OS Security Basics

CS642: Computer Security

Professor Ristenpart

http://www.cs.wisc.edu/~rist/

rist at cs dot wisc dot edu
We start with some basics about operating system security:

- Multics
- Multi-level security
- Security policies
- Access controls
- UNIX permissions
Take yourself back to the 1960’s

http://fyeahhippies.tumblr.com/post/135907376
Take yourself back to the 1960’s

Time-share multiuser computers coming into use

GE-645
36 bit address space
Up to 4 processors
Magnetic tape drives
Supported virtual memory in hardware

Courtesy of http://aficionadous.blogspot.com/
Multiplexed Information and Computing Service (Multics)

Project to develop operating system for time-shared systems

- MIT project MAC, Bell Labs, and GE
- ~100 installations at greatest extent
- Last one shut down in 2000 (Canadian department of defense)

“A small but useful hardware complement would be 2 CPU units, 128K of core, 4 million words of high speed drum, 16 million words of disc, 8 tapes, 2 card readers, 2 line printers, 1 card punch and 30 consoles.”

[Vyssotsky, Corbato, Graham 1965]
Multics: ancestor to many OS’s

Lots of innovations in design

- Use of segmentation and virtual memory with hardware support
- SMP (shared memory multiprocessor)
- Written in PL/1 (high level language)

Significant attention paid to security
Multi-level security

• Military and other government entities want to use time-sharing too
Classification levels

- Top secret
- Secret
- Confidential
- Unclassified
Classification levels and compartmentalization

Top secret
Secret
Confidential
Unclassified

European
Special intelligence
Classification levels and compartmentalization

• Security level (L,C)
  – L is classification level (Top secret, secret, ...)
  – C is compartment (Europe, Special intelligence...)

Dominance relationship:

\( (L_1, C_1) \leq (L_2, C_2) \)

\( L_1 < L_2 \)

\( C_1 \) subset of \( C_2 \)
Bell-Lapadula Confidentiality Model

“no reads up”, “no writes down”

<table>
<thead>
<tr>
<th>Simple security condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>User with ((L_1,C_1)) can read file with ((L_2,C_2)) if</td>
</tr>
<tr>
<td>((L_1,C_1) \geq (L_2,C_2))</td>
</tr>
<tr>
<td>((L_1,C_1) \leq (L_2,C_2))</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>*-property</th>
</tr>
</thead>
<tbody>
<tr>
<td>User with ((L_1,C_1)) can write file with ((L_2,C_2)) if</td>
</tr>
<tr>
<td>((L_1,C_1) \geq (L_2,C_2))</td>
</tr>
<tr>
<td>((L_1,C_1) \leq (L_2,C_2))</td>
</tr>
</tbody>
</table>
Say we have just Bell-Lapadula in effect... what could go wrong?
Biba integrity model

“no read down”, “no writes up”

Simple integrity condition

User with $(L_1,C_1)$ can read file with $(L_2,C_2)$ if

$$(L_1,C_1) \leq (L_2,C_2)$$

$$(L_1,C_1) \not\geq (L_2,C_2)$$

*-property

User with $(L_1,C_1)$ can write file with $(L_2,C_2)$ if

$$(L_1,C_1) \not\leq (L_2,C_2)$$

$$(L_1,C_1) \geq (L_2,C_2)$$
If we combine them... one can only communicate in same classification
Other policy models

• Take-grant protection model
• Chinese wall
• Clarke-Wilson integrity model
• etc.

A good reference is:
Bishop, Computer Security: Art and Science
Multics: ancestor to many OS’s

Lots of innovations in design

• Use of segmentation and virtual memory with hardware support
• SMP (shared memory multiprocessor)
• Written in PL/1 (high level language)

Significant attention paid to security
Multics: security mechanisms

Protection rings 0-7 in which processes execute

• Lower number = higher privilege
• Ring 0 is “hardcore” supervisor
• Inherit privileges over higher levels

Protection rings included in all typical CPUs today and used by all operating systems
Multics: security mechanisms

Segments

- Virtual memory
- Program and data items stored in a segment
- Descriptor control field (read only, write only, execute only, ...)
- Segments access controlled
Multics: security mechanisms

Enciphered passwords

- Couldn’t find the algorithm
- Later ones used DES, but Multics predates DES

\[ pw = 12345 \]

\[ pw \rightarrow h(pw) \]
From reading:
A Large-Scale Study of Web Password Habits, by Florencio and Herley
Karger and Schell: security analysis of Multics

• Classic red teaming example

We have concluded that AFDSC cannot run an open multi-level secure system on Multics at this time. As we have seen above, a malicious user can penetrate the system at will with relatively minimal effort. However, Multics does provide AFDSC with a basis for a benign multi-level system in which all users are determined to be trustworthy to some degree. For example, with certain enhancements, Multics could serve AFDSC in a two-level security mode with both Secret and Top Secret cleared users simultaneously accessing the system. Such a system, of course, would depend on the administrative determination that since all users are cleared at least to Secret, there would be no malicious users attempting to penetrate the security controls.
In the long term, it is felt that Multics can be developed into an open secure multi-level system by restructuring the operating system to include a security kernel. Such restructuring is essential since malicious users cannot be ruled out in an open system. The
Reference monitors / security kernels

• System component that monitors (hopefully all) accesses to data for security violations

• Reference monitors may be:
  – kernel
  – hypervisor
  – within applications (Apache)
Circumventing access controls: covert channels

\[(L_1, C_1) \geq (L_2, C_2)\]

- Process 1 (L1,C1)
- Process 2 (L2,C2)
- Reference monitor

send M to Process 2

Fail
Circumventing access controls: covert channels

\[(L1, C1) \geq (L2, C2)\]

**Process 1** 
(L1, C1)

- Write to my file on disk
- Ok

**Reference monitor**

**Process 2** 
(L2, C2)

- Read from my file on disk
- Ok

**Hard disk**

### Process 1
- Sends a 1 bit to Process 2 by writing lots of bits to files it controls on hard disk
- Sends a 0 bit by idling

### Process 2
- Measures time to read from its files on disk
- Longer read time = 1 bit sent
- Shorter read time = 0 bit sent
Covert channels one of unsolved MLS problems
Access controls
## Access control matrix

<table>
<thead>
<tr>
<th>Subjects</th>
<th>file 1</th>
<th>file 2</th>
<th>...</th>
<th>file n</th>
</tr>
</thead>
<tbody>
<tr>
<td>user 1</td>
<td>read, write</td>
<td>read, write, own</td>
<td>read</td>
<td></td>
</tr>
<tr>
<td>user 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>user m</td>
<td>append</td>
<td>read, execute</td>
<td>read, write, own</td>
<td></td>
</tr>
</tbody>
</table>

User $i$ has permissions for file $j$ as indicated in cell $[i,j]$

Due originally to Lampson in 1971
Two common implementation paradigms

<table>
<thead>
<tr>
<th></th>
<th>file 1</th>
<th>file 2</th>
<th>...</th>
<th>file n</th>
</tr>
</thead>
<tbody>
<tr>
<td>user 1</td>
<td>read, write</td>
<td>read, write, own</td>
<td>read</td>
<td></td>
</tr>
<tr>
<td>user 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>user m</td>
<td>append</td>
<td>read, execute</td>
<td>read, write, own</td>
<td></td>
</tr>
</tbody>
</table>

1) Access control lists
   Column stored with file

2) Capabilities
   Row stored for each user
   Unforgeable tickets given to user
## ACLs compared to Capabilities

<table>
<thead>
<tr>
<th>ACLs requires authenticating user</th>
<th>Token-based approach avoids need for auth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processes must be given permissions</td>
<td>Tokens can be passed around</td>
</tr>
<tr>
<td>Reference monitor must protect permission setting</td>
<td>Reference monitor must manage tokens</td>
</tr>
</tbody>
</table>
UNIX-style file system
UNIX-style file system ACLs

Permissions:
- Directory?
- Owner (r,w,x), group (r,w,x), all (r, w, x)

Owner (rist)
Group (staff)
Who uses capabilities?

- Amoeba: distributed operating system (1990’s)
- Eros (extremely reliable operating system)

- IBM System 38
- Intel iAPX 432

Capabilities are used in various ways inside modern systems all over

(From Wikipedia)
Delegation

Need to give a process, other user access

In ACL, process run by user inherits user’s permissions

In Cap, process can pass around token
Revocation

Take away access from user or process

In ACL, remove user from list

In Cap, more difficult

Reference monitor must know where tokens are

Using pointer indirection
Permissions:
- Directory?
- Owner (r,w,x), group (r,w,x), all (r, w, x)

Owner (rist)

Group (staff)
Roles (groups)

Group is a set of users

Administrator  User  Guest

Simplifies assignment of permissions at scale

User 1  Administrator  /etc/passwd
User 2  User  /usr/local/
User 3  Guest  /tmp/
UNIX file permissions

- Owner, group
- Permissions set by owner / root
- Resolving permissions:
  - If user=owner, then owner privileges
  - If user in group, then group privileges
  - Otherwise, all privileges
UNIX Process permissions

- Process (normally) runs with permissions of user that invoked process

/etc/passwd is owned by root

Users shouldn’t be able to write to it generally
How do you reset your password?
Process permissions continued

UID 0 is root

Real user ID (RUID) --
same as UID of parent (who started process)

Effective user ID (EUID) --
from set user ID bit of file being executed or due to sys call

Saved user ID (SUID) --
place to save the previous UID if one temporarily changes it

Also SGID, EGID, etc..
Executable files have 3 setuid bits

- Setuid bit – set EUID of process to owner’s ID
- Setgid bit – set EGID of process to group’s ID
- sticky bit:
  - 0 means user with write on directory can rename/remove file
  - 1 means only file owner, directory owner, root can do so

So passwd is a setuid program

program runs at permission level of owner, not user that runs it
How do you reset your password?
seteuid system call

uid = getuid();
eid = geteuid();
seteuid(uid);  // Drop privileges
...
seteuid(eid);  // Raise privileges
file = fopen("/etc/passwd", "w");
...
seteuid(uid);  // drop privileges

seteuid can:
- go to SUID or RUID always
- any ID if EUID is 0
Details of setuid more complicated

Chen, Wagner, Dean “Setuid Demystified”

(a) An FSA describing setuid in Linux 2.4.18

Figure 4: Three finite state automata describing the setuid system call in Linux, Solaris, and FreeBSD, respectively.

Ellipses represent states of the FSA, where a notation like "R=1,E=0,S=1" indicates that euid = 0 and ruid = suid.

Each transition is labelled with the system call it corresponds to. To avoid cluttering the diagram, we omit the error states and (in Linux) the capability bits that otherwise would appear in our deduced model.
Setuid allows necessarily privilege escalation but...

- Source of many privilege escalation vulnerabilities

Buffer overflow (next lecture) in local setuid program gives privilege escalation

Race conditions
Race conditions
Time-of-check-to-time-of-use (TOCTTOU)

if( access("/tmp/myfile", R_OK) != 0 ) {
   exit(-1);
}

file = open( "/tmp/myfile", "r" );
read( file, buf, 100 );
close( file );
print( "%s\n", buf );
Say program is setuid root:
access checks RUID, but open only checks EUID

```c
access("/tmp/myfile", R_OK)

ln –s /tmp/myfile /home/root/.ssh/id_rsa

open( "tmp/myfile", "r" );

print( "%s\n", buf );
```

Prints out the root’s secret key…
Better code

euid = geteuid();
ruid = getuid();
seteuid(ruid); // drop privileges
file = open("/tmp/myfile", "r");
read(file, buf, 100);
close(file);
print("%s\n", buf);
Summary

• Multics: seminal multi-user operating system
  – many security features
  – significant auditing performed, achieved high security certifications
• MLS security principles
  – covert channels
• Access controls (matrixes, ACLs, capabilities)
• UNIX style file and process permissions