Attacks on Android 7 File Based Encryption

Ronan Loftus[1] & Marwin Baumann[1]

[1]Systems and Network Engineering MSc.
University of Amsterdam

Research Project 1, 2017

Table of Contents

Encryption Since Android 3.0

'Full Disk' Encryption:

- Encrypts the data partition

Major problem:

- Needs user interaction after reboot

New in Android 7.0

File Based Encryption:

- Still only encrypts the data partition
- Each file encrypted with separate key
- Per user encryption

Why?

Why do people want to encrypt their devices?

Why?

Why do people want to encrypt their devices?

To protect data at rest.

- When device is lost/stolen keep your personal data confidential
- Businesses can feel more comfortable keeping sensitive data on employee devices

What's the question?

Our primary research question:

Is Android 7 File Based Encryption vulnerable to the same attacks as Full Disk Encryption in previous Android versions?

What's the question?

Our primary research question:

Is Android 7 File Based Encryption vulnerable to the same attacks as Full Disk Encryption in previous Android versions?

Kind of. . .

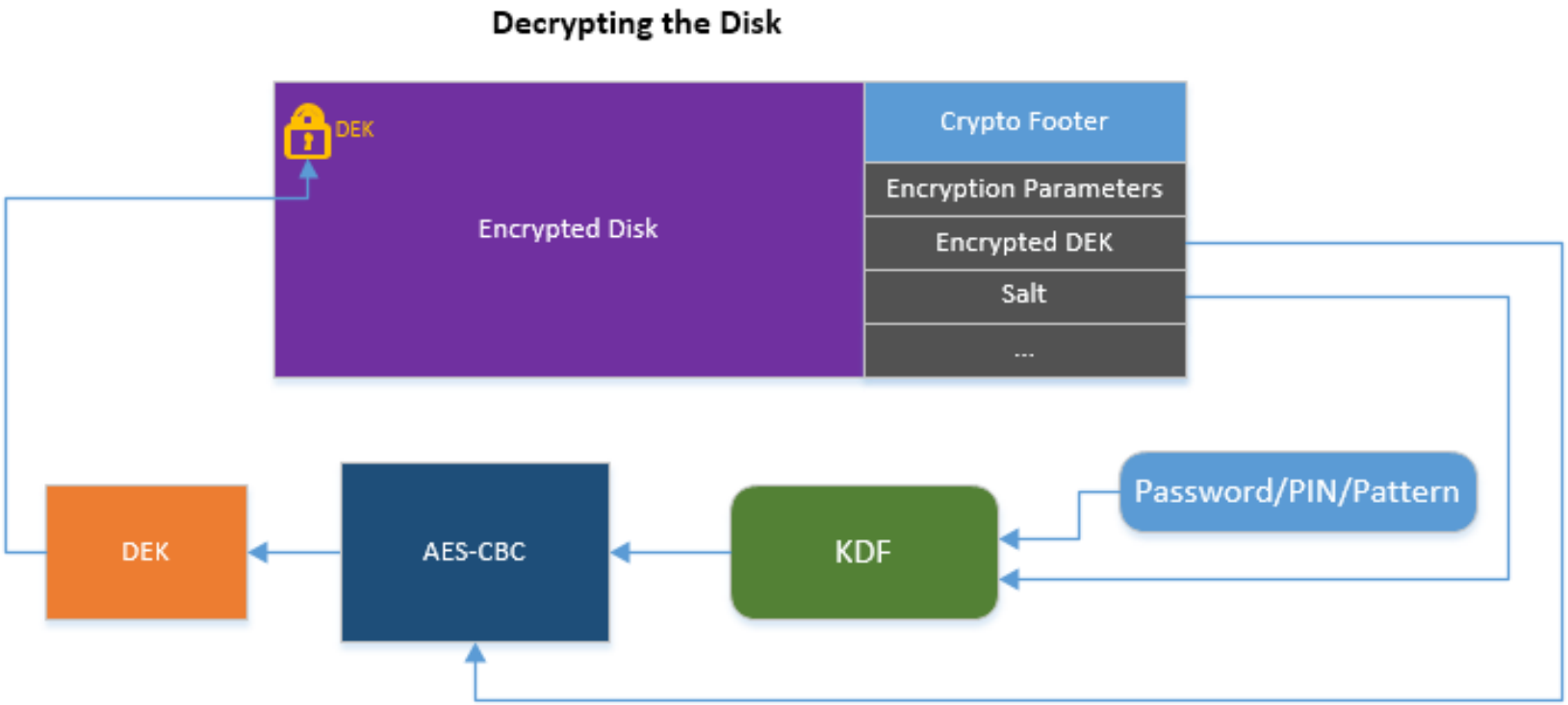
Table of Contents

How does Full Disk Encryption work?

- Uses dm-crypt
- (u)Randomly created master key (DEK) encrypts data partition using AES-128 (CBC)
- DEK encrypted with KEK using, at least, AES-128 (CBC)

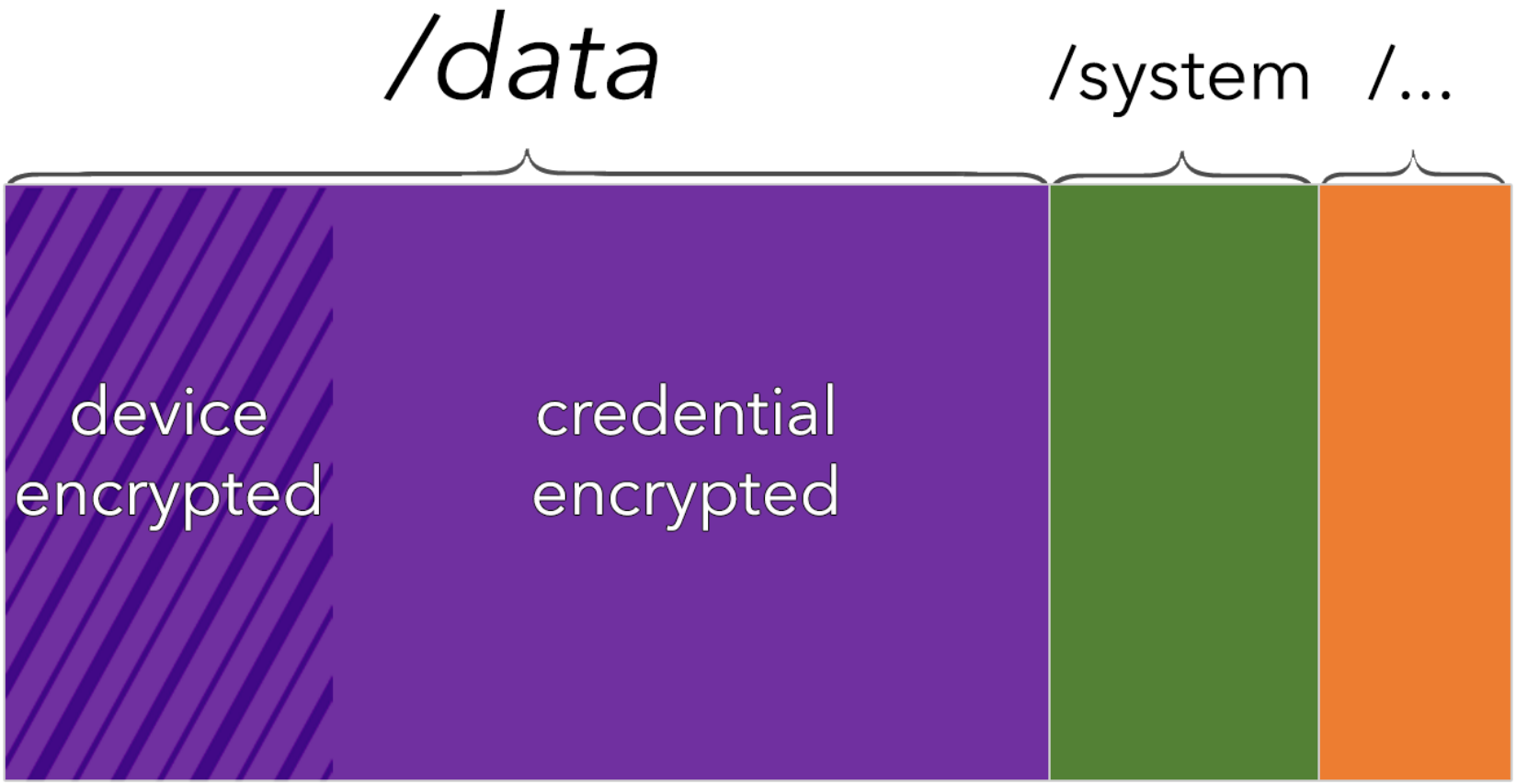
Master key is static until partition wiped.

Full Disk Encryption overview



How does File Based Encryption work?

Two areas encrypted differently



This solves the big problem with Full Disk Encryption mentioned earlier.

How does File Based Encryption work?

Many keys

Uses native ext4 filesystem level encryption

- 512 bit master key is encrypted using AES-256 in GCM mode
- File names encrypted using AES-256 in CBC-CTS mode
- File contents encrypted using AES-256 in XTS mode

Master key still static!

File Based Encryption overview

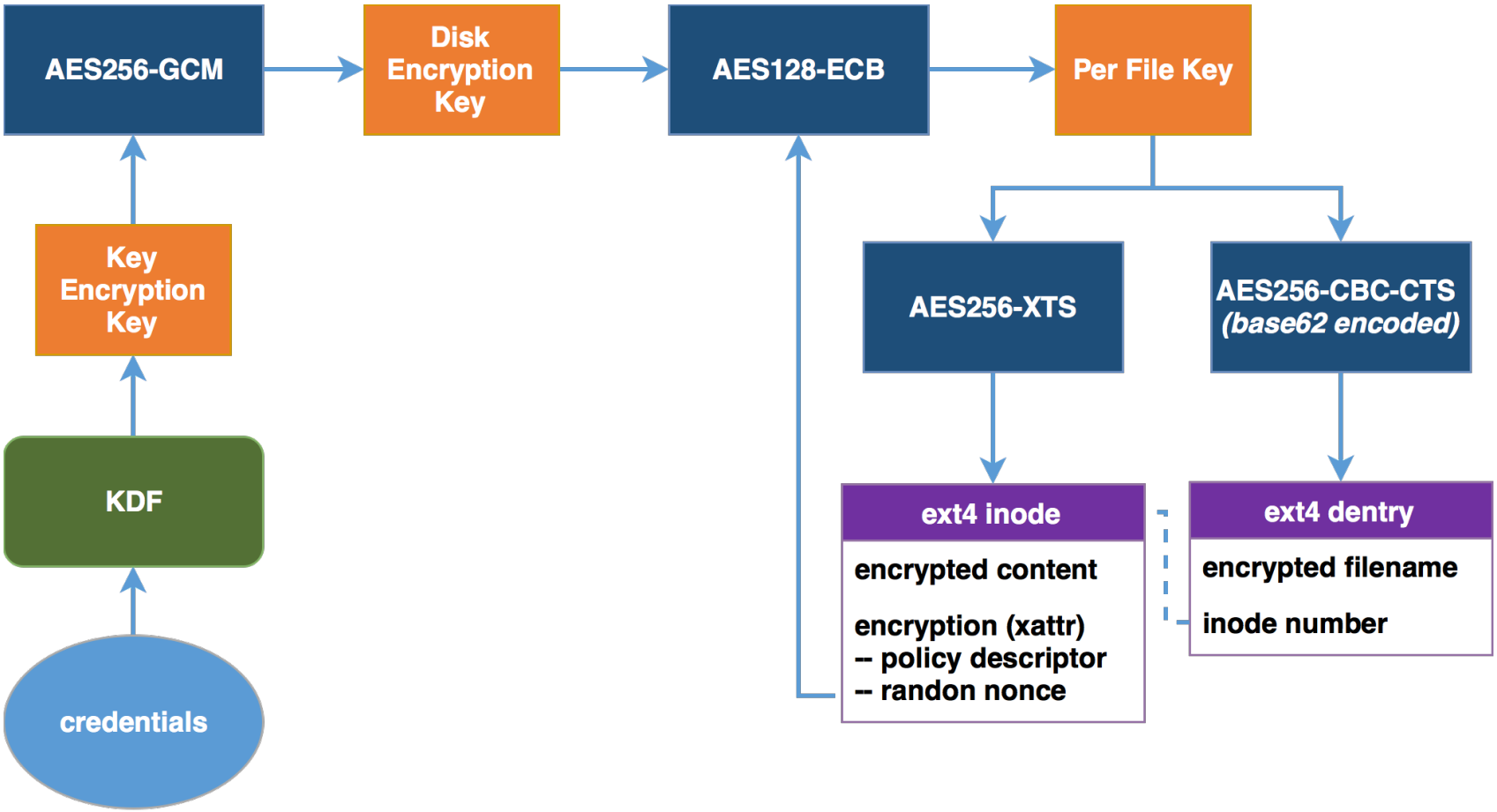


Table of Contents

Madness

Let's attack the cryptosystem directly . . .

Madness

Let's attack the cryptosystem directly . . .

Nope!

Cold Boot

- Data remanence attacks rely on cryptographic keys being kept in memory
- Trusted Execution Environment (TEE)
  secure area of the main processor
- Since TEE, keys not stored in RAM

Brute Force (online)

Enumerate all the combinations. Always possible in theory!

Attack:

- Using Android Debug Bridge
- Using On-the-Go protocol
- Using robot

Brute Force (offline)

Qualcomm

no TEE:

- Image partitions and start cracking

Brute Force (offline)

Qualcomm

no TEE:

- Image partitions and start cracking

with TEE: not possible, unless the device has a Qualcomm chip (≈60% of Android devices)

- The key derivation function is not actually bound to the hardware in Qualcomm chips
- Been patched in AOSP but still exists in hardware so a downgrade attack is still viable for Full Disk Encryption

Brute Force (semi-online)

- Try to offload some of the work from the device
- Make the device do the hardware specific work then compute the rest on a more powerful machine

Evil Maid

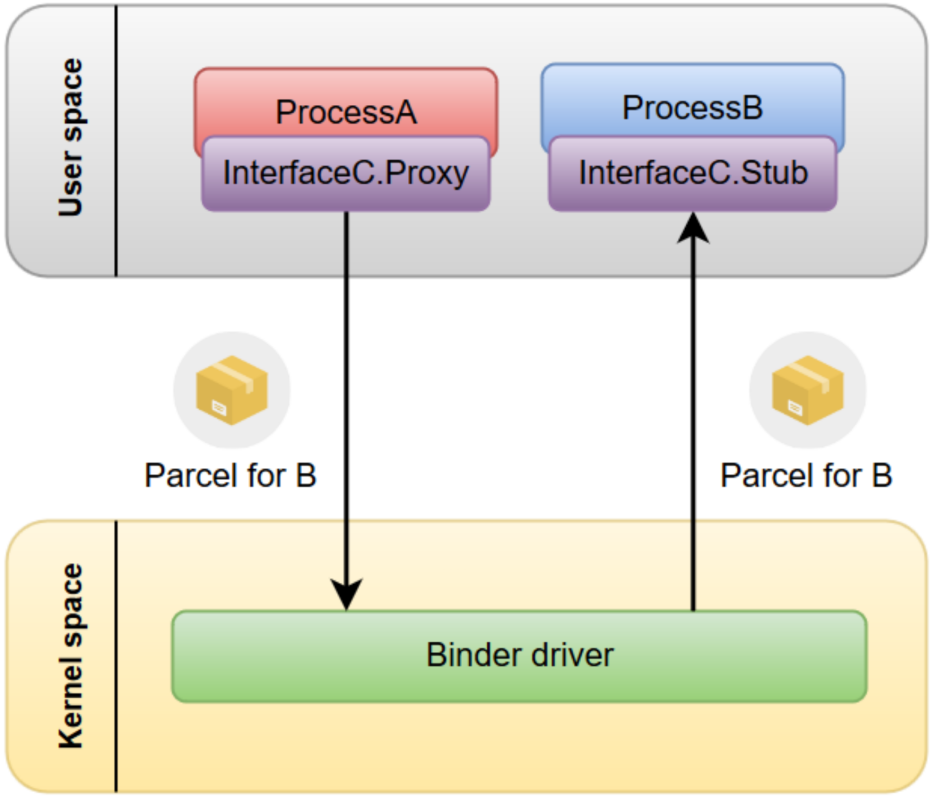
Classic attack on encrypted devices. Just install a keylogger!

- Capture users authentication credentials using a non encrypted part of the device
- Install a new keyboard
- Subvert code displaying PIN prompts

Evil Maid

Subvert "Binder"

- Input Method Editor



COM.ANDROID.INPUTMETHOD.LATIN

Fingerprints

- Becoming far more common for users to authenticate to their cryptosystem via fingerprint
- With trivial modification to the source, sensor will authenticate anything it can read

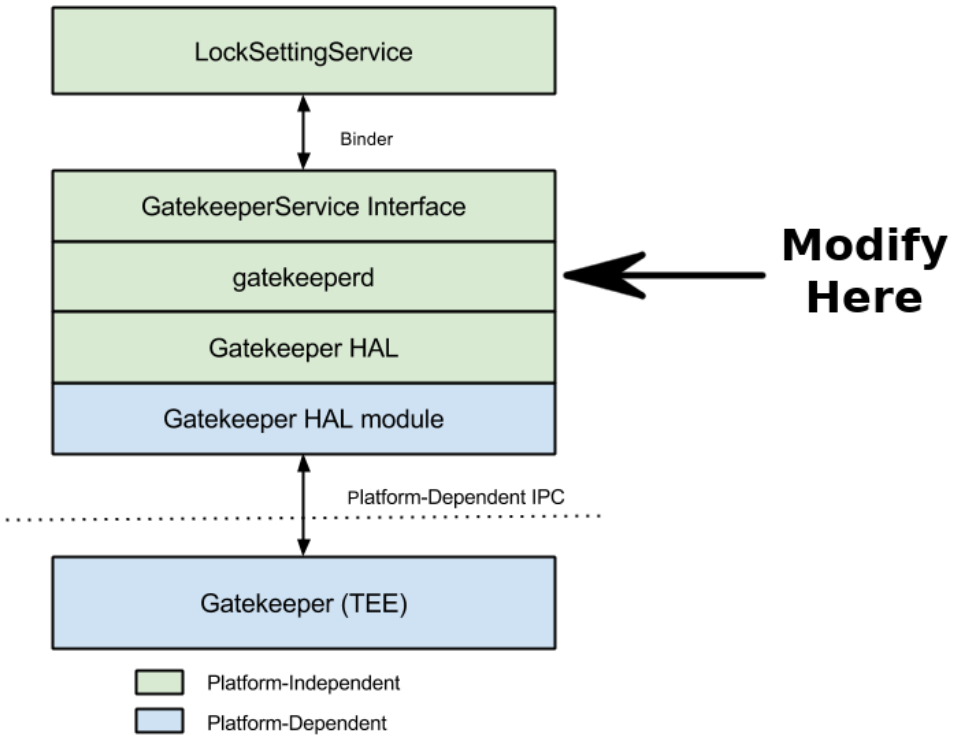
Table of Contents

Cold Boot

- Since Android 7.0 devices MUST come with a hardware backed keystore (TEE)
- This renders remanence attacks obsolete!

Brute Force (online)

- Rate limits have been updated since 7.0
- Try to subvert the "Gatekeeper"



Brute Force (offline)

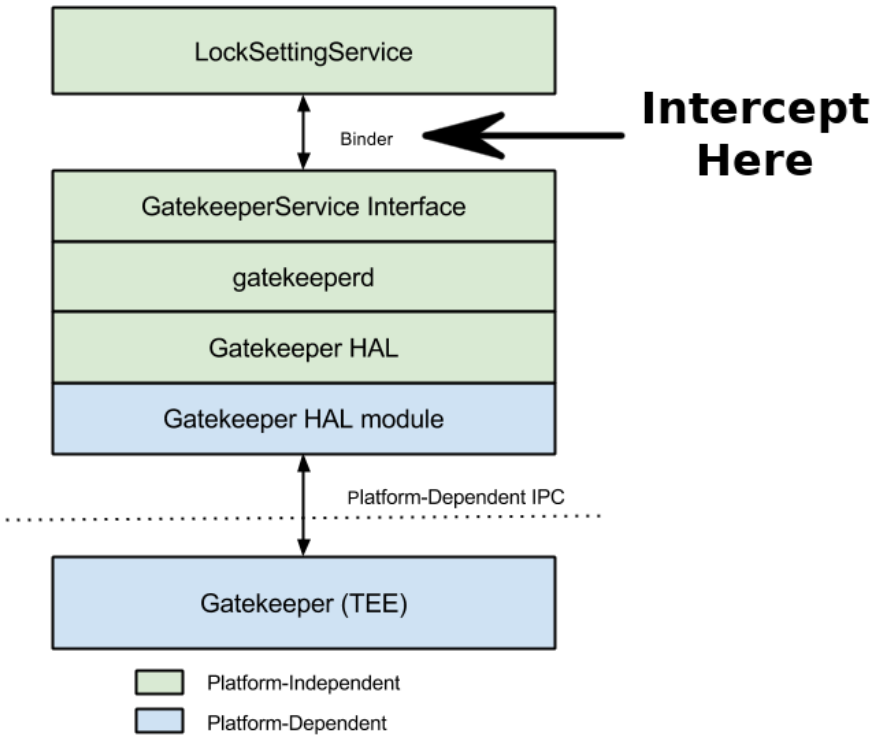
- Qualcomm vulnerability has been software patched
- Downgrade attack would still be possible but . . .

Brute Force (semi-online)

- We theorise that is still possible
- Untested

Evil Maid

- Hook the binder
- Dump the credentials to the disk
- Can also Send over network (out of scope)



IPCThreadState.cpp

Listing 1: IPCThreadState::talkWithDriver

```
1 ioctl(mProcess->mDriverFD, BINDER_WRITE_READ, &bwr)
```

Listing 2: evil_ioctl

```
1 int evil_ioctl(int fd, int op_type,
      binder_write_read* bwr)
2 {
3     int res = ioctl(fd, op_type, bwr);
4     evil_reply_manipulation(bwr);
5     return res;
6 }
```

Fingerprints

Phone contains modified fingerprintd binary.

Listing 3: FingerprintDaemonProxy.cpp

```
1     callback->onAuthenticated(device,
2 //                msg->data.authenticated.finger.fid,
3                   0x1a4, // non-zero id
4                   msg->data.authenticated.finger.gid);
```

Summary of Attacks

Attack needs these conditions to succeed:

Attack	7.0+	Requirement
Online BF	X [1]	- [2]
Offline BF	X	-
Semi-online BF	✓	unlocked bootloader
Cold Boot	X	unlocked bootloader
Evil Maid	✓	custom recovery
Fingerprints	✓	custom recovery

[1] unless can subvert gatekeeper
[2] ADB enabled allows automation

Recommendations

With great power . . .

To AOSP:

- Encrypt more of the device to reduce attack surface
- Encrypt (sensitive) binder communications

Recommendations

With great power . . .

To the user:

- Turn off root access

Recommendations

With great power . . .

To the user:

- Turn off root access
- Use the stock recovery

Recommendations

With great power . . .

To the user:

- Turn off root access
- Use the stock recovery
- [re]Lock your bootloader

Recommendations

With great power . . .

To the user:

- Turn off root access
- Use the stock recovery
- [re]Lock your bootloader
- Use long complex password

Recommendations

With great power . . .

To the user:

- Turn off root access
- Use the stock recovery
- [re]Lock your bootloader
- Use long complex password
- Have no fun with your device!

That's All Folks

Questions?

Comments?

Critisicms?