Android Bootloader and Verified Boot

Lecture 7

Security of Mobile Devices

2019

Bootloader

Recovery

Verified Boot

Bibliography

- Software that runs when device is powered up
- Proprietary and specific to the SoC
- Initialize hardware
- Find and start the OS
- Separate bootloader for each booting stage

- Supported by most bootloaders
- Special hardware key combination while booting
- `adb reboot bootloader`
- Flashing raw partition images
- Booting transient system images

- Default on customer devices
- Cannot flash or boot images
- Flash only images signed by device manufacturer
- Unlocking bootloader:
  - Removes fastboot restrictions
  - Removes signature check
  - Format userdata partition

- Minimal Linux-based OS
- RAM disk with low-level tools
- Minimal UI
- Stored on the recovery partition
- Apply updates - OTA packages
  - Patch of the system files and updater script
  - Code-signed using device manufacturer's private key
  - Recovery includes public key and verifies OTA
  - OTA from trusted source

- Flashed in fastboot/download mode
- No OTA signature verification
- Modify main OS
- Root access through ADB
- Obtain raw partition data

Custom Recovery - Attack

- Encrypted data partition:
  - Install rootkit on system partition
  - Access to decrypted user data when in main OS
  - Remote access
- Verified boot
  - Verify boot partition with key stored in hardware
  - Can prevent rootkit attack
  - Limit damage done by malicious system partition

- Linux kernel framework
- Generic way to implement virtual block devices
  - Linux's Logical Volume Manager
  - Full disk encryption
  - RAID arrays
  - Distributed replicated storage
- Mapping a virtual block device to one or more physical ones
- May modify the data in transfer (dm-crypt)

- Android verified boot based on dm-verity
  - Device-mapper block integrity checking target
- Verifies the integrity of each device block when read
  - Success -> Block is read
  - Fail -> IO error

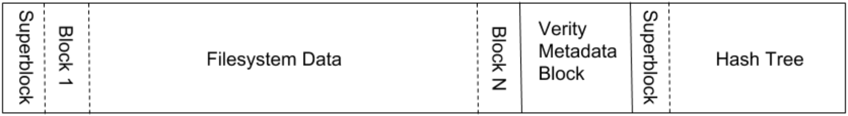
- Uses a Merkle tree:
  - Hashes (SHA256) of all device blocks (4k)
  - Leaf nodes - hashes of physical device blocks
  - Intemediate nodes - hashes of child nodes
  - Root node - based on all hashes of lower levels
- A change in a single device block -> change root hash
- To verify all device blocks -> verify root hash

- When a block is read:
  - Verify hash by traversing the precalcuated hash tree
  - After that, the block is cached
  - Subsequent reads to the block - no verification
- Device needs to be mounted read-only
- Mounting read-write -> integrity check fail

- Recommended for partitions with system files
  - Modified only by OS update
  - Integrity check failure -> OS or disk corruption
  - Malware modified a system file
- Well integrated with Android
  - Only the user partition is mounted read-write
  - OS files on system partition

- From Android 4.4
- Implemented differently from one from the Linux kernel
- RSA public key
  - On boot partition - verity_key
  - Verify dm-verity mapping table
    - Location of target device
    - Offset of the hash table
    - Root hash
    - Salt

- Verity metadata block:
  - On disk after last filesystem block
  - Includes mapping table and signature
- Verifiable partition:
  - `verify` flag in fstab file



- Filesystem manager encounters `verify` flag
- Loads verity metadata from device
- Verifies mapping table signature with verity key
- Success -> Parses dm-verity mapping table
- Passes table to Linux device-mapper
- Creates virtual dm-verity block device

- Virtual block device mounted instead of physical device
- All block reads are verified using the hash tree
- Integrity verification and I/O error:
  - When modifying a file
  - When adding a file
  - When remounting partition as read-write

Trusted boot partition

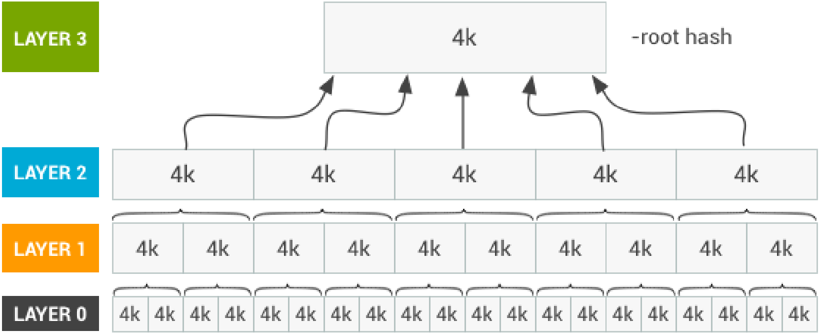
- Boot partition: kernel (dm-verity), RAM disk, verity key
- Needs to be trusted
- Verification is device-specific
- Implemented in the bootloader
- Using signature verification key stored in hardware

Enabling Verified Boot - Steps

1. Generate hash tree
2. Create dm-verity mapping table
3. Sign the table
4. Generate and write verity metadata block on device

- Using `veritysetup`
  - Included in cryptsetup
  - Cryptographic volume management tools package
  - Works directy with block devices or system images
  - Writes hash table in a file
- Hash tree stored on the same target device
- Offset - location after the verity metadata block
- Specify offset when running `veritysetup`

- Steps:
  1. Choose random salt
  2. Divide system image in blocks (4k)
  3. For each block, generate salted SHA256 hash
  4. Concatenate all hashes to form a level
  5. Pad this level with 0s (4k boundary)
  6. Concatenate level to the hash tree
  7. Repeat steps 2-6 based on the previous level until obtaining a single hash



Source: http://source.android.com

Create Mapping Table

- Root hash used to create mapping table
- Table includes:
  - dm-verity version
  - Undelying data and hash device
  - Data and hash block sizes
  - Data and hash disk offsets
  - Hash algorithm
  - Root hash
  - Salt

- Using 2048 bit RSA key
  - In mincrypt format
  - Serialization of RSAPublickey structure
  - In the boot partition - `/verity_key` file
- PKCS#1 v1.5 signature
- Table + signature -> 32k Verity Metadata Block

- Magic number - sanity check
- Version - can be extended
- Table signature
- Table length
- Table
- Padding - to 32k

Modify fstab File

- To enable integrity verification
- Add `verify` flag for system partition

```
marlin:/ $ cat /vendor/etc/fstab.marlin
# Android fstab file.
#<src> <mnt_point> <type> <mnt_flags and options> <fs_mgr_flags>
/dev/block/platform/soc/624000.ufshc/by-name/system /system ext4 ro,barrier=1
wait,slotselect,verify
```

- When booting, virtual dm-verity device is created
- Mounted at /system instead of the physical device

- Any modification to the system partiton
- Any OTA without verity metadata update
- Compatible OTA -> Update hash tree and metadata

- Verifies software device integrity: bootloader, boot partition, system partition
- Each booting stage verifies integrity and autenticity of next stage before execution
- Boot states: GREEN, YELLOW, ORANGE, RED
- Device state: LOCKED, UNLOCKED

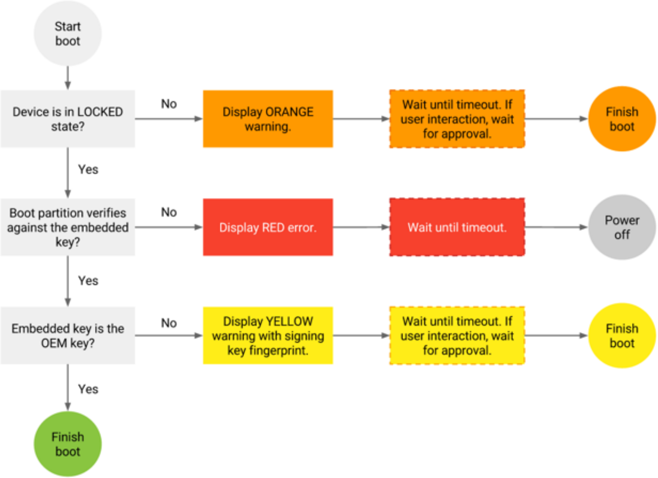
Verification Keys

- Bootloader verified by hardware root of trust
- Verify boot and recovery partitions - OEM key
  - May be protected using a Hardware Security Module
- LOCKED device
  1. OEM key
  2. Certificate embedded in partition signature
- UNLOCKED device
  - Image may be signed with other keys

- GREEN
  - Full chain of trust
  - Bootloader, boot partition and all verified partitions
- YELLOW
  - Boot partition verified using embedded certificate
  - Display warning and public key fingerprint
  - Continue booting
- ORANGE
  - Device integrity is not verified
  - Display warning, continue booting
- RED
  - Failed device verification
  - Warning and stop boot process

- LOCKED
  - Device cannot be flashed
  - May boot into GREEN, YELLOW, or RED states
- UNLOCKED
  - Device can be flashed freely
  - Device not verified
  - Always boots into ORANGE state

Verified Boot Flow



Source: http://source.android.com

Bootloader

Recovery

Verified Boot

Bibliography

Bibliography

- Android Security Internals, Nicolay Elenkov, 2015
- Android Hacker's Handbook, Joshua J. Drake, 2014
- `https://source.android.com/security/verifiedboot/`

Keywords

- Bootloader
- Fastboot mode
- Locked bootloader
- Signed images
- Recovery OS
- OTA packages
- Custom recovery
- Device mapper
- Verified boot
- dm-verity
- Hash tree
- Mapping table