Finding Name Resolution Vulnerabilities in Programs

Hayawardh Vijayakumar, Joshua Schiffman, Trent Jaeger

CSE543 - Introduction to Computer and Network Security
Module: System Vulnerabilities

Professor Trent Jaeger
More Vulnerabilities

• When you ask for a file from the OS
  – What could go wrong?

What could possibly go wrong?
Links

- **Link**: Add new path to an inode
  - Multiple names for a single inode
- Convenient, but problematic
  - What inode does a name really refer to?
- Programmers can prevent use
  - open(file, O_NOFOLLOW, mode)
    - Prevents open from following a link
  - How does the programmer know when not to follow a file system link?
- **Adversaries can cause vulnerabilities** - how?
Link Traversal Attacks

- Adversary can trick victim to use their link
- Adversary creates link to target file
  - `ln -s /etc/shadow file`
- Invokes victim with access to target
  - `trusted_dump file < *passwd-entry*`
- Unless victim checks the destination of the link, bad things could happen
  - To overwrite `/etc/shadow`
- We will show that these are difficult to eradicate in programs
TOCTTOU Race Condition

- Victim checks the properties of a resource at a particular name (time-of-check)
- Adversary changes the binding of that name to a different resource (race)
  - Via a symbolic link is the most common
- Victim is tricked into using a resource of the adversary's choice (time-of-use)
  - E.g., the /etc/shadow resource was chosen in this case
  - Called time-of-check-to-time-of-use attack
    - TOCTTOU attack
A Simple Program

01 SOCKET_DIR=/tmp/.X11-unix
...
02 set_up_socket_dir ()
{
03 if [ "$VERBOSE" != no ]; then
04 log_begin_msg "Setting up X server socket directory"
05 fi
06 if [ -e SOCKET_DIR ] && [ ! -d SOCKET_DIR ]; then
07 mv SOCKET_DIR SOCKET_DIR.$$
08 fi
09 mkdir -p SOCKET_DIR
10 chown root:root SOCKET_DIR
11 chmod 1777 SOCKET_DIR
12 do_restorecon SOCKET_DIR
13 [ "$VERBOSE" != no ] && log_end_msg 0 || return 0
14
}
A Simple Program

```
01 SOCKET_DIR=/tmp/.X11-unix
...
02 set_up_socket_dir ()
{
03 if [ "$VERBOSE" != no ]; then
04 log_begin_msg "Setting up X server socket directory"
05 fi
06 if [ -e $SOCKET_DIR ] && [ ! -d $SOCKET_DIR ]; then
07 mv $SOCKET_DIR $SOCKET_DIR.$$
08 fi
09 mkdir -p $SOCKET_DIR
10 chown root:root $SOCKET_DIR
11 chmod 1777 $SOCKET_DIR
12 do_restorecon $SOCKET_DIR
13 [ "$VERBOSE" != no ] && log_end_msg 0 || return 0
14 }
```

Can mkdir fail then?
Link Traversal TOCTTOU Attack

- Adversary who has access to `/tmp` can create directory for `/tmp/.X11-unix`
  - victim code does not detect that problem
- Adversary can make this a link to `/etc/shadow`, and later code makes this file world-writeable
- Two parts to the attack
  - Change the namespace binding
  - Race condition to insert link between `mv` and `mkdir`
- Adversary ability to change namespace binding is fundamental to this attack
  - Race conditions are much easier to create than you might think
Creating a File

- When you create a file
  - `creat(pathname, mode)`
  - `open(pathname, O_CREAT, mode)`
  - What could go wrong?

- What happens if file is already there?
File Squatting

• For directories where create access is shared with adversaries
  – Adversaries may predict the names of files/directories

• Create sub-directory in advance
  – E.g., Adversaries predicted the .X11-unix directory in /tmp

• Also, works for files
  – Adversary binds name to a file of their choice before the victim can
  – Then, the victim uses the adversary’s file instead

• **Current Defense:** Check for existence on creation
  – open( name, O_CREAT | O_EXCL)
  – Always possible?
Search Path

• When you open a file with a relative path name
  – The system tries to be smart about figuring out where to find the file
  – PATH for executables
  – LD_PRELOAD to find libraries

• Also, the system has an algorithm for finding files
  – Specific order to search for directories
Windows: Library Loads

- SafeDIIISearchMode
  - 1. The directory from which the application loaded.
  - 2. The system directory. Use the `GetSystemDirectory` function to get the path of this directory.
  - 3. The 16-bit system directory. There is no function that obtains the path of this directory, but it is searched.
  - 4. The Windows directory. Use the `GetWindowsDirectory` function to get the path of this directory.
  - 5. The current directory.
  - 6. The directories that are listed in the PATH environment variable. Note that this does not include the per-application path specified by the `App Paths` registry key. The `App Paths` key is not used when computing the DLL search path.
Search Path Vulnerability

- Adversaries may craft malicious names using search path.

- Attack steps
  - If the adversary can plant a malicious library in the user’s home directory.
  - And start a privileged program from the user’s home directory.
  - The dynamic linker will request libraries using a name whose prefix is the user’s home directory.
  - Enabling the adversary to supply code to root processes.

- Sirefef Malware
  - Adobe installer loaded from user’s home dir.
  - Where a malware library has been written.
Directory Traversal Attack

- **Maliciously-crafted names**
  - Multiple ways of naming lots of things
    - Files
      - `/x/data or /y/z/.../x/data` or `/y/z/%2e%2e/x/data`
    - Lots of others -- URLs, DNS names, middleware-specific, etc.

- **Get access to resources that the adversary normally cannot (but, victim can)**
  - E.g., Windows system files
  - These are called *Confused Deputy* attacks

- **Trick process into accessing untrusted resources where safe are expected**
  - E.g., untrusted PHP files
  - These are called *File Inclusion* attacks
Safe Name Usage

• Canonicalization
  – Conversion to a single, “standard” name

• Rules of thumb
  – Do not rely on names -- or anything -- from remote user
    • At least not blindly
  – Be careful if your program may be started by user in their own directory
    • Environment variables
  – Convert them -- correctly -- to canonical format
    • Enable checking against your rules
  – Check resources are expected (e.g., inodes instead of filenames)
    • Check that these right resources with stat commands
Multi-binding

- One name may refer to multiple resources
- Victim adds mapping of name to resource
- So does adversary
  - Name server allows multiple bindings to name
- Name server chooses either resource
  - E.g., Chooses resource at random
- Is this for real?
  - Yes, Android Intents and D-Bus methods both allow such binds

- Current Defense: ???
  - Prevent use of adversary-controlled resources
Resource Attacks

• Systems allow for the **sharing of resources**
  – So a process may access a resource that is under an adversary’s control
  – Even when expecting a safe resource

• **What kinds of resources are under an adversary’s control?**
  – /tmp and user files
  – logs
  – /etc?
  – many others

• **Problem:** programmers do not know the deployment environments of their programs accurately
Common Problems?

- Do these attacks have a common problem?
Name Resolution

- Processes often use *names* to obtain access to *system resources*

- A *nameserver* (e.g., OS) performs *name resolution* using *namespace bindings* (e.g., directory) to convert a *name* (e.g., *filename*) into a system *resource* (e.g., *file*)
  - Filesystem, System V IPC, …

Diagram:
- **Name (filename)**
- **P**
- **open(“/var/mail/root”)**
- **Namespace (filesystem)**
  - **/**
  - **var**
  - **mail**
  - **root**
- **Resource (file)**
- **Bindings (directories)**
## Vulnerability Prevalence

<table>
<thead>
<tr>
<th>Attack Class</th>
<th>CWE class</th>
<th>CVE Count</th>
<th>&lt;2007</th>
<th>2007-12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Untrusted Search Path</td>
<td>CWE-426</td>
<td>109</td>
<td>329</td>
<td></td>
</tr>
<tr>
<td>Untrusted Library Load</td>
<td>CWE-426</td>
<td>97</td>
<td>91</td>
<td></td>
</tr>
<tr>
<td>File/IPC squat</td>
<td>CWE-283</td>
<td>13</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>Directory Traversal</td>
<td>CWE-22</td>
<td>1057</td>
<td>1514</td>
<td></td>
</tr>
<tr>
<td>PHP File Inclusion</td>
<td>CWE-98</td>
<td>1112</td>
<td>1020</td>
<td></td>
</tr>
<tr>
<td>Link Following</td>
<td>CWE-59</td>
<td>480</td>
<td>357</td>
<td></td>
</tr>
<tr>
<td>TOCTTOU Races</td>
<td>CWE-362</td>
<td>17</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>Signal Races</td>
<td>CWE-479</td>
<td>9</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

| % Total CVEs               | 12.40%    | 9.41%    |

---

**Source:** Systems and Internet Infrastructure Security Laboratory (SIIS)
Attacks on Name Resolution

• Improper Binding Attack
  ‣ Adversary controls bindings to redirect victim to a resource not under adversary’s control (confused deputy)
  ‣ Victim expects low integrity/secrecy, gets high instead
    • E.g., Link traversal attacks, including TOCTTOU races
Attacks on Name Resolution

• Improper Resource Attack
  ‣ Adversary controls final resource
  ‣ Victim expects high integrity, gets low integrity instead
    • E.g., Untrusted search paths, File squatting

```bash
mail
open("/var/mail/root")
/var
root
```

```
mail
A
```
Attacks on Name Resolution

• Improper Name
  ‣ Adversary controls name
  ‣ Victim expects one type of resource (low or high) and gets other
    • E.g., Directory Traversal, File Inclusion
Binding Defenses

• More extensive defenses
  – Safe sequence of system calls [Tsafrir et al FAST 2008]
    • lstat (get inode #)
      Resolve name/link to inode
    • access (check)
      Check that requestor has access
    • open (use -- get fd)
      Does the open work?
    • fstat (get inode # of fd)
      Does the fd match lstat’s inode
    • Does this work?
  – Access safe resources only using safe names [Chari et al NDSS 2010]
    • Prevents use of untrusted links to access safe resources
    • What is a safe name? What is a safe resource?
    • Can work for root-user, but is limited without program information

• Any binding defense must:
  – it must have side information about the programs it protects, it must protect only a subset of all programs, it must be vulnerable to DoS attacks, it must have false-positives, or it must fail to prevent some race condition exploits
Binding Defenses

• More extensive defenses
  – Safe sequence of system calls
    • lstat (get inode #) Secret file (not accessible)
    • access (check) Public file
    • open (use -- get fd) Secret file
    • fstat (get inode # of fd) Secret file
    • Does this work? No! [Cai et al, Oakland 2009]

  – Access safe resources only using safe names [Chari et al NDSS 2010]
    • Prevents use of untrusted links to access safe resources
    • What is a safe name? What is a safe resource?
    • Can work for root-user, but is limited without program information

• Any binding defense must:
  – it must have side information about the programs it protects, it must protect only a subset of all programs, it must be vulnerable to DoS attacks, it must have false-positives, or it must fail to prevent some race condition exploits
Program: Complicated

- Check for symbolic link (lstat)
- Check for lstat-open race
- Check for inode recycling
- Do checks for each path component (safe_open)
  - /, var, mail, …
- Cannot expect programmers to get this right!

```c
/* fail if file is a symbolic link */
int open_no_symlink(char *fname)
{
    struct stat lbuf, buf;
    int fd = 0;
    lstat(fname, &lbuf);
    if (S_ISLNK(lbuf.st_mode))
        error("File is a symbolic link!");
    fd = open(fname);
    fstat(fd, &buf);
    if ((buf.st_dev != lbuf.st_dev) ||
        (buf.st_ino != lbuf.st_ino))
        error("Race detected!");
    lstat(fname, &lbuf);
    if ((buf.st_dev != lbuf.st_dev) ||
        (buf.st_ino != lbuf.st_ino))
        error("Cryogenic sleep race!");
    return fd;
}
```
Goal

• Find name resolution vulnerabilities in programs
  ‣ So programs can be fixed to perform correct checks

```c
if ((stat("/var/mail/root", st)) == 0 && !S_ISLNK(st->st_mode))
```

• Or access control policies can be tightened

```bash
root@mantra:/var# ls -l | grep mail
drwxr-xr-x 2 root mail 4096 2012-03-29 02:05 mail
```

• System defenses can be improved (Process Firewall)
Our Solution

• Runtime analysis with active adversary in the OS
  ‣ OS models adversaries using access control permissions

• Only produce attack test case when that program’s adversary can modify directory (i.e., change bindings or resources) used in resolution
  ‣ Dynamically changes namespace to generate attack test case
  ‣ Later detects if program was vulnerable to the attack test case or not
  ‣ Finally, rolls back changes to namespace
Part 1 - Launch Attacks

1. Find bindings
2. Find adversary access
3. Launch attack (modify namespace)
4. Continue system call

```c
fd = open("/var/mail/root", O_APPEND);
delete("/var/mail/root");
symlink("/etc/passwd", "/var/mail/root")
```

User-space

Kernel

Adversary (group mail)

```
```
Part 2 – Detect Vulnerability

1. Victim accepts resource
2. Record vulnerability
3. Rollback namespace
4. Restart system call

Victim (user root)

write(fd)

User-space

Kernel

/etc
/var
/mail
/root (symbolic link)

passwd

Systems and Internet Infrastructure Security Laboratory (SIIS)
Impact

• Our solution guarantees
  ‣ No false positives where name resolution not accessible to program adversary
  ‣ No false positives where program defends itself from name resolution attacks already

• Can also create realistic attack test cases at runtime

• Any bug we find indicates a problematic name resolution!
Launch an Attack

- Example
  - Adversary $a$ creates higher branch file for victim $v$ for name $n$
  - Process $u$ receives lower branch if $a$ is not adversary of $u$ for same $n$

```
\text{\begin{minipage}{0.9\textwidth}
  $\mathcal{U}$ \hspace{1cm} $\mathcal{V}$
  \\
  $n$ \hspace{1cm} $n$
  \\
  Nameserver
  \hspace{1cm}
  $\alpha$

  $r'$ : original resource
  $r' : r$ modified by $a$
  $\alpha$ not adversary of $u$
  $a$ adversary of $v$
  \\
  Higher Branch
  \hspace{1cm}
  Lower Branch
\end{minipage}}$
```
STING Detects TOCTTOU Races

• STING can deterministically create races, as it is in the OS

```bash
SOCKET_DIR=/tmp/.X11-unix

set_up_socket_dir () {
    if [ "$VERBOSE" != no ]; then
        log_begin_msg "Setting up $SOCKET_DIR..."
    fi
    if [ -e $SOCKET_DIR ] && [ ! -d $SOCKET_DIR ]; then
        mv $SOCKET_DIR $SOCKET_DIR.$$
    fi
    mkdir -p $SOCKET_DIR
    chown root:root $SOCKET_DIR
    chmod 1777 $SOCKET_DIR
    do_restorecon $SOCKET_DIR
    [ "$VERBOSE" != no ] && log_end_msg 0 || return 0
}

ln -s /etc/passwd /tmp/.X11-unix
```
STING Creates Scenarios

- That do not occur in normal runtime

Adversary

Victim

```
hayawardh@mantra:~$ touch my_file
hayawardh@mantra:~$ ln -s my_file none
hayawardh@mantra:~$ ln -s my_file fusectl
```

```
root@mantra:/home/haywardh# mountall

root@mantra:/home/haywardh# cat /etc/mtab
/dev/sda1 / ext3 rw,relatime,errors=remount-ro,user_xattr 0 0
proc /proc proc rw 0 0
binfmt_misc /proc/sys/fs/binfmt_misc binfmt_misc rw,noxexec,nosuid,nodev 0 0
sysfs /sys sysfs rw,noxexec,nosuid,nodev 0 0
/home/haywardh/my_file /sys/fs/fuse/connections fusectl rw 0 0
none /sys/kernel/debug debugfs rw 0 0
none /sys/kernel/security securityfs rw 0 0
udev /dev/udev tmpfs rw,nodev=0755 0 0
devpts /dev/pts devpts rw,noxexec,nosuid,gid=5,mode=0620 0 0
tmpfs /run tmpfs rw,nosuid,nodev,size=10%,mode=0755 0 0
none /run/lock tmpfs rw,noxexec,nosuid,nodev,size=5242880 0 0
none /run/shm tmpfs rw,nosuid,nodev 0 0
/dev/sda6 /mnt/sda6 ext3 rw 0 0
/dev/sda2 /mnt/sda2 ext3 rw 0 0
/dev/sda5 /usr/src ext3 rw 0 0
```
Vulnerabilities by Entrypoint

• Under DAC adversary model
  ‣ Only 12.8% (Fedora) and 19.4% (Ubuntu) of total name resolution entrypoints were accessible to adversaries
  ‣ Only 1.0% (Fedora) and 3.0% (Ubuntu) of total name resolutions were vulnerable

<table>
<thead>
<tr>
<th>Adversary model</th>
<th>Total Resolutions</th>
<th>Adversary Access</th>
<th>Vulnerable</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAC - Ubuntu</td>
<td>690</td>
<td>134 (19.4%)</td>
<td>21 (3.0%)</td>
</tr>
<tr>
<td>DAC - Fedora</td>
<td>514</td>
<td>66 (12.8%)</td>
<td>5 (1.0%)</td>
</tr>
</tbody>
</table>
Types of Successful Attacks

- Some entry points were vulnerable to multiple attack types

<table>
<thead>
<tr>
<th>Type of vulnerability</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symlink following</td>
<td>22</td>
</tr>
<tr>
<td>Hardlink following</td>
<td>14</td>
</tr>
<tr>
<td>File squatting</td>
<td>10</td>
</tr>
<tr>
<td>Untrusted search</td>
<td>6</td>
</tr>
<tr>
<td>Race conditions</td>
<td>7</td>
</tr>
</tbody>
</table>
## Vulnerabilities

<table>
<thead>
<tr>
<th>Program</th>
<th>Vuln. Entry</th>
<th>Priv. Escalation DAC: uid-&gt;uid</th>
<th>Distribution</th>
<th>Previously known</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbus-daemon</td>
<td>2</td>
<td>messagebus-&gt;root</td>
<td>Ubuntu</td>
<td>Unknown</td>
</tr>
<tr>
<td>landscape</td>
<td>4</td>
<td>landscape-&gt;root</td>
<td>Ubuntu</td>
<td>Unknown</td>
</tr>
<tr>
<td>Startup scripts</td>
<td>4</td>
<td>various-&gt;root</td>
<td>Ubuntu</td>
<td>Unknown</td>
</tr>
<tr>
<td>mysql</td>
<td>2</td>
<td>mysql-&gt;root</td>
<td>Ubuntu</td>
<td>Known</td>
</tr>
<tr>
<td>mysql_upgrade</td>
<td>1</td>
<td>mysql-&gt;root</td>
<td>Unknown</td>
<td>Unknown</td>
</tr>
<tr>
<td>tomcat</td>
<td>2</td>
<td>tomcat6-&gt;root</td>
<td>Ubuntu</td>
<td>Known</td>
</tr>
<tr>
<td>lightdm</td>
<td>1</td>
<td>*-&gt;root</td>
<td>Ubuntu</td>
<td>Unknown</td>
</tr>
<tr>
<td>bluetooth-applet</td>
<td>1</td>
<td>*-&gt;user</td>
<td>Ubuntu</td>
<td>Unknown</td>
</tr>
<tr>
<td>java (openjdk)</td>
<td>1</td>
<td>*-&gt;user</td>
<td>Both</td>
<td>Known</td>
</tr>
<tr>
<td>mountall</td>
<td>1</td>
<td>*-&gt;root</td>
<td>Ubuntu</td>
<td>Unknown</td>
</tr>
<tr>
<td>colorod</td>
<td>1</td>
<td>*-&gt;user</td>
<td>Ubuntu</td>
<td>Unknown</td>
</tr>
<tr>
<td>mailutils</td>
<td>1</td>
<td>mail-&gt;root</td>
<td>Ubuntu</td>
<td>Unknown</td>
</tr>
<tr>
<td>bsd-mailx</td>
<td>1</td>
<td>mail-&gt;root</td>
<td>Fedora</td>
<td>Unknown</td>
</tr>
<tr>
<td>cupsd</td>
<td>1</td>
<td>cups-&gt;root</td>
<td>Fedora</td>
<td>Known</td>
</tr>
<tr>
<td>abrt-server</td>
<td>1</td>
<td>abrt-&gt;root</td>
<td>Fedora</td>
<td>Unknown</td>
</tr>
<tr>
<td>yum</td>
<td>1</td>
<td>sync-&gt;root</td>
<td>Fedora</td>
<td>Unknown</td>
</tr>
<tr>
<td>x2gostartagent</td>
<td>1</td>
<td>*-&gt;user</td>
<td>Extra</td>
<td>Unknown</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>20</strong></td>
<td></td>
<td><strong>21</strong></td>
<td>Unknown</td>
</tr>
</tbody>
</table>
Conclusions

• **Name resolution** is a fundamental process
  ‣ But, has long been vulnerable to various attacks

• It is both **difficult to** prevent name resolution attacks and **find program vulnerabilities**
  ‣ Finding vulnerabilities requires an active adversary

• **STING** is a system that finds name resolution vulnerabilities in programs
  ‣ By producing **malicious test case when a program’s adversary can modify a directory used in resolution**

• Found 21 previously-unknown vulnerabilities

• Next step, defense via **Process Firewall**
Thank You!

• Questions?
• E-mail for contact: tjaeger@cse.psu.edu