a) An FSA describing setuid in Linux 2.4.18
... really complicated

(c) An FSA describing setresuid in Linux
Outline for Lecture 3

1. Wrap up “What is a process?”

2. Abstract approaches to access control (5.2)

3. UNIX notions of users, ownership, and permissions (5.1)

4. suid Permissions
suid Permission: Necessity and Danger

- Passwords stored in `/etc/shadow`, which is owned by `root`
- To change my password, I need to edit that file!
- Maybe add a syscall to kernel?
  - We’d have to add a ton of syscalls… violating P8: Small Trusted Base

**Solution:** Special permission on a program that allows anyone to “run it as root.”

(Actually, anyone can run file with owner as uid.)
The End