Running daemons non-root

Simon J. Gerraty
Juniper Networks, Inc.
2019

Imagine something very witty here

Agenda

- Introduction
- Daemons need privileges
- Approach
- Progress
- Further work
- Q&A

Introduction

- Running daemons non-root was goal 20 years ago
- Hacking kernel always an option
- Modern FreeBSD offers better solutions
  - Capabilities (Capsicum)
  - Mandatory Access Control (MAC)

Daemons need privileges

- open AF_UNIX sockets in protected dirs
- open raw sockets
- bind reserved ports
- set fib (routing instance)
- read[write] routing socket
- set sysctl values
- tweak rlimits
- configure devices
  - read/write /dev/mem
CLI needs privileges too

- setuid
  - open MGD management socket
  - run ping, traceroute with restricted options
- careful to drop privs when not needed
  - raising privs controlled by MGD (uses fine grained permissions control)
- better if simply run as user?
  - possibly safer to remain setuid for opening management socket then permanently drop privs

Goal

- run daemons as unprivileged user
  - minimize collateral damage from bugs and exploits
  - use of Verified Exec mitigates local exploits
- allow controlled and specific privilege escalation
  - just enough to do the operations needed
- allow gradual transition
  - potentially one daemon at a time
- many filesystem related privileges could be addressed by redesign
  - subdir of /var/run/ with group write permissions
  - makes transition more disruptive

Hack the kernel?

- simple (for some value of simple) if brutal
- Cheswick and Bellovin [ChBe94] took this approach
- maintenance nightmare
- we did not go there ;-)

Capabilities

- Capsicum offers light weight Capabilities mode
- In FreeBSD; capabilities can be passed/inherited like file descriptors
- Mostly aimed at limiting what a process can do
- Run process in a sandbox with no escape
- Need for proxy to handle global lookups
- Launchd can simplify granting capabilities
  - centralized configuration and control
- Can use capabilities without entering Capability mode
- Non-trivial redesign
Capsicum Chromium example

- Watson et al [WALK10] provide a comparison of methods used to sandbox Chromium browser:

<table>
<thead>
<tr>
<th>OS</th>
<th>Model</th>
<th>Line count</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>ACLs</td>
<td>22,350</td>
<td>Windows ACLs and SIDs</td>
</tr>
<tr>
<td>Linux</td>
<td>chroot</td>
<td>605</td>
<td>setuid root helper sandboxes renderer</td>
</tr>
<tr>
<td>Mac OS X</td>
<td>Seatbelt</td>
<td>560</td>
<td>Path-based MAC sandbox</td>
</tr>
<tr>
<td>Linux</td>
<td>SELinux</td>
<td>200</td>
<td>Restricted sandbox type enforcement domain</td>
</tr>
<tr>
<td>Linux</td>
<td>seccomp</td>
<td>11,301</td>
<td>seccomp and userspace syscall wrapper</td>
</tr>
<tr>
<td>FreeBSD</td>
<td>Capsicum</td>
<td>100</td>
<td>Capsicum sandboxing using cap_enter</td>
</tr>
</tbody>
</table>

- Chrome design lends itself to this treatment

Mandatory Access Control - MAC

- Framework to control interactions between subjects and objects
  - subjects and objects may be labeled
- Key to success is suitable mac_* API calls throughout the kernel
  - checks for whether current process cr_uid == 0 (ie are we root) replaced with calls to priv_check_cred which calls various mac_check_*
  - MAC modules register to receive mac_* calls.
- Generally serves to limit access
  - mac_priv_grant is the exception!
  - priv_check_cred hence mac_priv_grant; have no visibility to object of interest, only subject requesting.

MAC continued

- MAC labels are free form text meaningful to one or more MAC modules.
- MAC modules/tools can set labels on many subjects and system objects
- Latest mac_veriexec can bind labels to verified file objects
  - limited to immutable files
  - mac_veriexec just stores labels, it does not use them

Approach: mac_grantbylabel

- New mac module to leverage mac_priv_grant and labels via mac_veriexec
- Initially minimize code changes to Junos
  - remove explicit checks for running as root if GBL label set
  - add uid and gid tokens to jlaunchd parser
    - if app has GBL label run as specified user
    - eases upgrade/downgrade issues
- allow addressing daemons one at a time
- eventually tackle filesystem layout changes
Recap: **mac_veriexec**

- reimplemention of Verified Exec (from NetBSD originally) as MAC module
- `sbin/veriexec` loads signed manifest content
  - `ioctl` to `/dev/veriexec` feeds `mac_veriexec`
- manifest provides *fingerprint* (hash) flags and more:
  - `sbin/veriexec` sha256=cafebabe... trusted
  - `sbin/verify-sig` sha256=2cafebabe... no_ptrace
  - `usr/bin/python` sha256=deadbeef... indirect
  - `usr/libexec/ftpd` sha256=0ffedead... no_fips

Recap: **mac_veriexec cont.**

- fingerprint and other data tracked per inode (`dev, fileid, gen`)
- fingerprint evaluation status cached in `vnode->v_label`
  - evaluation optimized for verified filesystem

**Use mac_veriexec to**

- prevent *unsigned*
  - apps running
  - kernel modules loading
  - shared libs linking
- indirect prevents direct execution of interpreters eg Python, Ruby etc.
- no_ptrace prevents ptrace of sensitive apps
- no_fips prevents apps running in FIPS mode
- trusted (implies no_ptrace) allowed to write `/dev/veriexec`
- Junos package system uses `veriexec -x $file` to test for verified

**maclabel set via veriexec**

- labels are free-form text (meaningful only to relevant MAC module)
- comma separated list of `module/value` tokens:

  ```
  $ grep label= manifest
  usr/sbin/snmpd sha256=efffeea6babe... label=gbl/daemon
  usr/sbin/rpd sha256=cee8c666... label=gbl/daemon,gbl/rtsock
  ```

- `gbl/daemon` maps to several `GBL_*` bits
- latest `veriexec` passes them to kernel (`mac_veriexec`) for storage along with hash (fingerprint) and flags
**priv_check_cred at a glance**

- in the long ago; kernel just checked for super user: `cred->cr_uid == 0`
- replaced with calls to `priv_check(td, priv)` or `priv_check_cred(cred, priv, flags)`
  - `mac_priv_check(cred, priv)` can say *NO*
  - `prison_priv_check(cred, priv)` can say *NO*
  - if `suser_enabled` and `cr_uid == 0` *YES*
  - `mac_priv_grant(cred, priv)` can say *YES!*
  - default result: *NO (EPERM)*

**mac_grantbylabel**

- simple MAC module
- during `exec(2)` ask `mac_veriexec` for label associated with `curproc->p_textvp`
- parse label and any `gbl/*` tokens set `GBL_*` bits in module specific label (stored in `curproc->p_textvp->v_label`)
- `gbl_label_t` is `uint32_t` for trivial storage
- when `priv_check_cred` calls `mac_priv_grant` check if label contains relevant bit and return success if so.

**Privileged operations**

- `sys/priv.h` lists over 200 separate `PRIV_*`
- `mac_grantbylabel` compresses these into `GBL_*` each of which represents a group:

  ```c
  case PRIV_NETINET_BINDANY:
  case PRIV_NETINET_RESERVEDPORT: /* socket bind low port */
  case PRIV_NETINET_REUSEPORT:
    if (label & GBL_BIND)
      rc = 0;
    break;
  ```

- so far 7 `GBL_*` bits cover the privileges our daemons need.

**Run CLI as user?**

- set label on CLI so it can open MGD management socket?
  - unlike daemons CLI is much more exposed to user, might be safer to rely on setuid to open socket then permanently drop
  - note: `priv_check_cred` hence `mac_priv_grant` have no visibility to object
- set label on `ping` and `traceroute` so they can operate without root privs.
  - again more potential for abuse
- bottom line; leave as is
Run daemons non-root

- label for necessary privs in manifest entry
- tweak jlaunchd.conf entry to specify [default] uid and gid to use
- jlaunchd ignores uid if no GBL label set for daemon
- remove explicit checks for uid 0
- can migrate one at a time
- minimal code change during transition

Progress

- proof of concept complete ?
- if rpd can work non-root anything can ;-)  
- chassisd might be as or more challenging
- more interesting applications of GBL labels also tested

Further work

- each daemon needs testing to ensure all privs accounted for
  - huge effort from multiple teams
- avoiding/reducing need for filesystem privs would be best
  - requires re-work of runtime environment
  - more extensive code changes
  - best tackled after majority of daemons addressed
- possibly use mac_vnode_*_check to limit scope of remaining filesystem privs
- Other applications ...

Python

- Junos has run only signed code since 2005
- Allowing unsigned Python (or Ruby etc) is insane!!!
- Shipped Python interpreter (/usr/bin/python) cannot be run directly
  - all scripts must be signed
  - all imports must be signed
- For internal developers we have an unrestricted interpreter

Running unsigned Python

- Some customers want ability to run unsigned python too
  - provide un-restricted python ?
  - turn off veriexec ?
  - allow self signing ?
- mac_grantbylabel can help
Running unsigned Python within limits

- Zero Touch Provisioning (ZTP) is a popular use-case
- Data center users want to leverage Python
- Self signing won't work until trust anchors installed
- Allow only specific application (eg. `dhclient`) to run unsigned python
  - new PRIV_VERIEXEC_*
  - `mac_veriexec` can call `mac_priv_grant` as needed
  - `mac_grantbylabel` can allow override of PRIV_VERIEXEC_* (such as indirect flag) if have GBL_VERIEXEC
  - totally scary and evil but alternatives are far worse
- As with all privileges granted by `mac_grantbylabel` cannot be inherited.

Running unsigned Python cont.

- suitably labeled app tries to directly exec interpreter in child process
- `mac_veriexec` spots indirect flag and calls `mac_priv_grant(PRIV_VERIEXEC_DIRECT)`
- `mac_grantbylabel` checks v_label for GBL_VERIEXEC
  return success if set, after setting GBL_VERIEXEC in curproc->p_label

- child (running interpreter) tries to read unsigned script
- `mac_veriexec` spots failure of O_VERIFY and calls
  `mac_priv_grant(PRIV_VERIEXEC_NOVERIFY)`
- `mac_grantbylabel` checks p_label for GBL_VERIEXEC
  return success if set.

exec_script

- API to seamlessly deal with unsigned scripts

  ```c
  int execv_script(const char *interpreter, char * const *argv);
  ```

- if (script = argv[0]) is signed, simply `execv(script, argv)`

- if we have suitable GBL_VERIEXEC in label
  - if interpreter not provided, obtain from start of script (eg. `#!/usr/bin/python`)
  - syslog running script via interpreter
  - `execv(interpreter, argv)`
Q&A

- Questions

[ChBe94] William R. Cheswick; Steven M. Bellovin: *Firewalls and Internet Security*. Addison-Wesley 1994
Reading, Massachusetts

[WALK10] Robert N. M. Watson; Jonathan Anderson; Ben Laurie; Kris Kennaway: *Capsicum: practical capabilities for UNIX*. 2010

Author: sjg@juniper.net
Revision: $Id: run-daemons-non-root-slides.txt,v 3022cc0b0c8b1 2019-05-18 13:00:29Z sjg $
Copyright: Juniper Networks, Inc.