Breakin' bad

Reviewing Qualcomm ARM64 TZ and hw-enabled Secure Boot on Android (4-9.x)

About me

Bjoern Kerler

- Reverse Engineer and Cryptoanalyst
- I like to break software and hardware :)

- https://github.com/bkerler
- https://twitter.com/viperbjk

Some initial words

1. Not all technical details given in this talk are precise. Some may be wrong. I wrote from what I remember what happened the last two years in my spare time.

2. Some things are very very simplied and are much more complex in reality, but it at least gives you a hint where to start your research (which I mostly didn‘t have).

3. This talk is about devices using AOSP or close to AOSP. It does not feature Samsung‘s or non-AOSP weird modifications.

Topics of this presentation

1. Diving into the past (Android 4.x – 5.x)

   Level: Can do, I know TWRP and Magisk !

2. Attacking Android 6-9 HW Crypto

   (Reversing Aboot and TZ from a physical attacker perspective)

   Level: I like reversing Blackboxes ! Who needs sleep ?!?!
   I am a reversing robot !

3. Who isn't affected/Recommendations

   (QC SDM platform / EDKII and AVBv2)

   Level: Relax. Most probably you are safe if you own a newer device.

Diving into the past

Feb.
2019

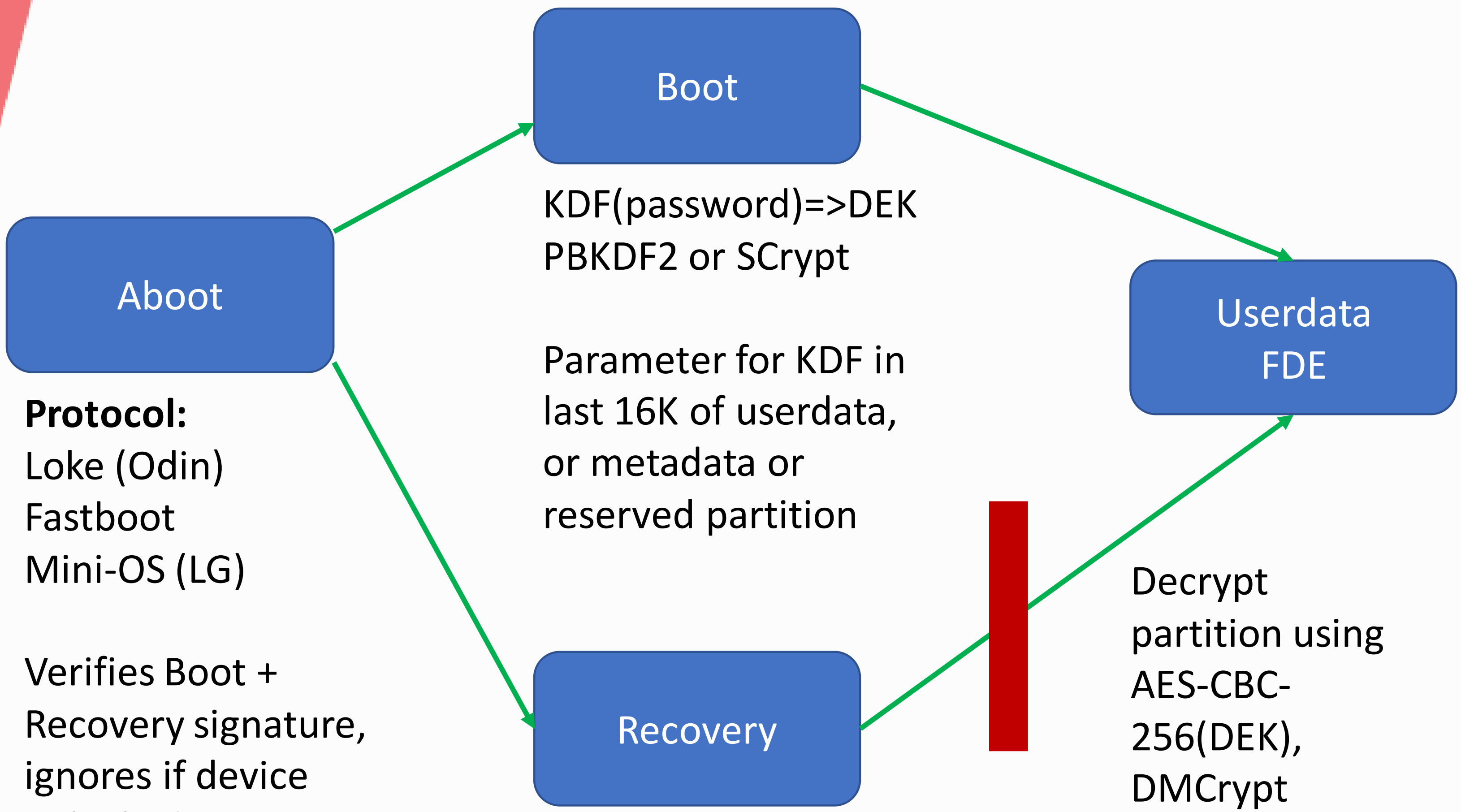
No Con

Android 4.4 - 5

Feb.
2019

No Con

Android < 6, Secure boot



KDF(password)=>DEK
PBKDF2 or SCrypt

Parameter for KDF in last 16K of userdata, or metadata or reserved partition

Protocol:
Loke (Odin)
Fastboot
Mini-OS (LG)

Verifies Boot + Recovery signature, ignores if device unlocked

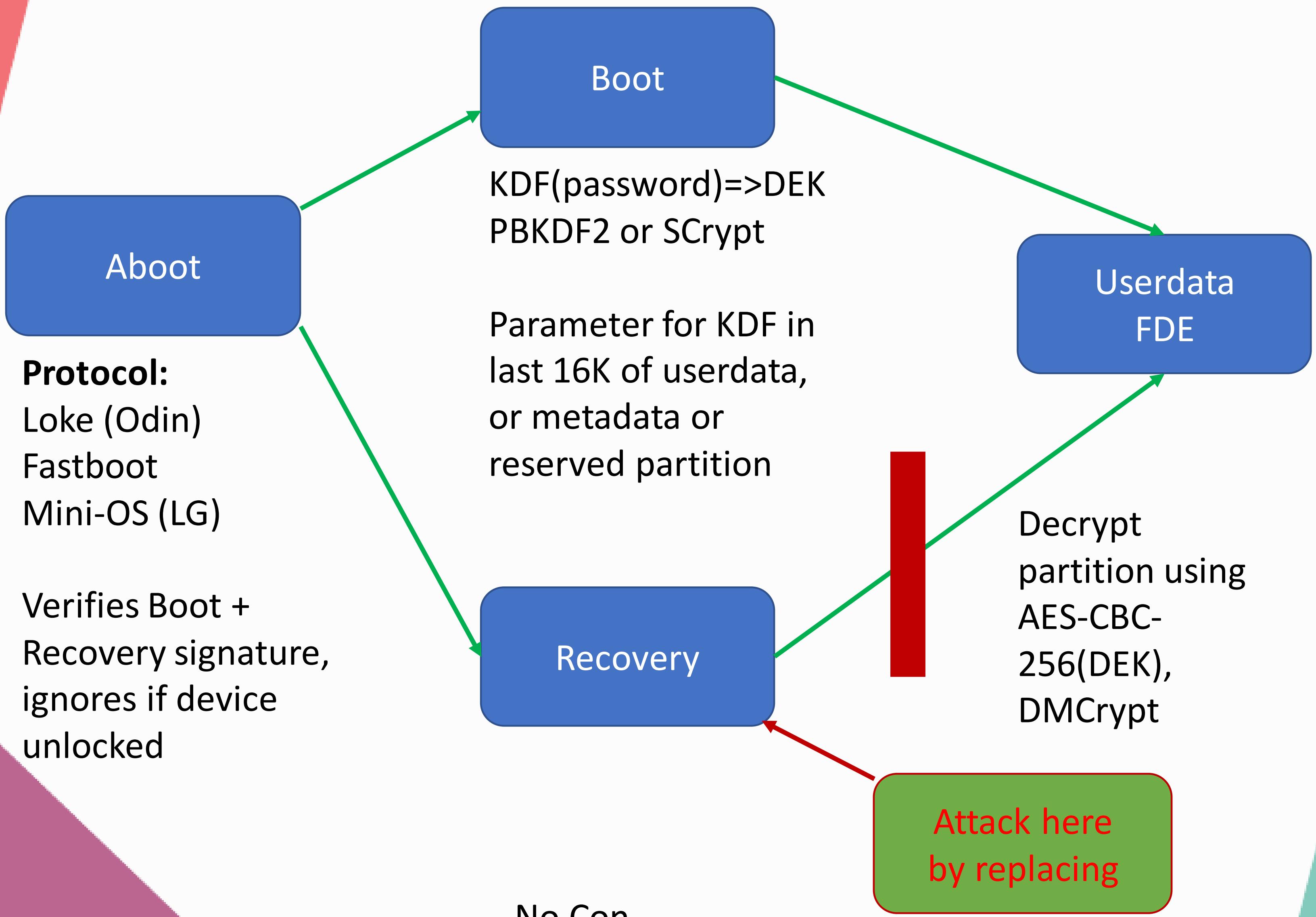
Decrypt partition using AES-CBC-256(DEK), DMCrypt

DEK = Device Encryption Key

Feb. 2019

No Con

Android < 6, Non-hw Secure boot



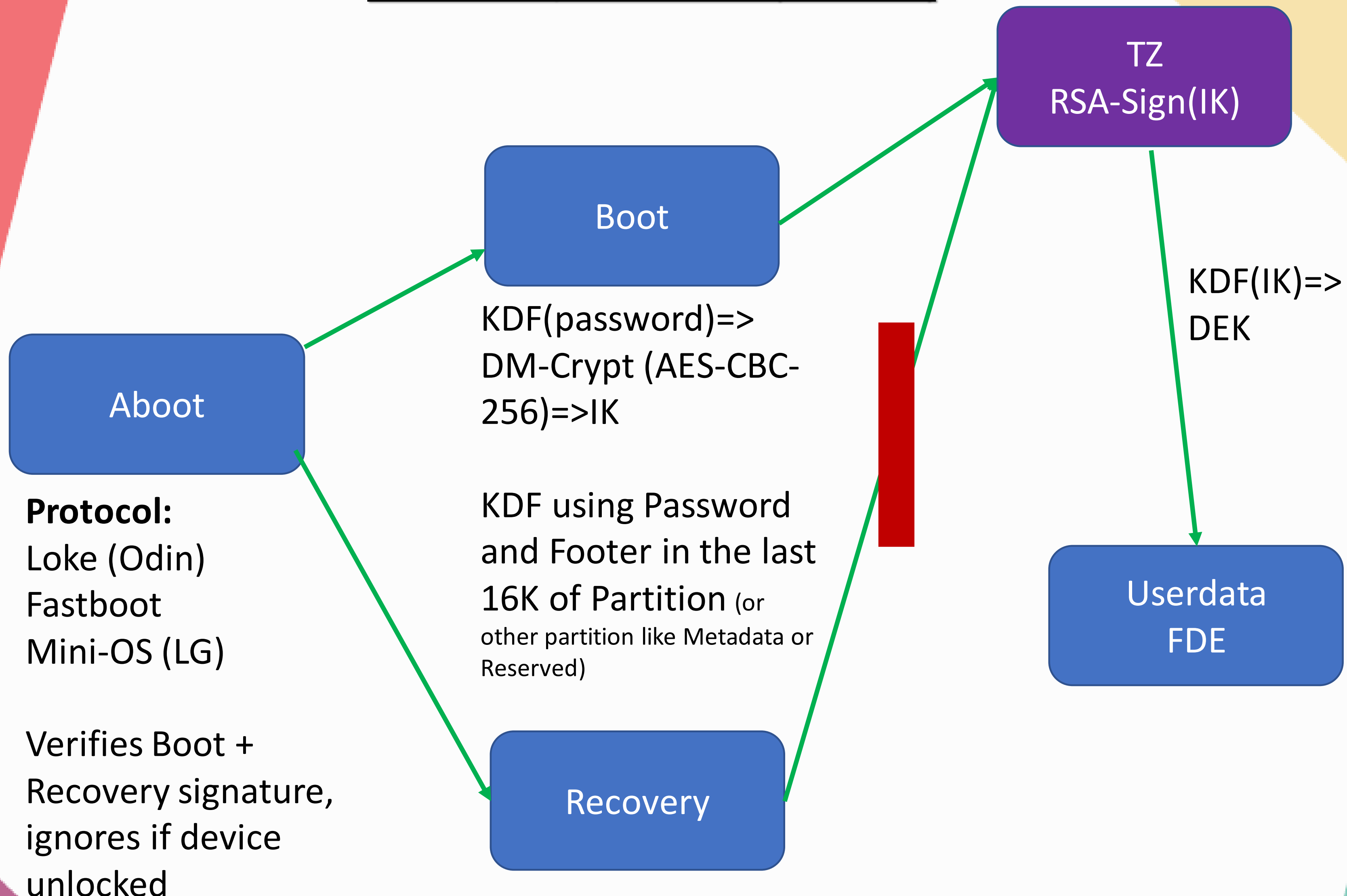
How to attack ?

Implies following :

- Unlock bootloader to flash own recovery
- Ideally, patch existing ramdisk in boot/recovery
- Dump partition, bruteforce password offline

Android 4-6 (non-AVB)

Android 4-6, Secure boot (Non-AVB)

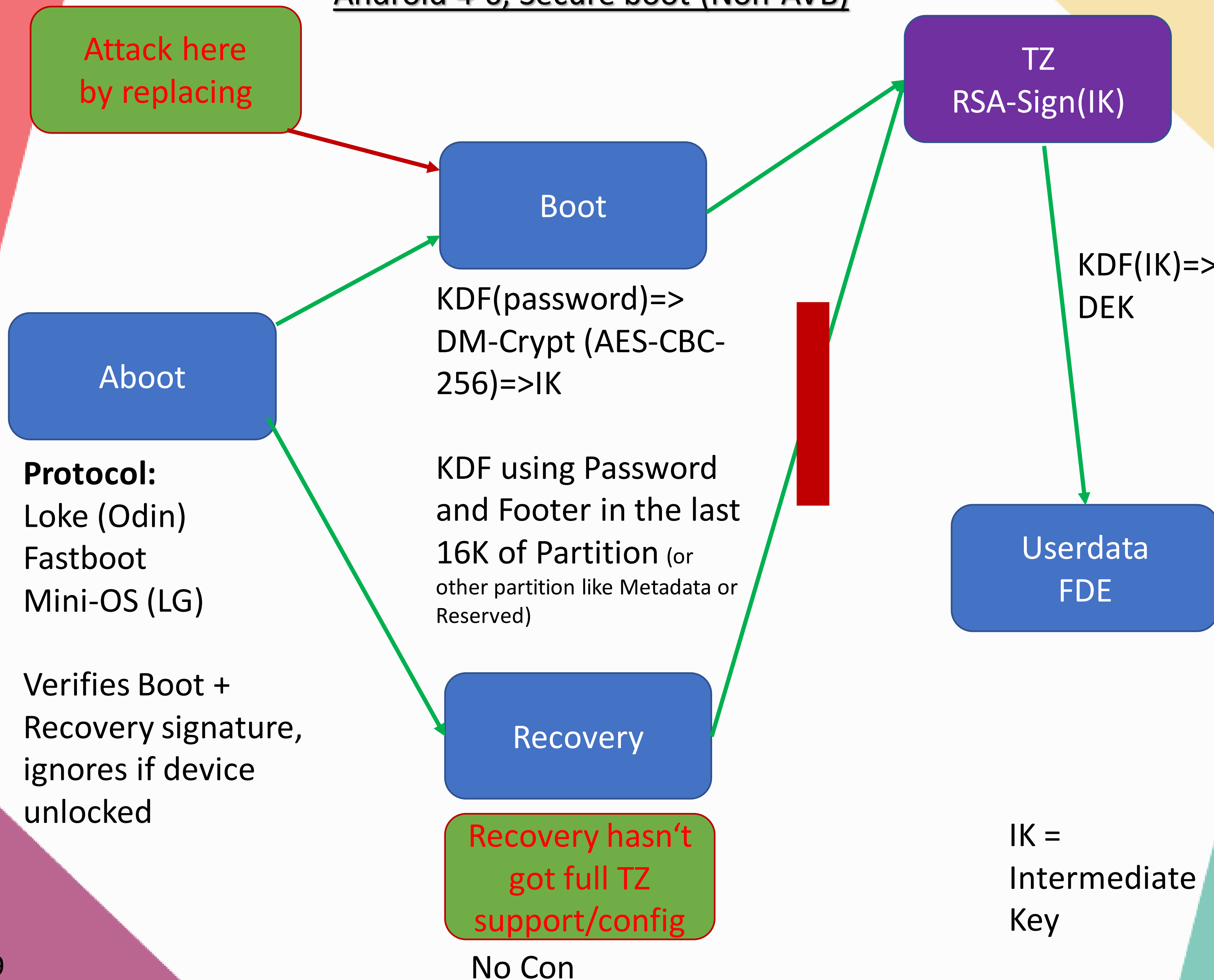


IK = Intermediate Key

Feb. 2019

No Con

Android 4-6, Secure boot (Non-AVB)



Feb. 2019

How to attack ?

Implies following :

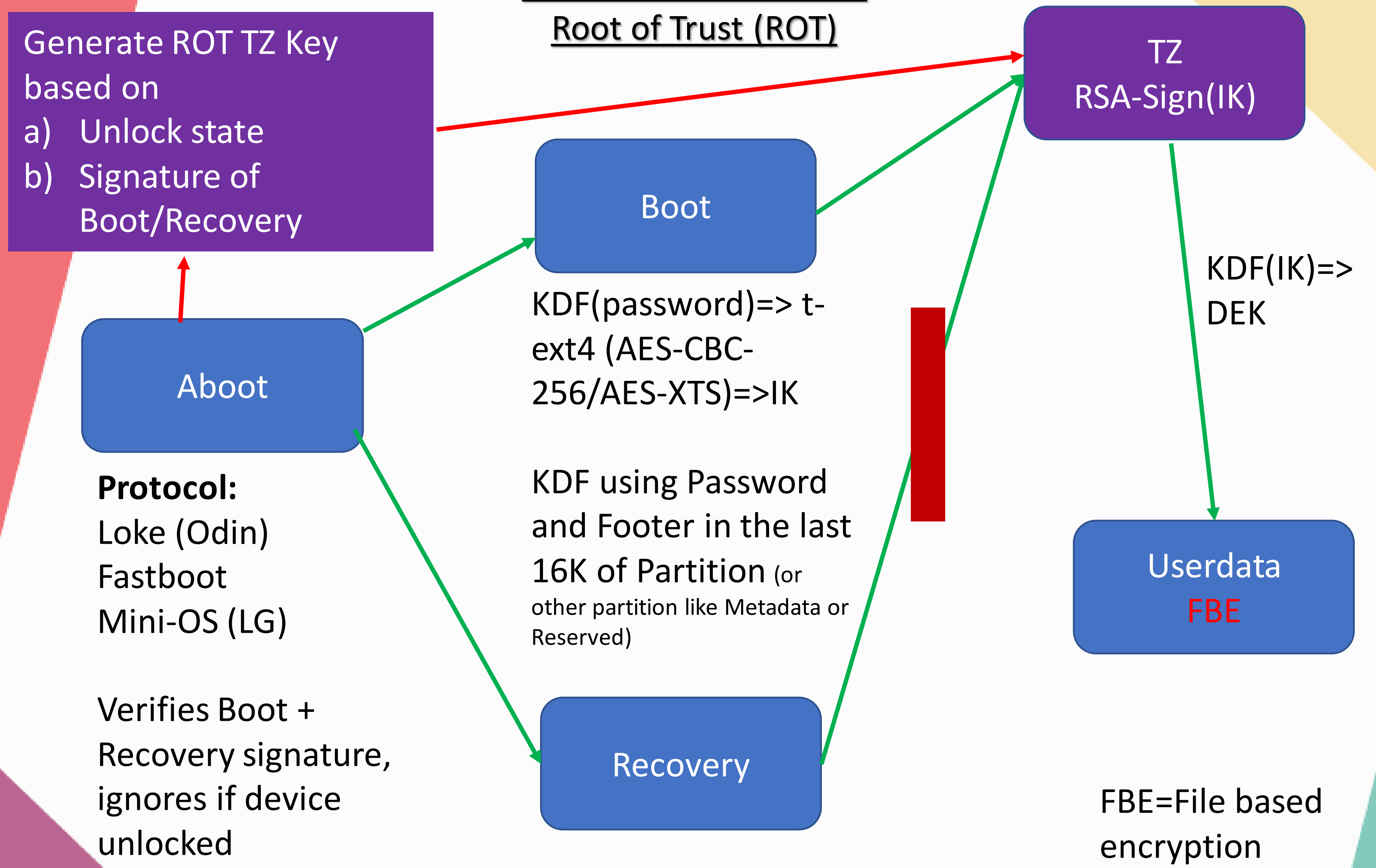
- Unlock bootloader to flash own recovery
- Ideally, patch existing ramdisk in boot/recovery
- Dump partition, bruteforce password online (using the device, as signing cannot work offline, except TZ/KM Keys are known)

Android
7 / 8.x / 9.x
AVB v1

Feb.
2019

As I thought initially by reading official documents

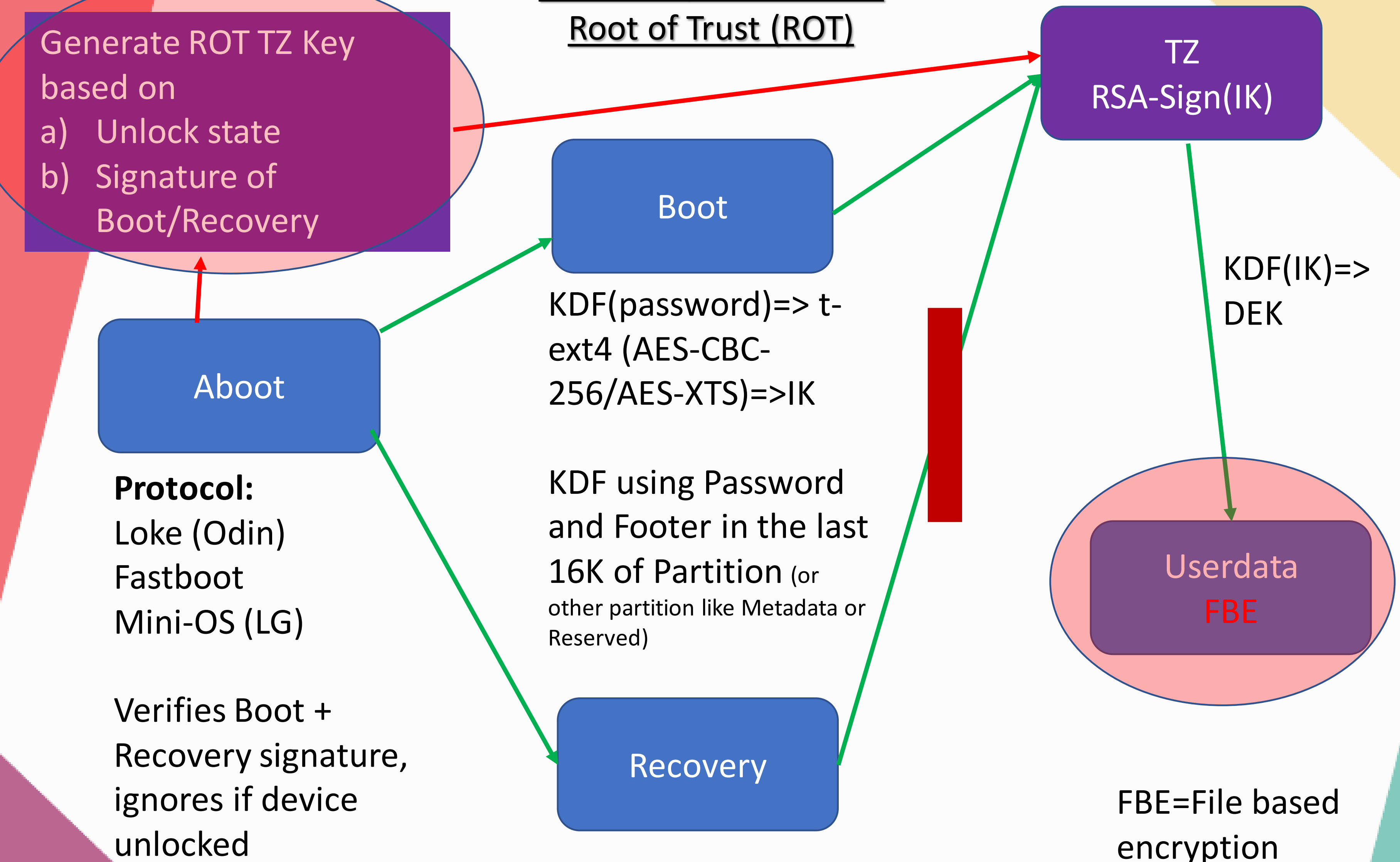
Android 7, Secure boot
Root of Trust (ROT)



Feb. 2019

No Con

Android 7, Secure boot Root of Trust (ROT)



Feb. 2019

No Con

So lets start reversing ….

Helpful for Qualcomm (QC) TZ research

- Sometimes, searching for “QFIL“ using „duckduckgo“, „bing“ or „baidu“, you will find firmware that also contains TZ with debug symbols, often referred as “qsee.elf“.

- However, leaked debug binaries especially for newer QC chipsets have become very rare.

Porting aboot stripped binaries to debug symbols

- Grab aboot source code and recompile : https://source.codeaurora.org/quic/la/kernel/lk (Make sure the branch matches the one in the stock aboot)

- Use a signature generator (sigmake/F.L.I.R.T, etc.) on debug binaries to create function names based on patterns and use IDA / Binary Ninja to remap on stock aboot.

- This won‘t help identify oem modded functions of course, but will help in better understanding the code.

Example targeted aboot code

https://source.codeaurora.org/quic/la/kernel/lk/tree/app/aboot/aboot.c?h=LA.HB.1.1.5.c2-02410-8x96.0

```c
static void verify_signed_bootimg(uint32_t bootimg_addr, uint32_t bootimg_size)
{
	int ret;

#if !VERIFIED_BOOT
#if IMAGE_VERIF_ALGO_SHA1
	uint32_t auth_algo = CRYPTO_AUTH_ALG_SHA1;
#else
	uint32_t auth_algo = CRYPTO_AUTH_ALG_SHA256;
#endif
#endif

	/* Assume device is rooted at this time. */
	device.is_tampered = 1;

	dprintf(INFO, "Authenticating boot image (%d): start\n", bootimg_size);

#if VERIFIED_BOOT
	uint32_t bootstate;
	if(boot_into_recovery &&
		(!partition_multislot_is_supported()))
	{
		ret = boot_verify_image((unsigned char *)bootimg_addr,
				bootimg_size, "/recovery", &bootstate);
	}
	else
	{
		ret = boot_verify_image((unsigned char *)bootimg_addr,
				bootimg_size, "/boot", &bootstate);
	}
	boot_verify_print_state();
#else
	ret = image_verify((unsigned char *)bootimg_addr,
					   (unsigned char *)(bootimg_addr + bootimg_size),
					   bootimg_size,
					   auth_algo);
#endif
	dprintf(INFO, "Authenticating boot image: done return value = %d\n", ret);

	if (ret)
	{
```

Root of Trust technical details

Results :

1. Signature matched and bootloader locked (green,0)
2. Signature makes sense and bootloader unlocked (orange,1)
3. Signature makes sense but bootloader locked (red,3)
4. Signature is wrong (red,3)

SHA256 Hash
(header(page_size)
+ kernel(kernelsize,padded)
+ ramdisk(ramdisksize,padded)
+ other(secondsize,padded)
+ qcdt (qcdtsize,padded)
+ meta(target_mount + length))

Compare Hash with RSA Signature Hash at end of recovery/boot

Recovery or Boot

verifies

Aboot (LK)

Root of Trust technical details

TZ (QSEE) via KM

Results :

1. **Signature matched and bootloader locked (green,0)**
2. Signature makes sense and bootloader unlocked (orange,1)
3. Signature makes sense but bootloader locked (red,3)
4. Signature is wrong (red,3)

Send ROT hash to TZ :

SHA256 (Recovery/boot RSA Public Key + DWORD(0x1))

SHA256 Hash
(header(page_size)
+ kernel(kernelsize,padded)
+ ramdisk(ramdisksize,padded)
+ other(secondsize,padded)
+ qcdt (qcdtsize,padded)
+ meta(target_mount + length))

Compare Hash with RSA Signature Hash at end of recovery/boot

Recovery or Boot

Aboot (LK)

Root of Trust technical details

TZ (QSEE) via KM

Results :

1. Signature matched and bootloader locked (green,0)
2. **Signature makes sense and bootloader unlocked (orange,1)**
3. Signature makes sense but bootloader locked (red,3)
4. Signature is wrong (red,3)

Send ROT hash to TZ :

SHA256 (Recovery/boot RSA Public Key + DWORD(0x0))

SHA256 Hash
(header(page_size)
+ kernel(kernelsize,padded)
+ ramdisk(ramdisksize,padded)
+ other(secondsize,padded)
+ qcdt (qcdtsize,padded)
+ meta(target_mount + length))

Compare Hash with RSA Signature Hash at end of recovery/boot

Recovery or Boot

Aboot (LK)

Root of Trust technical details

TZ (QSEE) via KM

Results :

1. Signature matched and bootloader locked (green,0)
2. Signature makes sense and bootloader unlocked (orange,1)
3. **Signature makes sense but bootloader locked (red,3)**
4. **Signature is wrong (red,3)**

Send ROT hash to TZ :

SHA256 (0x0)

SHA256 Hash
(header(page_size)
+ kernel(kernelsize,padded)
+ ramdisk(ramdisksize,padded)
+ other(secondsize,padded)
+ qcdt (qcdtsize,padded)
+ meta(target_mount + length))

Compare Hash with RSA Signature Hash at end of recovery/boot

Recovery or Boot

Aboot (LK)

What does that mean ?

Implies following changes :

- If boot signature doesn‘t match content
  => ROT-Key different => Cannot decrypt

- If device unlocked (if it wasn‘t before)
  => ROT-Key different => Cannot decrypt

- TZ has timeouts and wipes keys in SSD after 30 tries to prevent online (on-device) bruteforce attack

=> We need an aboot or kernel+tz temp or permanent exploit or signing attack/key

Verifying AVB v1 and v2 on your own:

- https://github.com/bkerler/dump_avb_signature

```
c:\dump_avb_signature>python verify_signature.py --file boot.img

Boot Signature Tool (c) B.Kerler 2017-2018
------------------------------------------------------
Kernel=0x00001000, length=0x00C2D000
Ramdisk=0x00C2E000, length=0x0024F000
Second=0x00E7D000, length=0x00000000
QCDT=0x00E7D000, length=0x00000000
Signature start=0x00E7D000
ID: bc2422e354257957e8e5dfeff85e6c376ea0f936000000000000000000000000

Image-Target: b'/boot'
Image-Length: 0xe7d000
Signature-Length: 0xe7d000
b'0\r\x13\x05/boot\x02\x04\x00\xe7\xd0\x00'

Image-Hash: b'0d1159f185859a72f9e08044882d54feb45111d470dd88690b9f773f9cfa7ff3'
Signature-Hash: b'0d1159f185859a72f9e08044882d54feb45111d470dd88690b9f773f9cfa7ff3'
AVB-Status: VERIFIED, 0

Signature-RSA-Modulus (n): e8eb784d2f4d54917a7bb33bdbe76967e4d1e43361a6f482aa62eb10338ba7660feba0a0428999b3e2b84e43c1fdb58ac67
dba1514bb4750338e9d2b8a1c2b1311adc9e61b1c9d167ea87ecdce0c93173a4bf680a5cbfc575b10f7436f1cddbbccf7ca4f96ebbb9d33f7d6ed66da4370c
ed249eefa2cca6a4ff74f8d5ce6ea17990f3550db40cd11b319c84d5573265ae4c63a483a53ed08d9377b2bccaf50c5a10163cfa4a2ed547f6b00be53ce360
d47dda2cdd29ccf702346c2370938eda62540046797d13723452b9907b2bd10ae7a1d5f8e14d4ba23534f8dd0fb1484a1c8696aa997543a40146586a76e981
e4f937b40beaebaa706a684ce91a96eea49
Signature-RSA-Exponent (e): 010001

TZ Root of trust (locked): b'3bd07d4fc1cea0698c699c8f3a167a80dbd96affc402a6fc924ea46edc2e8381'
TZ Root of trust (unlocked): b'80d72c4aafdd168c9bb44be519124ea518d79c18ac030d99f13343f595ee18a4'
```

Making your own rooted ramdisk with rotfake:

- https://github.com/bkerler/android_universal

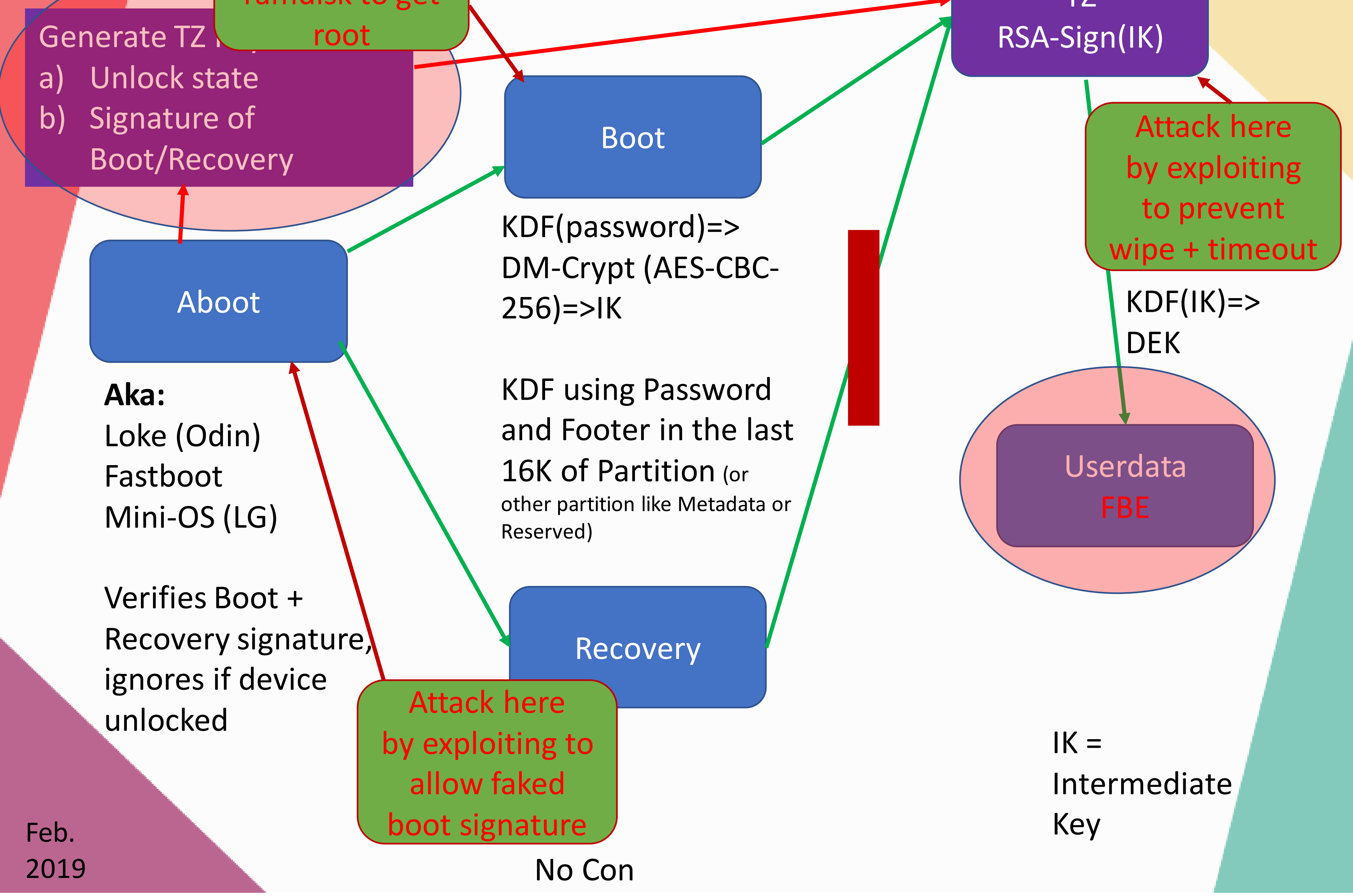
- Patches stock boot image, adds hidden root shell, signs with google test keys and makes faked rot boot image (you will need to patch aboot and proper resign or use aboot exploit before flashing of course)

How to attack ?

Regular approach as on android <=6 (no avb) won‘t work !

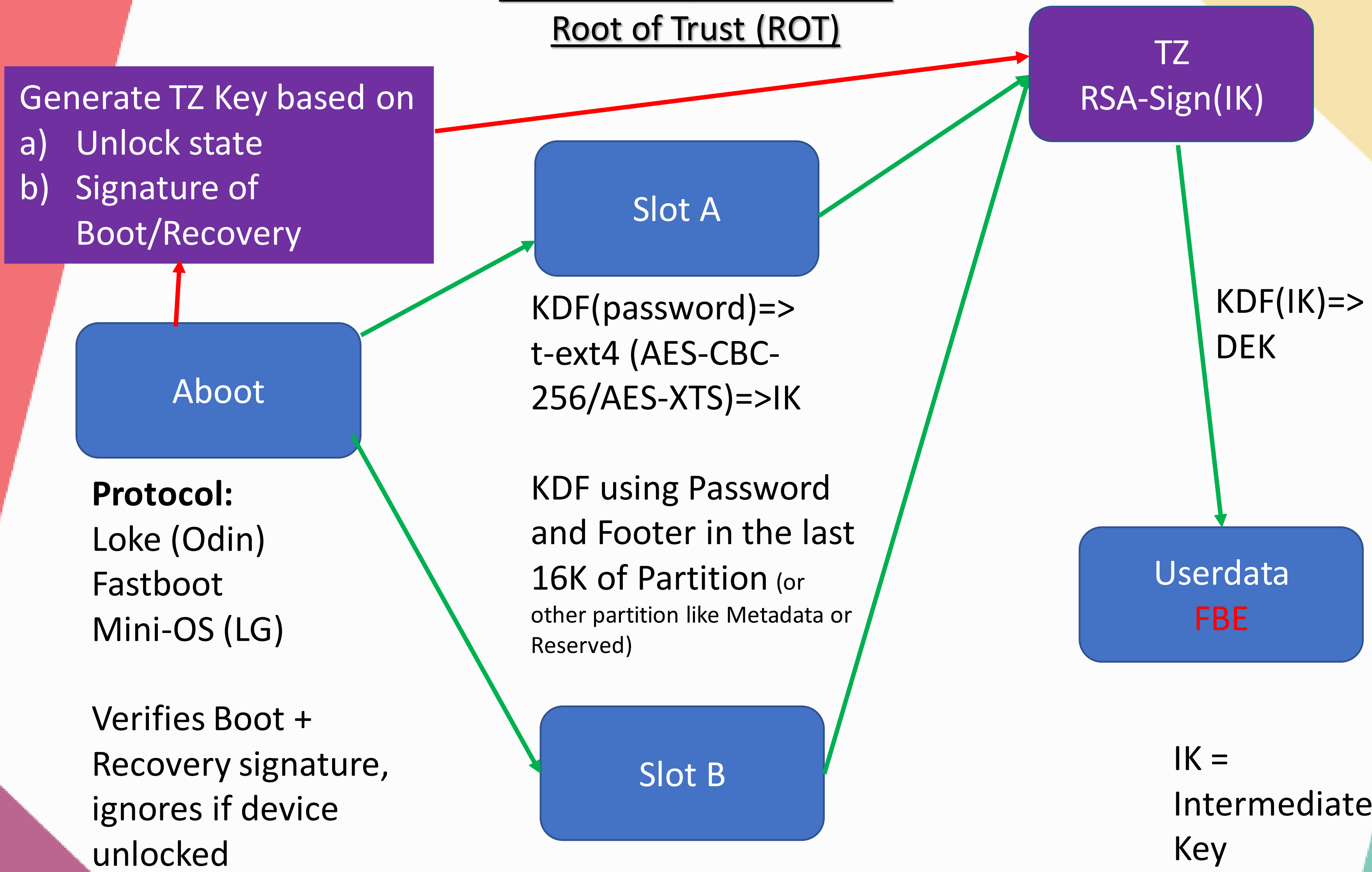
- ~~Unlock bootloader to flash own recovery~~
- ~~Ideally, patch existing ramdisk in boot/recovery~~
- ~~Dump partition, bruteforce password online (using the device, as signing cannot work offline, except TZ Rsa Keys are known)~~

Android 7, Secure boot Root of Trust (ROT)



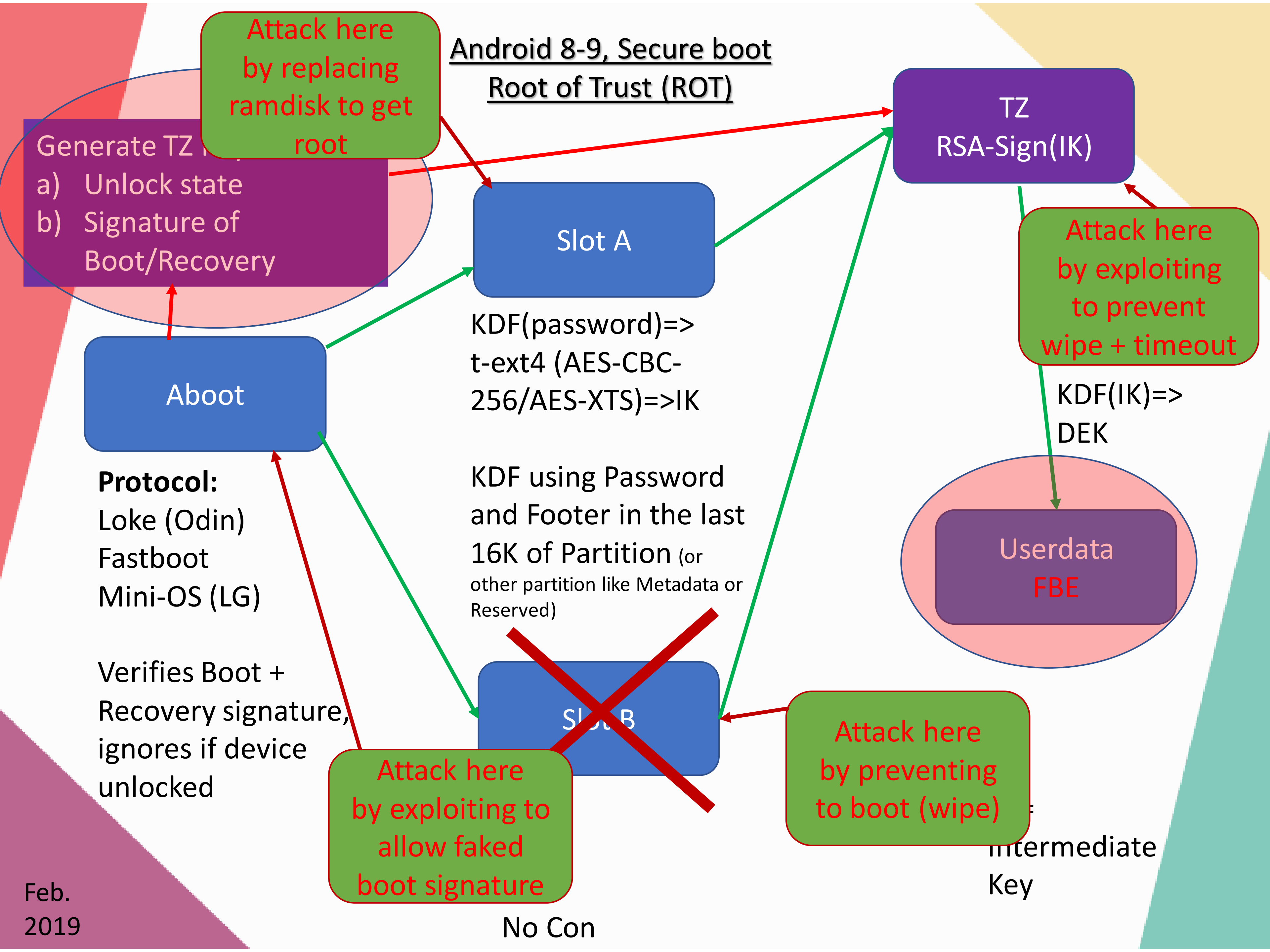
Android One

Android One, Secure boot Root of Trust (ROT)



Feb. 2019

No Con


Android 8-9, Secure boot
Root of Trust (ROT)
Attack here by replacing ramdisk to get root
Generate TZ
a) Unlock state
b) Signature of Boot/Recovery
TZ
RSA-Sign(IK)
Slot A
Attack here by exploiting to prevent wipe + timeout
Aboot
KDF(password)=> t-ext4 (AES-CBC-256/AES-XTS)=>IK
KDF(IK)=> DEK
Protocol:
Loke (Odin)
Fastboot
Mini-OS (LG)
KDF using Password and Footer in the last 16K of Partition (or other partition like Metadata or Reserved)
Userdata
FBE
Verifies Boot + Recovery signature, ignores if device unlocked
Attack here by preventing to boot (wipe)
Attack here by exploiting to allow faked boot signature
Key
Feb. 2019
No Con

Attacking Secure-Boot and Secure-Startup on QC-based devices

Feb.
2019

No Con

Patch kernel to allow further tz research

- Add your own SCM commands to allow tz hotpatching (scm.h)
- Enable devmem/devmemk for flexible reading/patching (kernel config)
- Disable QC patched tz protections in kernel source code

For tz debugging:

- Enable tz debugging: “mount -t debugfs debugfs /d/“ then have a look at “/d/tzdbg/qsee_log“.
- If you do something bad in tz mem space (not allowed or crash), device will reboot.
- If it crashes fast, it‘s tz. If it takes time to crash, it‘s kernel (thanks to Sean Beaupre for this magic hint).
- Some devices have tz uart pinout on gpio.

- See my example kernel patch sets for multiple devices : https://github.com/bkerler/twrp_tz_fixes

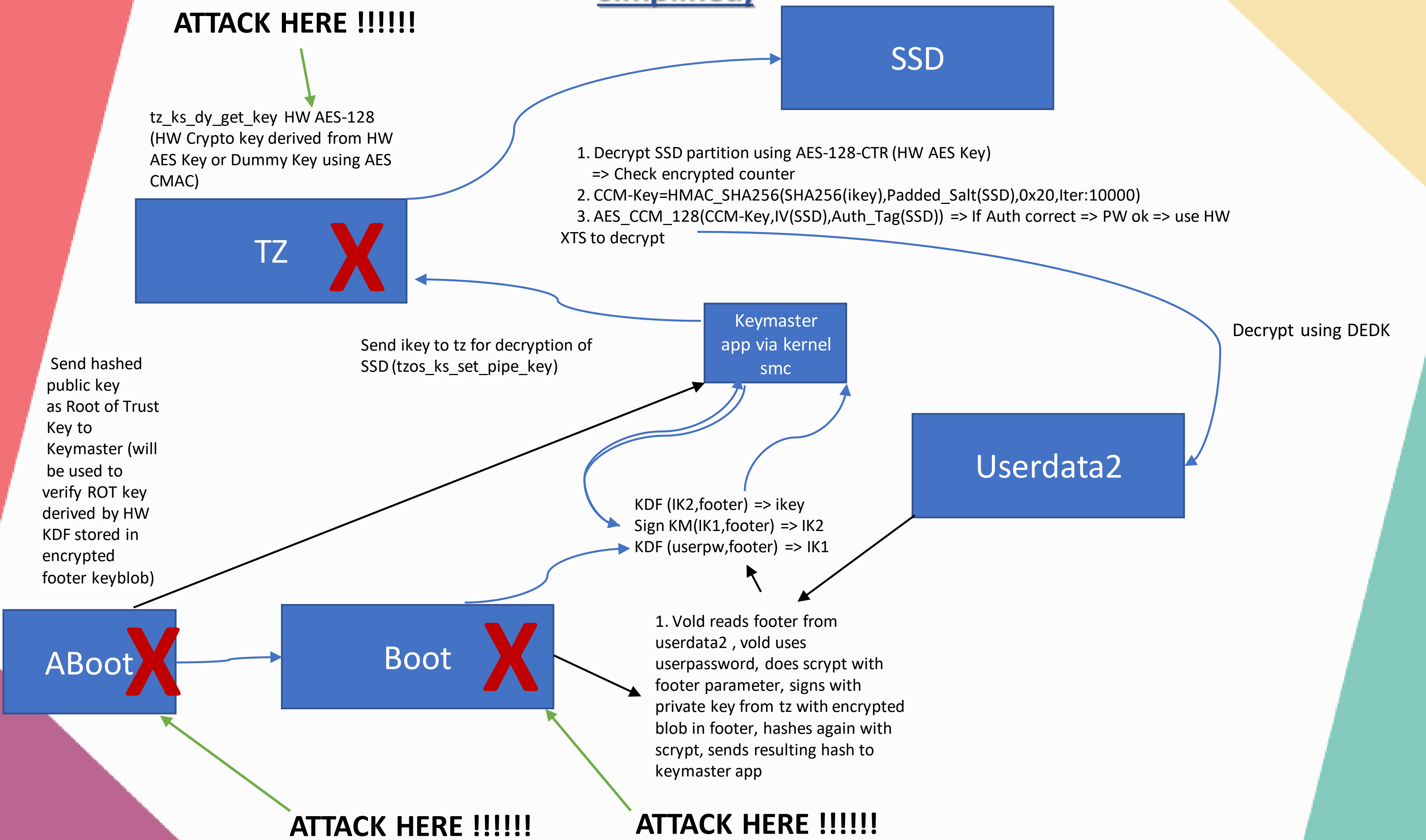
Some initial explanation

- SSD partition contains the encrypted tz keystore keys (I only saw the ICE Key in fact on my device after decrypting, but there is space for more).

- "QSEE_Dynamic:Keystore Entry Encryption Key" is used to decrypt SSD data

- "KM CPHR HW Crypto key derived from SHK" is used to decrypt the crypto footer using KM (KeyMaster)

Understanding Qualcomm Password Verification and HW Key Derivation (overly simplified)



To conclude on Avb v1

To inject own ramdisk, you need an :

a) aboot Exploit via Ram, patch selinux, patch init binary

or

b) aboot Exploit (Signature Hash and Length check) via physical, own boot ramdisk, resign with any private key, replace public key with stock to make root of trust work

Attacking tz is only needed if

- Secure Startup is used and PW is unknown (otherwise „default_password“ is the password)

For all online bruteforce attacks, partition “ssd“ and partition containing footer is critical (“userdata“, “metadata“, “reserved“, etc.)

- Hw derived encryption keys need to be extracted for offline bruteforce / decryption of ssd / userdata partition

Demo Time !

Attacking TZ and QC ROT/AVB v1 on a bootloader locked, secure startup enabled device with Android 8.1 (Stock, not HDK/Development) and patch level February 2019

Don’t panic !

My attack only works on improperly fused devices or devices using test/leaked keys. (Except someone has 0-days of course)

However some oems still misconfigure their fuses on Qualcomm Chips

Affected devices exposed to secure boot / key extraction attack

Affected devices

All devices with “Secure Boot” disabled or unfused in the QC Chip QFPROM

For most devices, this applies most of the time for MSM89xx devices with PK_Hash starting with: “cc3153a80293939b” as these are normally shipped with improper fuse settings

Affected devices

QFPROM register address	QC Chip Type
0x00058098	MSM8916, MSM8929, MSM8939, MSM8952
0x000a01d0	MSM8917, MSM8937, MSM8940, MSM8953, MSM8976, MDM9607
0xfc4b83f8	MSM8992, MSM8994
0x00070378	MSM8996
0x00780350	MSM8998

if the QFPROM register is set to 0x0 (Secure boot disabled)

Affected devices

In order to test, use: https://github.com/bkerler/EDL

Start device into 9008 / EDL (Sahara, loader not needed) mode, “pip3 install pyusb, capstone, keystone-engine” and run
(use option –qfp for qfprom dumping)

```
c:\EDL>python edl.py -loader none

Qualcomm Sahara / Firehose Client (c) B.Kerler 2018.


Trying with no loader given ...
Waiting for the device
Device detected :)
Mode detected: Sahara


------------------------
HWID:              0x000460e100000000 (MSM_ID:0x000460e1,OEM_ID:0x0000,MODEL_ID:0x0000)
PK_HASH:           0xcc3153a80293939b90d02d3bf8b23e0292e452fef662c74998421adad42a380f
Serial:            0x5ed55f9e
SBL Version:       0x00000000

Unfused device detected, so any loader should be fine...
```

Devices already confirmed to be vulnerable to this attack

Vendor	Models
BQ	Aquaris X, X Pro, X5, X5 Plus, C, V, U2, U2 Lite, U Plus, E5 4G, M5.5
Xiaomi	Mi 2
Gigaset	ME
Oneplus	One, X
ZTE	Z831
Infinix	Hot 6 Pro (X608)

Who isn't affected

All devices with QC SDM chipsets (SDM660, SDM845, etc.) should be fine by now

- TZ is now double signed, makes it hard to attack (Signer is QC and OEM)

- These devices normally use EDKII not LK for aboot

In order to research edk2 aboot (for service functionality that shouldn‘t be there), code segment in elf binary needs to be uncompressed (lz4) ☺

All devices with QC SDM chipsets (SDM660, SDM845, etc.) should be fine by now

- Some vendors hardware encrypt their firmware

(Bad thing as researchers won‘t have access to proprietary code, making the process of finding and reporting bugs very difficult but not impossible)

- AVB v2 introduces rollback protection and moves part of ROT process to vbmeta partition, making it harder to physically attack by flashing, and crypto footer will be encrypted.

Further remarks on my QC research

Feb.
2019

Further remarks on my research

Don’t ask for details, I won’t provide except for affected vendors (in order to protect users)

- Hotpatching TZ in memory on a running device is possible (independent on secure boot fuses).

  You have to mess with XPU2 and MemoryMapping (most areas are r-x, write protected after sbl/tz init)=>(If not disabled in the right order, device will reboot, preventing any attack).

Further remarks on my research

Don’t ask for details, I won’t provide except for affected vendors (in order to protect users)

- Dumping QC BootRom (PBL) on MSM8xxx is possible (independent on secure boot fuses).

  Using EDL loaders with peek command. Research it and report bugs to Qualcomm. Don‘t think that PBL can be updated using OTA as it‘s read-only after fusing at factory afaik. For details on attacks see AlephSecurity blog.

Further remarks on my research

Don’t ask for details, I won’t provide except for affected vendors (in order to protect users)

- On improper fused devices, both cloning and offline (no hardware needed) bruteforce attacks are possible. (depending on secure boot fuses)

Two TZ and two KM keys are needed (on MSM8974, two TZ Keys [AES Key and HMAC Auth Key] were sufficient).

Further remarks on my research

Don't ask for details, I won't provide except for affected vendors (in order to protect users)

- ROT Key should be used for deriving hw keys.

  At least in my tests for decrypting the device, ROT key was be different but still worked for decryption after patching some checks in tz.

- Re-Enabling JTAG is a thing.

  Reversing, it looks like using a specific signature on specific qc partitions, JTAG gets re-enabled even on production devices but then uses a hardcoded dummy hw key.

Further remarks on my research

Don’t ask for details, I won’t provide except for affected vendors (in order to protect users)

- Permanently disabling EDL is a thing.

  Yes, some do that by fusing it to qfprom.

Or in other words

Google and Qualcomm Security are doing amazing work, making the newest devices very secure.

However, still a lot has to be done.

My very own opinion

Every device owner should have the right to have **full** access to their own data.

Only having a crippled useless backup interface, improper or impossible bootloader unlocks and custom ramdisks with **risk of bricks**, annoying orange screen on startup, no proper root shell and companys and bad individuals attacking user privacy with massive use of ad sdks / malware or rootkits pushed me to break my own devices to make them usable for research.

My recommendations

1. Listen to the research and modding community.

2. Do allow proper unlocking of devices with no restrictions (or annoying nags) and possibility to have a root shell on locked devices (authenticate the user first before offering that functionality) to ensure privacy of users (by enabling researchers to detect misuse). Stop protecting criminals and making money with user data (without users consent).

3. Stop putting service/backdoor functionality in end user devices (only development should have that).

4. Enforce signing of kernel modules and all partitions.

5. Open source should be the way to go (at least for everything critical involving encryption and boot protection).

Code + PoC

https://github.com/bkerler/android_universal
https://github.com/bkerler/bootimg
https://github.com/bkerler/EDL
https://github.com/bkerler/twrp_tz_fixes
https://github.com/bkerler/dump_avb_signature
https://github.com/bkerler/exploit_me
https://github.com/bkerler/qc_signer (to be released)

Recommended Lecture and my hall of fame

You girls/guys rock, and I mean it:

http://bits-please.blogspot.com (Gal Beniamini, for older MSM8974 TZ research)
@beaups (Sean Beaupre, a huge thanks for your help)
http://blog.azimuthsecurity.com (Dan Rosenberg)
https://alephsecurity.com/ (EDL/PBL attacks, Roee Hay)
RedNaga, Jon Sawyer,
All darkcellular Slack Members
!!!!!! All GTFO contributers !!!!!!
Qualcomm Security Guys (blog more, share more documentation openly !),
Google Zero Members (Thanks for the blogs),
All colleagues/reversers (national and international) fighting against crime
My family for taking care of me and enabling my research
All people I forgot and I'm sorry for ☺

No 0-day exploits were used for this talk

☺

That’s it folks !

Thanks for listening.