CS 423Operating System Design: Introduction to Linux Kernel Programming (MP4 Walkthrough)

Jack Chen

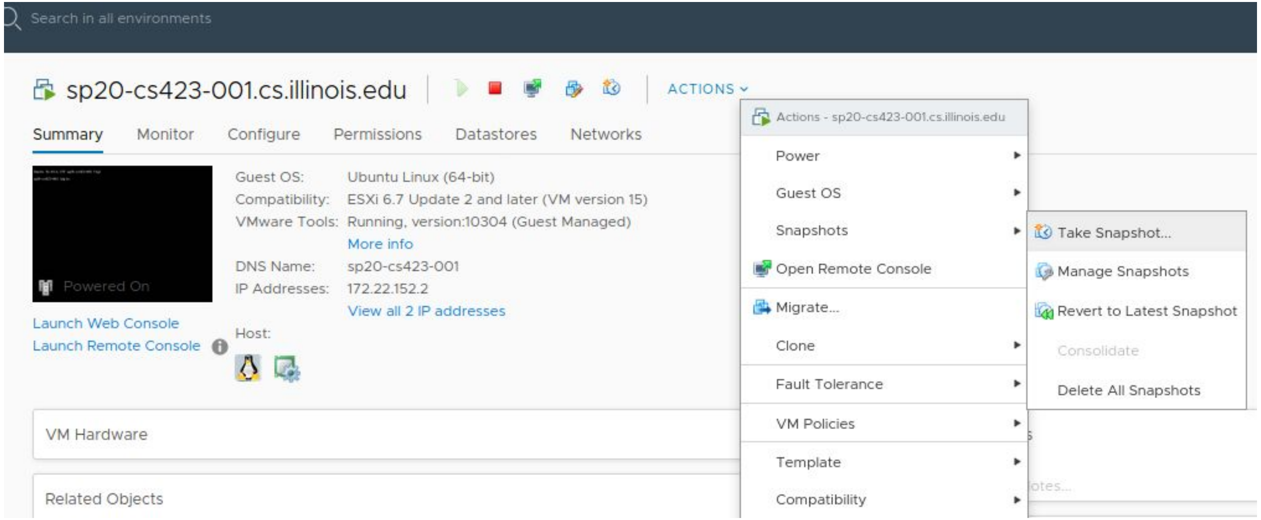
Some content taken from a previous year's walkthrough by Prof. Adam Bates

Preliminaries

Take stable snapshots before starting this MP

Your security module will affect kernel boot
- Work incrementally
- Start with empty functions, add logic in small doses

How to Take a SnapShot

1. Connect to school vpn
2. Login https://vc.cs.illinois.edu/ui/
3. 

Goals of this MP

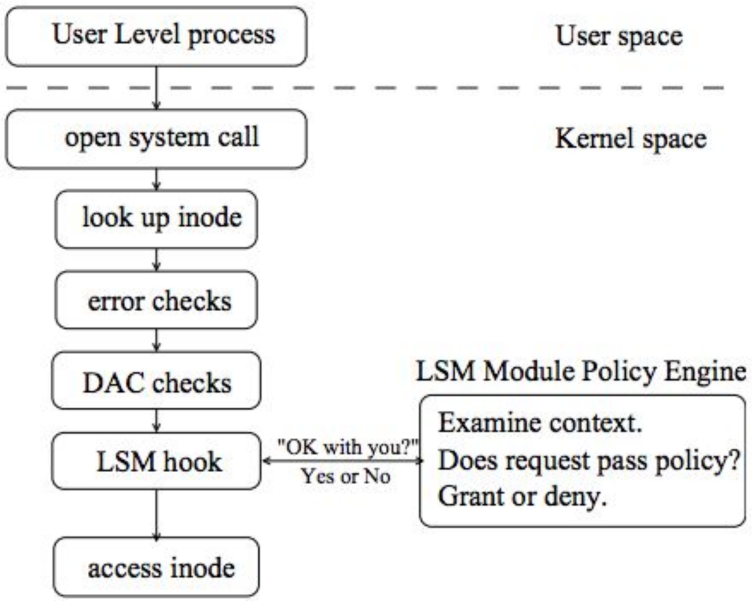
- Understand Linux Security Modules
- Understand basic concepts behind Mandatory Access Control (MAC)
- Understand and use filesystem extended attributes
- Add custom kernel configuration parameters and boot parameters
- Derive a least privilege policy for /usr/bin/passwd

Linux Security Module

- Came out of a presentation that the NSA did in 2001
  - Security Enhanced Linux (SELinux)
- Kernel provided support for Discretionary Access Control
- Did not provide framework for different security models w/o changes to core kernel code
- Linux Security Modules (LSM) proposed as a solution
  - Not to be fooled by the term “module”
  - LSMs are NOT* loadable at runtime

How Do LSMs Work?

Hooks inserted throughout important functionalities of the kernel



In which context does the LSM run?

In the kernel context just before the kernel fulfills a request

```
union security_list_options {
        int (*binder_set_context_mgr)(struct task_struct *mgr);
        int (*binder_transaction)(struct task_struct *from,
                                        struct task_struct *to);
        int (*binder_transfer_binder)(struct task_struct *from,
                                        struct task_struct *to);
        int (*binder_transfer_file)(struct task_struct *from,
                                        struct task_struct *to,
                                        struct file *file);

        int (*ptrace_access_check)(struct task_struct *child,
                                        unsigned int mode);
        int (*ptrace_traceme)(struct task_struct *parent);
        int (*capget)(struct task_struct *target, kernel_cap_t *effective,
                        kernel_cap_t *inheritable, kernel_cap_t *permitted);
        int (*capset)(struct cred *new, const struct cred *old,
                        const kernel_cap_t *effective,
                        const kernel_cap_t *inheritable,
                        const kernel_cap_t *permitted);
        int (*capable)(const struct cred *cred, struct user_namespace *ns,
```

Major and Minor LSM

- Major LSM
  - Only one major LSM can run in the system
  - Examples: SELinux, Smack, etc.
  - Can access opaque security fields (blobs)
    - Data structures reserved only for the use of major LSMs
- Minor LSM
  - Can be stacked
  - Does not have access to the security blobs
  - Examples: YAMA

What is “Security Blobs” ?

- Reserved fields in various kernel data structures
  - task_struct, inode, sk_buff, file, linux_binprm
- Controlled by the major security module running
  - struct cred is the security context of a thread
  - task->cred->security is the tasks’s security blob
  - A task can only modify its own credentials
    - No need for locks in this case!
    - Need rcu read locks to access another tasks’s credentials

MAC, Mandatory Access Control

- Access rights are based on regulations defined by a central authority
- Strictly enforced by the kernel
- Label objects by sensitivity
  - e.g., unclassified, confidential, secret, top secret
- Label users (subjects) by, e.g., clearance
- Grant access based on combination of subject and object labels

Labeling our System

- We will developed a major security module
- To keep things simple, we will focus on tasks that carry the label target
- We will focus on only labeling inodes
  - We can use the security blobs
  - We will also use extended filesystem attributes
- How do we label our tasks then?
  - We will use the inode label of the binary that is used to launch the process

File System Extended Attributes

- Provides custom file attributes that are not interpreted by the file system
- Convention: attributes under the prefix security will be used for interpretation by an LSM
- We will be using security.mp4
- Set an attribute:
  - setfattr -n security.mp4 -v target target_binary
  - setfattr -n <prefix>.<suffix> -v <value> <file>
- List attributes:
  - getfattr -d -m - <file>

```
jianyan2@sp20-cs423-001:~$ sudo setfattr -n security.mp4 -v target userapp
[sudo] password for jianyan2:
jianyan2@sp20-cs423-001:~$ getfattr -d -m - userapp
# file: userapp
security.mp4="target"

jianyan2@sp20-cs423-001:~$
```

MP4 Challenges

- Label management
  - How to assign and maintain labels
  - How to transfer labels from inodes to tasks
- Access control
  - Who gets to access what
  - Enforce MAC policy
- Kernel configuration
  - Kconfig environment
  - Change boot parameters

Step 1: Compile your kernel

- Customize kernel configuration using the Kconfig environment
- Go to the linux source code folder in MP0
- Add custom config option to security/mp4/Kconfig

```
config SECURITY_MP4_LSM
        bool "CS423 machine problem 4 support"
        depends on NET
        depends on SECURITY
        select NETLABEL
        select SECURITY_NETWORK
        default n
        help
          This selects the cs423 machine problem 4 security lsm to be
          compiled with the kernel.
          If you are unsure how to answer this question, answer N.
```

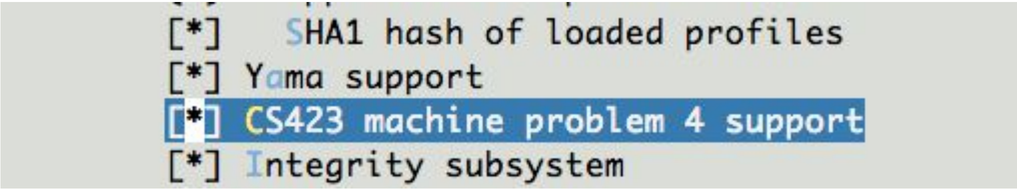
Step 1: Compile your kernel

- Now when you run make oldconfig, make will ask you whether to enable
  - CONFIG_SECURITY_MP4_LSM
- You can also use it for static compiler macros in your code. e.g.

```
#ifdef CONFIG_SECURITY_MP4_LSM
void do_something(void) { printf("MP4 active\n"); }
#else
void do_something(void) { }
#endif
```

Step 1: Compile your kernel

- You can also use make menuconfig to see your config option visually




```
> make menuconfig
```

In linux source code root level

- You might want to turn DEBUG_INFO off to speed up the generation of the .deb files

Step 1: Compile your kernel

- After the first compilation, you do not need to recompile the entire kernel again
- Reminder: make clean removes all of the object files and will cause the entire kernel to be recompiled
- For incremental builds, just: `make`
- To produce .deb files again:
- make bindeb-pkg LOCALVERSION=…

Step 1: Boot params

- Next we want to enable the mp4 module as the major security module in our system

- The kernel accepts the key-value pair security=<module> as part of its boot parameters

- Update /etc/default/grub:

    GRUB_CMDLINE_LINUX_DEFAULT="security=mp4"

- sudo update-grub (Don't forget this)

Step 2.0: Implementation

- We will implement our module in three steps:

    1. Register the module and enable it as the only major security module (Provided to you at no cost in mp4.c)
    2. Implement the labels initialization and management
    3. Implement the mandatory access control logic

- We provide you with helper functions in mp4_given.h

- Use pr_info, pr_err, pr_debug, pr_warn macros

- #define pr_fmt(fmt) "cs423_mp4: " fmt

Step 2.1: Startup

- We provide you with the startup code to get your started
- We will implement the following security hooks:

```
static struct security_hook_list mp4_hooks[] = {

    LSM_HOOK_INIT(inode_init_security, mp4_inode_init_security),
    LSM_HOOK_INIT(inode_permission, mp4_inode_permission),

    LSM_HOOK_INIT(bprm_set_creds, mp4_bprm_set_creds),

    LSM_HOOK_INIT(cred_alloc_blank, mp4_cred_alloc_blank),
    LSM_HOOK_INIT(cred_free, mp4_cred_free),
    LSM_HOOK_INIT(cred_prepare, mp4_cred_prepare)
};
```

Step 2.2: Label Semantics - Test Points!

```c
/* mp4 labels along with their semantics */
#define MP4_NO_ACCESS 0       /* may not be accessed by target,
                               * but may by everyone other */

#define MP4_READ_OBJ 1        /* object may be read by anyone */

#define MP4_READ_WRITE 2      /* object may read/written/appended by the target,
                               * but can only be read by others */

#define MP4_WRITE_OBJ 3       /* object may be written/appended by the target,
                               * but not read, and only read by others */

#define MP4_EXEC_OBJ 4        /* object may be read and executed by all */

/* NOTE: FOR DIRECTORIES, ONLY CHECK ACCESS FOR THE TARGET SID, ALL OTHER NON
 * TARGET PROCESSES SHOULD DEFAULT TO THE LINUX REGULAR ACCESS CONTROL
 */
#define MP4_READ_DIR 5        /* for directories that can be read/exec/access
                               * by all */

#define MP4_RW_DIR 6          /* for directory that may be modified by the target
                               * program */
```

Step 2.2: Label Map

```
if (strcmp(cred_ctx, "read-only") == 0)
	return MP4_READ_OBJ;
else if (strcmp(cred_ctx, "read-write") == 0)
	return MP4_READ_WRITE;
else if (strcmp(cred_ctx, "exec") == 0)
	return MP4_EXEC_OBJ;
else if (strcmp(cred_ctx, "target") == 0)
	return MP4_TARGET_SID;
else if (strcmp(cred_ctx, "dir") == 0)
	return MP4_READ_DIR;
else if (strcmp(cred_ctx, "dir-write") == 0)
	 return MP4_RW_DIR;
else
	return MP4_NO_ACCESS;
```

Step 2.2: Label Management

- We are interested in three operations:
  1. Allocate/free/copy subject security blobs
  2. When a process starts, check the inode of the binary that launches it.
     a. If it is labeled with target, mark task_struct as target
     b. mp4_bprm_set_creds
  3. Assign read-write label to inodes created by the target application
     a. mp4_inode_init_security

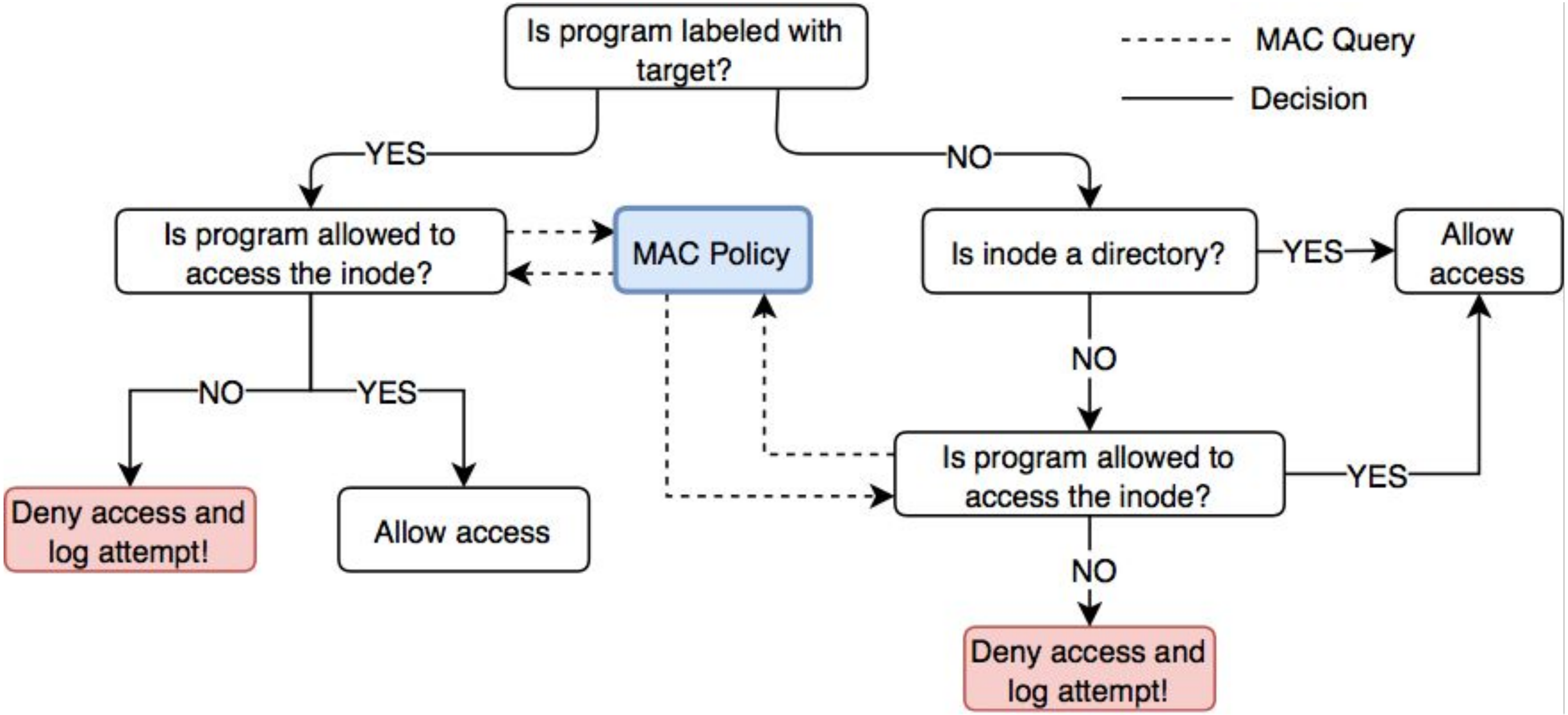
Step 2.2: Obtain Inode's extended Attributes

- Given an struct inode *, we can ask for its struct dentry *
- You can query some kernel functions if there is something you need to know
  - This is important if you don't know how much memory to allocate
  - Watch for the ERANGE errno
- It is very important to put back a dentry after you use it
  - dput(dentry);

Step 2.3: Implement Access Control

- Translate label semantics into code
  - mp4_inode_permission
- Operation masks are in linux/fs.h
- Obtain current task's subject blob using current_cred()
- To speed things up during boot, we want to skip certain directories
  - Obtain inode's path (hint: use dentry!)
  - Call mp4_should_skip_path from mp4_given.h

Step 2.3: Implement Access Control



Step 2.3: Implement Access Control

- You MUST log attempts that are denied access
- To minimize the chances of bricking your machine:
  - Always take a snapshot that takes you back to stable state
  - Implement AC logic, but always return access granted and print appropriate messages
  - Check you messages, if all is according to plan, update your code to return appropriate values
  - Test your return codes

Step 3: Testing

- Test your security module on simple functions
  - vim, cat, etc.
  - avoid operation critical programs (ls, cd, bash, etc.)
  - Note: to grant read access to /home/netid/file.txt …
    - must have access to all three of /home, /home/netid/, and /home/netid/file.txt
- Always restore your system state to a place where all labels are removed before you reboot

```
jianyan2@sp20-cs423-001:~$ sudo setfattr -x security.mp4 userapp
jianyan2@sp20-cs423-001:~$ getfattr -d -m - userapp
jianyan2@sp20-cs423-001:~$
```

Step 3: Testing

- Suggested method of testing:
  - Create two scripts: mp4_test.perm and mp4_test.perm.unload
  - source first script to load, source the other to unload
- In mp4_test.perm:
  - setfattr -n security.mp4 -v target /usr/bin/cat
  - …
  - setfattr -n security.mp4 -v read-only /home/netid/file.txt
- In mp4_test.perm.unload, undo everything before reboot:
  - setfattr -x security.mp4 /usr/bin/cat
  - ..
  - setfattr -x security.mp4 /home/netid/file.txt

Final Step: Obtain Policy

- Goal is to obtain least privilege policy for the program /usr/bin/passwd
- DO NOT TRY TO CHANGE THE PASSWORD FOR YOUR NETID ACCOUNT

```
jianyan2@sp20-cs423-001:~$ passwd
Current Kerberos password:
Current Password:
passwd: Authentication token manipulation error
passwd: password unchanged
jianyan2@sp20-cs423-001:~$
```

- Create dummy user account and change its password
- Use strace to obtain the set of files and access requests that passwd uses
  - sudo apt install strace
- Generate passwd.perm and passwd.perm.unload based on the outcome
- Test your module's enforcement of the policy!

Final Tips

- Where to turn when things get confusing?
  - There are 5 other LSM's in the source code of your kernel… use them as a reference!
    - AppArmor, SELinux, Smack, TOMOYO Linux, Yama
    - E.g. linux/security/yama/yama_lsm.c
  - The bookkeeping your LSM will need to do is very similar to what others need to do, because you are using the same interface.

Final Tips

- Your mp4_cred_alloc_blank hook will share many similarities with selinux_cred_alloc_blank… just don't blindly copy code without understanding it first, or you're going to create even more trouble for yourself!

```
static int selinux_cred_alloc_blank(struct cred *cred, gfp_t gfp)
{
        struct task_security_struct *tsec;

        tsec = kzalloc(sizeof(struct task_security_struct), gfp);
        if (!tsec)
                return -ENOMEM;

        cred->security = tsec;
        return 0;
}
```

Bye

```
jianyan2@sp20-cs423-001:~$ sudo adduser dummy
Adding user `dummy' ...
Adding new group `dummy' (1000) ...
Adding new user `dummy' (1000) with group `dummy' ...
The home directory `/home/dummy' already exists.  Not copying from `/etc/skel'.
Current Kerberos password:
New password:
BAD PASSWORD: The password is shorter than 8 characters
Retype new password:
Current Kerberos password:
Password unchanged
passwd: Authentication token manipulation error
passwd: password unchanged
Try again? [y/N]
Changing the user information for dummy
Enter the new value, or press ENTER for the default
        Full Name []: dummy
        Room Number []: -1
        Work Phone []: -1
        Home Phone []: -1
        Other []: -1
Is the information correct? [Y/n] Y
jianyan2@sp20-cs423-001:~$
jianyan2@sp20-cs423-001:~$
jianyan2@sp20-cs423-001:~$ users
jianyan2
jianyan2@sp20-cs423-001:~$ su dummy
Password:
su: Authentication failure
jianyan2@sp20-cs423-001:~$ sudo su dummy
dummy@sp20-cs423-001:/home/jianyan2$ exit
exit
jianyan2@sp20-cs423-001:~$ sudo deluser dummy
Removing user `dummy' ...
Warning: group `dummy' has no more members.
Done.
jianyan2@sp20-cs423-001:~$ sudo su dummy
No passwd entry for user 'dummy'
jianyan2@sp20-cs423-001:~$
```