LXFI, a software fault isolation system

Junyuan Zeng

Department of Computer Science
The University of Texas at Dallas

April 4\textsuperscript{th}, 2012
Outline

1. Overview
   - problems

2. Software Fault Isolation
   - LXFI

3. Reference
   - reference
Outline

1. Overview
   - problems

2. Software Fault Isolation
   - LXFI

3. Reference
   - reference
Outline

1. Overview
   - problems

2. Software Fault Isolation
   - LXFI

3. Reference
   - reference
Linux kernel is vulnerable

- vulnerabilities in Linux are routinely discovered: 145 vulnerabilities in Linux kernel (CVE 2010)
- many exploits attack kernel modules: 67% of Linux kernel vulnerabilities (CVE 2010)
privilege escalation attack

- a type of attack that takes advantage of programming errors or design flaws to grant the attacker elevated access to the network and its associated data and applications
privilege escalation attack

- A type of attack that takes advantage of programming errors or design flaws to grant the attacker elevated access to the network and its associated data and applications.

- Example: buffer overflow, set current UID to root.
problems

one approach: type safe languages

- write kernel and modules in Java, C#
- no reference to UID object => cannot directly change UID
problems

one approach: type safe languages

- write kernel and modules in Java, C#
- no reference to UID object => cannot directly change UID

Most kernels are not written in type safe language!
Outline

1. Overview
   - problems

2. Software Fault Isolation
   - LXFI

3. Reference
   - reference
Outline

1. Overview
   - problems

2. Software Fault Isolation
   - LXFI

3. Reference
   - reference
software fault isolation system for Linux kernel modules

annotation language for:
- argument integrity: programmer controls what arguments a module can pass to functions
- callback integrity: kernel invokes callback only if the module could have invoked callback directly
- specifying privilege separation within a module
LXFI

Compile time:
- Programmer annotates core kernel
- LXFI translates annotations to runtime checks

Runtime:
- LXFI performs checks

When annotations capture all implicit rules, compromised module cannot violate rules to gain additional privileges.

Using compiler plugins:
- Provide safe default: reject a module if it calls an unannotated API

Consulting a dynamic table of capabilities for each module
example: privilege escalation attack

```c
void spin_lock_init(spinlock_t *lock) {
    lock->v = 0;
}
```

Core Kernel

Spin_module

Privilege escalation!
enforce argument integrity

- **spin_lock_init**: three annotations are required

<table>
<thead>
<tr>
<th>Part</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>capability</td>
<td><code>write(ptr,size)</code></td>
<td>Write <code>[ptr,ptr+size]</code></td>
</tr>
<tr>
<td>capability action</td>
<td><code>check(cap)</code></td>
<td>Checks <code>cap</code></td>
</tr>
<tr>
<td>location</td>
<td><code>pre(action)</code></td>
<td>Perform <code>action</code> before function call</td>
</tr>
</tbody>
</table>
example about enforce argument integrity

```c
void spin_lock_init(spinlock_t *lock)
pre(check(write(lock, sizeof(spinlock_t))))
```

- Core Kernel
- Spin_module
  - capability table
  - write(mylock, 8)
  - lxfi_check_write(mylock, 8);
  - spin_lock_init(mylock)
  - lxfi_check_write(&cur_proc->uid, 8);
  - spin_lock_init(&cur_proc->uid)

Privilege escalation prevented
where does the capability come from

- granted on allocation
- two more annotations are required

<table>
<thead>
<tr>
<th>Part</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>capability</td>
<td>write(ptr, size)</td>
<td>Write [ptr, ptr+size]</td>
</tr>
<tr>
<td>capability action</td>
<td>check(cap)</td>
<td>Check cap</td>
</tr>
<tr>
<td></td>
<td>copy(cap)</td>
<td>Grant a copy of cap</td>
</tr>
<tr>
<td>location</td>
<td>pre(action)</td>
<td>Perform action before function call</td>
</tr>
<tr>
<td></td>
<td>post(action)</td>
<td>Perform action after function return</td>
</tr>
</tbody>
</table>
Example: grant spinlock.

```c
void *kmalloc(size)
post(copy(write(return, size)))
```

```
spinlock_t *mylock = kmalloc(8);
lxfi_copy_write(mylock, 8);
```

Core Kernel

Spin_module

LXFI Runtime

Capability table

Write(mylock, 8)
where happens when memory is freed

- need to revoke capability to safely reuse memory
- one more annotations are required

<table>
<thead>
<tr>
<th>Part</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>capability</td>
<td>write(ptr,size)</td>
<td>Write [ptr,ptr+size]</td>
</tr>
<tr>
<td>capability action</td>
<td>copy(cap)</td>
<td>Grant a copy of cap</td>
</tr>
<tr>
<td>capability action</td>
<td>check(cap)</td>
<td>Check cap</td>
</tr>
<tr>
<td>capability action</td>
<td>transfer(cap)</td>
<td>Revoke cap from all modules, and grant</td>
</tr>
<tr>
<td>location</td>
<td>pre(action)</td>
<td>Perform action before function call</td>
</tr>
<tr>
<td>location</td>
<td>post(action)</td>
<td>Perform action after function return</td>
</tr>
</tbody>
</table>
example: safely free a spinlock

```c
void kfree(void *p)
pre(transfer(write(p, no_size)))
```

```
lxfi_transfer_write(mylock, -1);
kfree(mylock);
```

```
write(mylock, 8)
```
why is spin_module able to call spin_lock_init, kmalloc, kfree

- call capability
- granted initially according to the module’s symbol table
  - trust module author not to call unnecessary functions

<table>
<thead>
<tr>
<th>Part</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>capability</td>
<td>write(ptr,size)</td>
<td>Write [ptr,ptr+size]</td>
</tr>
<tr>
<td></td>
<td>call(a)</td>
<td>Call a</td>
</tr>
<tr>
<td>capability action</td>
<td>copy(cap)</td>
<td>Grant a copy of cap</td>
</tr>
<tr>
<td></td>
<td>check(cap)</td>
<td>Check cap</td>
</tr>
<tr>
<td></td>
<td>transfer(cap)</td>
<td>Revoke cap from all modules, and grant</td>
</tr>
<tr>
<td>location</td>
<td>pre(action)</td>
<td>Perform action before function call</td>
</tr>
<tr>
<td></td>
<td>post(action)</td>
<td>Perform action after function return</td>
</tr>
</tbody>
</table>
example: all for spin_lock_init argument integrity
other software fault isolation systems

- SFI [SOSP93]: memory safety
- XFI [OSDI06]: no argument checks
- BGI [SOSP09]:
  - error-prone and time-consuming
  - works if kernel code is well-structured
    (not linux)
Outline

1. Overview
   - problems

2. Software Fault Isolation
   - LXFI

3. Reference
   - reference
1. Overview
   - problems

2. Software Fault Isolation
   - LXFI

3. Reference
   - reference
Yangdong Mao etc, SOSP’11, Software fault isolation with API integrity and multi-principal modules