HKG18-124 Android Verified Boot 2.0 and U-boot

Igor Opaniuk, Texas Instruments
Agenda

● Android Verified Boot 2.0 highlights
● Current status of AVB 2.0 integration in U-boot
● Tamper-evident storage and TEE
● What is next?
● Appendix
Android Verified Boot 2.0: highlights

- Verified Boot establishes a chain of trust from the bootloader to system image
- Integrity checking:
  - Boot: Linux kernel + ramdisk. RAW hashing of the whole partition and comparing with hash stored in VBMeta image
  - System/Vendor: verifying only root hash of **dm-verity** hashtrees
- Provides capabilities for rollback protection
- Out of scope: integrity of the bootloader (U-boot in our case)
AVB 2.0 integration into U-boot

- VBMeta image
- AVB 2.0 libs: libavb/libavb_ab
- Implementation of AvbOps
- Subset of avb commands in U-boot shell
- Invocation of verification
- Example of tampering partitions
- Enable on your device
VBMeta image

- Contains crypto descriptors (hashes, hash metadata, public keys) for verification of other partitions
- VBMeta structure is versioned
- Max image size - 64K
- Dedicated partition is used for this image:
  - Also could be integrated into **boot** partition

<table>
<thead>
<tr>
<th>Header data - fixed size (256 bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authentication data - variable size</td>
</tr>
<tr>
<td>Hash and signature for vbmeta authentication</td>
</tr>
<tr>
<td>Auxiliary data - variable size:</td>
</tr>
<tr>
<td>the public key used to create the signature and descriptors.</td>
</tr>
</tbody>
</table>
libavb/libavb_ab

- Provided under MIT and BSD-3 clause license.
- Requires C99-compliant C compiler.
- Equivalent types available in a normal C runtime should be defined. At least things like `uint8_t`, `uint64_t`, and `bool` (with `false`, `true` keywords) must be present.
- Proper implementation of these runtime functions must be implemented: `avb_memcmp()`, `avb_memset()`, `avb_printv()`, `avb_abort()`, `avb_malloc()`, `avb_free()`, `avb_strlen()`.
Implementation of AvbOps

.read_from_partition() - Reads N bytes from a partition identified by a name.
.write_to_partition() - Writes N bytes to a partition identified by a name.
.validate_vbmeta_public_key() - Checks if the given public ‘vbmeta’ partition is trusted.
.get_unique_guid_for_partition() - Gets the GUID for a partition identified by a string name.

AVB 2.0 specification recommends to use tamper-evident storage for storing rollback indexes and device state (LOCKED/UNLOCKED), so currently there are only stubs instead of full implementation for these ops (as we plan to use RPMB partitions and TEE, work in progress):

.read_rollback_index() - Gets the rollback index for a given index location.
.write_rollback_index() - Sets the rollback index to a given location.
.read_is_device_unlocked() - Gets where the device is unlocked.
AVB commands in U-boot shell

Provides CLI interface to invoke AVB 2.0 verification of partitions + for different testing purposes:

```
avb init <dev> - initialize avb 2.0 for <dev>
avb verify - run verification process using hash data from vbmeta structure
avb read_rb <num> - read rollback index at location <num>
avb write_rb <num> <rb> - write rollback index <rb> to <num>
avb is_unlocked - returns unlock status of the device
avb get_uuid <partname> - read and print uuid of partition <partname>
avb read_part <partname> <offset> <num> <addr> - read <num> bytes from partition <partname> to buffer <addr>
avb write_part <partname> <offset> <num> <addr> - write <num> bytes to <partname> by <offset> using data from <addr>
```
Kernel cmdline updates

- Substitution of (based UUID received from `get_unique_guid_for_partition()`):
  - $(ANDROID_SYSTEM_PARTUUID)
  - $(ANDROID_BOOT_PARTUUID)
  - $(ANDROID_VBMETA_PARTUUID)

- Additional params should be concatenated to the kernel cmdline:
  - `AVB_VBMETA_IMAGE_FLAGS_HASHTREE_ENABLED` - concatenating all AvbKernelCmdlineDescriptor data
  - `androidboot.vbmeta.device_state`: “locked” or “unlocked”
  - `androidboot.vbmeta.device`
  - `androidboot.vbmeta.{hash_alg, size, digest}`
  - `androidboot.vbmeta.avb_version`
Partitions tampering (example)

- **boot** or **system/vendor** (dm-verity metadata section)
  
  => avb init 1
  => avb verify

  avb_slot_verify.c:175: ERROR: boot: Hash of data does not match digest in descriptor.

  Slot verification result: ERROR_IO

- **vbmeta**:
  
  => avb init 1
  => avb verify

  avb_vbmeta_image.c:206: ERROR: Hash does not match!
  avb_slot_verify.c:388: ERROR: vbmeta: Error verifying vbmeta image: HASH_MISMATCH

  Slot verification result: ERROR_IO
Enable on your device (U-boot env)

```bash
avb_verify=avb init $mmcdev; avb verify;

if run avb_verify; then
    echo AVB verification OK. Continue boot;
    set bootargs $bootargs $avb_bootargs;
else
    echo AVB verification failed;
    exit;
fi;
```
Enable on your device (U-boot env)

emmc_android_boot=
  echo Trying to boot Android from eMMC ...;
  ...
  run avb_verify;
  mmc read ${fdtaddr} ${fdt_start} ${fdt_size};
  mmc read ${loadaddr} ${boot_start} ${boot_size};
  bootm $loadaddr $loadaddr $fdtaddr;
Enable on your device (AOSP build)

To switch on automatic generation of vbmeta partition in AOSP build, add these lines to device configuration mk file:

BOARD_AVB_ENABLE := true
BOARD_AVB_ALGORITHM := SHA512_RSA4096
BOARD_BOOTIMAGE_PARTITION_SIZE := <boot partition size>
Tamper-evident storage and TEE

- Leverage OP-TEE for crypto services (MAC/HMAC)
- OP-TEE portable client library
- Possible limitations
OP-TEE RPMB Secure Storage

● From AVB 2.0 README.md: tamper-evident storage must be used for stored rollback indexes, keys used for verification, device state (whether the device is LOCKED or UNLOCKED), and named persistent values.

● **Suggestion:** Use RPMB partition on eMMC.

● **Problem:** private key, used for HMACs can be disclosed.

● **Solution:** Use RPMB secure storage implementation in OP-TEE
OP-TEE portable client library

- Currently in active development.
- OP-TEE portable client library provides communication capabilities with OP-TEE in bootloader environments. Similar idea as [AOSP Queueless Trusty IPC](https://example.com).
- Designed to be bootloader-independent. The suggested approach to porting is to copy all header and C files into the bootloader and integrate as needed (same idea, as with libavb).
- Following GlobalPlatform TEE Client API.
- A full-value TEE supplicant will be added later.
OP-TEE for crypto services (limitations)

- Obviously, making advantage of TEE makes sense only when U-boot is used as non secure bootloader and there is another bootloader which is secure and loads OP-TEE (i.e. ARM Trusted Firmware).
- On some SoCs OP-TEE booting is “triggered” from Non-Secure U-boot. In this case OP-TEE is loaded by Secure Monitor (which can be used for integrity checking of bootloader itself etc), and U-boot just loads OP-TEE blob to RAM and triggers Secure Monitor via SMC calling convention asking to load TEE.
  - The problem is that TEE “installing” can be done only when U-boot `bootm` command consumes FIT image with custom `tee` section and proper `handler` is called, and AVB verification should be done before `bootm` invocations (“chicken or the egg” causality dilemma).
TEE loading by Non Secure U-boot (TI AM57xx)

1. `bootm {fit_addr}`
   - SPL (Second program loader)
   - U-boot

2. custom handler for tee loadable section in FIT: `secure_tee_install()`

3. boot Linux kernel loaded from FIT image
What is next?

- Storing rollback indexes in RPMB and leverage TEE
  - OP-TEE portable client library to obtain access to OP-TEE services + TEE supplicant
  - OP-TEE Supplicant in U-boot to handle RPC requests from RPMB-based secure storage

- Verification of A/B slots (used in seamless updates)
  - A/B system updates use two sets of partitions referred to as slots (normally slot A and slot B). The system runs from the current slot while the partitions in the unused slot are not accessed by the running system during normal operation. This approach makes updates fault resistant by keeping the unused slot as a fallback

- Integrate AVB debugging capabilities into fastboot (fastboot oem subset)
  - Read/write rollback indexes
  - "Lock/Unlock" bootloader
References

1. AOSP AVB 2.0 README.md
2. OP-TEE RPMB Secure Storage
Thank You

igor.opaniuk@linaro.org

#HKG18

HKG18 keynotes and videos on: connect.linaro.org

For further information: www.linaro.org
Appendix: SPL highlights

- **arch_cpu_init()**
  - initializes some registers, the watchdog, the DMA, etc;
- **board_early_init_f()**
  - initializes M-cores if they exist and the pads of the UART;
- **timer_init()**
  - initializes CPU timers and clock sources;
- **preloader_console_init()**
  - initializes serial port communications and prints the message "U-Boot SPL ...";
- **spl_dram_init()**
  - sets board-specific DRAM configuration
- **memset()** zeros BSS memory;
- **board_init_r()** continues the boot, loading the second stage of the boot-loader.
Appendix: bootm internals
Appendix: AVB 2.0 chained partitions

Chained partitions: $xyz \rightarrow key1\_pub$
(signed by key0)
Appendix: dm-verity hashtrees
## Appendix: A/B support

<table>
<thead>
<tr>
<th>vbmeta_a</th>
<th>boot_a</th>
<th>system_a</th>
<th>xyz_a</th>
<th>vbmeta_b</th>
<th>boot_b</th>
<th>system_b</th>
<th>xyz_b</th>
<th>userdata</th>
<th>--</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hash for boot</td>
<td>payload</td>
<td>payload</td>
<td>hashtree</td>
<td>vbmeta</td>
<td>payload</td>
<td>hashtree</td>
<td>Rollback index: 42</td>
<td>Chained partitions:</td>
<td>(signed by key0)</td>
</tr>
<tr>
<td>Hash for boot</td>
<td>payload</td>
<td>payload</td>
<td>hashtree</td>
<td>vbmeta</td>
<td>payload</td>
<td>hashtree</td>
<td>Rollback index: 43</td>
<td>Chained partitions:</td>
<td>(signed by key0)</td>
</tr>
<tr>
<td>(signed by key0)</td>
<td>(signed by key0)</td>
<td>(signed by key0)</td>
<td>(signed by key0)</td>
<td>(signed by key0)</td>
<td>(signed by key0)</td>
<td>(signed by key0)</td>
<td>(signed by key0)</td>
<td>(signed by key0)</td>
<td>(signed by key0)</td>
</tr>
</tbody>
</table>
Appendix: avbtool

$ avbtool make_vbmeta_image
   [--output OUTPUT]
   [--algorithm ALGORITHM] [--key /path/to/key_used_for_signing_or_pub_key]
   [--public_key_metadata /path/to/pkmd.bin] [--rollback_index NUMBER]
   [--include_descriptors_from_image /path/to/image.bin]
   [--setup_rootfs_from_kernel /path/to/image.bin]
   [--chain_partition part_name:rollback_index_location:/path/to/key1.bin]
   [--signing_helper /path/to/external/signer]
   [--signing_helper_with_files /path/to/external/signer_with_files]
   [--print_required_libavb_version]
   [--append_to_release_string STR]
Appendix: Recommended bootflow

START

Valid OS found? (Accept verification errors and any key)

Is device locked?

Valid OS found? (Accept only embedded verification key)

Is verification key used settable by user?

Warn about custom OS. Dismiss after 10 seconds

Warn about OS not being verified. Dismiss after 10 seconds

Update Stored Rollback Indexes

Boot OS

Cannot boot
Enter repair mode