Becoming the Evil Maid

Breaking Android FDE for Fun and Profit

david@sigma-star.at

2024-11-11

Hi!

David Gstir

- Finds vulns, writes code and manages things @ sigma star gmbh
- Engineering and consulting around Linux, embedded devices and security
- Security audits on broad range of topics
- Trainings

A Curious Request

It started out by a DM I've received:

```
X: Hi! *Y* told me you could maybe help me recover data from
   my Android phone.
me: Sure, do you have a backup?
X: Well yes, but it is not a regular backup. It's a
   low-level disk dump of the eMMC and it is encrypted
me: Uhhhmmm oh... Then I have more questions!
```

A Quest!

Long story short after meeting in person:

- Samsung Galaxy S21 was running Android 11
- Got stuck in boot loop
- Created low-level dump of internal (encrypted) storage
- Flashed stock Android 12 (latest at that time)

Evil Maid Attacks

Similar to an Evil Maid attack:

- Physical access to device
- Goal is to recover data from encrypted storage
- Differences:
  - Storage is outside of device (should not make a difference)
  - We already know the passcode (owner gave it to us)

Pre-Knowledge

- Android FDE is pretty much the same as Linux
- Android 10+ uses file-based encryption, but is thoroughly documented
- I worked plenty on that, so probably not an issue
- To decrypt all we need is a key (famous last words... ;)
- Assume: keys stored as *encrypted blobs* somewhere on disk and can only be decrypted by ARM TrustZone
- User passcode or biometrics have to be involved at some point
- Not much knowledge about how Android uses TrustZone

Time to change this!

A First Attempt

- First idea: restore backup and try to boot
- Fail: attempting to downgrade to stock Android 11 does not work
- Samsung flipped an efuse with Android 12 upgrade that prevents downgrades
- Only done in case of major security vulns
- Reason was paper: *Trust Dies in Darkness: Shedding Light on Samsung's TrustZone Keymaster Design* by Shakevsky et al
- They found AES-GCM IV reuse attack in key blob mechanism of ARM TrustZone
- Would have made my task that much easier
- However: They open sourced their tool `keybuster` with a bunch of details from their reversing effort

New Plan

- Live off the land: use existing Android 12 on device to mount encrypted backup
- Something similar needs to happen during upgrade from 11 -> 12
- We need rooted device to do that - no issue with Android 12
- Big unknowns:
  - Can we still decrypt key blobs from backup after flashing stock Android 12?
  - What did Samsung change that I do not know (and do not get with their OSS code)?

Android Storage Encryption

Since Android 9 there are *3* layers of storage encryption:

1. Metadata encryption
2. Device encrypted (DE) storage
3. Credential encrypted (CE) storage

Metadata Encryption

- Lowest encryption layer and first to unlock during boot
- Called `dm-default-key` - pretty much the same as `dm-crypt` in Linux
- Encrypts storage blocks and sits beneath filesystem
- Key is added to Kernel via device mapper ioctls: `DM_DEV_CREATE`, `DM_TABLE_LOAD`, `DM_DEV_SUSPEND`

Device Encrypted Storage

- Second layer of encryption
- Encrypts part of storage that need to be accessible right after boot (before lock code is provided)
- Uses `fscrypt` which is part of Linux (Google upstreamed it)
- Encrypts individual files of a filesystem based on per-directory policy
- Implemented only by some filesystems (`ext4`, `f2fs`, `ubifs`, ...)
- Key is added to Kernel via `fscrypt` ioctl: `FS_IOC_ADD_ENCRYPTION_KEY`

Credential Encrypted Storage

- Last encryption layer protecting user data (profile)
- Also uses `fscrypt`, so similar to DE storage
- However requires *biometrics or passcode to unlock*
- Will be hardest part as requires (more) interaction with TrustZone

High-Level Mount Logic

Relevant parts of mount flow during boot. Mainly done by `vold` service:

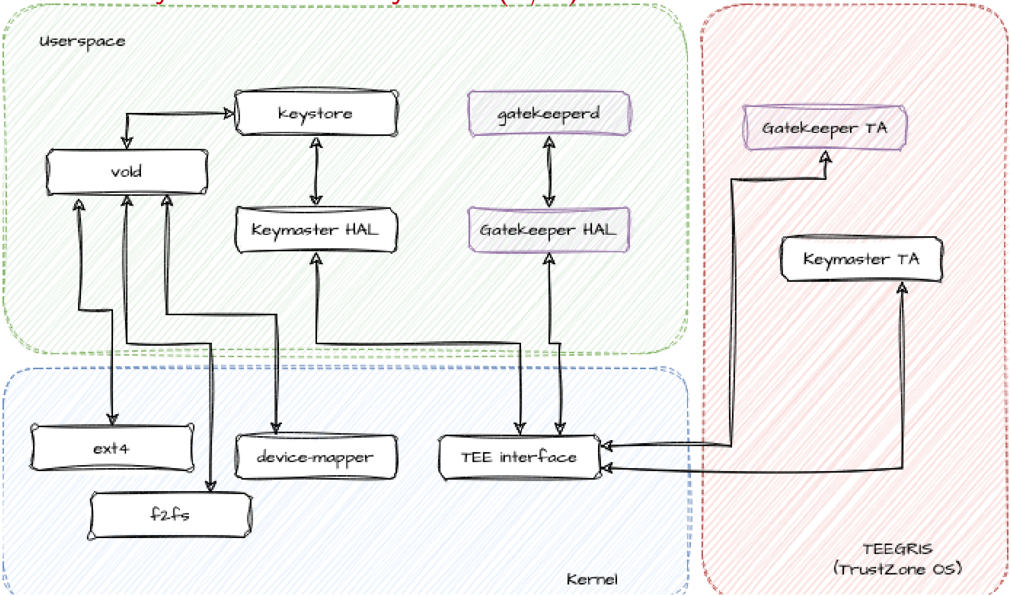
- Mount `/metadata` (is not encrypted)
- Unwrap metadata key
- Attach DM volume `userdata` using `dm-default-key`
- Mount volume as `/data`
- Unwrap DE key and add as `fscrypt` key
- Unwrap CE key and add as `fscrypt` key

Master of Keys: Android Keystore (2/2)

The Android Keystore API manages key storage:

- Keymaster TA (Trusted App) in TEE (Trusted Exec. Env. aka TrustZone) is doing unwrap
- Called via a Kernel interface by *Keymaster HAL*
- Samsung extra: `libkeymaster_helper.so` used by *Keymaster HAL*
- Trick from `keybuster`: bypass Keymaster HAL checks by simply using `libkeymaster_helper.so`

Master of Keys: Android Keystore (2/2)

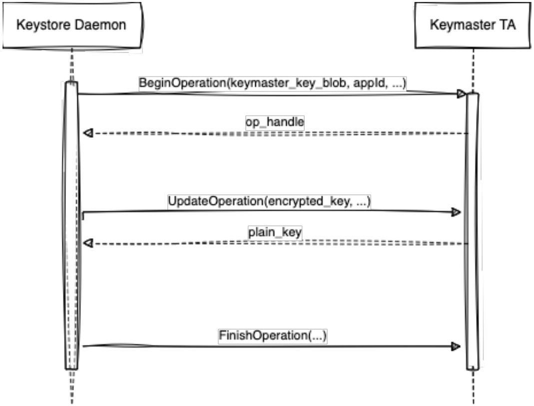


Yo Android! Where're Your Keys At?

For `dm-default-key` searching AOSP source reveals key loaded from files in `/metadata/vold/metadata_encryption/key/`:

- `secdiscardable`: used the generate `AppID` (logic implemented in `vold`)
- `stretching`: contains `nopassword` so we can ignore it
- `encrypted_key`: the key we want to decrypt
- `keymaster_key_blob`: the key used by Keymaster TA to decrypt `encrypted_key`; is encrypted with Keymaster internal key

Unwrap, Please! (1/2)



Unwrap, Please! (2/2)

Reversing some functions from `libkeymaster_helper.so` gives us:

- `nwd_begin(...)`: starts unwrap with key encryption key
- `nwd_update(...)`: performs unwrap with key blob yielding plaintext key
- `nwd_finish(...)`: does cleanup

Full Unwrap Logic

Pseudocode of unwrap logic using `libkeymaster_helper.so`:

```
unwrap_vold_key() {
    secdiscard = read_file("./secdiscardable");
    app_id = generate_appid(secdiscard);
    keyblob = read_file("./encrypted_key");
    kek = read_file("./keymaster_key_blob");
    in_params = generate_in_params(keyblob[:12] /* nonce */);
    dummy = {0};
    nwd_begin(KM_PURPOSE_DECRYPT, kek, in_params, NULL, &dummy, &handle);
    nwd_update(handle, NULL, keyblob[12:], NULL, NULL, &dummy_cnt, &dummy,
               &plain_key);
    nwd_finish(handle, NULL, NULL, NULL, NULL, NULL, &dummy, NULL);
}
```

Key Blob Parameters

`in_params` for Keymaster TA:

- 256-bit AES key
- 128-bit GCM MAC (no padding, 128-bit min MAC length)
- Nonce
- Tag AppID: needs the `AppID` generated from `secdiscardable` file
- Tag `TAG_NO_AUTH_REQUIRED`: no user credentials needed
- Tag `TAG_ROLLBACK_RESISTANCE` (if possible, re-tries without afterwards)

It's Alive!

This allows to configure `dm-default-keys` and attach it.

Minor complications to fix:

- Android 12 added a string prefix to the key blobs, Android 11 does not have this
- Had to fix small bugs in `keybuster` (e.g. wrong constants)
- Had to find proper parameters to `dm-default-key` (used `dmctl table userdata`)

However, most folders contain garbage -> `fscrypt` encrypted

Delving Deeper

Now we can mount `userdata` partition (`/data`) which holds the key blobs for `fscrypt`:

- `/data/misc/vold/user_keys/de/0/`: user DE (device encrypted)
- `/data/misc/vold/user_keys/ce/0`: user CE (credential encrypted)
- `/data/unencrypted/key`: needed to access above folders
- Unwrapping keys in `/data/unencrypted/key` and `/data/misc/vold/user_keys/de/0/` only required minimal changes to unwrap logic

Pause: Where We're At?

- At this point we have access to the whole OS from the backup
- We *never* needed to supply the user passphrase
- We do need the TrustZone as only it can unwrap key blobs
- Flashing Android 12 did not invalidate key blobs from backup

Getting Into CE Storage

- Next challenge is getting CE key unwrapped
- It will now require the passphrase (which we have)
- Derivation logic is much more involved and requires talking to TEE again
- This time an additional TA is involved: Gatekeeper TA

Lucky Again :-D

The brilliant people at Quarkslab started a similar endeavor in parallel to mine:

- Used a different approach of breaking secure boot (patched boot chain and TZ OS)
- Great documentation of their work!

A Bit More To Do...

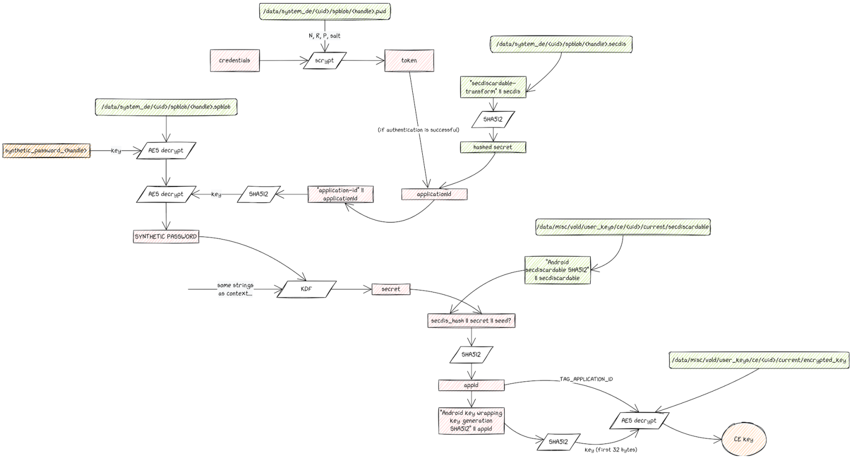


Figure 1: Key unwrap with passphrase and TrustZone, source: Quarkslab

All Done?

- Nope!
- Involves more reverse engineering - yay!
- I'm currently working on Gatekeeper TA integration
- Check sigma-star.at/blog for in-depth blog post soon
- Check back next year ;-)

Summary

- We can decrypt whole OS of a low-level disk dump only with the device
- Resetting device does not stop us yet
- Without the device this would not work though
- When you loose your device, only your passcode will protect you
- How secure is your passcode? ;-)