Application White Listing

Steve Grubb
Red Hat

What is it?

What is it?

- Only applications on a whitelist can execute (e.g. - only things we know about)

How does a program execute?

- Bash checks if internal command and handles it
- If its a subshell,
  - forks and starts reading lines and performing them
- Else it: forks, sets up pipes, calls execve(filename, argv, envp)
- Kernel has a list of supported formats
  - ia_32aout
  - Flat
  - Aout
  - Script
  - Em86
  - Elf
  - elf_fdpic
- It iterates through each handler until one accepts the file

How does a program execute?

- If its a script
  - It must start with: !# interpreter [optional arg]
  - Re-execs as: interpreter [optional-arg] filename argv
  - If execve fails with ENOEXEC, Bash
    - Checks to see if its a directory
    - Checks to see if execute bit is set
    - Opens file and reads it
      - Interprets it as a shell script

- If its an ELF file...

How does a program execute?

- Kernel opens and reads the file (ELF image)
- Kernel inspects the file and notes that its interpreter is ld.so
- Kernel loads ld.so into program's address space
- ld.so initializes and looks at the program's ELF image
- ld.so locates the library names
- Looks for RPATH record (not normally there)
- ld.so consults LD_LIBRARY_PATH to locate the first library (not normally used)
- Checks /etc/ld.so.cache
- ld.so opens, mmaps, and reads library
- ld.so resolves symbols
- Continues this until all libraries and libraries dependencies are loaded.
- Jumps to init and then main

What are the attack points

- Without privileges
  - Downloading malware/escalation tools
  - Changing search paths by environmental variables
  - Code injection via LD_PRELOAD
- With privileges
  - Modifying/replacing applications
  - Installing new applications
  - Inject malware into running processes via ptrace
  - Change ELF interpreter in existing apps

Demonstrate Launches

Fanotify

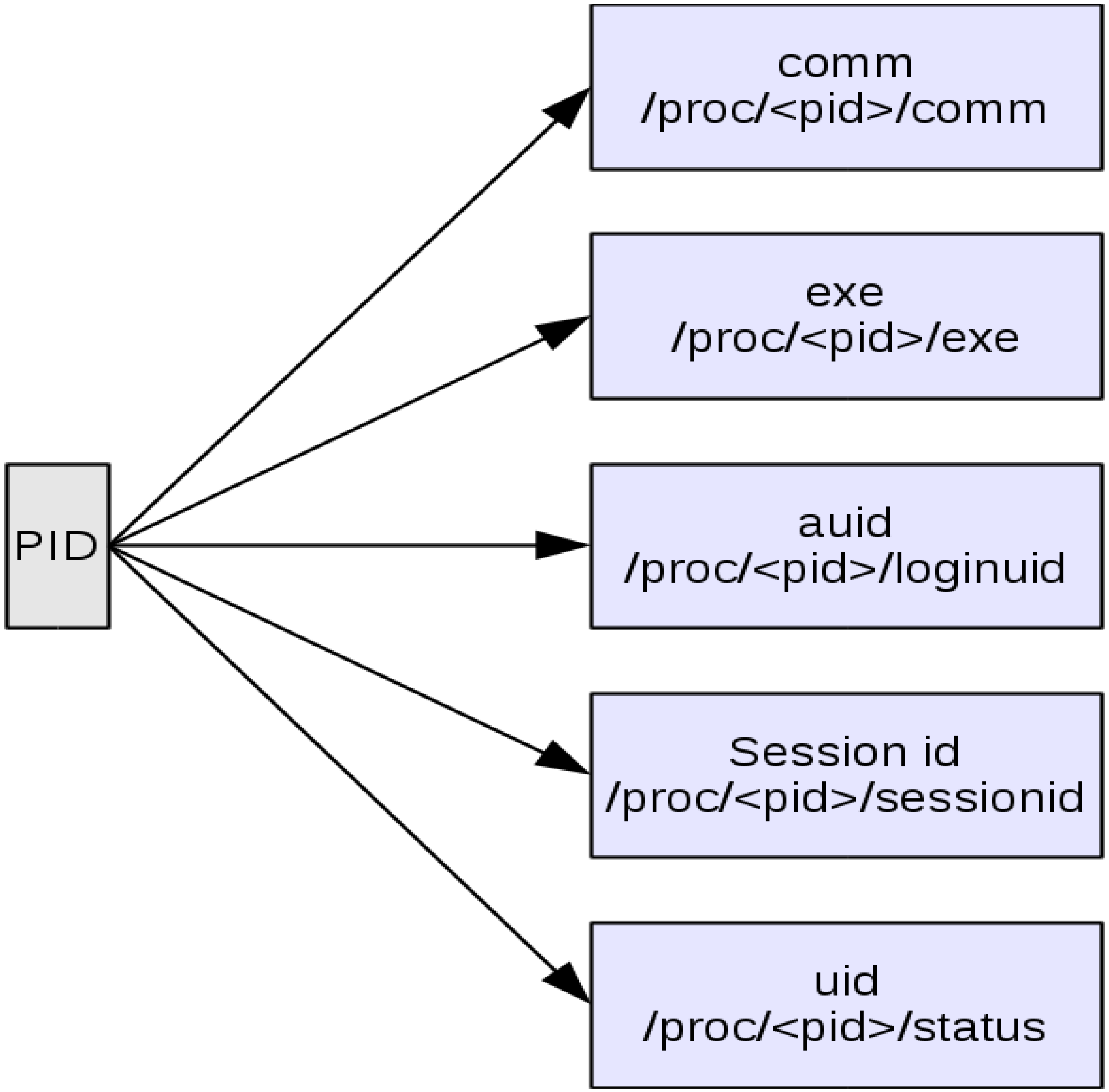
- File Access Notifications
  - Available since Linux 2.6.37
  - Allows recursive monitoring within a mount point
  - Allows user space to say yes or no to file access
  - Hands the monitor an open file descriptor for reading
  - Originally designed for virus scanning
- Drawbacks
  - No notification on deletions, renames, or file moves
  - Requires CAP_SYS_ADMIN

Fanotify Event

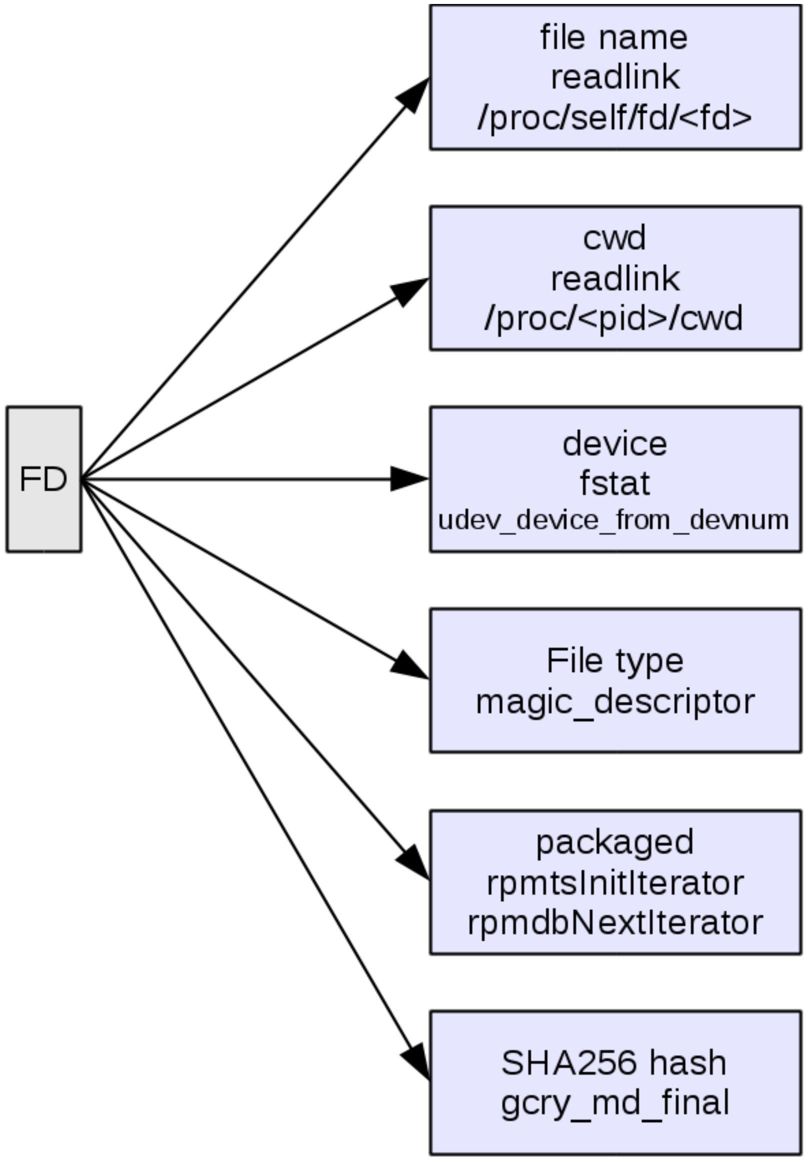
- Open a descriptor with fanotify_init(2)
- Passes a struct back to user space when something happens

```
struct fanotify_event_metadata {
       __u32 event_len;
       __u8 vers;
       __u8 reserved;
       __u16 metadata_len;
       __aligned_u64 mask;
       __s32 fd;
       __s32 pid;
};
```

What can we get from that?



What else can we get from that?



Access control policy

- Current policy is in the following format
  - decision subject= object=
  - decision pattern=
  - Decision
    - allow, allow_audit, deny, deny_audit
  - Subject attributes
    - All, auid, uid, sessionid, pid, comm, exe, exe_dir, exe_type, exe_device, pattern
  - Object attributes
    - All, path, dir, device, ftype, sha256hash

Can have multiple subject and objects, they are “anded”

Subject statements

- all – no args
- auid = number or name
- uid = number or name
- sessionid = number
- pid = number
- comm = string up to 15 characters
- exe = full path to executable
- exe_dir = full path to directory or execdirs, systemdirs, untrusted
- exe_type = mime type (file --mime-type /path-to-file)
- exe_device – full path to device (/dev/sr0)

Object Statements

- all – no args
- path = string, full path
- dir = full path to directory or execdirs, systemdirs, unpackaged
- device = /dev/something
- ftype = mime type
- Sha256hash = hex number

execdirs: /usr, /bin, /sbin, /lib, /lib64, /usr/libexec
systemdirs: execdirs + /etc

Patterns

<u>Normal</u>

```
dec=allow auid=4325 pid=4490 exe=/usr/bin/bash file=/usr/bin/ls
dec=allow auid=4325 pid=4490 exe=/usr/bin/bash file=/usr/lib64/ld-2.21.so
dec=allow auid=4325 pid=4490 exe=/usr/bin/ls file=/etc/ld.so.cache
dec=allow auid=4325 pid=4490 exe=/usr/bin/ls file=/usr/lib64/libselinux.so.1
dec=allow auid=4325 pid=4490 exe=/usr/bin/ls file=/usr/lib64/libcap.so.2.24
```

<u>ld.so started</u>

```
dec=allow auid=4325 pid=31684 exe=/usr/bin/bash file=/usr/lib64/ld-2.21.so
dec=allow auid=4325 pid=31684 exe=/usr/lib64/ld-2.21.so file=/usr/bin/ls
dec=allow auid=4325 pid=31684 exe=/usr/lib64/ld-2.21.so file=/etc/ld.so.cache
dec=allow auid=4325 pid=31684 exe=/usr/lib64/ld-2.21.so file=/usr/lib64/libselinux.so.1
dec=allow auid=4325 pid=31684 exe=/usr/lib64/ld-2.21.so file=/usr/lib64/libcap.so.2.24
```

Patterns

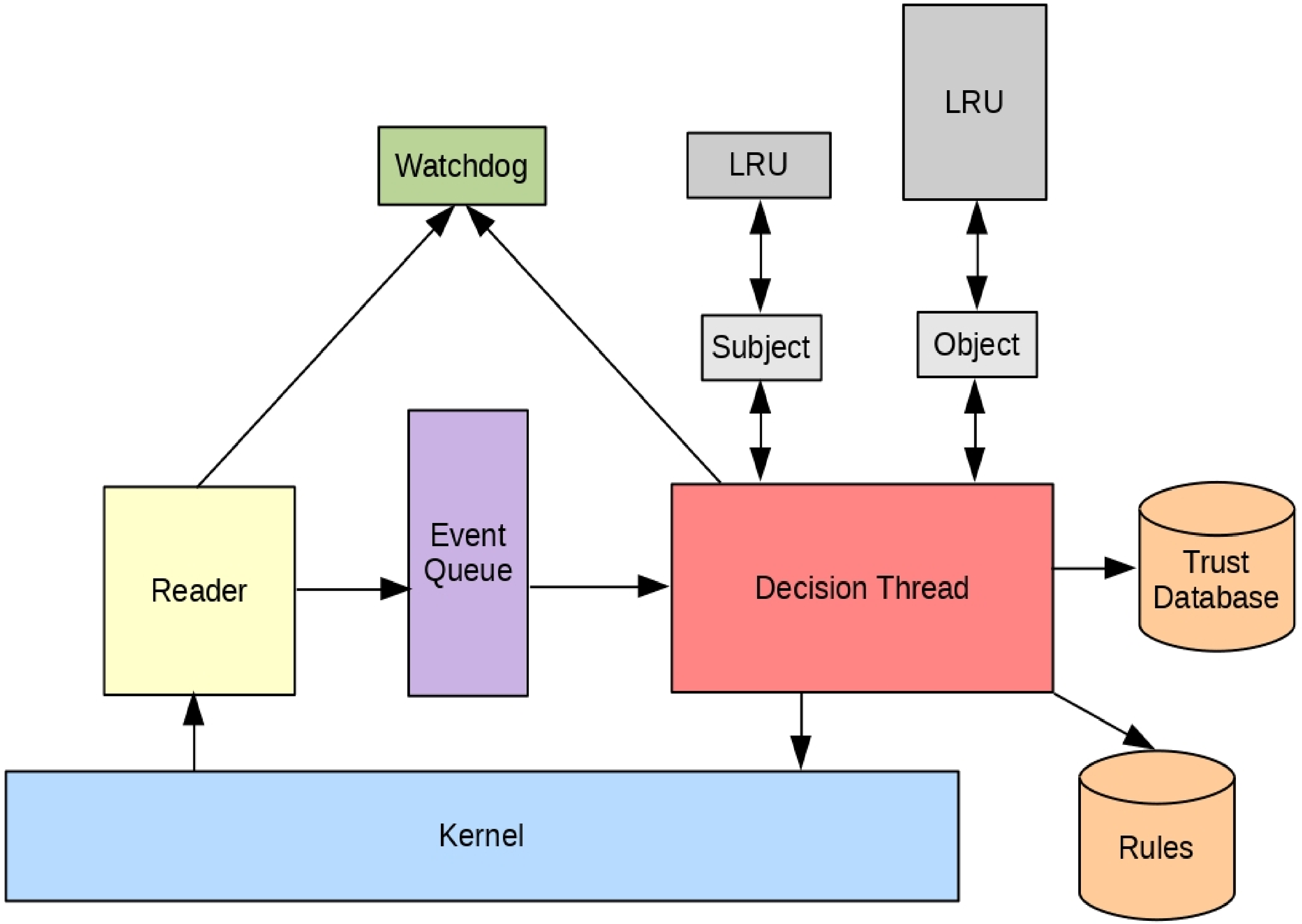
LD_PRELOAD

```
dec=allow auid=4325 pid=31728 exe=/usr/bin/bash file=/usr/bin/ls
dec=allow auid=4325 pid=31728 exe=/usr/bin/bash file=/usr/lib64/ld-2.21.so
dec=allow auid=4325 pid=31728 exe=/usr/bin/ls
file=/usr/lib64/libaudit.so.1.0.0
dec=allow auid=4325 pid=31728 exe=/usr/bin/ls file=/etc/ld.so.cache
dec=allow auid=4325 pid=31728 exe=/usr/bin/ls file=/usr/lib64/libselinux.so.1
dec=allow auid=4325 pid=31728 exe=/usr/bin/ls file=/usr/lib64/libcap.so.2.24
```

Sample policy

```
# Prevent execution by ld.so
deny_audit pattern=ld_so all

# Don't allow LD_PRELOAD
deny_audit pattern=ld_preload all

# Don't allow unpackaged executables
deny_audit exe_dir=execdirs exe=untrusted all

# Only allow system ELF Applications
allow all dir=execdirs ftype=application/x-executable
deny_audit all ftype=application/x-executable

# Only allow system ELF libs
allow all dir=execdirs ftype=application/x-sharedlib
deny_audit all ftype=application/x-sharedlib

# Only allow system python executables and libs
allow all dir=execdirs ftype=text/x-python
allow exe=/usr/bin/python2.7 dir=execdirs ftype=text/x-python
deny_audit  all ftype=text/x-python
```

Design


Watchdog
LRU
LRU
Subject
Object
Reader
Event Queue
Decision Thread
Trust Database
Rules
Kernel

Shipped policy design goals

- 1) No bypass of security by executing programs via ld.so.
- 2) No injection of code by LD_PRELOAD
- 3) All approved executables must be packaged or trusted. Unpackaged or untrusted programs can't run.
- 4) Elf and python files/shared objects must come from system directories.
  - This prevents LD_LIBRARY & PYTHON_LIBRARY redirection to an attacker controlled dir.
- 5) Other languages are not allowed or must be enabled.

Stats report

```
Allowed accesses: 14354
Denied accesses: 0

File access attempts from oldest to newest as of Thu Sep 29 19:00:49 2016

        FILE                                                    ATTEMPTS
---------------------------------------------------------------------------
/usr/lib64/libnspr4.so                                              5
/usr/sbin/unix_chkpwd                                               3
/usr/lib64/libcrypt-2.23.so                                         4
/usr/lib64/libaudit.so.1.0.0                                        4
/usr/lib64/libcap-ng.so.0.0.0                                       4

---

Object queue size: 4096
Object slots in use: 3073
Object hits: 4104
Object misses: 5949
Object evictions: 2876
```

Demo

Findings (so far...)

- Some applications are putting code in your homedir
  - Kodi
  - R Studio
  - libreoffice

Refinements

- Fanotify needs kernel work
  - Need to know open is because of execve
    - Improved cache management
    - Required for accurate pattern matching
  - Really wished we could get notification on process exit
    - Improve cache management
  - More efficient if we had a stat buf passed in event
- Needs to handle yum/dnf/rpm install/update/remove
- Other trust sources besides rpm database such as SWID

Questions?

sgrubb @redhat.com

github.com/stevegrubb/fapolicyd