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

Professor Trent Jaeger
MAC in Commercial OSes

• We have learned that MAC is necessary to enforce security

• How do we add MAC enforcement effectively to a commercial OS?
Linux

- Circa 2000 - Uses traditional UNIX “Mode Bits”
Security Concerns

- Various attacks were being launched against Windows systems, essentially compromising all.
- Concerns that Linux could also be prone
  - “Inevitability of Failure” paper
    - Any system with network facing daemons running as root was likely vulnerable
  - Why is that?
Security Concerns

• Various attacks were being launched against Windows systems, essentially compromising all
• Concerns that Linux could also be prone
  ‣ “Inevitability of Failure” paper
    • Any system with network facing daemons running as root was likely vulnerable
  ‣ What can we do?
Approx. Secure OS

• Maybe Linux cannot be a “secure” OS, but perhaps we can approximate a secure OS closely enough
  ‣ What is required to be a secure OS?
• Security Policy
  ‣ Info Flow or Least Privilege?
• Reference Monitor
  ‣ Complete Mediation, Tamperproof, Validation
• Formal Assurance
  ‣ Validate that OS with reference monitor implementation enforces security policy
• Can we do this?
Approx. Secure OS

- **Secure Linux Project** - 2001
- Group of systems security researchers working on refactoring various security features into Linux
  - But, especially a reference monitor
- A variety of different projects were underway
  - Argus Pitbull, Security-Enhanced Linux, Subdomain (AppArmor), grsecurity, RSBAC, …
- Presented ideas to Linus
  - All were different
  - Each group argued that its idea was best
- **What would you do if you were Linus?**
Linux Security Modules

• “All problems in computer science problem can be solved by another level of indirection”
  ‣ Attributed to Butler Lampson

• Linus asked for another level of indirection to host access control enforcement
  ‣ And the Linux Security Modules project was born
Linux Security Modules

- Defines a authorization interface to enable a chosen security module to make access control decisions
- Focus on mediation
- Let LSM module implementations determine the security policy and how they satisfy the reference monitor concept
Reference Monitor

• Defines a set of requirements on reference validation mechanisms
  ‣ To enforce access control policies correctly
• Complete mediation
  ‣ The *reference validation mechanism* must always be invoked (before executing security-sensitive operations)
• Tamperproof
  ‣ The *reference validation mechanism* must be tamperproof
• Verifiable
  ‣ The *reference validation mechanism* must be small enough to be subject to analysis and tests, the completeness of which can be assured
Access Policy Enforcement

• A protection system uses a reference validation mechanism to produce and evaluate authorization queries

  ‣ Interface: Mediate security-sensitive operations by building authorization queries to evaluate
  ‣ Module: Determine relevant protection state entry (ACLs, capabilities) to evaluate authorization query
  ‣ Manage: Manage the assignment of objects and subjects (processes) to the protection state

• How do we know whether a reference validation mechanism is correct?
Security-Sensitive Operations

• Broadly, operations that enable interaction among processes that violate secrecy, integrity, availability

• Which of these are security-sensitive? Why?
  ‣ Read a file (read)
  ‣ Get the process id of a process (getpid)
  ‣ Read file metadata (stat)
  ‣ Fork a child process (fork)
  ‣ Get the metadata of a file you have already opened? (fstat)
  ‣ Modify the data segment size? (brk)

• Require protection for all of CIA?
Linux Security Modules

• Reference validation mechanism for Linux
  ▸ Upstreamed in Linux 2.6
  ▸ Support modular enforcement - you choose
    • SELinux, AppArmor, POSIX Capabilities, SMACK, ...

• 150+ authorization hooks
  ▸ Mediate security-sensitive operations on
    • Files, dirs/links, IPC, network, semaphores, shared memory, ...
  ▸ Variety of operations per data type
    • Control access to read of file data and file metadata separately

• Hooks are restrictive - in addition to DAC security
Linux Security Modules

```c
#include <fcntl.h>

#define FILE_SECURITY_LEVEL 0

ssize_t vfs_read(...) {
    ...
    ret = security_file_permission(file, ...);
    if (!ret) {
        ret = file->f_op->read(file, ...);
    }
    ...
}
```

Security check function

Security sensitive operation
Linux Security Modules

- Register (install) module
- Load policy (open and write to special file)
- Produce authorization queries at hooks
Linux Security Modules

- Attacks on “register”
- Attacks on “install policy”
- Attacks on “system calls”
Linux Security Modules

- To prevent attacks on registration
- And attacks on function pointers of LSM
- LSMs are now statically compiled into the kernel
Does LSM satisfy reference monitor concept?
LSM & Reference Monitor

- Does LSM satisfy reference monitor concept?
  ◦ Tamperproof
  - Can MAC policy be tampered?
  - Can kernel be tampered?
DAC

- Suppose J owns $O_1$ and $O_2$ - Is $O_1$ secret in a DAC system?

<table>
<thead>
<tr>
<th></th>
<th>$O_1$</th>
<th>$O_2$</th>
<th>$O_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>J</td>
<td>R</td>
<td>R</td>
<td>RW</td>
</tr>
<tr>
<td>$S_2$</td>
<td>-</td>
<td>R</td>
<td>RW</td>
</tr>
<tr>
<td>$S_3$</td>
<td>-</td>
<td>R</td>
<td>RW</td>
</tr>
</tbody>
</table>
Access Control Administration

There are two central ways to manage a policy

1. Discretionary - Object “owners” define policy
   - Users have discretion over who has access to what objects and when (trusted users)
   - Canonical example, the UNIX filesystem
     - RWX assigned by file owners

2. Mandatory - Environment defines policy
   - OS distributor and/or administrators define a system policy that cannot be modified by normal users (or their processes)
   - Typically, information flow policies are mandatory
     - More later…
LSM & Reference Monitor

- Does LSM satisfy reference monitor concept?
  - Tamperproof
    - Can MAC policy be tampered?
    - Can kernel be tampered?
  - Verifiable
    - How large is kernel?
    - Can we perform complete testing?
LSM & Reference Monitor

• Does LSM satisfy reference monitor concept?
  ‣ Tamperproof
    • Can MAC policy be tampered?
    • Can kernel be tampered?
  ‣ Verifiable
    • How large is kernel?
    • Can we perform complete testing?
  ‣ Complete Mediation
    • What is a security-sensitive operation?
    • Do we mediate all paths to such operations?
LSM & Complete Mediation

• What is a security-sensitive operation?
  ‣ Instructions? Which?
  ‣ Structure member accesses? To what data?
  ‣ Data types whose instances may be controlled?
    • Inodes, files, IPCs, tasks, ...

• Approaches
  ‣ Mediation: Check that authorization hook dominates all control-flow paths to structure member access on security-sensitive data type
  ‣ Consistency: Check that every structure member access that is mediated once is always mediated
    • Several bugs found - some years later
LSM & Complete Mediation

- Static analysis of Zhang, Edwards, and Jaeger [USENIX Security 2002]
  - Based on a tool called CQUAL
- Found a TOCTTOU bug
  - Authorize filp in sys_fcntl
  - But pass fd again to fcntl_getlk
- Many supplementary analyses were necessary to support CQUAL

```c
/* from fs/fcntl.c */
long sys_fcntl(unsigned int fd,
              unsigned int cmd,
              unsigned long arg)
{
    struct file * filp;
    ...
    filp = fget(fd);
    ...
    err = security_ops->file_ops
         ->fcntl(filp, cmd, arg);
    ...
    err = do_fcntl(fd, cmd, arg, filp);
    ...
}
static long
do_fcntl(unsigned int fd,
         unsigned int cmd,
         unsigned long arg,
         struct file * filp) {
...
switch(cmd){
  case F_SETLK:
    err = fcntl_setlk(fd, ...);}
...
}

/* from fs/locks.c */
cntl_getlk(fd, ...) { struct file * filp; ...
    filp = fget(fd);
    /* operate on filp */
    ...}
```

Figure 8: Code path from Linux 2.4.9 containing an exploitable type error.
LSM Enforcement

• Several LSMs have been deployed
  ‣ Most prominent: AppArmor, SELinux, Smack, TOMOYO
• The most comprehensive is SELinux
  ‣ Used by RedHat Fedora and some others
LSM Enforcement

• Several LSMs have been deployed
  ‣ Most prominent: AppArmor, SELinux, Smack, TOMOYO

• The most comprehensive is SELinux
  ‣ Created by the NSA - Result of many years work
  ‣ Used by RedHat Fedora and some others
SELinux Challenges

• (1) Protection state definition
  ‣ Per program access control policy
  ‣ Thousands of rules - produced by runtime auditing

• (2) Assigning objects and subjects (processes) to labels
  ‣ Policy module per program on install
  ‣ Control how a new program obtains its label
    • Different approach to setuid problem
Setuid Problem

• In Setuid, program runs with UID of file owner
  ‣ Usually root, so too many permissions
    • SELinux - run with permissions of program
  ‣ Anyone can start any setuid program
    • Limit to authorized processes by label
SELinux Transition State

- 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)
Take Away

• **Goal**: Build authorization into operating systems
  ‣ Multics and Linux

• **Requirements**: Reference monitor
  ‣ Satisfy reference monitor concept

• Multics
  ‣ Hierarchical Rings for Protection
  ‣ Call/Access Bracket Policies (in addition to MLS)

• Linux
  ‣ Did not enforce security *(DAC, Setuid, root daemons)*
  ‣ So, the Linux Security Modules framework was added
  ‣ Approximates reference monitor assuming network threats only -- some challenges in ensuring complete mediation