Practical Techniques to Obviate Setuid-to-Root Binaries

Bhushan Jain, Chia-Che Tsai, Jitin John, Donald Porter
OSCAR Lab
Computer Science Department
Stony Brook University
Setuid-root and Privilege Escalation

/bin/mount /dev/sda2 /disk1
Setuid-root and Privilege Escalation

Root

Kernel

/bin/mount /dev/sda2 /disk1

/* Parse arguments */

sys_mount(args);
Setuid-root and Privilege Escalation

/bin/mount /dev/sda2 /disk1

/* Parse arguments */
sys_mount(args);

Root

Kernel

sys_mount() {
  if(!capable(CAP_SYS_ADMIN))
    return -EPERM;
  do_mount();
}
Setuid-root and Privilege Escalation

/bin/mount /dev/cdrom /cdrom

/* Parse arguments */

sys_mount(args);

User

Kernel

sys_mount() {
    if(!capable(CAP_SYS_ADMIN))
        return -EPERM;
    do_mount();
}
Setuid-root and Privilege Escalation

```
#include <sys/capability.h>

sys_mount() {
    if (!capable(CAP_SYS_ADMIN))
        return -EPERM;
    do_mount();
}
```

```
/* Parse arguments */
sys_mount(args);
```

```
/bin/mount /dev/cdrom /cdrom
```

User

Kernel

```
sys_mount() {
    if (!capable(CAP_SYS_ADMIN))
        return -EPERM;
    do_mount();
}
```
```
sys_mount() {
    if(!capable(CAP_SYS_ADMIN))
        return -EPERM;
    do_mount();
}

/* Parse arguments */
sys_mount(args);
```

```
/bin/mount /dev/cdrom /cdrom
```

Setuid to Root

User to Kernel

Kernel
```
sys_mount() {
    if(!capable(CAP_SYS_ADMIN))
        return -EPERM;
    do_mount();
}
```
Setuid-root and Privilege Escalation

```
sys_mount() {
    if(!capable(CAP_SYS_ADMIN))
        return -EPERM;
    do_mount(); }

/* Parse arguments */
sys_mount(args);
```

```
/dev/cdrom /cdrom
iso9660 user,ro 0 0
```

```
/etc/fstab
```

**User**

```
/bin/mount /dev/cdrom /cdrom
```

**Kernel**

```
sys_mount() {
    if(!capable(CAP_SYS_ADMIN))
        return -EPERM;
    do_mount(); }```

Setuid to Root
Setuid-root and Privilege Escalation

```
/bin/mount /dev/cdrom /cdrom
iso9660 user,ro 0 0
/etc/fstab

/* Parse arguments */
if(ruid == 0 ||
  user_mount_ok(args))
  sys_mount(args);
```

User

Kernel
```
sys_mount() {
  if(!capable(CAP_SYS_ADMIN))
    return -EPERM;
  do_mount();
}
```
Setuid-root and Privilege Escalation

```
sys_mount() {
  if(!capable(CAP_SYS_ADMIN))
    return -EPERM;
  do_mount(); }

/* Exploit Vulnerability */
fd=open("rootkit.ko")
finit_module(fd);

/* Setuid to Root */
User
/dev/cdrom /cdrom
iso9660 user,ro 0 0
/etc/fstab

Kernel

/bin/mount /dev/cdrom /cdrom
```

sys_mount() {
  if(!capable(CAP_SYS_ADMIN))
    return -EPERM;
  do_mount();
}

/* Exploit Vulnerability */
fd=open("rootkit.ko")
finit_module(fd);

User /dev/cdrom /cdrom iso9660 user,ro 0 0
/etc/fstab

Setuid to Root

Kernel /bin/mount /dev/cdrom /cdrom

Setuid-root and Privilege Escalation
sys_mount() {
    if(!capable(CAP_SYS_ADMIN))
        return -EPERM;
        do_mount();
}

/* Parse arguments */
if(ruid == 0 || user_mount_ok(args))
    sys_mount(args);
/* Exploit Vulnerability */
fd = open("rootkit.ko")
    finit_module(fd);

sys_finit_module() {
    if(!capable(CAP_SYS_MODULE))
        return -EPERM;
    do_init_module();
}

/rootkit.ko

Setuid to Root

User

root has all capabilities

Kernel sys_finit_module() {
    if(!capable(CAP_SYS_MODULE))
        return -EPERM;
    do_init_module();
}
How is Setuid-Root Used in Practice?

- 26 Binaries on 89% systems
- 83 Binaries on <0.89% systems
Can we get rid of setuid-to-root?

- Surprisingly feasible to obviate setuid-root
  - ~10 underlying privileged abstractions

- Protego prototype change 715 LoC in kernel
  - De-privileged 12,732 lines of trusted binary code
  - < 2% kernel compile time overhead over Linux 3.6.0
  - Ongoing investigation of long tail
Outline

- Background
- Insights and design principles
- Protego overview and examples
- Evaluation
Linux Capabilities

- Linux file POSIX capabilities
  - Not same as pointers with access control
  - Divide root privilege into 36 different capabilities

- Enforce least privilege for administrator
  - Too coarse for untrusted user
  - Many privileged actions with just `CAP_NET_ADMIN`

Need to think about least privilege for untrusted user
Efforts to Mitigate Setuid-Root Risks

- Ubuntu/Fedora try to limit use of setuid-root
  - Privilege Bracketing, consolidation, fs permissions
  - Not able to completely eliminate setuid-root
- Some binaries have point alternatives
- SELinux enforces relatively fine-grained security
  - Still too liberal for least privilege of user
  - SELinux introduces substantial complexity
What can we do about setuid-root risk?
How do we approach this problem?

- Studied 28 in detail
  - Order by popularity
- Study policies in binaries
  - Why is root needed?
  - Simpler alternative in kernel?
- Goal: Non-admin never raises privilege
Setuid-Root: Unix Security Duct Tape

- Kernel policy mismatch with system policy
  - Kernel: only \texttt{root} can mount anywhere
  - System: any \texttt{user} can mount at safe locations

- Point solutions used as duct tape
  - Setuid binary \texttt{mount} bridges the gap

Generally setuid patches kernel and system policies
Interface Designs can Thwart Least Privilege

- Interface design choice may need more privilege
- `dmcrypt-get-device` use privileged ioctl
  - Reports physical device under encrypted device
  - Also discloses the private key
  - Can get same info from `/sys` without privilege
  - Maintainers agreed to use `/sys` interface.

Sometimes setuid indicates programmer error
Protego Design

- No need for trusted apps to enforce system policy
  - Inform kernel about system policy
  - Enforce system policy using Linux Security Module
  - Policies orthogonal to AppArmor, SELinux, etc.

- Object-based policies for unprivileged users
- Adjust the interfaces that need more privilege
- Maintain backwards compatibility for user
<table>
<thead>
<tr>
<th>Privileged Interface</th>
<th>Used by</th>
<th>What do we do?</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mount, umount</code></td>
<td>3</td>
<td>Whitelist safe locations and options</td>
</tr>
<tr>
<td><code>socket (ping)</code></td>
<td>5</td>
<td>Apply firewall rules on raw sockets</td>
</tr>
<tr>
<td><code>Credential databases (passwd)</code></td>
<td>5</td>
<td>Fragment to per-user or pergroup files, matching DAC granularity.</td>
</tr>
<tr>
<td><code>ioctl (pppd)</code></td>
<td>2</td>
<td>Add LSM hooks to verify new routes</td>
</tr>
<tr>
<td><code>bind (mail)</code></td>
<td>3</td>
<td>Map low port to (binary, userid) pair</td>
</tr>
<tr>
<td><code>setuid, setgid (sudo)</code></td>
<td>7</td>
<td><strong>Delegation Framework</strong>: LSM hooks to check delegation rules &amp; recency</td>
</tr>
<tr>
<td><code>Video driver control state (X)</code></td>
<td>1</td>
<td><strong>Kernel Mode Switching</strong>: Context switches video devices in the kernel</td>
</tr>
<tr>
<td><code>/dev/pts* terminal slaves (pt_chown)</code></td>
<td>1</td>
<td>Deprecated since kernel 2.1</td>
</tr>
<tr>
<td><code>Host private ssh key (ssh-keysign)</code></td>
<td>1</td>
<td>Restrict file access to specific binaries</td>
</tr>
</tbody>
</table>

A few abstractions, many binaries
Example 1: Protego mount

This technique works for 3/28 setuid-root binaries
Example 1: Protego mount

This technique works for 3/28 setuid-root binaries
Example 1: Protego mount

/dev/cdrom /cdrom iso9660 user,ro 0 0
/etc/fstab

Root

Kernel

Protego LSM

This technique works for 3/28 setuid-root binaries
Example 1: Protego mount

```
/dev/cdrom /cdrom iso9660 user,ro 0 0
```

This technique works for 3/28 setuid-root binaries
Example 1: Protego mount

```
/dev/cdrom /cdrom iso9660 user,ro 0 0
```

```
/etc/fstab /*Parse /etc/fstab*/
```

```
/proc/mnt_policy
```

Privileged Daemon

Root

Kernel

Protego LSM

This technique works for 3/28 setuid-root binaries
Example 1: Protego mount

```
/dev/cdrom  /cdrom
iso9660  user,ro 0 0

/etc/fstab

/*Parse /etc/fstab*/

/proc/mnt_policy
```

Unprivileged user

Privileged Daemon

Root

Kernel

Protego LSM

This technique works for 3/28 setuid-root binaries
Example 1: Protego mount

```
/dev/cdrom /cdrom iso9660 user,ro 0 0
/etc/fstab
```

Privileged Daemon

```
/*Parse /etc/fstab*/
```

Root

```
mount /dev/cdrom /cdrom
```

Unprivileged user

```
/proc/mnt_policy
```

Protego LSM

This technique works for 3/28 setuid-root binaries
Example 1: Protego mount

```
mount /dev/cdrom /cdrom iso9660 user,ro 0 0
```

```
/*Parse /etc/fstab*/
```

```
sys_mount(args);
```

```
mount /dev/cdrom /cdrom
```

This technique works for 3/28 setuid-root binaries
Example 1: Protego mount

```
/proc/mnt_policy

sys_mount() {  
  if(!security_mount_ok(args))  
    return -EPERM;  
  do_mount(args);  
}
```

```
/*Parse /etc/fstab*/

/etc/fstab

Root

Privileged Daemon

mount /dev/cdrom /cdrom

Unprivileged user

Example 1: Protego mount

This technique works for 3/28 setuid-root binaries
Example 1: Protego mount

```
Example 1: Protego mount

/proc/mnt_policy

sys_mount() {
    if(!security_mount_ok(args))
        return -EPERM;
    do_mount(args);
}

sys_mount(args);

mount /dev/cdrom /cdrom

/*Parse /etc/fstab*/

/etc/fstab

/dev/cdrom /cdrom iso9660 user,ro 0 0

/proc/mnt_policy

Protego LSM

Privileged Daemon

Unprivileged user

Root

This technique works for 3/28 setuid-root binaries

Example 1: Protego mount

/proc/mnt_policy

sys_mount() {
    if(!security_mount_ok(args))
        return -EPERM;
    do_mount(args);
}

sys_mount(args);

mount /dev/cdrom /cdrom

/*Parse /etc/fstab*/

/etc/fstab

/dev/cdrom /cdrom iso9660 user,ro 0 0

/proc/mnt_policy

Protego LSM

Privileged Daemon

Unprivileged user

Root

This technique works for 3/28 setuid-root binaries
Underlying Problem with Setuid-bit

- Setuid bit is user/subject-based mechanism
  - “Does the user have privilege to take this action?”
  - E.g., Who can mount a device?
- Object-based security policies are better fit
  - “Can any user take an action on this object?”
  - E.g., Which device can be safely mounted and where?
Example 2: Linux ping

- Raw socket: unimplemented protocols in kernel
- TCP/UDP packets can be faked by raw sockets
  - Appear from socket owned by another process
- Kernel: \texttt{CAP\_NET\_RAW} to create raw socket
- System: Allow sending safe packets on raw socket
  - \texttt{ping} uses raw sockets for ICMP packets
Example 2: Protego ping

- Allow raw socket creation for all users
- Whitelist set of all safe packets on raw sockets
  - Use packet filtering framework (**netfilter**/BPF)
  - Limit raw socket protocol and message types
- For ping, allow ICMP echo request and reply

This technique works for 5/28 setuid-root binaries
Central credential databases

- /etc/passwd, /etc/group, /etc/shadow
- All owned and modified by root

Current: only root can modify credentials

System: every user should modify own credential

- Setuid-root passwd enforces this policy
Example 3: Protego passwd

- Access control at record granularity
  - Not entire database

- Split shared database into per-account files
  - /etc/passwd ➔ per-user file under /etc/passwds
    - drwxr-xr-x root root /etc/passwds
    - rw------- bjain group /etc/passwds/bjain

- Backwards compatibility: update /etc/passwd
  - Trusted daemon monitors /etc/passwds/* files

This technique works for 5/28 setuid-root binaries
Evaluation

- How is overall system security affected?
- Is Protego functionally identical to Linux?
- What is cost for unprivileged applications?
  - **Test machine**: Linux 3.6, 4-core 3.40 GHz Intel Core i7 CPU, 4GB memory and a 250GB, 7200 RPM SATA disk
- How to deprivilege the remaining 91 binaries?
Protego’s Addition to TCB

Changed LoC in Protego = 2,598
Protego’s Addition to TCB

Changed LoC in Protego = 2,598
- Untrusted Utilities = 108
- Trusted Services = 1,600
- Kernel = 715
## Protego’s Addition to TCB

<table>
<thead>
<tr>
<th>Category</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Changed LoC in Protego</td>
<td>2,598</td>
</tr>
<tr>
<td>Untrusted Utilities</td>
<td>108</td>
</tr>
<tr>
<td>Trusted Services</td>
<td>1,600</td>
</tr>
<tr>
<td>Kernel</td>
<td>715</td>
</tr>
<tr>
<td>• LSM</td>
<td>200</td>
</tr>
<tr>
<td>• Other</td>
<td>515</td>
</tr>
</tbody>
</table>
Protego’s Addition to TCB

- Changed LoC in Protego = 2,598
  - Untrusted Utilities = 108
  - Trusted Services = 1,600
  - Kernel
    - LSM = 200
    - Other = 515
  - Total Addition to TCB = 2,315
Protego’s Addition to TCB

Changed LoC in Protego = 2,598

- Untrusted Utilities = 108
- Trusted Services = 1,600
- Kernel = 715
  - LSM = 200
  - Other = 515

Total Addition to TCB = 2,315

Changes are small enough to be easily audited
Overall Security of Protego

De-privileged LoC by Protego = 15,047

Total Addition to TCB by Protego = 2,315

Net LoC de-privileged = 12,732

Protego reduces TCB by more than 12K LoC
Manual tests for Linux functional equivalence

Automate command-line utilities using `gcov`
Micro-Benchmarks

Micro-benchmark overheads

% Execution Time Overhead over Linux 3.6.0

syscall  read  write  open/close  sig.install  sig. overhead  Prot. Fault  fork+exit  fork+execve  Local/TCP lat.  Local UDP lat.  mount/unmount  setuid  setgid  ioctl  bind

Protego
Micro-Benchmarks

% Execution Time Overhead over Linux 3.6.0

Micro-benchmark overheads

~ 0% overhead on basic Linux operations
Micro-Benchmarks

% Execution Time Overhead over Linux 3.6.0

~ 0% overhead on basic Linux operations

< 7.4% overhead in worst case

Micro-benchmark overheads

Protego
Micro-Benchmarks

- < 7.4% overhead in worst case
- ~ 0% overhead on basic Linux operations
- < 2.5% overhead in changed system calls

% Execution Time Overhead over Linux 3.6.0

- syscall
- read
- write
- open/close
- sig.install
- sig.overhead
- Prot.
- Fault
- fork+exit
- fork+execve
- Local/TCP lat.
- Local UDP lat.
- mount/unmount
- setuid
- setgid
- ioctl
- bind

Protego
Macro-Benchmark Applications

![Macro-benchmark overheads](chart)

- **Postal Mail server (Min/msg)**: Overhead is minimal.
- **Kernel Compile**: A moderate overhead of about 2%.
- **ApacheBench 100 conc. req**: A significant overhead of about 8%.

The chart illustrates the execution time overheads compared to Linux 3.6.0.
Macro-Benchmark Application

% Execution Time Overhead over Linux 3.6.0

Macro-benchmark overheads

~ 0% overhead for bind privilege

- Postal Mail server (Min/msg)
- Kernel Compile
- ApacheBench 100 conc. req
Macro-Benchmark Applications

Macro-benchmark overheads

~ 0% overhead for bind privilege

2-4% overhead for unprivileged applications

% Execution Time Overhead over Linux 3.6.0

0 1 2 3 4 5 6 7 8 9 10

Postal Mail server (Min/msg)  Kernel Compile  ApacheBench 100 conc. req
Toward Zero Setuid-To-Root Binaries
Toward Zero Setuid-To-Root Binaries

91 Binaries on < 10% systems
Toward Zero Setuid-To-Root Binaries

91 Binaries on < 10% systems
77 Binaries use Protego Interfaces
Toward Zero Setuid-To-Root Binaries

- 6 use namespaces that are not privileged since 3.8
- 3 reboot system, load kernel modules, or configure network
  - May use delegation framework in Protego
- 5 VirtualBox binaries access custom kernel module
  - Requires additional work to identify a sensible policy

91 Binaries on < 10% systems
77 Binaries use Protego Interfaces
New kernel abstractions may need setuid-to-root
- Until we precisely understand safe function subsets
- Kernel namespaces needed root from 2.6.23 to 3.8
  - Incrementally deployed

Software with companion kernel modules
- Setuid-root as security blanket
Conclusions

- Setuid: Duct tape between kernel & system policy
- Instead, enforce system policies in the kernel
- Protego shows feasibility of eliminating suid-root
  - Same functionality, low overhead, compatible w/ LSM
- New applications should justify need for setuid

bpjain@cs.stonybrook.edu
http://protego.cs.stonybrook.edu