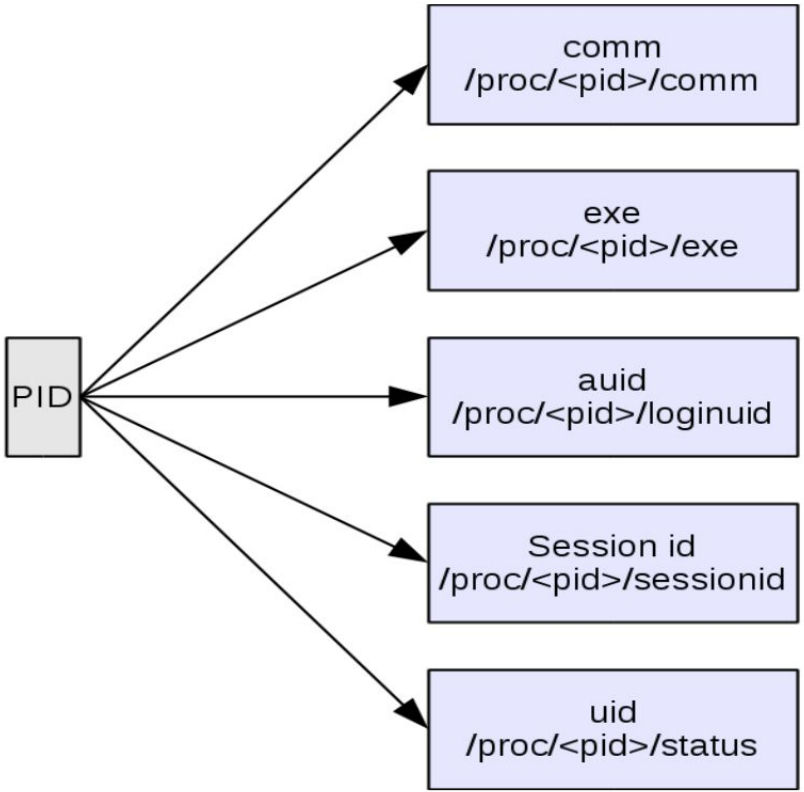
# fapolicyd for fun and profit

Derek Thurston

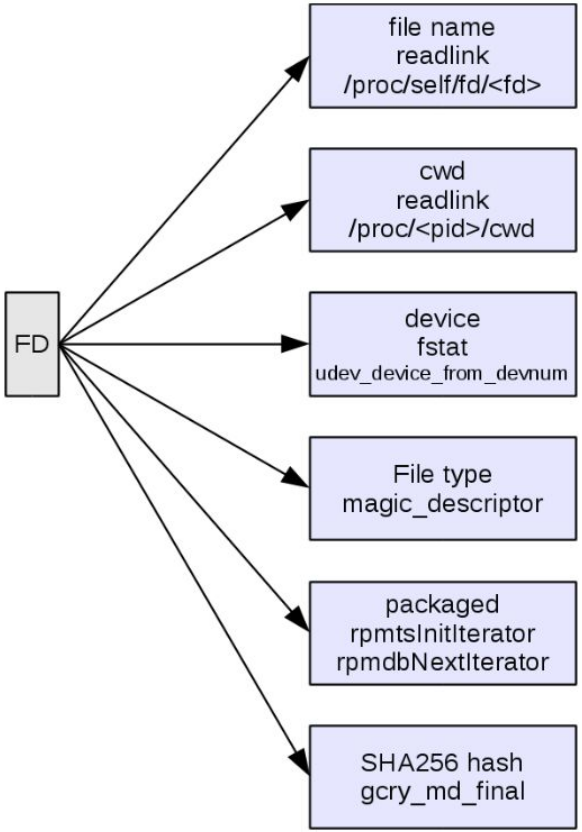
# First, How do programs execute on Linux?

1. Bash checks if it's an internal command (alias, bind, builtin, command, declare, echo, enable, help, let, local, logout, printf, read, shopt, type, typeset, ulimit or unalias), and handles it.
2. If its a subshell:
   a. it forks and starts reading lines and performing them
3. Else: forks, sets up pipes, calls execve (filename, argv, envp)
4. The Linux Kernel has a list of supported formats
   a. ia_32aout
   b. Flat
   c. Aout
   d. Script
   e. Elf
5. It iterates through each handler until one accepts the file

# What can we get from that?



# What else can we get from that?



# What is fapolicyd?

The “File Access Policy Daemon”

The fapolicyd software framework controls the execution of applications based on a user-defined policy.

Allow or Deny execution rules can be defined based on a

- path
- hash
- MIME type
- or trust

Needs kernel >= 4.2 (Must support FANOTIFY_OPEN_EXEC_PERM)

# What is fapolicyd (cont)?

The fapolicyd framework is made up of the following components:

- fapolicyd service
- fapolicyd command-line utilities
- fapolicyd RPM plugin
- fapolicyd rule language

# What is fapolicyd (cont)?

The fapolicyd service configuration is located in the /etc/fapolicyd/ directory with the following structure:

- The fapolicyd.rules file contains allow and deny execution rules.
- The fapolicyd.conf file contains daemon’s configuration options. This file is useful primarily for performance-tuning purposes.
- The fapolicyd.trust file contains list of trusted files/binaries for the application whitelisting daemon.

# fanotify

The fanotify API provides notification and interception of filesystem events.

– Available since Linux 2.6.37

– Allows recursive monitoring within a mount point

– Allows user space to say yes or no to file access

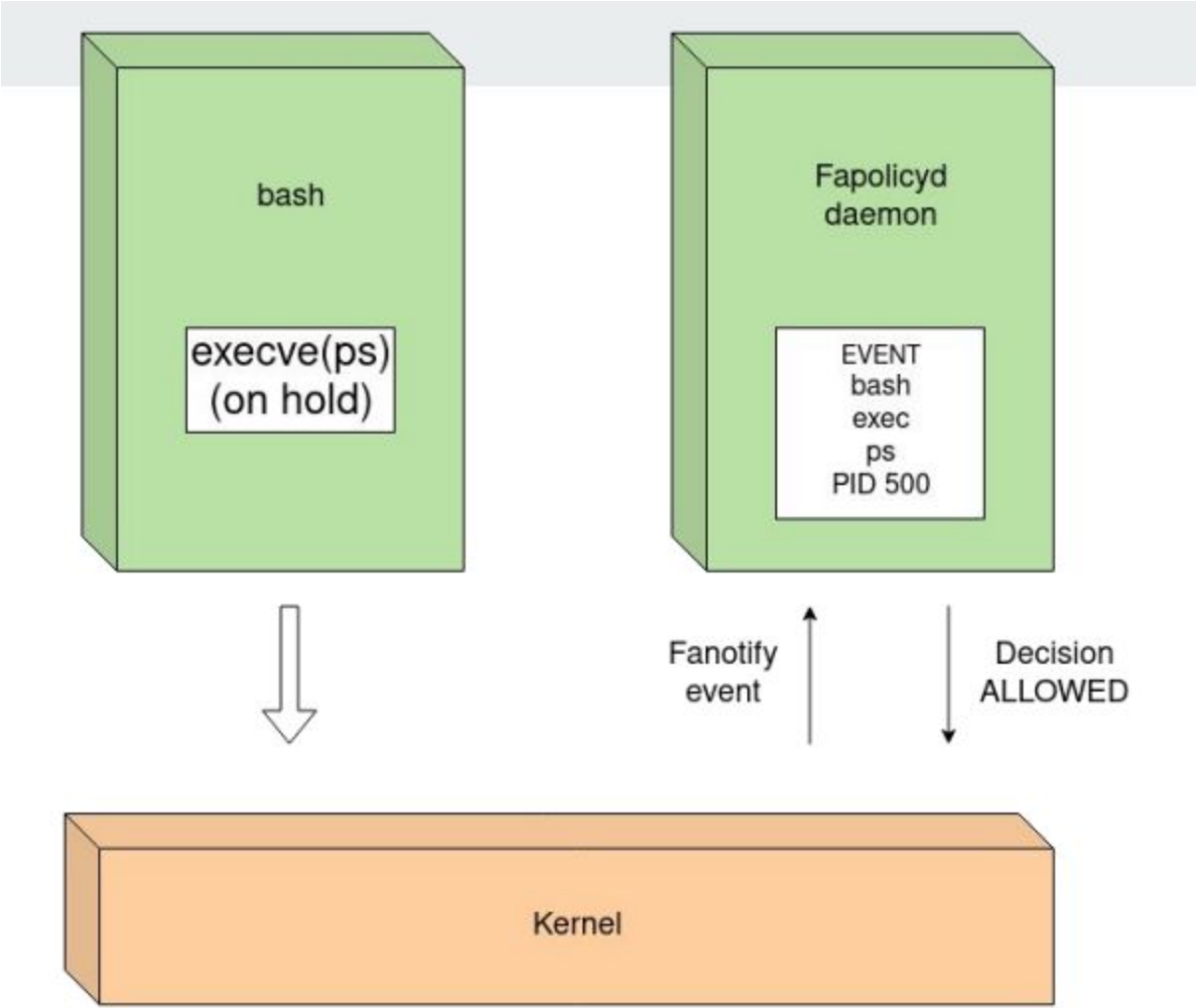
– Hands the monitor an open file descriptor for reading

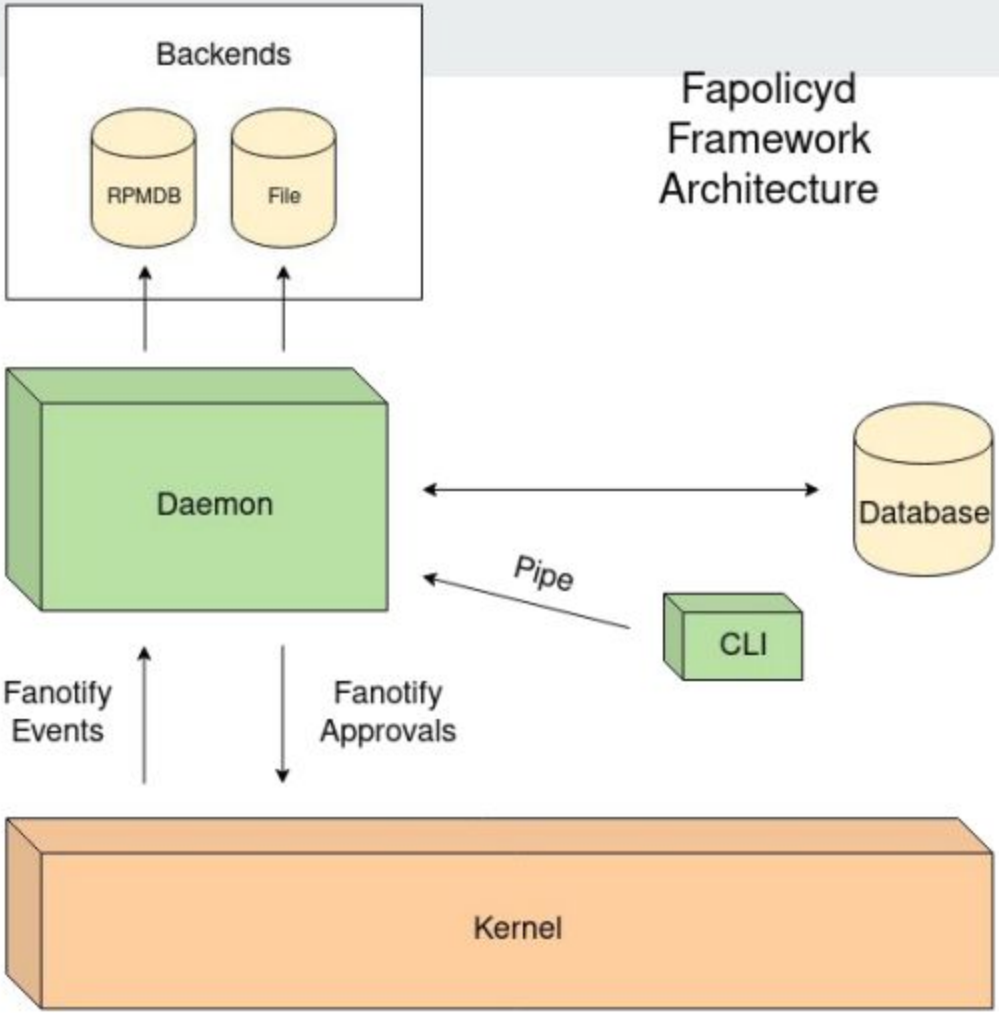
– Originally designed for virus scanning

- Drawbacks

– No notification on deletions, renames, or file moves (only the following system calls are used with this API: fanotify_init(2), fanotify_mark(2), read(2), write(2), and close(2).)

– Requires CAP_SYS_ADMIN


bash
execve(ps)
(on hold)
Fapolicyd
daemon
EVENT
bash
exec
ps
PID 500
Fanotify
event
Decision
ALLOWED
Kernel


Backends
RPMDB
File
Fapolicyd
Framework
Architecture
Daemon
Database
Pipe
CLI
Fanotify
Events
Fanotify
Approvals
Kernel

# Security

# A comparison to other approaches

Antivirus is a Denylisting approach – We define/detect known malware – Much more “out there” that we don’t know about

MAC’s – Restrict based on behavior or subject / object rules around information flow/access. – Provenance is not taken into account

Application Allowlisting – It’s simpler to say this is what we know about

# 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

# 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.

# fapolicy-cli

```
[root@f35 ~]# fapolicyd-cli --list

->
%languages=application/x-bytecode.ocaml,application/x-bytecode.python,application/java-archive,text/x-java,application/x-java-applet,application/javascript,text/javascript,text/x-awk,text/x-gawk,text/x-lisp,application/x-elc,text/x-lua,text/x-m4,text/x-perl,text/x-php,text/x-python,text/x-R,text/x-ruby,text/x-script.guile,text/x-tcl,text/x-luatex,text/x-systemtap

1. allow perm=any uid=0 : dir=/var/tmp/

2. allow perm=any uid=0 trust=1 : all

3. deny_audit perm=any pattern=ld_so : all

4. allow perm=open exe=/usr/bin/rpm : all

5. allow perm=open exe=/usr/bin/python3.10 comm=dnf : all

6. deny_audit perm=any all : ftype=application/x-bad-elf

7. allow perm=open all : ftype=application/x-sharedlib trust=1

8. deny_audit perm=open all : ftype=application/x-sharedlib

9. allow perm=execute all : trust=1

10. allow perm=any uid=0 : dir=/tmp/ansible

11. allow perm=any uid=0 : dir=/root/.ansible/tmp/

12. allow perm=open all : ftype=%languages trust=1
```

# 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
```

# Installation

# yum install fapolicyd

# systemctl enable - - now fapolicyd

Or

https://github.com/linux-application-whitelisting/fapolicyd/blob/main/BUILD.md

# Configuration

The fapolicyd service configuration is located in the /etc/fapolicyd/ directory with the following structure:

- The fapolicyd.rules file contains allow and deny execution rules.
- The fapolicyd.conf file contains daemon’s configuration options. This file is useful primarily for performance-tuning purposes.

# Writing rules

You can use one of the ways for fapolicyd integrity checking:

- file-size checking
- comparing SHA-256 hashes
- Integrity Measurement Architecture (IMA) subsystem

By default, fapolicyd does no integrity checking. Integrity checking based on the file size is fast, but an attacker can replace the content of the file and preserve its byte size. Computing and checking SHA-256 checksums is more secure, but it affects the performance of the system. The integrity = ima option in fapolicyd.conf requires support for files extended attributes (also known as *xattr*) on all file systems containing executable files.

# 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

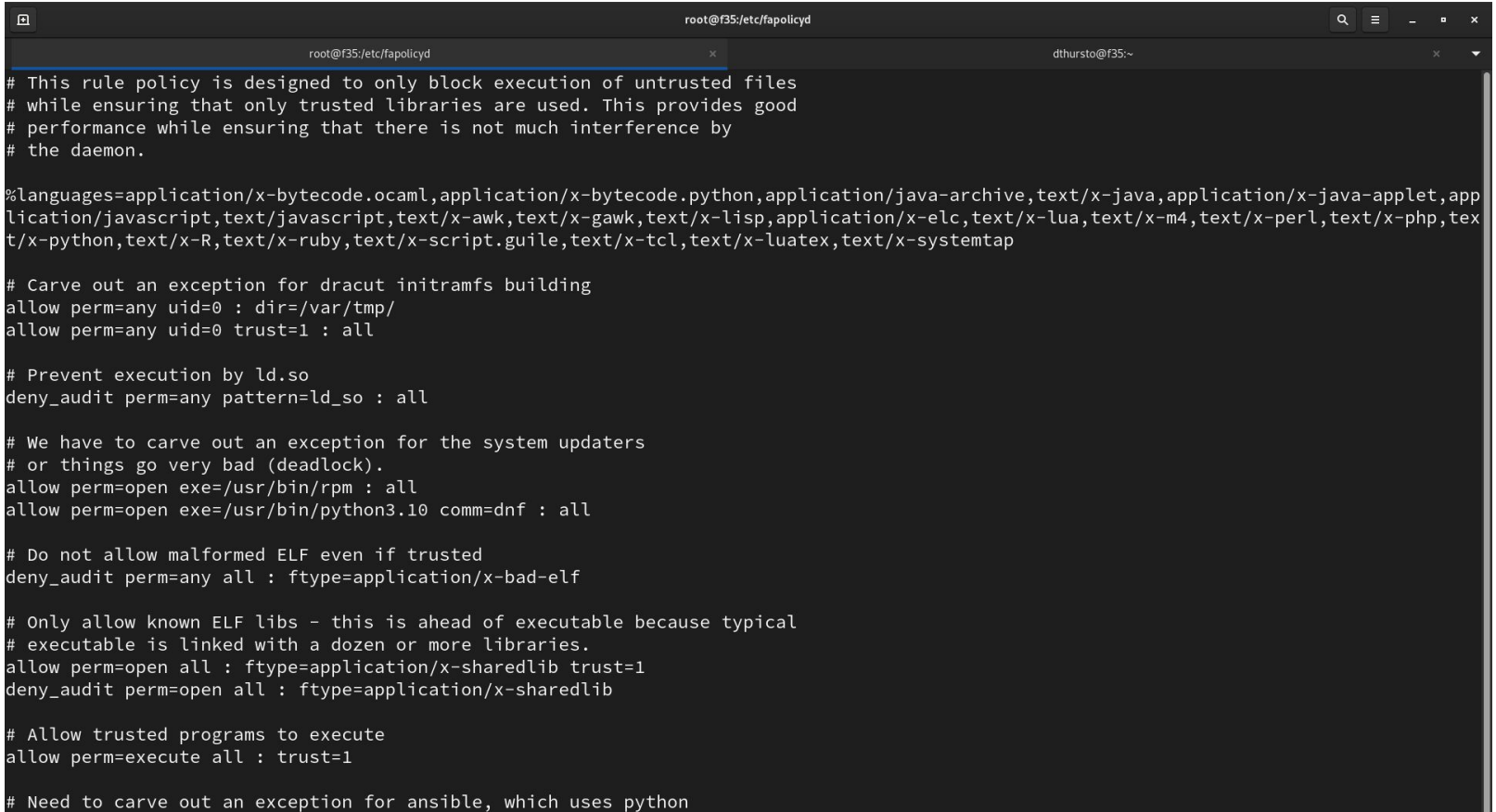
execdirs: /usr, /bin, /sbin, /lib, /lib64, /usr/libexec

systemdirs: execdirs + /etc

# Patterns

```
rule=9 dec=allow perm=execute auid=1000 pid=27432 exe=/usr/bin/bash : path=/usr/bin/ls ftype=application/x-executable trust=1

rule=17 dec=allow perm=open auid=1000 pid=27432 exe=/usr/bin/bash : path=/usr/bin/ls ftype=application/x-executable trust=1

rule=9 dec=allow perm=execute auid=1000 pid=27432 exe=/usr/bin/bash : path=/usr/lib64/ld-linux-x86-64.so.2
ftype=application/x-sharedlib trust=1

rule=7 dec=allow perm=open auid=1000 pid=27432 exe=/usr/bin/bash : path=/usr/lib64/ld-linux-x86-64.so.2
ftype=application/x-sharedlib trust=1

rule=17 dec=allow perm=open auid=1000 pid=27432 exe=/usr/bin/ls : path=/etc/ld.so.cache ftype=application/octet-stream trust=0

rule=7 dec=allow perm=open auid=1000 pid=27432 exe=/usr/bin/ls : path=/usr/lib64/libselinux.so.1 ftype=application/x-sharedlib
trust=1

rule=7 dec=allow perm=open auid=1000 pid=27432 exe=/usr/bin/ls : path=/usr/lib64/libcap.so.2.48 ftype=application/x-sharedlib
trust=1

rule=17 dec=allow perm=open auid=1000 pid=27432 exe=/usr/bin/ls : path=/usr/lib64/libc.so.6 ftype=application/x-executable trust=1

rule=7 dec=allow perm=open auid=1000 pid=27432 exe=/usr/bin/ls : path=/usr/lib64/libpcre2-8.so.0.10.2 ftype=application/x-sharedlib
trust=1
```

# Sample policy (/etc/fapolicyd/fapolicyd.rules)




```
# This rule policy is designed to only block execution of untrusted files
# while ensuring that only trusted libraries are used. This provides good
# performance while ensuring that there is not much interference by
# the daemon.

%languages=application/x-bytecode.ocaml,application/x-bytecode.python,application/java-archive,text/x-java,application/x-java-applet,app
lication/javascript,text/javascript,text/x-awk,text/x-gawk,text/x-lisp,application/x-elc,text/x-lua,text/x-m4,text/x-perl,text/x-php,tex
t/x-python,text/x-R,text/x-ruby,text/x-script.guile,text/x-tcl,text/x-luatex,text/x-systemtap

# Carve out an exception for dracut initramfs building
allow perm=any uid=0 : dir=/var/tmp/
allow perm=any uid=0 trust=1 : all

# Prevent execution by ld.so
deny_audit perm=any pattern=ld_so : all

# We have to carve out an exception for the system updaters
# or things go very bad (deadlock).
allow perm=open exe=/usr/bin/rpm : all
allow perm=open exe=/usr/bin/python3.10 comm=dnf : all

# Do not allow malformed ELF even if trusted
deny_audit perm=any all : ftype=application/x-bad-elf

# Only allow known ELF libs - this is ahead of executable because typical
# executable is linked with a dozen or more libraries.
allow perm=open all : ftype=application/x-sharedlib trust=1
deny_audit perm=open all : ftype=application/x-sharedlib

# Allow trusted programs to execute
allow perm=execute all : trust=1

# Need to carve out an exception for ansible, which uses python
```

# Generate rules

You can test by starting the daemon from the command line. Before starting the daemon, `cp /usr/bin/ls /usr/bin/my-ls` just to setup for testing.

When testing new policy, its highly recommended to use the permissive mode to make sure nothing bad happens. It really is not too hard to deadlock your system. Continuing on with the tutorial, as root start the daemon as follows:

/usr/sbin/fapolicyd --permissive --debug

In permissive + debug mode you will see dec=deny which means "decision is to deny". But the program will actually be allowed to run.

# enable fapolicyd integrity checks

1. Open the /etc/fapolicyd/fapolicyd.conf file in a text editor of your choice, for example:
   # vi /etc/fapolicyd/fapolicyd.conf
2. Change the value of the integrity option from none to sha256, save the file, and exit the editor:
   integrity = sha256
3. Restart the fapolicyd service:
   # systemctl restart fapolicyd

# Real life example

install

ausearch --start today -m fanotify -i

https://github.com/linux-application-whitelisting/fapolicyd/issues/84

# FAQ

1. Can this work with other distributions?

Absolutely! There is a backend API that any trust source has to implement. This API is located in fapolicyd-backend.h. A new backend needs an init, load, and destroy function. So, someone who knows the debian package database, for example, could implement a new backend and send a pull request. We are looking for collaborators.

# FAQ (cont)

2. Can SE Linux or AppArmor do this instead?

SE Linux is modeling how an application behaves. It is not concerned about where the application came from or whether it's known to the system. Basically, anything in /bin gets bin_t type by default which is not a very restrictive label. MAC systems serve a different purpose. Fapolicyd by design cares solely about if this is a known application/library. These are complimentary security subsystems. There is more information about application whitelisting use cases at the following NIST website:

https://www.nist.gov/publications/guide-application-whitelisting

# DEMO TIME

# demo steps

# systemctl status fapolicyd.service

# yum install fapolicyd

# systemctl enable fapolicyd --now

# systemctl status fapolicyd.service

# cd /etc/fapolicyd/

# cat fapolicyd.rules

# cat fapolicyd.trust

# cat fapolicyd.conf

# ausearch --start today -m fanotify -i

# fapolicyd-cli --list

# fapolicyd-cli --help

# demo steps (cont)

```
$ cp /bin/ls /tmp

$ /tmp/ls

-bash: /tmp/ls: Operation not permitted

# fapolicyd-cli --file add /tmp/ls

# fapolicyd-cli --file add /tmp/ls

# fapolicyd-cli --update

$ /tmp/ls

# stop fapolicyd

# vi /etc/fapolicyd/fapolicyd.trust

# fapolicyd-cli --update

$ /tmp/ls -l

# systemctl stop fapolicyd.service

# systemctl status fapolicyd.service

# fapolicyd --debug 2> fapolicy.out &

$ /tmp/ls -l

# fg

# less fapolicy.out

# vi /etc/fapolicyd/fapolicyd.rules
```

stop fapolicyd.service

systemctl status fapolicyd.service

run in debug mode now:

# fapolicyd --debug 2> fapolicy.output &

# references


https://static.sched.com/hosted_files/lssna19/ce/application-whitlisting-sgrubb.pdf

https://people.redhat.com/sgrubb/files/application-whitelisting-2018.pdf

https://github.com/linux-application-whitelisting/fapolicyd

https://www.redhat.com/en/blog/stop-unauthorized-applications-rhel-8s-file-access-policy-daemon

https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-167.pdf

https://rsroka.fedorapeople.org/fapolicyd-fosdem.pdf

https://github.com/dthurston/fapolicyd-configuration