Adding verification to FreeBSD loader

aka; loader verified exec

Simon J. Gerraty

Juniper Networks, Inc.

2018

Imagine something very witty here

Agenda

- Introduction
- Verified Exec in Junos
  - Secure boot
- Manifests
- loader veriexec
- Q&A

Veriexec in Junos

- Introduced in Junos 7.2 (2005) for FIPS-140-2
  - originally from NetBSD
  - added support for signed manifests
  - relied on raising `securelevel`
  - Junos kernel approximately FreeBSD 4.2
- General release in Junos 7.5 (2005)
  - added `boot -x` safety belt; never needed
- Blocks *script kiddies*
- Mitigates famous vulnerabilities

Veriexec in BSDX

- re-implemented as `mac_veriexec` for FreeBSD 10
- avoids kernel hacks
- suitable for up-streaming

Veriexec manifests

- list of pathnames, hashes, flags and labels:

```
sbin/init sha1=d88f88c24d91b87e6c072d5bce60582ada890cfa
sbin/veriexec sha1=5a8b6e3944185c98795986e24a260a711b6a024a no_ptrace trusted
usr/bin/python sha1=0234c35ac932d2dc8738e84128ec1d552df9d501 indirect
```

- Junos manifests add `uid` and other fields:

```
sbin/veriexec sha1=958a4da868abb2e2aa913cece234beb688085b4c uid=0 gid=0 mode=555 no_ptrace trusted

usr/sbin/adaemon  sha1=cafebabe... label=maclabel(7)
```

- support for `sha256` hashes

BSDX (XML) packages

- `package.xml` contains all meta data
  - various tags and toggles allow package system to do it's job
- signed `manifest` providing fingerprints for content
- most content is in an iso image (`cd9660`)
  - iso image has it's own signed `manifest` for its content
- some packages provide modules that need to be pre-loaded

BSDX kernel package

- kernel package is somewhat atypical:

```
(cd /packages/sets/active/os-kernel && find * -type f)
boot/miibus.ko
boot/if_fxp.ko
boot/if_igb.ko
boot/if_ixlv.ko
boot/loader.conf
boot/if_em.ko
boot/contents.izo
boot/kernel
manifest
manifest.ecerts
manifest.esig
package.xml
```

BSDX runtime package

- most packages look more like this:

```
(cd /packages/sets/active/os-runtime && find * -type f)
contents/contents.izo
contents/contents.symlinks
contents/files.tar
manifest
manifest.ecerts
manifest.esig
package.xml
scripts/mounted.sh
scripts/downgrade.sh
```

BSDX modules package

- of more interest to the `loader`:

```
(cd /packages/sets/active/junos-modules && find * -type f)
boot/hmac_drbg.ko
boot/fips_core.ko
boot/sdk_core.ko
boot/loader.conf
boot/init.4th
boot/junosprocfs.ko
boot/mac_fips.ko
boot/mac_sdk.ko
contents/contents.izo
contents/contents.symlinks
manifest
manifest.certs
manifest.ecerts
manifest.esig
manifest.sig
package.xml
```

X.509 certificate chains

- X.509 certificate chains allow tracing keys to a trust anchor

```
JuniperRootCA (trust anchor)
 \
 EngineeringCA (intermediate CA)
  \
  PackageCA (intermediate CA)
   \   \
    \  PackageProduction_2018 (signing for releases)
     \
      PackageDevelopment_2018 (signing key for developers)
```

- CA private keys never accessible from network
- Signing private keys stored in signing server/HSM

Manifest signatures

- each manifest is signed:

```
manifest                data
manifest.esig           EC signature
manifest.ecerts         X.509 certificate chain
```

- RSA+SHA1 (`.sig`) deprecated since 2014
  - Junos ignores `.sig` if `.esig` supported

Userland `veriexec`

- must be `root` to run
- verifies signature using supplied certificate chain
  - may need to load extensions to handle 3rd party certificates
  - rejects `manifest` if unverified
- opens each path referenced by `manifest`
  - passes file descriptor, hash, flags and label to kernel
- kernel tracks files by `dev,inode,gen`
  - multiple names and symlinks *just work*
  - copy does *not*

Loader

- loads kernel and modules
  - cannot have *secure boot* if `loader` does not verify
  - only recently practical
- limited functionality and resources
  - filesystem support is *minimal*
- deals with each file only once

Loader verification - goals

- verify everything possible
  - allow for mutable `loader.conf`
  - allow for tunable behavior
- retain flexibility of X.509 certificates
  - key to simple upgrade/downgrade
  - `loader` itself may be signed by whatever means prior boot stage wants
- minimize impact to size, boot time and complexity
- find `manifest` automatically
  - allow explicit load as well

Loader verification - design

- simple data store
  - `manifest` content has to be in memory for verification; so keep it
  - need to track path prefix per `manifest`
    - strictly pathname based lookup
    - verification status tracked by `dev,inode`
  - ordered (by prefix length) linked list of manifest content

BearSSL

- new SSL library by Thomas Pornin; designed for embedded environments
  - library does no memory allocations
  - provides all functionality needed for X.509 certificates and signature verification
- written in forth
- at least an order of magnitude smaller than OpenSSL
- depending on primary boot stage, `loader` may be limited to 640Kb
  - using OpenSSL would have added at least 3Mb to loader
  - using BearSSL less than 100Kb

Fingerprint data store

- a linked list with each element being:

```
struct fingerprint_info {
    char *fi_prefix;           /**< manifest entries relative to */
    char *fi_skip;             /**< manifest entries prefixed with  */
    const char *fi_data;       /**< manifest data */
    size_t fi_prefix_len;      /**< length of prefix */
    size_t fi_skip_len;        /**< length of skip */
    dev_t fi_dev;              /**< device id  */
    LIST_ENTRY(fingerprint_info) entries;
};
```

- list ordered by length of `fi_prefix`; longest and most recent first

Self tests

- FIPS compliance requires running Known Answer Tests before use
  - Test each supported hash method
  - Test verifying each supported signature type
  - OpenPGP signatures can also be supported

```
FreeBSD/x86 bootstrap loader, Revision 1.1
(sjg@kaos.jnpr.net, Sun Nov 19 19:12:21 PST 2017)
Testing hash: sha1                              Passed
Testing hash: sha256                            Passed
Testing verify certificate: EngineeringEcCA     Passed
Testing verify OpenPGP signature:               Passed
```

Loading Junos BSDX

- support for multiple packages and package sets complicates `loader` task
- `loader` sees:

```
/packages/sets/active/boot/os-kernel/kernel
/packages/sets/active/boot/os-kernel/contents.iso
/packages/sets/active/boot/netstack/netstack.ko
```

- which are really:

```
/packages/sets/active/os-kernel -> /packages/db/os-kernel-$version
/packages/sets/active/boot/os-kernel -> /packages/db/os-kernel-$version/boot
/packages/db/os-kernel-$version/manifest
/packages/db/os-kernel-$version/manifest.esig
/packages/db/os-kernel-$version/manifest.ecerts
/packages/db/os-kernel-$version/boot/kernel
```

Loading Junos BSDX example

```
Verified /boot/manifest signed by PackageDevelopmentEc_2018
Verified /boot/boot.4th
Verified /boot/platform.4th
Verified /boot/loader.rc
Verified /boot/junos-menu.4th
...
Unverified: /boot/device.hints: no entry
Verified /packages/sets/active/boot/junos-modules/../manifest signed by PackageDevelopmentEc_2018
Verified /packages/sets/active/boot/junos-modules/loader.conf
Verified /packages/sets/active/boot/junos-modules/init.4th
Unverified: /boot/ffp.cookie: no entry
Verified /packages/sets/active/boot/os-kernel/../manifest signed by PackageDevelopmentEc_2018
Verified /packages/sets/active/boot/os-kernel/loader.conf
Verified /packages/sets/active/boot/os-crypto/../manifest signed by PackageDevelopmentEc_2018
Verified /packages/sets/active/boot/os-crypto/loader.conf
...
Verified /packages/sets/active/boot/os-kernel/kernel
/packages/sets/active/boot/os-kernel/kernel text=0x46f678 data=0x44720+0x30e42c syms=[0x4+0x61eb0+0x4+0x7fe79]
...
Verified /packages/sets/active/boot/os-kernel/contents.izo
/packages/sets/active/boot/os-kernel/contents.izo size=0x7a0200
```

Verify APIs

- loader `sys/boot/common/verify.c`:

```
int verify_file(int fd, const char *filename, off_t off, int severity);
static int find_manifest(const char *name);
int load_manifest(const char *name, const char *prefix,
        const char *skip, struct stat *stp);
```

- `libve`:

```
unsigned char *verify_asc(const char *sigfile, int flags);
unsigned char *verify_sig(const char *sigfile, int flags);
void fingerprint_info_add(const char *filename, const char *prefix,
        const char *skip, const char *data, struct stat *stp);
int verify_fd(int fd, const char *path, off_t off, struct stat *stp);
```

Verifying a file - `is_verified`

- loader tracks status of each file it has checked
  - simple linked list - most recent first
  - keyed by `dev,ino` of file as reported by `fstat`
    - had to add support for `st_dev` and `st_ino` to `ufs_stat`
      - `st_ino` is simple
      - `st_dev` is trickier I ended up cramming `fs_id` (64bit) into `st_dev` (32bit)

Verifying a file - `find_manifest`

- to verify `/packages/sets/active/boot/os-kernel/kernel`
- `verify_file` calls `find_manifest`; looks for `manifest.esig` and `../manifest.esig` relative to file to be verified
- will find `/packages/db/os-kernel-$version/boot/../manifest.esig`
- if `manifest` not already in data store
  - attempt to verify using corresponding `.ecerts`
  - if successful add `manifest` to data store
  - `fi_prefix = "/packages/sets/active/boot/os-kernel"`
  - regardless; result of signature verification is recorded
- if `manifest` is not verified, nothing in it can be

Verifying a signature

- `verify_sig` uses `manifest.*certs` for `manifest.*sig`
  - returns content of `manifest` if verified.
- BearSSL does not allow ignoring certificate validity period
- `loader` cannot trust `time` to be accurate anyway
  - use `st_mtime` of files to update time used for verification.
    - added `st_mtime` to `ufs_stat`

Verifying an OpenPGP signature

- X.509 certificates are great for vendors like Juniper or FreeBSD.org
- OpenPGP is simpler for self signing
- `verify_asc` uses `manifest.asc` and embedded public key(s)
  - returns content of `manifest` if verified

Verifying a file - `verify_fd`

- `verify_file` calls `verify_fd`
  - try to lookup `kernel` in fingerprint data store
  - in Junos we actually want to look for `boot/kernel`
  - hence; `fi_skip = "boot"`
- if found, we have `sha1=deadbeef....cafebabe`
  - tells us the desired value and the method to be used
  - hash file and compare, if they match; file is verified
- record and return status; success or reason for failure

Verify failure

- verification can fail for multiple reasons
  - `VE_FINGERPRINT_WRONG` hash does not match `manifest`; always results in failure
  - `VE_FINGERPRINT_NONE` no matching `manifest` entry found
    - may result in failure depending on `severity` and threshold setting
  - `VE_FINGERPRINT_UNKNOWN` matching `manifest` entry found but no (recognized) hash.
    - may result in failure depending on `manifest` and threshold setting

Verify file - `severity`

- `severity` arg to `verify_file` indicates importance of verification:

```
#define VE_GUESS        -1          /* let verify_file work it out */
#define VE_TRY          0           /* we don't mind if unverified */
#define VE_WANT         1           /* we want this verified */
#define VE_MUST         2           /* this must be verified */
```

- `VE_MUST` used for kernel, modules etc
- `VE_GUESS` used by most callers
  - `VE_TRY` used for `*.conf`, `*.hints` etc.
  - `VE_WANT` used for rest
- if verification status not `VE_FINGERPRINT_WRONG` and `severity` less than accept threshold, return success.

Controlling loader settings

- for FIPS mode we want strict enforcement
  - only accept `VE_FINGERPRINT_NONE` for `VE_TRY`
- for debugging/experimenting we might want very lax enforcement
- default is in between
  - accept `VE_FINGERPRINT_NONE` up to `VE_WANT`
- how to configure without compromising security?

Tweak packages: `loader-ve-*`

- since this implementation is strictly pathname based we can leverage verified pathnames to communicate to `loader`
- `loader-ve-strict` set *strict* enforcement
  - contains `init.4th` that attempts to load file `loader.ve.strict`
  - `loader` can spot the pattern `loader.ve.*` and interpret the extension
    - set accept threshold to `VE_WANT`
    - check result of self-tests; if they failed `panic`
- `loader-ve-off` turn verification *off*
  - some folk think they are safe in their data center

Performance

- `loader` does not read modules in a manner conducive to hashing
  - `verify_fd` has to read whole file, then rewind to original offset this does not matter for small files, but hurts for `kernel` etc.
  - overhead is about 3% for Junos booting from Compact Flash.

Optimized API for modules

- `libve` provides an API to reduce hashing overhead:

```
struct vectx* vectx_open(int, const char *, off_t, struct stat *, int *);
ssize_t vectx_read(struct vectx *, void *, size_t);
off_t vectx_lseek(struct vectx *, off_t, int);
int vectx_close(struct vectx *);
```

  - can hash file as side-effect of reading
  - requires *extensive* re-work of `loader` (eg `load_elf.c`)
  - verification happens at close
    - only use for modules
    - `panic` on failure ?

Loader is OS version agnostic

- as a standalone application, `loader` does not care about OS version
- `loader` from `stable/11` can boot `stable/6`
- since `loader` needs to be signed specially for secure-boot using same binary for many releases can help.

Q&A

- Questions

Author: sjg@juniper.net
Revision: $Id: loader-veriexec-slides.txt,v 0d045147724a 2018-06-01 20:10:40Z sjg $