CSE 127: Computer Security

Least privilege and privilege separation

Deian Stefan

Slides adopted from John Mitchell, Dan Boneh, and Stefan Savage
This week...

• How to build secure systems
  ➤ Least privilege and privilege separation
  ➤ Sandboxing and isolation

• Key is underlying principles not mechanisms
  ➤ We’re going to look at systems techniques
  ➤ Other ways to achieve similar goals: language-based
Principles of secure design

• Principle of least privilege
• Privilege separation
• Defense in depth
  ➤
  ➤
• Keep it simple
Principles of secure design

• Principle of least privilege

• Privilege separation

• Defense in depth
  ➤ Use more than one security mechanism
  ➤ Fail securely/closed

• Keep it simple
Principle of Least Privilege

Defn:

- What’s a privilege?
Principle of Least Privilege

**Defn:** A system should only have the minimal privileges needed for its intended purposes

• What’s a privilege?
Principle of Least Privilege

Defn: A system should only have the minimal privileges needed for its intended purposes

• What’s a privilege?
  ➤ Ability to access (e.g., read or write) a resource
What’s the problem with this defn?

- Talking about a huge, monolith system is not really useful

- Why?
Breaking a system into components

- Compartmentalization and isolation
  - Separate the system into isolated compartments
  - Limit interaction between compartments

- Why is this more meaningful?
How do we break things apart?
Map compartment to user ids!

- Recall: permissions in UNIX granted according to UID
  - A process may access files, network sockets, ....

- Each process has UID

- Each file has ACL
  - Grants permissions to users according to UIDs and roles (owner, group, other)
  - Everything is a file!
How many UIDs does a process have?

- A: one
- B: two
- C: three
- D: four
Process UIDs

- Real user ID (RUID)
  - same as the user ID of parent (unless changed)
  - used to determine which user started the process

- Effective user ID (EUID)
  - from setuid bit on the file being executed, or syscall
  - determines the permissions for process

- Saved user ID (SUID)
  - Used to save and restore EUID
Process UIDs

• **Real user ID (RUID)**
  ➢ same as the user ID of parent (unless changed)
  ➢ used to determine which user started the process

• **Effective user ID (EUID)**
  ➢
  ➢

• **Saved user ID (SUID)**
  ➢
Process UIDs

• Real user ID (RUID)
  ➤ same as the user ID of parent (unless changed)
  ➤ used to determine which user started the process

• Effective user ID (EUID)
  ➤ from setuid bit on the file being executed, or syscall
  ➤ determines the permissions for process

• Saved user ID (SUID)
Process UIDs

• **Real user ID (RUID)**
  ➢ same as the user ID of parent (unless changed)
  ➢ used to determine which user started the process

• **Effective user ID (EUID)**
  ➢ from setuid bit on the file being executed, or syscall
  ➢ determines the permissions for process

• **Saved user ID (SUID)**
  ➢ Used to save and restore EUID
SetUID demystified (a bit)

- **Root**
  - ID=0 for superuser root; can access any file

- **fork and exec system calls**

- **setuid system call**
  - seteuid(newid) can set EUID to
SetUID demystified (a bit)

- Root
  - ID=0 for superuser root; can access any file

- fork and exec system calls
  - Inherit three IDs, except exec of file with setuid bit

- setuid system call
  - seteuid(newid) can set EUID to
SetUID demystified (a bit)

- **Root**
  - ID=0 for superuser root; can access any file

- **fork and exec system calls**
  - Inherit three IDs, except exec of file with setuid bit

- **setuid system call**
  - seteuid(newid) can set EUID to
    - Real ID or saved ID, regardless of current EUID
    - Any ID, if EUID is root
SetUID demystified (a bit)

• There are actually 3 bits:
  ➤ setuid - set EUID of process to ID of file owner
  ➤ setgid - set EGID of process to GID of file
  ➤ sticky bit
    ➤ on:
    ➤ off:
SetUID demystified (a bit)

• There are actually 3 bits:
  ➤ setuid - set EUID of process to ID of file owner
  ➤ setgid - set EGID of process to GID of file
  ➤ sticky bit
    ➤ on: only file owner, directory owner, and root can rename or remove file in the directory
    ➤ off: if user has write permission on directory, can rename or remove files, even if not owner
Where have you seen this?

-rwsr-xr-x 1 root root 55440 Jul 28  2018 /usr/bin/passwd

drwxrwxrwx 16 root root  700 Feb  6 17:38 /tmp/
Why are EUIDs even a thing?

We can drop and elevate privileges!
Why are EUIDs even a thing?

We can drop and elevate privileges!

```
...;
...;
exec( );
```

```
RUID 25

...;
...;
```

```
SetUID

Owner 18

program

Owner 18

-rw-r--r--

file

Owner 25

-rw-r--r--

file

Owner 18

i=getuid();
setuid(i);
...
...
```

```
...;
...;
Owner 18
```

```
Owner 18
```

```
Owner 25
```

```
Owner 25
```

```
RUID 25
```

```
EUID 18
```

```
EUID 25
```

```
EUID 25
```
Example 1: Mail agent

• Requirements
  ➤ Receive and send email over external network
  ➤ Place incoming email into local user inbox files

• Sendmail
  ➤ Monolithic design
  ➤ Historical source of many vulnerabilities

• Qmail
  ➤ Compartmentalized design
qmail design

• Isolation based on OS isolation
  ➢ Separate modules run as separate “users”
  ➢ Each user only has access to specific resources

• Least privilege
  ➢ Minimal privileges for each UID
  ➢ Only one “setuid” program
  ➢ Only one “root” program
structure of qmail

- qmail-smtpd
- qmail-local
- qmail-remote
- qmail-rspawn
- qmail-send
- qmail-lspawn
- qmail-inject

Incoming external mail

Incoming internal mail
structure of qmail

- qmail-smtpd
- qmail-local
- qmail-remote
- qmail-rspawn
- qmail-send
- qmail-lspawn
- qmail-queue
- qmail-inject

**Incoming external mail**

**Incoming internal mail**

-setuid-
structure of qmail

- qmail-smtpd
- qmail-local
- qmail-remote
- qmail-remote
- qmail-rspawn
- qmail-lspawn
- qmail-send
- qmail-queue
- qmail-inject

Incoming external mail
setuid

Incoming internal mail

root
structure of qmail
structure of qmail

- qmaild
  - qmail-smtpd
  - qmail-local
  - qmail-remote
  - qmail-lspawn
  - qmail-rspawn
  - qmail-send
  - qmail-inject

  Reads incoming mail directories
  Splits message into header, body
  Signals qmail-send

  qmailq
  - qmail-queue

  qmailr
  - qmail-rspawn
  - qmail-remote

  qmails
  - qmails

  qmail-local
  - qmail-local

  setuid user
  - root
  - user

  user
structure of qmail

- qmail-send signals
  - qmail-lspawn if local
  - qmail-remote if remote
structure of qmail

- qmaild
  - qmail-smtpd
  - qmail-local
  - qmail-remote
  - qmail-rspawn
  - qmailq
  - qmail-queue
  - qmail-send
  - qmail-local
  - qmail-remote
  - qmail-rspawn
  - qmail-inject

qmail-lspawn
- Spawns qmail-local
- qmail-local runs with ID of user receiving local mail
structure of qmail

- **qmaild**: 
  - Handles alias expansion
  - Delivers local mail
  - Calls qmail-queue if needed

- **qmail-smtpd**: User
- **qmail-local**: Setuid user
- **qmail-remote**: User
- **qmail-send**: Root
- **qmail-dspawn**: 
  - qmail-rspawn
  - qmail-remote
- **qmail-inject**: 
  - qmail-queue
  - qmail-local
structure of qmail

- qmail-smtpd
- qmail-local
- qmail-remote
- qmail-lspawn
- qmail-rspawn
- qmail-send
- qmail-inject
- qmail-queue
- qmailq
- qmailr
- qmailr
- qmail-remote
- qmail-local
- qmail-rspawn
- qmail-lspawn

- root
- setuid user
- user
- user

qmail-remote
- Delivers message to remote MTA
Android design

• Isolation: Each app runs with own UID (own VM)
  ➤ Provides memory protection
  ➤ Communication limited to using UNIX domain sockets + reference monitor checks permissions
  ➤ Only ping and zygote run as root

• Least Privilege: Applications announces permission
  ➤ User grants access at install time + runtime
okws design

• Isolation: each service runs with own UID
  ➤ Each service run in a chroot jail, restricted to
  ➤ Communication limited to structured RPC between service and DB

• Least privilege
  ➤ Each UID is unique non privileged user
  ➤ Only okld (launcher daemon) runs as root
okws design

External HTTP connections (Port 80)

Internal Site Management (Port 11277)

pubd  okd  okld

svc₁  svc₂  svc₃

data₁  data₂  DB

KEY:
- site-specific
- OKWS helper
- RPC
- HTTP
- "Parent Of"
- SQL
### okws design

<table>
<thead>
<tr>
<th>process</th>
<th>chroot jail</th>
<th>run directory</th>
<th>uid</th>
<th>gid</th>
</tr>
</thead>
<tbody>
<tr>
<td>okld</td>
<td>/var/okws/run</td>
<td>/</td>
<td>root</td>
<td>wheel</td>
</tr>
<tr>
<td>pubd</td>
<td>/var/okws/htdocs</td>
<td>/</td>
<td>www</td>
<td>www</td>
</tr>
<tr>
<td>oklogd</td>
<td>/var/okws/log</td>
<td>/</td>
<td>oklogd</td>
<td>oklogd</td>
</tr>
<tr>
<td>okd</td>
<td>/var/okws/run</td>
<td>/</td>
<td>okd</td>
<td>okd</td>
</tr>
<tr>
<td>svc1</td>
<td>/var/okws/run</td>
<td>/cores/51001</td>
<td>51001</td>
<td>51001</td>
</tr>
<tr>
<td>svc2</td>
<td>/var/okws/run</td>
<td>/cores/51002</td>
<td>51002</td>
<td>51002</td>
</tr>
<tr>
<td>svc3</td>
<td>/var/okws/run</td>
<td>/cores/51003</td>
<td>51003</td>
<td>51003</td>
</tr>
</tbody>
</table>
Browser security architecture

• Browser is an execution environment
  ➢ Has access control policies similar to an OS

• Browser runs under control of an OS
  ➢ Use least privilege to keep the browser code secure against attacks that would break the browser enforcement of web security policy
What’s the security model?

Operating system

- Subject: Processes
  - Has User ID (UID, SID)
  - Discretionary access control
- Objects
  - File
  - Network
  - ...
- Vulnerabilities
  - Untrusted programs
  - Buffer overflow
  - ...

Web browser

- Subject: web content (JavaScript)
  - Has “Origin”
  - Mandatory access control
- Objects
  - Document object model
  - Frames
  - Cookies / localStorage
- Vulnerabilities
  - Cross-site scripting
  - Implementation bugs
  - ...

Chromium security architecture

• Browser ("kernel")
  ➤ Full privileges (file system, networking)

• Rendering engine
  ➤ Can have multiple processes
  ➤ Sandboxed

• One process per plugin
  ➤ Full privileges of browser
Privilege separation

<table>
<thead>
<tr>
<th>Rendering Engine</th>
<th>Browser Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTML parsing</td>
<td>Cookie database</td>
</tr>
<tr>
<td>CSS parsing</td>
<td>History database</td>
</tr>
<tr>
<td>Image decoding</td>
<td>Password database</td>
</tr>
<tr>
<td>JavaScript interpreter</td>
<td>Window management</td>
</tr>
<tr>
<td>Regular expressions</td>
<td>Location bar</td>
</tr>
<tr>
<td>Layout</td>
<td>Safe Browsing blacklist</td>
</tr>
<tr>
<td>Document Object Model</td>
<td>Network stack</td>
</tr>
<tr>
<td>Rendering</td>
<td>SSL/TLS</td>
</tr>
<tr>
<td>SVG</td>
<td>Disk cache</td>
</tr>
<tr>
<td>XML parsing</td>
<td>Download manager</td>
</tr>
<tr>
<td>XSLT</td>
<td>Clipboard</td>
</tr>
<tr>
<td>Both</td>
<td></td>
</tr>
<tr>
<td>URL parsing</td>
<td></td>
</tr>
<tr>
<td>Unicode parsing</td>
<td></td>
</tr>
</tbody>
</table>
Chrome Security Architecture

Process Level Snapshot

Utility Process
Launched for short-lived operations, and will run sandboxed or unsandboxed depending on the specific operation (e.g. printing).

GPU Process
The GPU process runs with the minimum access required for using GPU resources (e.g. low-integrity on Windows).

Browser Process
The browser process runs at full user privilege and brokers access to most system resources including the profile and any persistent data.

Browser Mitigations:
- IPC hardening and CL reviews
- Minimal active content (e.g. JS)
- Limited protocol parsing

Major Attack Surface:
- Web renderer IPC surface
- Network protocol parsing
- Process state confusion (e.g. navigation)
- Google services (e.g. extension syncing)

PPAPI Broker Process
The PPAPI broker is allowed by the user to perform limited privileged actions for the PPAPI process (e.g. update global Flash settings).

Renderer Processes
Renderer Mitigations:
- Tightest OS sandbox
- Scripting runtime binding integrity
- Memory partitions
- Internal origin enforcement

Major Attack Surface:
- Blink
- V8 (including RWX JIT)
- media (e.g. ffmpeg, libpng)
- WebRTC, WebGL, etc.

Extension
Elevated extension and app permissions are listed in manifest file as either optional or required.

WebUI
C++ generated settings and diagnostic pages (effective permissions are hard to quantify).

PPAPI Process
Native code Pepper plugins, including Pepper Flash (which has some elevated APIs).

NaCl Loader Process
Bound by the hosting renderer’s origin and an inner SFI sandbox. (Non SFI code is a work in progress.)

Elevated Web
Certain web processes implicitly receive limited elevated privileges (e.g. omnibox renderer, Chrome Web Store, file: URLs)

Web
Normal Web content runs at the low privilege, but origins can be granted limited increased permissions by the user.

Legend:
- Chrome IPC
- Minimum Ambient Permissions
- Limited Ambient Permissions
- Elevated Ambient Permissions
- Maximum Ambient Permissions
- Feature not supported on Android

Generic Mitigations:
- Process-level sandboxing
- DEP+ASLR (per-process on linux & cros)
- Stack canaries
- Runtime and Library Hardening

Non-JSON content

- "Chrome Security Architecture
- Process Level Snapshot"
Are UIDs enough?

- A: yes
- B: no
What else do we need?

• We need to confine code running in renderer
  ➢ Restrict code from reading the filesystem, talking to network, etc. if compromised

• On Linux this is done with seccomp-bpf
  ➢ seccomp - “secure computing mode”: no sys calls except exit, sigreturn, read, and write to already open FDs
  ➢ seccomp-bpf - syscall firewall filtering