University of BRISTOL

Linux Security Modules

Joseph Hallett

How do we get Linux to Grade B?

TSEC says Linux just offers discretionary access control
- Grade C (pushing into B with things like AppArmour)

How do we bring MAC policies to Linux?
- Needs to be small and testable and not impact people who don't want to use a MAC policy on their systems…

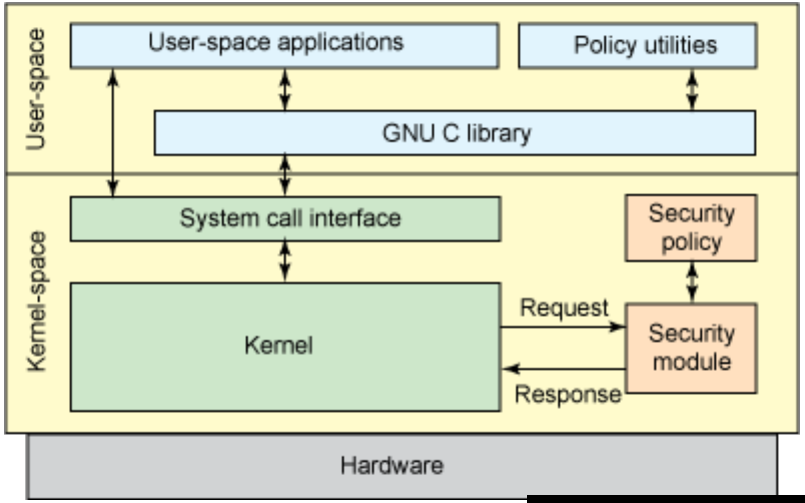
(This is/was a real research problem)

LSM Framework

Linux Security Module framework implements a reference monitor for Linux

Dynamically loadable kernel module hooks into system call checks

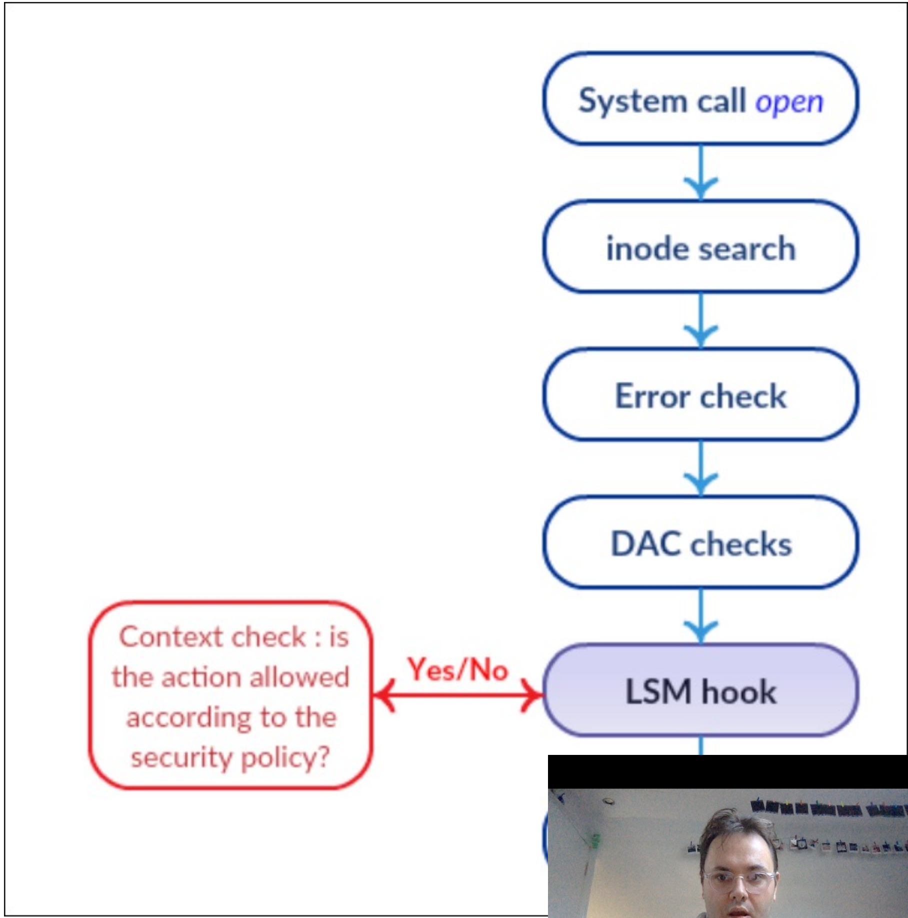
Framework is verified, modules are (in theory) small and verifiable



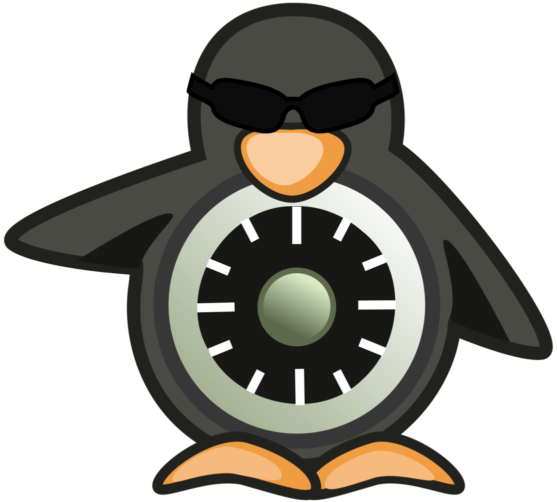
LSM Framework

Security hooks are invoked on the path between any subject action and an object
- For example, *processes* and *inodes*

Hook function returns access decision:
- `0`: access granted
- `ENOMEM`: no memory available
- `EPERM`: not enough privileges



SELinux



LSM framework developed (initially) by the NSA

- Open sourced in 2000
- Merged into Linux Kernel 2.6.0 in 2003
- Available in Fedora (Core 2) since 2004

Based around type-based enforcement and RBAC (role based access controls)

SELinux Hook Example

security/selinux/hooks.c

```c
/* file security operations */

static int selinux_revalidate_file_permission(struct file *file, int mask)
{
	const struct cred *cred = current_cred();
	struct inode *inode = file_inode(file);

	/* file_mask_to_av won't add FILE__WRITE if MAY_APPEND is set */
	if ((file->f_flags & O_APPEND) && (mask & MAY_WRITE))
		mask |= MAY_APPEND;

	return file_has_perm(cred, file,
			     file_mask_to_av(inode->i_mode, mask));
}

static int selinux_file_permission(struct file *file, int mask)
{
	struct inode *inode = file_inode(file);
	struct file_security_struct *fsec = selinux_file(file);
	struct inode_security_struct *isec;
	u32 sid = current_sid();

	if (!mask)
		/* No permission to check.  Existence test. */
		return 0;

	isec = inode_security(inode);
	if (sid == fsec->sid && fsec->isid == isec->s
	    fsec->pseqno == avc_policy_seqno(&selinux
		/* No change since file_open check. *
		return 0;

	return selinux_revalidate_file_permission(fil
}
```

bristol.ac.uk

SELinux Hook Example

security/selinux/hooks.c

```c
/* Check whether a task can use an open file descriptor to
   access an inode in a given way.  Check access to the
   descriptor itself, and then use dentry_has_perm to
   check a particular permission to the file.
   Access to the descriptor is implicitly granted if it
   has the same SID as the process.  If av is zero, then
   access to the file is not checked, e.g. for cases
   where only the descriptor is affected like seek. */
static int file_has_perm(const struct cred *cred,
                         struct file *file,
                         u32 av)
{
        struct file_security_struct *fsec = selinux_file(file);
        struct inode *inode = file_inode(file);
        struct common_audit_data ad;
        u32 sid = cred_sid(cred);
        int rc;

        ad.type = LSM_AUDIT_DATA_FILE;
        ad.u.file = file;

        if (sid != fsec->sid) {
                rc = avc_has_perm(&selinux_state,
                                  sid, fsec->sid,
                                  SECCLASS_FD,
                                  FD__USE,
                                  &ad);
                if (rc)
                        goto out;
        }

#ifdef CONFIG_BPF_SYSCALL
        rc = bpf_fd_pass(file, cred_sid(cred));
        if (rc)
                return rc;
#endif

        /* av is zero if only checking
        rc = 0;
        if (av)
                rc = inode_has_perm(cr

out:
        return rc;
}
```

bristol.ac.uk

SELinux Hooks

Management hooks
- Called to handle object lifecycle
- E.g. `security_inode_allocate` or `security_inode_free`
- Used to manage security information

Path-based hooks
- Related to pathnames (used to implement TSEC B1)

Object-based hooks
- Path kernel structure corresponding to objects (used to implement TSEC B2)

SELinux Pseudo Filesystem

Need a mechanism to interact with SELinux from userland

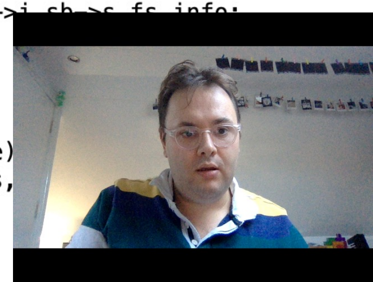
- To load policies and configuration
- To get audit data

Standard mountable virtual filesystem (like `/proc`)

- `security/selinux/selinuxfs.c`

```c
#define TMPBUFLEN	12
static ssize_t sel_read_enforce(struct file *filp, char __user *buf,
				size_t count, loff_t *ppos)
{
	struct selinux_fs_info *fsi = file_inode(filp)->
	char tmpbuf[TMPBUFLEN];
	ssize_t length;

	length = scnprintf(tmpbuf, TMPBUFLEN, "%d",
			   enforcing_enabled(fsi->state)
	return simple_read_from_buffer(buf, count, ppos,
}
```



SELinux Policies

All subjects get labeled with a security context
- User
- Domain
- Role

Rules describe what each *subject* domain can do with an *object* domain…

They get a bit complicated…

SELinux Policy Example

`/etc/passwd`
User information readable by any user

`/etc/shadow`
Password information readable only by root(ish)

SELinux Policy Example

/* Normal users are allowed to read normal files */

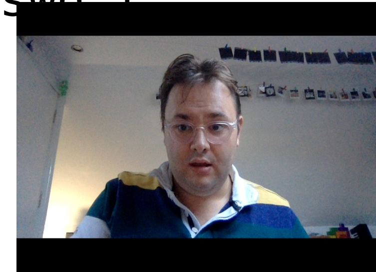
```
allow user_t public_t : file read
```

SELinux Policy Example

```
/* Normal users are allowed to read normal files */
allow user_t public_t : file read

/* Users in the password_t domain can r/w files in the
   password_data_t domain */
allow passwd_t passwd_data_t : file {read write}

/* Allow users to actually run the password program, and transition their domain */
allow user_t passwd_exec_t : file execute
allow user_t passwd_t : process transition
type transition user_t passwd_exec_t : process passwd_t
```



It's kinda complex!

5 lines of policy just to control access to 2 files

- Kinda makes sense…?

NSA gives a reference policy

- Hard to live with, and long (~20,000 rules)
- `echo 0 >/sys/fs/selinux/enforce`

Android implements its permissions system using its own SELinux policy

- Also long and confusing…

Rule design is really hard

SELinux Coloring Book

Rule design is hard…

So lets try and make it easy!

`https://people.redhat.com/duffy/selinux/selinux-coloring-book_A4-Stapled.pdf`