Making Linux Protection Mechanisms Egalitarian with UserFS

Taesoo Kim
and
Nickolai Zeldovich

MIT CSAIL
Overview:
How to build secure applications?

- **Simple** in principle:
  - reduce privileges of application components
  - enforce policy at lower level (e.g. OS kernel)

- **Difficult** in practice (unless root user):
  - cannot create new principals
  - cannot reduce privileges
This Talk:

How to help programmers to reduce privileges and enforce security policy in Linux?

by allocating and managing UIDs
Today’s Unix-like OS

• **UID is not a real user’s identity** anymore (instead, also use UID as **a protection principal**)
  
i.e. nobody, www-data, wheelfs, etc.

• **Existing protection mechanisms** are using UID as a security principal
  
i.e. filesystem permission
Running example: DokuWiki

DokuWiki is a standards compliant, simple to use Wiki, mainly aimed at developer teams, workgroups and small companies. It has a simple readable outside the Wiki and eases the creation of structured text required.

Read the DokuWiki Manual to unleash the full power of DokuWiki.

Download

DokuWiki is available at http://www.splitbrain.org/go/dokuwiki
Example: Security model of DokuWiki

• PHP based Wiki
• Run as a single UID

• Main features
  1) Wiki users
  2) Saving each page as a file
  3) ACL on each page
Example: Run DokuWiki

```
php
<UID:www-data>
```
Example: Alice write to the page1

ACL of DokuWiki Pages

<table>
<thead>
<tr>
<th>Path</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>/doku/pages/page1</td>
<td>Alice: r/w, Bob: r/-</td>
</tr>
<tr>
<td>/doku/pages/page2</td>
<td>Alice: r/-, Bob: r/w</td>
</tr>
</tbody>
</table>
Example: Alice write to the page1
Example: **Bob** write to the *page1*

**ACL of DokuWiki Pages**

<table>
<thead>
<tr>
<th>Path</th>
<th>Access Rights</th>
</tr>
</thead>
<tbody>
<tr>
<td>/doku/pages/page1</td>
<td>Alice: r/w</td>
</tr>
<tr>
<td>/doku/pages/page2</td>
<td>Alice: r/-</td>
</tr>
</tbody>
</table>
Example: Bob write to the *page1*

bob

write to *page1*

failed to write

php

<UID:www-data>

open()

/doku/conf/acl.php

<UID:www-data>
Example: Vulnerability when checking ACL

write() to page1

/doku/conf/acl.php

open() to <UID:www-data>

write() to <UID:www-data>

failed to write

bob

/doku/pages/page1

/<UID:www-data>
Example: Vulnerability when checking ACL

The ACL check happens **40 times** in DokuWiki’s code:
New, *potentially-buggy* code in every app.

CVE-2010-0288: Insufficient Permission Check
Strawman: Running php with different UID

alice

write to page1

bob

write to page1
Problem: Privilege separation is difficult in Unix

- Applications cannot
  - allocate new UIDs (e.g. adduser)
  - switch current UID (e.g. setuid) without root privilege
- Ironically,
  To reduce privilege, it requires root privilege
- Running DokuWiki as root is a security disaster
Problem: Privilege separation is difficult in Unix

Unix-like OS

DokuWiki

root

alice

bob

root

doku-wiki
doku-alice
doku-bob
taesoo

firefox

PHP

PHP

PHP

PHP
Goal of this work

Allowing any application to use these protection mechanisms without root privilege

- create a new principal
- reuse existing protection mechanisms
  - use chroot and firewall mechanisms
Outline

- Overview
- **Design**
- Example
- Implementation
- Evaluation
- Limitation
- Related work
- Conclusion
Design: UID allocation

• **Strawman:** pick a *previously unused* UID

• Challenges
  • *who* can call setuid()?
  
  • How to **reuse** UIDs?
  
  • How to make UIDs **persistent**?
Challenge: Who can call setuid()? 

- Current Linux
  - **Root** can switch to any UID with *setuid()*
  - **Non-root cannot** switch to new UID with *setuid()*

- Ideal system requirements
  - Need to **represent** privilege of each UID
  - Need to **specify** who can access each UID
  - Need to **pass** privilege between processes
Key Idea: UserFS

• Maintaining **UIDs** as **files** in /proc-like **filesystem**

  • Representing Privileges
    - each UID is represented by a **file**

  • Delegating Privileges
    - **change** permissions on the file
    - **send** the file descriptor via FD passing

• **Accountability**
  - track allocated UIDs of each user in a **directory**
Representing UIDs
Representing UIDs

```
mount UserFS at /userfs
```
Representing UIDs

represent UID number as a directory
Representing UIDs

“ctl file” to represent a privilege of each UID
Representing Privileges

- Each UID has only one **ctl file**
- Any process having the **file descriptor of the ctl**
  - can **change** current **UID** e.g. **setuid()**
  - can **pass** it through **Unix domain socket** e.g. **send()**
  - can **deallocate UID** by **deleting** the **ctl file** e.g. **unlink()**
Challenge: How to reuse UIDs?

• Ideally, unique ID to every principal

• Problem:
  - Linux use **32-bit** UID
  - **Reuse** previously allocated UID

• Solution:
  - Introduce **64-bit #gen**
  - Use **#gen** to detect unwanted UID reuse
Challenge: How to make UIDs persistent?

- For each UID, keep track of:
  - #gen
  - permissions of *ctl file*
  - creator’s UID

  in persistent database
# Managing UIDs

<table>
<thead>
<tr>
<th>File system</th>
<th>UserFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add a file</td>
<td>Allocate a UID</td>
</tr>
<tr>
<td>Delete a file</td>
<td>Deallocate a UID</td>
</tr>
<tr>
<td>Open a file</td>
<td>Gain the privilege of UID</td>
</tr>
<tr>
<td>Change permission</td>
<td>Delegate a privilege</td>
</tr>
</tbody>
</table>
Example: Using a Ufile

fd = open(/userfs/1000/ctl)

1) Setuid
   ioctl(fd, IOCTL_SETUID)

2) UID Allocation
   ioctl(fd, IOCTL_ALLOC, 2000)

3) Privilege Delegation
   sendmsg(receiver-socket, fd)
Outline

- Overview
- Design
- **Example**
- Implementation
- Evaluation
- Limitation
- Related work
- Conclusion
Example: Security model of UserFS-aware DokuWiki

Key idea:
Allocate **UID** for each Wiki user!

- Authenticate users with non-root daemon
- Use UID Sandboxing
- Reuse well-tested ACL of filesystem
Example: Authenticating users with non-root daemon

- Allocate new **doku-admin** UID (Wiki admin)

- When a new user **signs up**
  - **doku-admin** will **allocate a UID** for the user
  - **doku-admin** will gain **read permission** on **ctl file**

- When a user **logs in**
  - **login-mgr** (setuid to **doku-admin**) check id/passwd
  - open the **ctl file** of the Wiki user
  - **send** it through Unix domain socket
Example: Servicing DokuWiki with anonymous UID

```
<UID:anonymous>
```

```
<UID:httpd>
```

```
php
```

```
fork/exec
```

```
URL
```

```
DokuWiki
```

```
httpd
```

```
```
Example: Authenticating users with non-root daemon

```
alice (ID/PASS)

php

<UID:anony.>
```
Example: Authenticating users with non-root daemon

alice (ID/PASS)

php

<UID:anony.><UID:doku-admin>

login-mgr

fork/exec
Example: Authenticating users with non-root daemon

alice (ID/PASS) -> fork/exec php

<UID:anony.> <UID:doku-admin>

login-mgr

fork/exec

open() /var/doku/passwd
<Open: doku-admin:r/->

open() /userfs/501/ctl
<Open: doku-admin:r/->

fd=open() /userfs/501/ctl
Example: Authenticating users with non-root daemon

- Alice (ID/PASS) requests access.
- The request is sent to the `login-mgr` daemon.
- `login-mgr` forks and executes PHP code.
- PHP code queries the `/var/doku/passwd` file to authenticate Alice.
- If Alice is authenticated, the `ctl` file at `/userfs/501/ctl` is opened.
Example: Authenticating users with non-root daemon

alice (ID/PASS)

fork/exec

php

UID: doku-alice

setuid(fd)

open() /var/doku/passwd
<UID:doku-admin:r/->

ctl file

send(fd)

ctl file

UID: doku-alice

open() /userfs/501/ctl
<UID:doku-admin:r/->

php

UID: anony.

login-mgr

<UID:doku-admin>
Example: UID Sandboxing

• Initially, launch PHP with anonymous UID

• After a Wiki user logins

  change UID of PHP to Wiki user’s UID

  - login-mgr will send the file descriptor of ctl file
  - receive the file descriptor of the Wiki user
  - call setuid() with the received file descriptor
Example: UID Sandboxing

alice (ID/PASS) -> fork/exec php

<UID:anony.> <UID:doku-admin>

fork/exec php

<UID:doku-alice>

login-mgr

open() /var/doku/passwd <doku-admin:r/->

fd=open() /userfs/501/ctl <doku-admin:r/->

send(fd)

ctl file
Example: UID Sandboxing

alice (ID/PASS) -> fork/exec

php

<UID:anony.><UID:doku-admin>

fork/exec

log-in-mgr

open() /var/doku/passwd
<uid:<doku-admin:r/>>

fd=open() /userfs/501/ctl
<uid:<doku-admin:r/>>

ctl file

<UID:doku-alice>

send(fd)

<UID:doku-alice>
Example: UID Sandboxing

alice (ID/PASS)

fork/exec

php

<UID:anony.><UID:doku-admin>

login-mgr

<UID:doku-alice>

ctl file

send(fd)

open()

/var/doku/passwd

<uid:anony.><uid:doku-admin>

open()

/userfs/501/ctl

<uid:doku-admin>

100 LoC!
Example: Reusing well-tested ACL of filesystem

- Save each Wiki page as a file with owner’s UID
- Align ACL of Wiki page to the file permission
- OS will enforce security policy
Example: Reusing well-tested ACL of filesystem

```
php
<UID:doku-alice>
```
Example: Reusing well-tested ACL of filesystem

php

<UID:doku-alice>

write page1

/doku/pages/page1

<doku-alice:r/w>
Example: Reusing well-tested ACL of filesystem

Bug on checking ACL?
CVE-2010-0288: Insufficient Permission Check
Implementation

• A single kernel module on Linux 2.6.31
  - Using Linux Security Module (LSM)
    ex) file_permission, inode_setattr, socket_send/recvmsg
  - Using Netfilter (NF)
    ex) NF_INET_LOCAL_IN/OUT
  - Using Virtual File System (VFS)

• Minimal changes of the Linux kernel
  - < 3,000 LoC Kernel Module
  - < 1,500 LoC Library
Implementing Generation Number

- Keeping system-wide 64-bit #gen

- **Storing** #gen in ext. attributes for setuid binaries by hooking `inode_setattr()` of LSM

- **Checking** #gen when executing setuid binaries by hooking `file_permission()` of LSM
Implementing Database

- Maintaing `/etc/userfs/*` per UID
  - #gen
  - permission of `ctl file`
  - creator’s UID/GID

- Lazily update the database

- `mount.userfs` constructs `/userfs` after booting
Evaluation questions

• How easy is it to use UserFS?
  - Modified 5 applications, minimal code changes

• What kinds of security problems can it prevent?
  - Catches 5/6 attacks on one of the apps, DokuWiki

• What is the performance overhead?
  - Minimal overhead, see the paper
## Applying UserFS to existing applications

<table>
<thead>
<tr>
<th>Apps</th>
<th>LoC</th>
<th>Security enhancement</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTP Server</td>
<td>30 (+100 login-mgr)</td>
<td>Avoid root privilege</td>
</tr>
<tr>
<td>Chromium Browser</td>
<td>1</td>
<td>UID Sandboxing</td>
</tr>
<tr>
<td>DokuWiki</td>
<td>40 (+150 login-mgr)</td>
<td>Avoid root privilege UID Sandboxing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reuse OS protection mechanism</td>
</tr>
<tr>
<td>Cmdline Tool (su and bash)</td>
<td>15 (+60 bash)</td>
<td>Easier to switch privileges</td>
</tr>
<tr>
<td>Subsh (shell tools)</td>
<td>150</td>
<td>Easier to reduce privileges</td>
</tr>
</tbody>
</table>
# Applying UserFS to existing applications

<table>
<thead>
<tr>
<th>Apps</th>
<th>LoC</th>
<th>Security enhancement</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTP Server</td>
<td>30 (+100 login-mgr)</td>
<td>Avoid root privilege</td>
</tr>
<tr>
<td>Chromium Browser</td>
<td>1</td>
<td>UID Sandboxing</td>
</tr>
<tr>
<td>DokuWiki</td>
<td>40 (+150 login-mgr)</td>
<td>Avoid root privilege</td>
</tr>
<tr>
<td>Cmdline Tool</td>
<td>15 (+60 bash)</td>
<td>Easier to switch privileges</td>
</tr>
<tr>
<td>Subsh (shell tools)</td>
<td>150</td>
<td>Easier to reduce privileges</td>
</tr>
</tbody>
</table>

By changing `fork()` -> `ufork()`, Provide **UID** for each **renderer** process.
Vulnerabilities prevented

<table>
<thead>
<tr>
<th>Attack Vectors</th>
<th>CVE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Directory Traversal</td>
<td>CVE-2010-0287</td>
</tr>
<tr>
<td>Insufficient Permission Check</td>
<td>CVE-2010-0288</td>
</tr>
<tr>
<td>Cross Site Request Forgery</td>
<td>CVE-2010-0289</td>
</tr>
<tr>
<td>PHP Code Upload</td>
<td>CVE-2006-4675</td>
</tr>
<tr>
<td>PHP Code Injection</td>
<td>CVE-2006-4674</td>
</tr>
<tr>
<td></td>
<td>CVE-2009-1960</td>
</tr>
</tbody>
</table>

- Prevent **5 out of 6 vulnerabilities**
  - not intended to prevent **Cross Site Req. Forgery**
- Application can **rely on OS to enforce policy**
  - or can even **get rid of manual ACL check routine**
## Evaluation of DokuWiki

<table>
<thead>
<tr>
<th>LoC of modification</th>
<th>Fetching a wiki page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Login-mgr</td>
<td>150 LoC</td>
</tr>
<tr>
<td>DokuWiki</td>
<td>40 LoC</td>
</tr>
<tr>
<td>Without UserFS</td>
<td>45 ms</td>
</tr>
<tr>
<td>With UserFS</td>
<td>61 ms</td>
</tr>
</tbody>
</table>

- **40 LoC changes on DokuWiki (+150 LoC Login-mgr)**
  - excluding 530 LoC UserFS PHP extension

- **35% performance overhead with extra security checks**
  - invoking `login-mgr` in every request
  - could avoid overhead with `long-running daemon`
Limitation

• UID gen# only tracked for setuid binaries
  • Reused UID owner can look at old UID’s files
  • Applications should clean up sensitive files when deallocating UIDs

• GID allocation not implemented in prototype
  • Can emulate this by creating shared UID

• Future work: allow a process to have multiple concurrent UIDs (generalization of Unix GIDs)
Related Work

- **UID sandboxing**
  e.g. Android, Qmail

- **System call interposition**
  e.g. Ostia

- **New protection mechanisms** for Linux/Unix
  e.g. Flume, SELinux
  e.g. Capsicum – doesn’t reuse file permission checks, would be a good complement to UserFS

- **New OS**
  e.g. HiStar, ServiceOS, KeyKOS, VSTa ...
Conclusion

• Key idea:
  Representing **UID as files** in **/proc-like** filesystem

• Anyone
  can create a **new protection principal**
  can reuse **existing protection mechanisms**
  without losing **compatibility**

• The **first system** to provide **egalitarian OS protection mechanisms** for Linux
  ex) UID allocation, chroot and firewall