Secure Programming

Links

Ahmet Burak Can

Learning objectives

- Learn how symlinks work
- Learn how symlinks may fool a program
- Learn how race conditions happen
- Learn how symlinks and race conditions work together
- Learn how to defend against symlink attacks and how to avoid race conditions

Material Not Covered Here

- The following has been covered in other units:
  - File permissions
  - Umask
  - Setuid files
  - Relative and absolute paths
  - Environment variables
    - Path

Introduction to File System Vulnerabilities

- Most common attack vectors:
  - Symlink attacks (234 entries as of April 2004)
  - Directory traversal attacks (252 entries)
- Other attack vectors:
  - Information leakage
    - Recycled disk space and buffers
    - File descriptors
  - Insecure file permissions (configuration issue)
  - File system "mounting" issues (OS issue)

Symlink Attacks: Outline

- Symbolic links and hard links
- Basic symlink attack
  - Known or predictable file name
  - Defense: Randomness
- Symlink attacks on insecure temporary files

File System Links

- Hard links
  - Windows: CreateHardLink
  - UNIX: ln
- Symbolic links
  - UNIX:
    - ln -s
  - Windows:
    - a.k.a. "directory junctions" in NTFS
    - Manually: use Linkd.exe (Win 2K resource kit) or "junction" freeware
- Virtual Drives
  - "subst" command

Virtual Drives (Windows)

- Similar to symbolic links but limited functionality
- Effect
  - A drive letter ("X:") actually points to a directory on a physical drive
  - Overcomes limits on path length
    - NTFS has limit of 32000 characters
    - Windows has a limit of 256 characters
      - NT, 2000, XP
      - "Path too long" error
    - Malicious software can hide inside very long paths
      - Setup using a "subst" virtual drive
      - Remove drive, and virus scanner can't find it!
      - http://www.securiteam.com/windowsntfocus/5QP08156AQ.html

Subst Vulnerability

- Virtual drives persist after a user logs off (NT)
- If next user tries to use that same letter drive, they may use a folder of the attacker's choosing
  - Store confidential files
  - Run trojans
- Example: Network-mapped home drive
  - Mounting operation used to fail silently if the drive letter was already in use
  - CVE-1999-0824

Hard Links

- Indistinguishable from original entry (peer)
- May not refer to directories or span file systems
- Created link is subject to the same, normal file access permissions.
- Deleting a hard link doesn't delete the file unless all references to the file have been deleted
- A reference is either a hard link or an open file descriptor
- In Windows, a hard link exists at the NTFS file system level and is not supported by FAT32
  - Note: Different from a Windows shortcut

Example

```
% ls -al .localized
-rw-r--r-- root wheel .localized
% ln .localized pascal/hard.loc
% ls -al pascal/hard.loc
-rw-r--r-- root  wheel  hard.loc
% rm pascal/hard.loc
override rw-r--r--  root/wheel for pascal/hard.loc? yes
% ls -al .localized
-rw-r--r-- root wheel .localized
```

- Note that the hard link showed the same permissions
- Note that deleting the hard link didn't delete the file (the reference count was not zero)

Hard Link Vulnerability Example

- Fool audit logging programs by using a hard link to a sensitive file
- Audit trails record benign name instead of sensitive file access
- CVE-2002-0725 (NTFS)

Hard Links – Windows Lab

- Create a file "original.txt" and put some text in it
- Create a link to "original.txt" and call it "secondary.txt"
  - Use `fsutil hardlink create secondary.txt original.txt`
  - Or write a program to create a hard link using `CreateHardLink()` http://msdn.microsoft.com/library/default.asp?url=/library/en-us/fileio/base/hard_links_and_junctions.asp
- Show that you can edit "original.txt", save, and when you open "secondary.txt", it has the same changes.

Hard Links – UNIX Lab

- Create a file “original.txt” and put some text in it.
- Create a link to “original.txt” and call it “secondary.txt”
  - Use `ln original.txt secondary.txt`
- Show that you can edit “original.txt”, save, and when you open “secondary.txt”, it has the same changes.

Symbolic Links

- Windows:
  - Directory junctions apply to directories only
    - Can refer to directories on different computers
  - Jargon: "File system reparse points"
  - Contain parameters resolved at access time
  - Several operations, complex setup (see http://www.sysinternals.com/ntw2k/source/misc.shtml#junction)
- UNIX:
  - Contain a path, which is resolved at access time
  - May refer to directories and files
  - May span file systems
  - Permissions appear different from the original

Symbolic Link Example

- Using the same starting file as for the hard link example:

```
% ln -s .localized pascal/sym.loc

% ll pascal/sym.loc
lrwxr-xr-x  1 pascal  staff pascal/sym.loc -> .localized
```

- Note:
  - The “->”)
  - The permissions (see the “l”?)
  - The owner and group are different (they were root/wheel for ".localized")
  - Deleting the symlink doesn't delete the file

Power of Symbolic Links

- You can create links to files that don't exist yet
- Symlinks continue to exist after the files they point to have been renamed, moved or deleted
  - They are not updated
- You can create links to arbitrary files, even in file systems you can't see
- Symlinks can link to files located across partition and disk boundaries
- Example:
  - You can change the version of an application in use, or even an entire web site, just by changing a symlink
    - Very convenient!

Basic Attack

- Trick a process (with higher privileges) to operate on another file than the one it thinks it is.
- Example:
  - Create the link "temp -> /etc/password"
  - A privileged process executes
    - truncate("temp", 0)
      - The "truncate" call follows symlinks
      - Changes the length of the file "temp" to 0
  - But truncated /etc/password instead!
    - Note that the *contents* are deleted, not the file
- Can be used for write or read operations
  - Or deletion if the symlink is in the path and not the end point

Conditions of Vulnerability

- If you are operating in a secured directory, you don't need to worry about symlink attacks
- A secured directory is one with permissions of all the directories from the root of the file system to your directory, set such that only you (or root) can make changes in your secured directory
  - Example: /home/me (user home directories are usually set by default with secure permissions)
- You are at risk if you operate
  - In a shared directory such as /tmp
  - In someone else's directory, especially with elevated privileges
    - Example: an anti-virus program running as administrator

Example: CUPS Vulnerability

- CVE-2002-1366
- Common Unix Printing System (CUPS) 1.1.14 through 1.1.17 allows local users with lp privileges to create or overwrite arbitrary files via file race conditions.
- Predictable file name:
  - '/etc/cups/certs/<pid>'
  - Shared directory with users that have "lp" privilege
    - "lp" privilege could be gained through another exploit
- File manipulated using root privileges
    - Symlink redirected operations anywhere and allowed gaining root privileges

Suggested Workarounds

1. Store the file in a secured directory
   - It was stored in a shared directory
2. Relinquish root privileges before doing file operations (if not needed)
3. Use a random name
4. Create files with "umask 077"
   - New files will give no permissions to groups and others

- What about third-party components that you utilize?

Best Defenses (Both Windows and UNIX)

- If you are operating:
  - In someone else's directory, relinquish elevated privileges
    - If you are root (or administrator), set your effective user ID to that of the directory's owner for file operations in that directory
      - assuming that the directory is secured for that user; otherwise, you may endanger that user's files
    - If you are not root, you may be at risk of attacks against files you own elsewhere
      - don't operate on files in other user's directories
  - In a shared directory such as /tmp, consider using instead
    - A temporary directory inside your home directory
    - A secured directory for root or administrator temporary files

Windows Example

- Recursive deletion utility: "rd"
- Scenario:
  - Attacker makes a link from c:\temp\tempdir to c:\windows\system32 or any other sensitive directory
  - Administrator does "rd /s c:\temp"
  - Sensitive directory is erased!
    - (Example from Howard and Leblanc 2003)
- Note: Unlike the Windows "rd" utility, the UNIX command "`rm -rf`" does not traverse symlinked directories when deleting

Symlink File Write Example: Netscape 6.01

- The Netscape installer (running as root) creates a file in /tmp using a predictable name
- Attack: create a number of symlinks to cover the probable names, all pointing to a file to be deleted
  - The installer will delete the target!
    - CVE-2001-1066
- Which installers are you using that might have similar vulnerabilities?

Question

- Let's say you have a setuid program “myprogram” that uses a temporary file, /tmp/mytemp
- What happens when attacker does this?

```
ln -s /somedir/somefile  /tmp/mytemp
/usr/bin/myprogram &
```

a) Nothing

b) There's a symlink pointing to "/somedir/somefile", and "myprogram" runs

c) "myprogram" may perform operations on "/somedir/somefile" instead of the intended temporary file

Other Example: XFree86 startx

- CVE-1999-0433 Symlink vulnerability
- Xfree86 runs as root, creates a temporary file
- What will this do?
  - `ln -s /dev/hd0 /tmp/.tX0-lock`
  - `startx`
  - XFree86 will write its temporary file to the raw device, messing up the file system...
- Note: /dev/hd0 refers to hard disk

Symlink File Reading Example

- Digital Unix "msgchk" (setuid) checks to see if you have new messages, using your file `$HOME/.mh_profile`
- Predictable file name: `.mh_profile`
- Attacker does:
  - `ln -s target_to_read .mh_profile`
- msgchk displays part of target if privileges of process allow it, as an error message
  - CERT/CC VU#440539

Exercise

- Examine the installer script named "find_java.sh"
  - Location provided by instructor
- What does it do?
  - What should the variable $jvm contain?
    - What is the source of its value?
  - What should the variable $VAR contain?
    - What is the source of its value?
- Which program does the script really run?
  - How can this be exploited?

Exercise Answers

- The script executes a program named "java"
- Through links, another program may be invoked, masquerading as the real java
  - This program could check if it has root privileges and do nasty things when it does
    - During the install *or* later
    - Pass control to the real java when done, so as not to raise suspicions

Windows Defenses

- Check directory attributes
  - `DWORD GetFileAttributes(`
    `  LPCTSTR lpFileName`
    `);`
  - `FindFirstFile`
- The return value contains ORed attributes
- The "`FILE_REPARSE_POINT`" attribute indicates a directory junction
  - Is it OK to traverse it?
  - Where does it point to?
  - Should the program operate there?