Integrity-checked block devices with device mapper

Mandeep Baines
Will Drewry

Huh?

We'll talk about our efficient device mapper target that provides read-only access to blocks from another block device that are cryptographically hashed and checked against a known good "manifest" prior to use.

Why?

- General examples:
  - /boot or / is unmodified since last boot
    - Boot off a usb key
    - Xen hypervisor root and Intel's tboot
  - Downloaded fs image is right (gpg sig over a hash)
  - USB stick is unmodified
- Chrome OS case:
  - all mountpoints are W^X
  - Root partition needs to be integrity assured
  - kernel & its parameters are signed (verified boot flow)
  - <2 second boot budget with Atom processors
  - minimize untrusted data parsing during boot

Existing options (circa 2009)

- linux-ima
  - was per-file, full code hashing prior to exec
  - tpm-rooted guarantees were slow
- fuse implementation
  - same idea, but slow(er)
- custom initramfs/initrd for checking the root filesystem
  - (assuming a signed, checked kernel)
  - slow boot. at least linear with check code/config size
- ditto, but checking only specific files
  - filesystem parsing attacks and complexity were concerns

Introducing dm-verity

- We needed something
  - ... fast
  - ... we could configure from the kernel commandline*
  - ... we could boot directly to
  - ... with cryptographic assurances
  - ... with minimal attack surface
- Enter device-mapper
  - Can intercept every block request to our root partition
  - Benefits from caching layers
  - Minimizes copying
- Now, how can we speed it up?

Hash trees (or are they tries ...)

- Hash trees (Merkle trees) are well-known.
- Provide a tree of hashes where the leaf nodes are the real data.
- Minimal data is needed to verify data with a hash tree:
  - the "root node" hash
  - how to configure the tree:
    - data source, block size, tree depth, ...
- Configuring a device mapper target only really needs a
  - data source
- Fiddly but gets us a 1.6 second boot on an Atom processor, SSD and ~700 megabyte root filesystem.

- Enter the hash tries.

Hash tries

- Hash tries, or prefix (hash) tree, or ...
- Provide a standard mechanisms for indexing and organizing the tree
- Tree depth and number of nodes per leaf stop being options
- Use a "block id" to determine the path from root to leaf
- Allows short-circuiting neighboring verifications

- Now we're at ~1.2s boot time on the same hardware but with ~800 megabyte root filesystem. Yeah!

Other architectural points

- Lock-free
  - Tree nodes use an atomic_t enum for state that can only progress: UNALLOCATED ... VERIFIED
  - Allows for parallelizing the workload
- Parallel processing
  - one workqueue per cpu
  - one crypto context per cpu
- Error (hash mismatch) behavior is configurable:
  - only return EIO, panic, or ...
  - use a registered handler via the *notifier subsystem*
- Salting
  - A long-lived block may keep the same hash for a while
  - Add a configurable salt and rotate on-demand

Where does that leave us?

- 1.2s boot with a 891 megabyte root filesystem
- Assuming no I/O bottlenecks, the same system would take 9 seconds to pre-verify the whole filesystem with SHA1
- Platform and "user" configurable error handling
- "dd" bandwidth of 25.5 MB/s (on a Samsung Chromebook)
    - Cached ~ 377 MB/s

- We could still do better.

Moar speed

- Copying SHA1 from git into the kernel:
  - Saves another 300 milliseconds
- Any suggestions?

Example dm table

Setup /dev/sda3 with an integrity-checking overlay with the pregenerated hash tree appended after the verified data:

"0 1740800 verity payload=/dev/sda3 hashtree=/dev/sda3 hashstart=1740800 alg=sha1 root_hexdigest=716277..."

Boot-time integration

- Chrome OS
  - Speed matters to us
  - Added do_mount_dm.c (no initramfs+klibc)
    - (sent but not pulled; will resend)
    - Just like md="" we added dm=""
  - The root node hash ends up in dm=""
  - Chrome OS firmware/bootloader checks the signature over our kernel and kernel parameters.
- Chromium OS
  - Intel's tboot could be used to get a verified root block device in the same fashion
  - Or just a USB stick you carry in your pocket :)

Questions? Comments?