A Secure Environment for Untrusted Helper Applications
Confining the Wily Hacker

J. White Bear

Lecture Notes 9/21/11

We continued our discussion of some of the security vulnerabilities in Janus and discuss how to mitigate these problems by analyzing specific bugs, attacks, fixes and their limitations.

1. Bug Discussion

1.1 **Symlink Bug 1**

This bug allowed opening of arbitrary paths, specifically anything specified in the /tmp/* directory.

1.1 **Attacks**

The main threat would be a symlink redirecting to a restricted file that would allow the attacker to change their permissions or circumvent other checks implemented by Janus. eg /tmp/a -> /etc/passwd.

1.1 **Fixes/Strategies**

1. Restrict privileges to applications to use only certain files or folders.

   Limitations

   The limitations of this strategy are the difficult in anticipating which files/ folders will be needed across applications or may have valid request for read/writes that should be allowed outside of those identified.

2. On Unix systems additional flags could be utilized to blocks symlinks.
   - eg open(path, flags) using O_EXCL to block symlinks.

   Limitations

   This approach could potentially break existing and necessary symlinks. It doesn't universally ban symlinks and only restricts the last part of the file name- so symlinks could be embedded earlier in the path name.

3. Another global approach is to simply ban all symlinks.

   Limitations

This approach will still allow preexisting symlinks, that were implemented after the policy, to be active.

1.2 **Symlink Bug 2**

This bug allowed opening of arbitrary paths, specifically anything specified in the /tmp/* directory.

1.2 **Attacks**

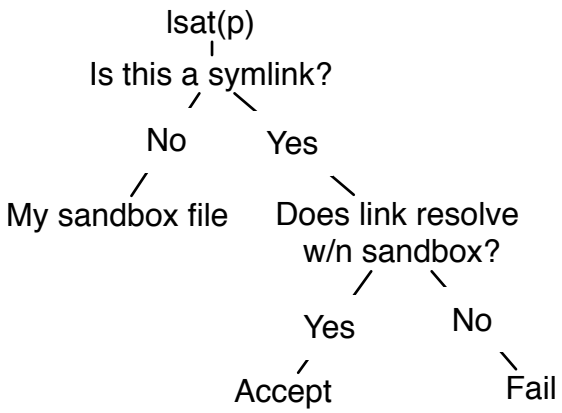
The creation of a seemingly benign symlinks that resolve to different paths or rename existing paths to resolve to different locations. This would be evident only after the path resolved so not as simple as checking the symlink. The symlink seems benign.

- /tmp/etc/passwd
- eg /tmp/d1/d2/a -> ../../etc/passwd
- renamed files in tmp
    - 1. /tmp/d1/d2 -> /tmp/d3
    - 2. /tmp/d3/a -> ../../etc/passwd

1.2 **Fixes/Strategies**

1. Check files on open to assure that they resolve to an authorized location. In this strategy on Unix, lstat(path) can be used to resolve the symlink.

   lstat validation



Limitations

There are several limitations to this strategy that relate back to the issues identified in the previous lecture with Janus.

1. Concurrency/Race Conditions/ TOCTTOU

A checked lstat link could still resolve in the kernel because this link is referenced as a globally accessible mutable state. An attacker could alter the reference after lstat or could arrange that the path refers to something different after it has been resolved.

2. This is difficult to mitigate because of the inability to alter the kernel and the application for compatibility and stability reasons.

3. The O_EXCL flag could be used with lstat(), but the full path is still not checked.

4. List of possible legacy options that could have been explored with Janus.

- open(path, flags)
  - O_EXCL
- lstat(path)/lstatat(path)
  - O_EXCL
- Openat(dir file descriptor, folder, path, flags)
  - used with specified relative directory
- Passing file descriptors between processes
  - To mitigate concurrency issues
- File descriptors bound to inodes
  - Always the latest directory stored
  - Inode would be immutable

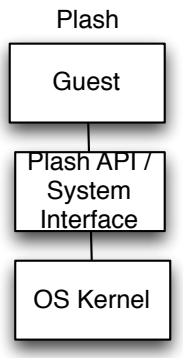
2. Solutions

2.1 **Solution openat()**

The openat() command using the O_EXCL flag to limit symlink access and restrict users at a directory level using an immutable file descriptor.

2.1 **Solution Plash http://plash.beasts.org/wiki/**

Plash, an updated and extended application using sandboxing principles similar to Janus that also includes kernel emulation and paravirtualization. Plash transforms guest input through the api into a kernel safe version, verified with the kernel emulation, and avoids some of the caveats of Janus.

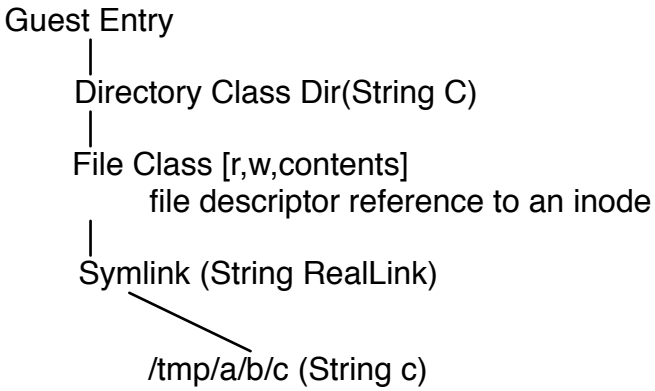


3. Plash

3.1 **Plash api**

The guest commands are run using a Unix emulation which translates commands into PlashGlibc which communicates with another api module called ServerProcess that sends commands to the kernel. Plash uses file descriptors for the file operations.

Eg. A guest enters the Plash sandbox environment requesting directory access. A new instance of the directory class is issued with specific permissions these permissions are inherited for each new level of access the guest might require and cannot exceed the parent's permissions.



3.2 **Plash Resolving Path Names**

The implementation of the way Plash resolves path names as strings and ensures that each level of the path name is allowed according to the permissions given to the parent.

```
resolve(dir d, path p)        {

for c in p.split ("/"):
        if (".") e =d.parent();
```

```
        e = d.child(c);
        switch(e):      //switch of file type
                dir
                        continue;
                symlink
                        d = resolve(d, e.readlink());
                default
                        access denied;
return d;
}
```

3.3 <u>Plash Attributes</u>

3.3 **Features**

- Symlink resolutions are done by the Unix emulation layer.
- Paravirtualization of the OS
- Separation of guest and host namespace.
    - Eg /tmp -> /tmp/guest1793
- Plash can intercept most system calls using its api

3.3 **Possible Vulnerabilites**

-Unix file descriptor permissions are never really revoked.
    - Could someone potentially exploit preexisting fd permissions in Plash?
- Dynamic loading in libc
    - An attacker could bypass user namespace using integer hexadecimal commands and go straight to the kernel.

4. Other Implementations

AppArmor is a viable solution however it requires modification of the kernel, but provides additional layer of security because it works within the native kernel. However, kernel modification is a significant barrier to mainstream adoption for non open source implementations and in open source kernels that may be more disparate.

5. Primer for Next Time

Think about and be prepared to discuss next time

-privilege separation
- browser, web server, ssh daemon

-How could you apply privilege separation techniques to build a secure email client?