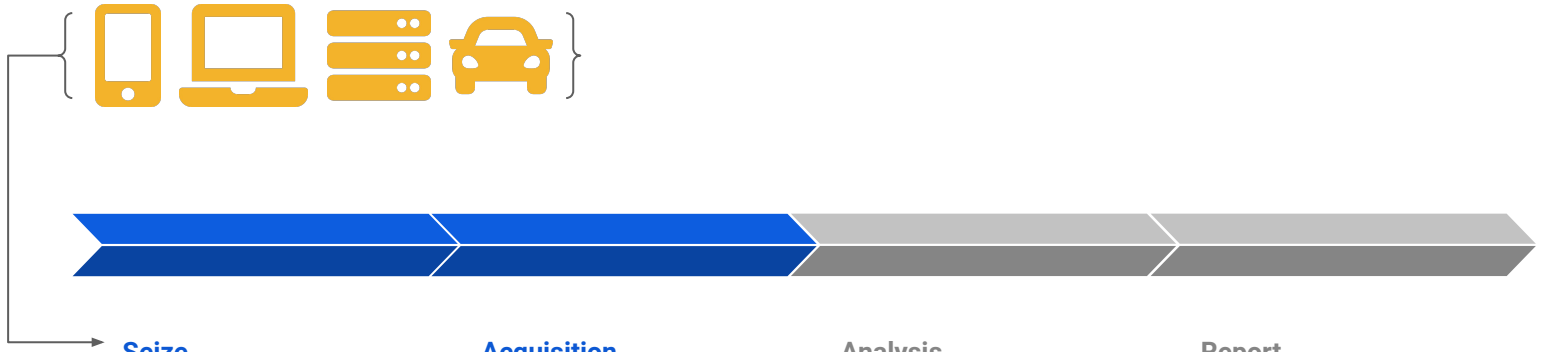
Chip Chop - Smashing the Mobile Phone Secure Chip for Fun and Digital Forensics

Gunnar Alendal
Norwegian University of Science and Technology (NTNU)
@gradoisageek


INVESTIGATION


eSE
SAMSUNG

Digital forensics (simplified)



Seize

Identifying and seizing, devices, hard drives, ...

e.g.

Acquisition

Extract data to be analysed

=

Digital Forensics Acquisition (DFA)

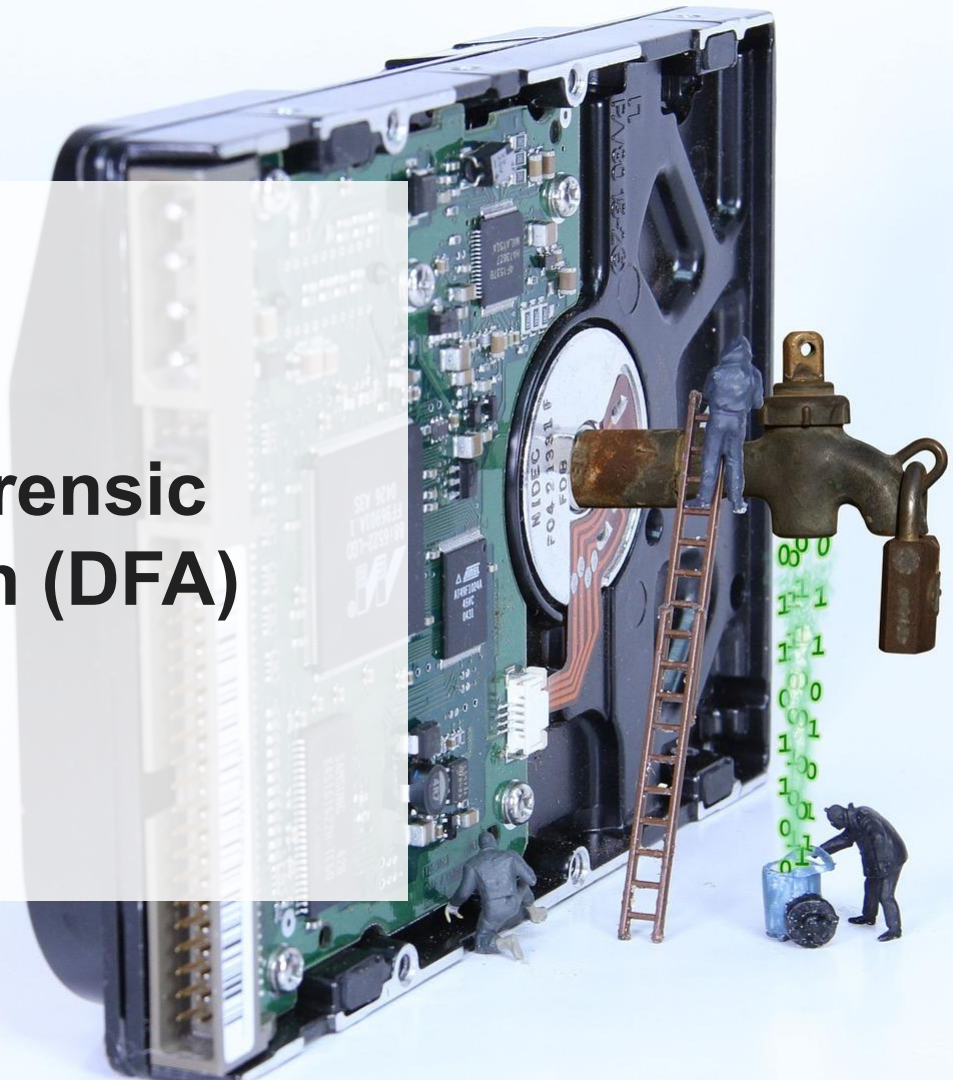
Analysis

Identify and analyse data relevant to investigation

Report

Report on positive and negative findings

Digital Forensic Acquisition (DFA)



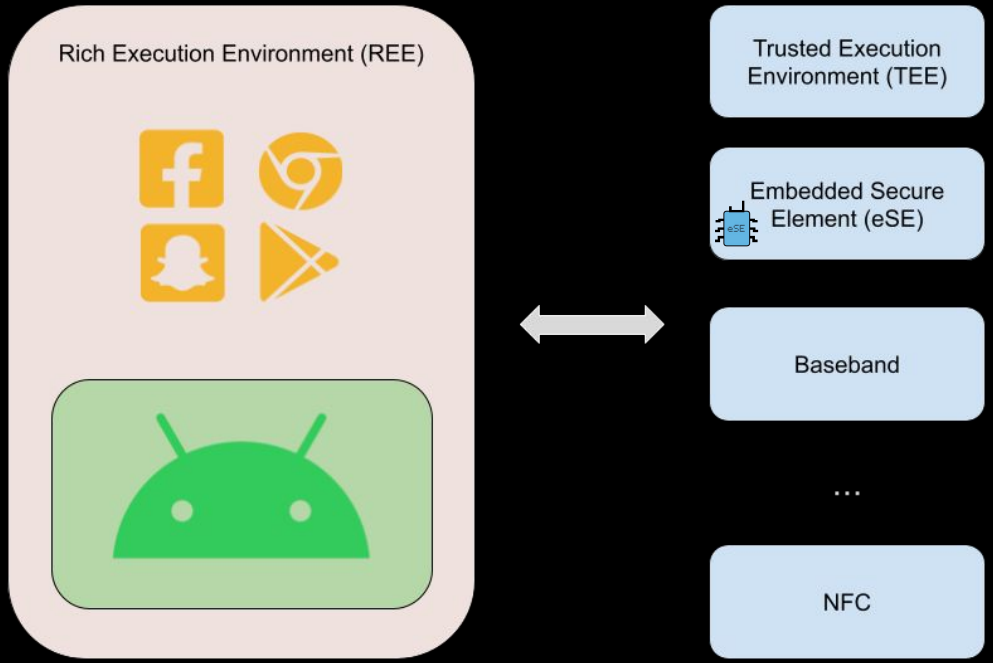
pixabay.com

Before

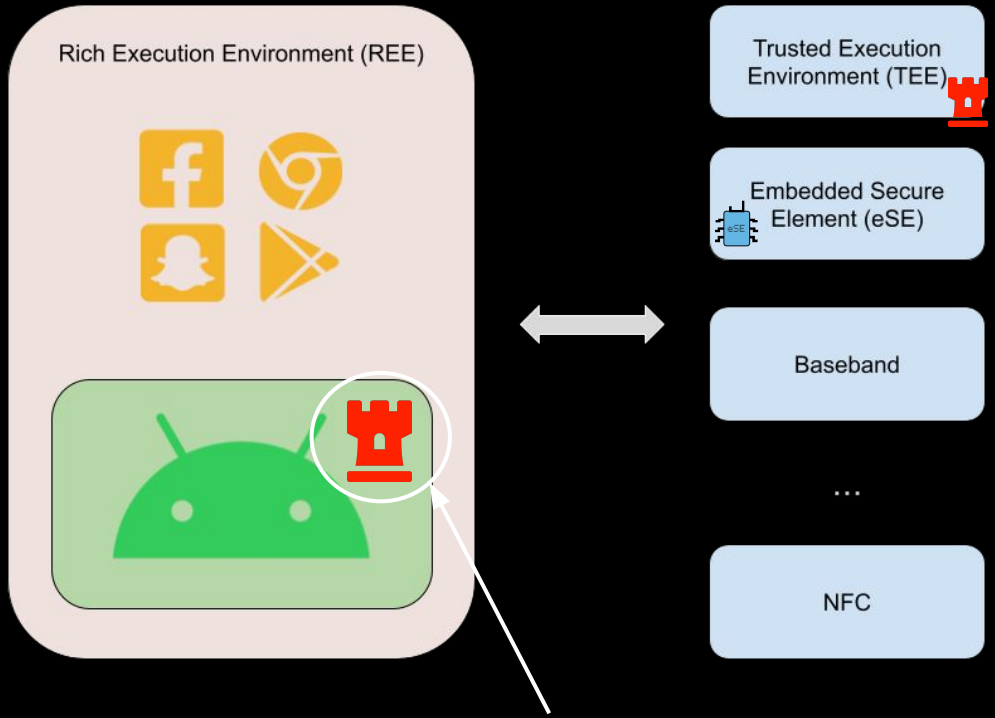
pixabay.com

Now

Android Security 101

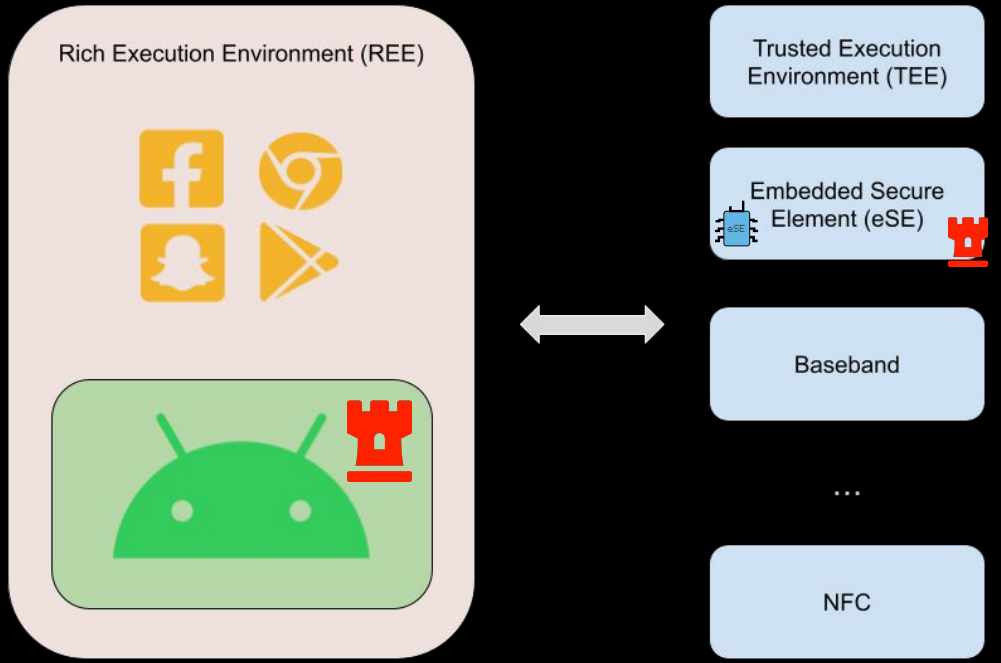


Untrusted & **Trusted** worlds



Towers preventing DFA <= Galaxy S10

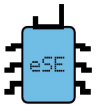
“Breaking Samsung's Root of Trust: Exploiting Samsung S10 S-Boot”
Jeff Chao / Black Hat 2020



Towers preventing DFA >= Galaxy S20

1 + 1 = Digital Forensic Acquisition

This talk

Break REE + break 

=> 

(embedded) Secure Element - eSE

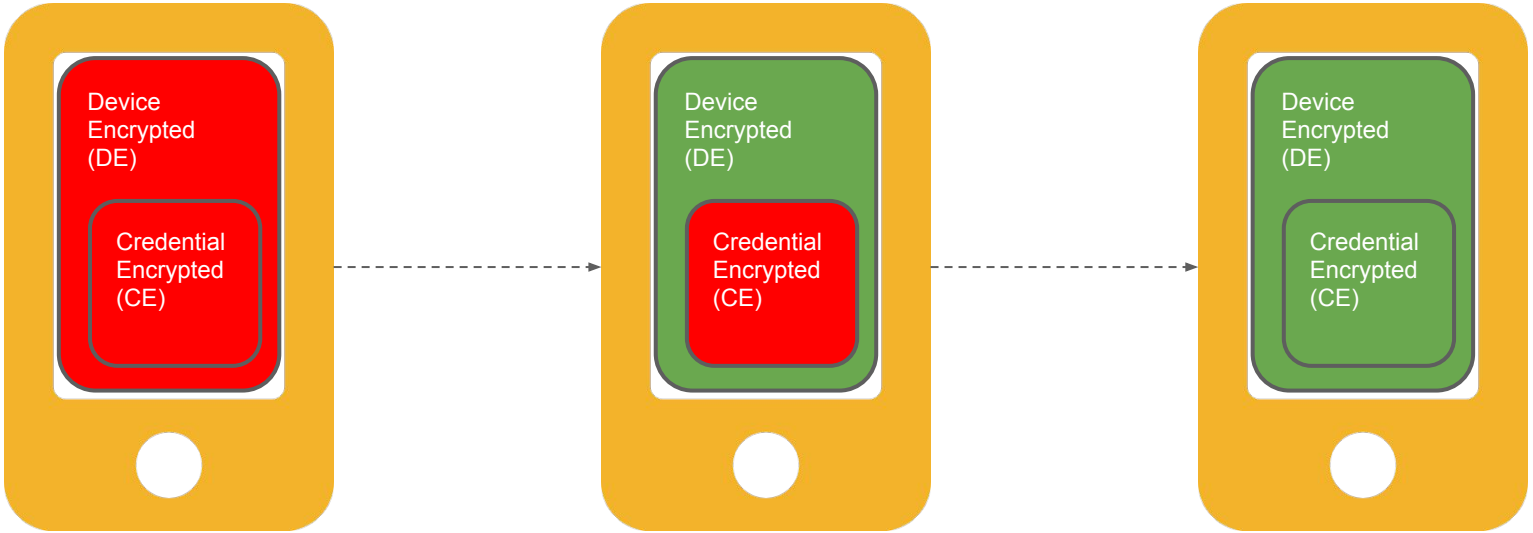
- Model in Galaxy S20 (Exynos): **S3K250AF ***
- Separate HW chip
- Protects encryption key material
- Prevents brute force from compromised system (“root”)

- Break eSE => gain access to encryption key material

* Full paper presented @DFRWS USA 2021:
“Chip Chop - Smashing the Mobile Phone Secure Chip for Fun and Digital Forensics”

Android File-based Encryption (FBE)

Android FBE States

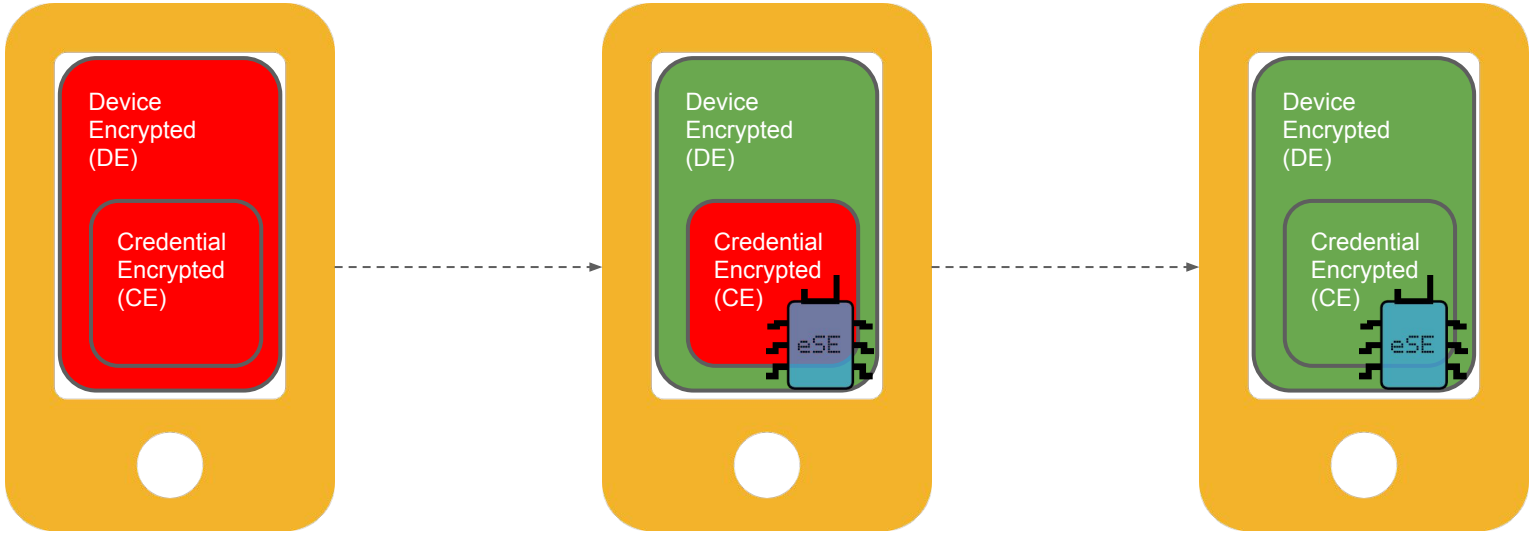


Device off

Power on / no unlock
Before-first-unlock (**BFU**)

Power on / first unlock
After-first-unlock (**AFU**)

Android FBE States & eSE

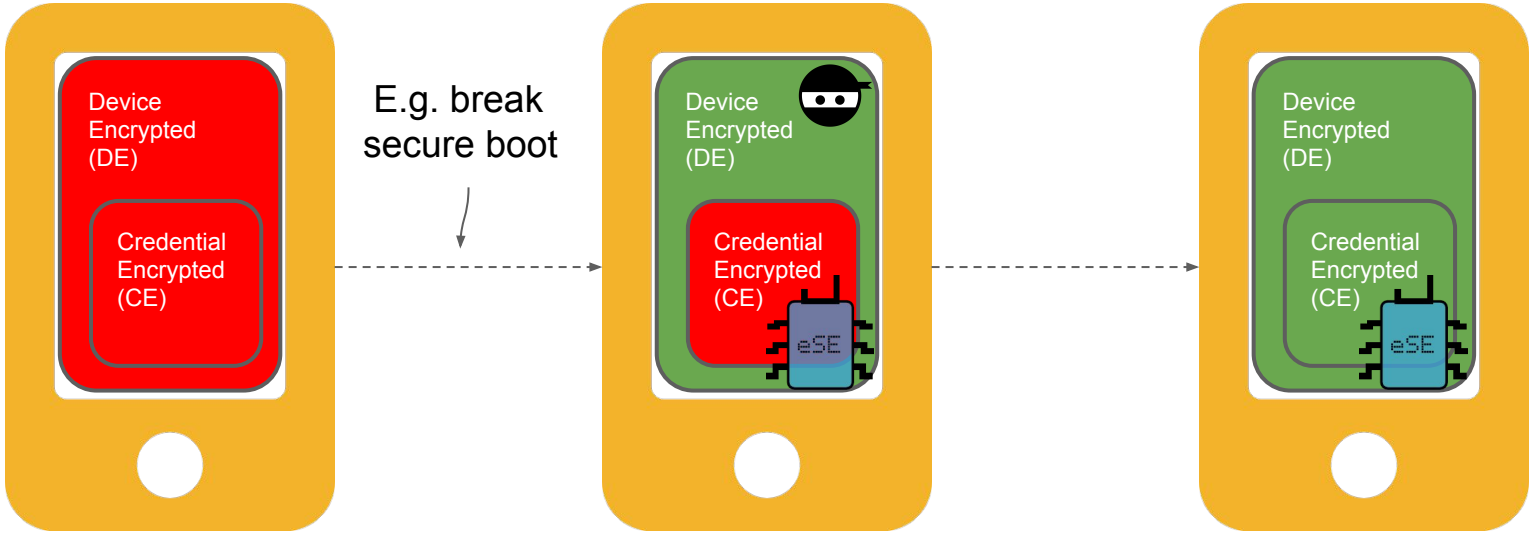


Device off

Power on / no unlock
Before-first-unlock (**BFU**)

Power on / first unlock
After-first-unlock (**AFU**)

Attack phase 1: “root” REE

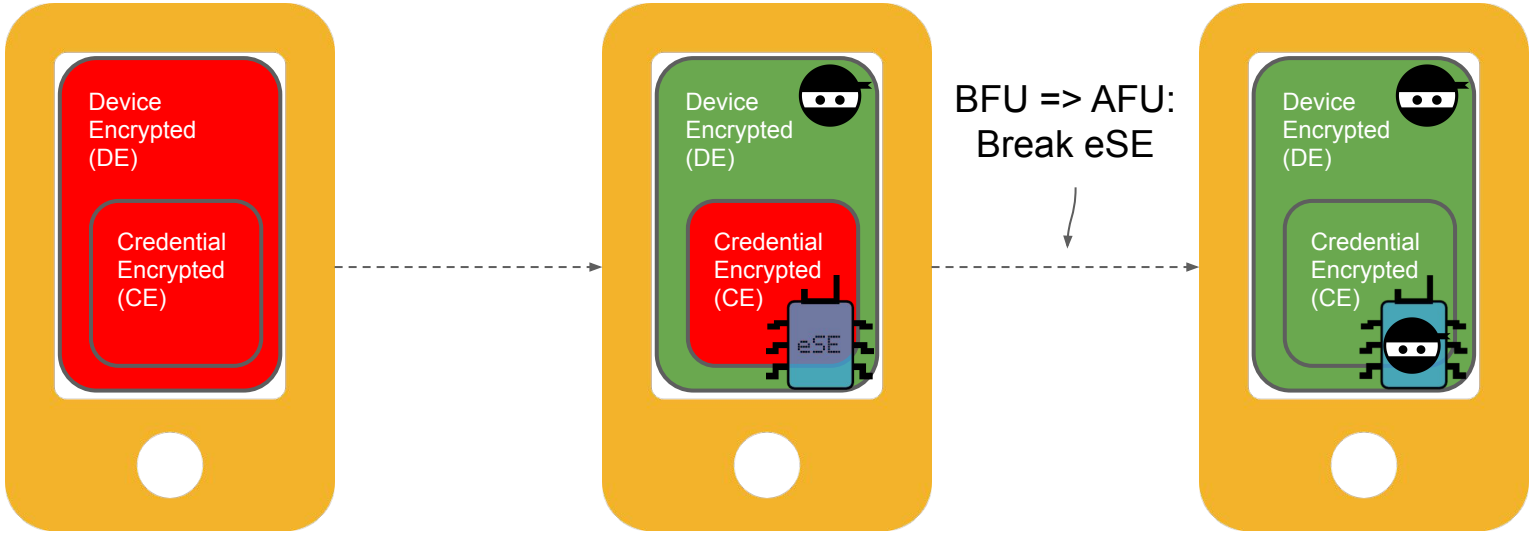


Device off

Power on / no unlock
Before-first-unlock (**BFU**)

Power on / first unlock
After-first-unlock (**AFU**)

Attack phase 2: eSE: Force BFU to AFU



Device off

Power on / no unlock
Before-first-unlock (**BFU**)

Power on / first unlock
After-first-unlock (**AFU**)

BFU => AFU
w/ weaver

pw/pin/pattern

+

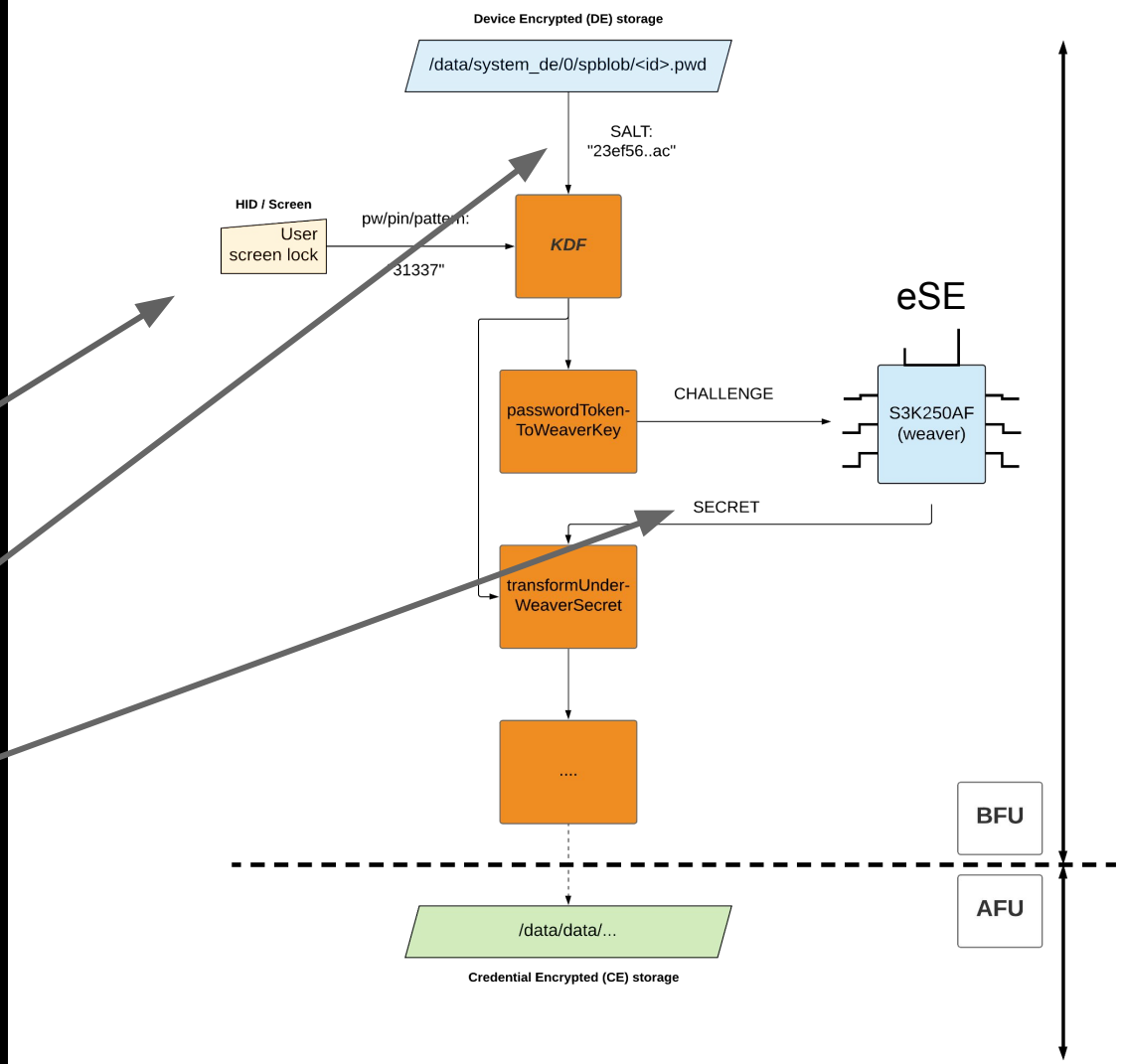
SALT (DE)

+

SECRET (eSE)

=

AFU



Brute force

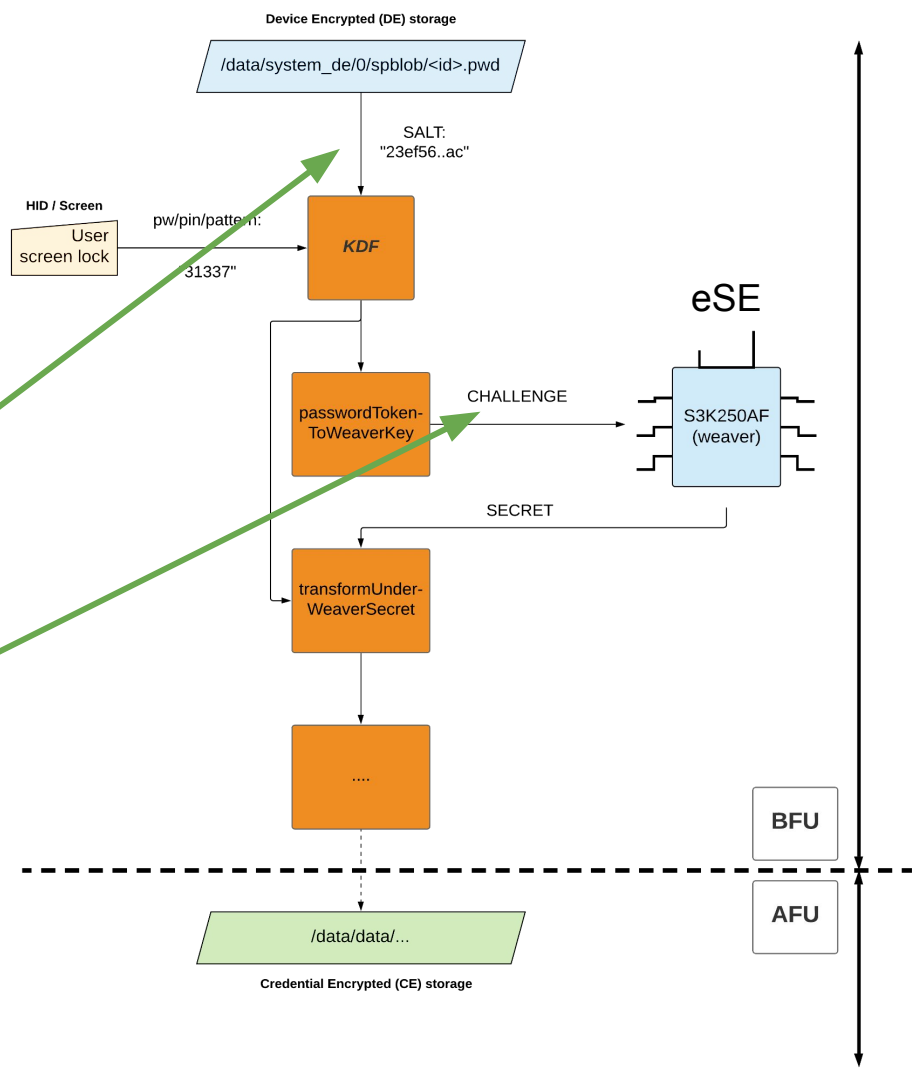
pw/pin/pattern

=

BruteForce(SALT (DE)

+

CHALLENGE (eSE))



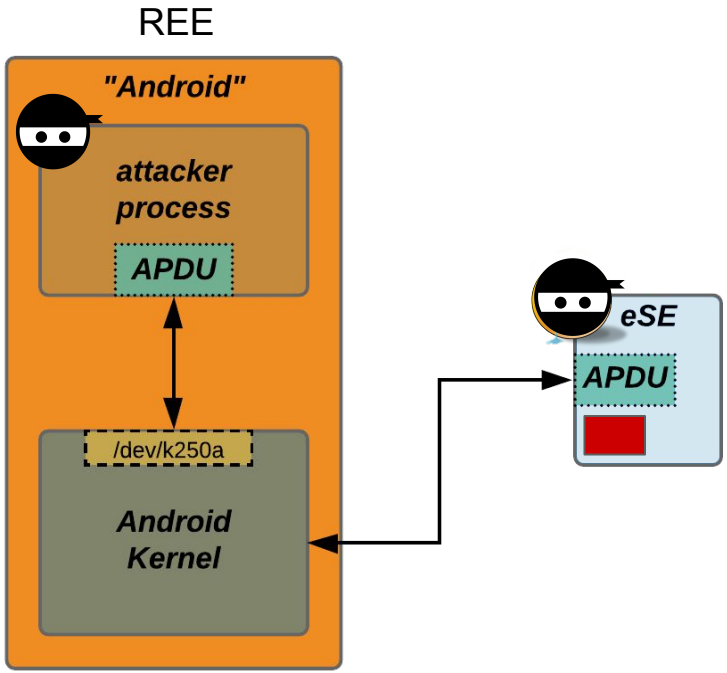
Attack Summary*

*Executive edition

Attacking the FBE (CE)

1. Break REE: “root” / SALT
2. Attack eSE

3. Get CHALLENGE + (SECRET)
4. Off-device brute force pw/pin/pattern



Off-device brute force pw/pin/pattern


```
for pin in all_pins:
  # KDF(PIN, SALT)
  computePasswordTokenRes = scrypt.hash(pin,SALT,N=scryptN,r=scryptR,p=scryptP,buflen=PASSWORD_TOKEN_LENGTH)
  # Generate CHALLENGE candidate
  sha512                  = hashlib.sha512(PERSONALISATION_WEAVER_KEY)
  sha512.update(computePasswordTokenRes)
  personalisedHash        = sha512.digest()

  # Compare candidate CHALLENGE with stolen CHALLENGE
  if personalisedHash[:stolenCHALLENGELen] ==stolenCHALLENGE:
        print("\n================================\n")
        print("     Correct pin is: %s"%pin)
        print("\n================================\n\n")
        print(" pwdToken         hash : " + computePasswordTokenRes.hex())
        print(" weaver CHALLENGE hash : " + personalisedHash[:stolenCHALLENGELen].hex())
```

The eSE attack

from 0 to 0-day

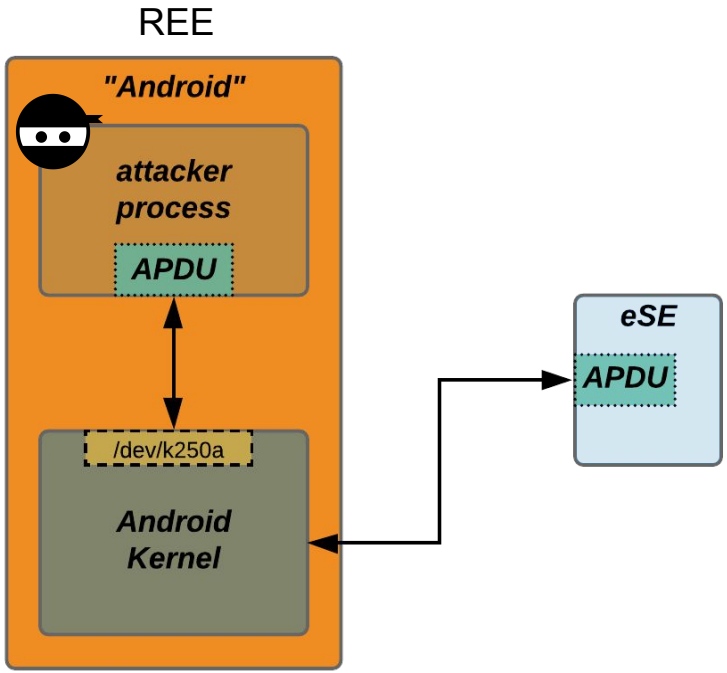
Enter S3K250AF eSE!

- Introduced 2020 in Galaxy S20 models (Exynos)
- Black box IC
- ARM BE8 THUMB
- 252 kB on-board flash + 16 kB RAM
- CC EAL 5+ certification
- Designed to protect against HW attacks, like Side-Channel attacks
- Brute force protection

- Features: **Weaver** / SecNVM / Device Attestation / Keystore / ..

eSE = “Black box”

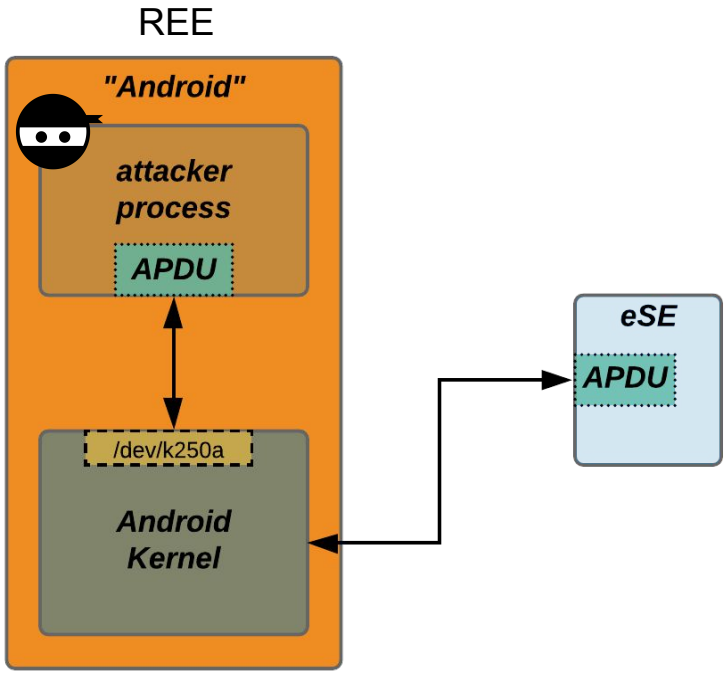
- REE talks to eSE
  - hermesd process
  - Frida instrumentation
  - Reimplement in chip_breaker
- Talks APDU
  - Just like a SIM card
  - APDU handlers in eSE FW
- Reverse engineer REE commands
  - REE .so + small FW part
  - We can talk “dirty” to it!
- But **no debug / info leak**
  - Locate **oracles**!




Info leak
Oracles needed

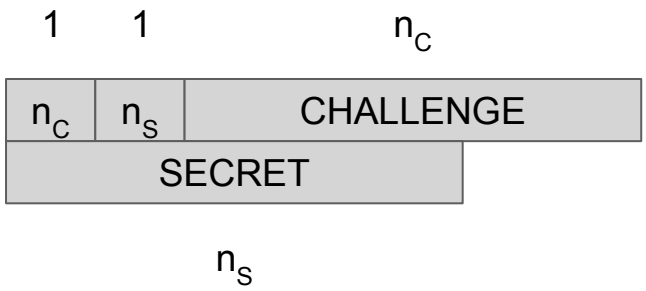
Oracle 1

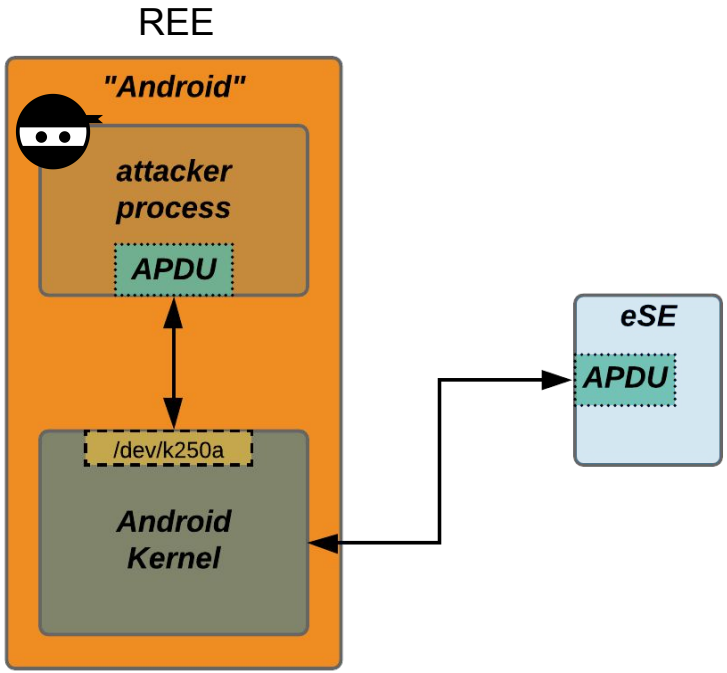
- APDU handler error:
  - APDU response w/error code
  - Error = APDU SW (Status Word)
- APDU handler crash:
  - No APDU response!



Oracle 2

- Promising eSE ADPU handlers:
  - `APDU_readWeaver`
    Send CHALLENGE
  - `APDU_writeWeaver`
    Set CHALLENGE / SECRET



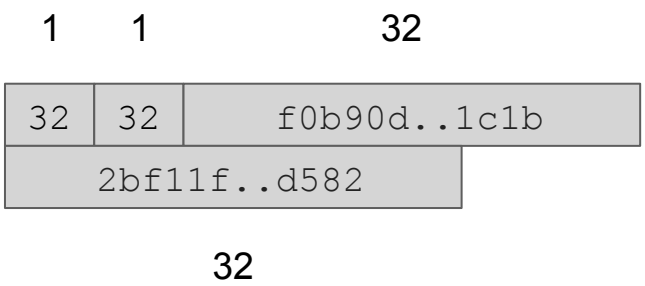


Oracle 2 (simplified)

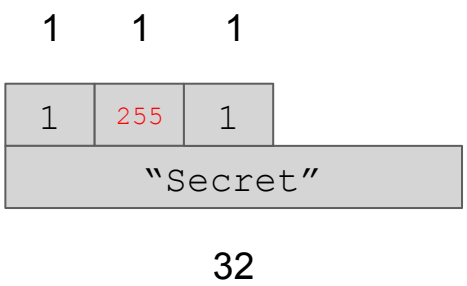
- `APDU_writeWeaver`

  First: Set CHALLENGE / SECRET

Normally



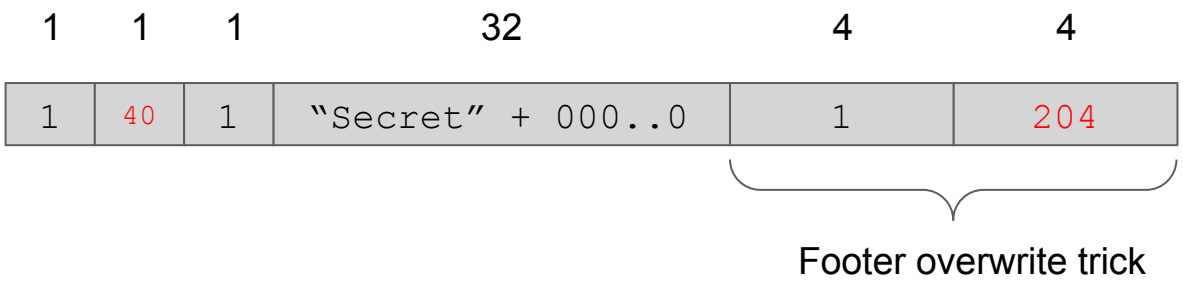
What if?



Oracle 2 (actual)

- `APDU_writeWeaver`

  First: Set CHALLENGE / SECRET

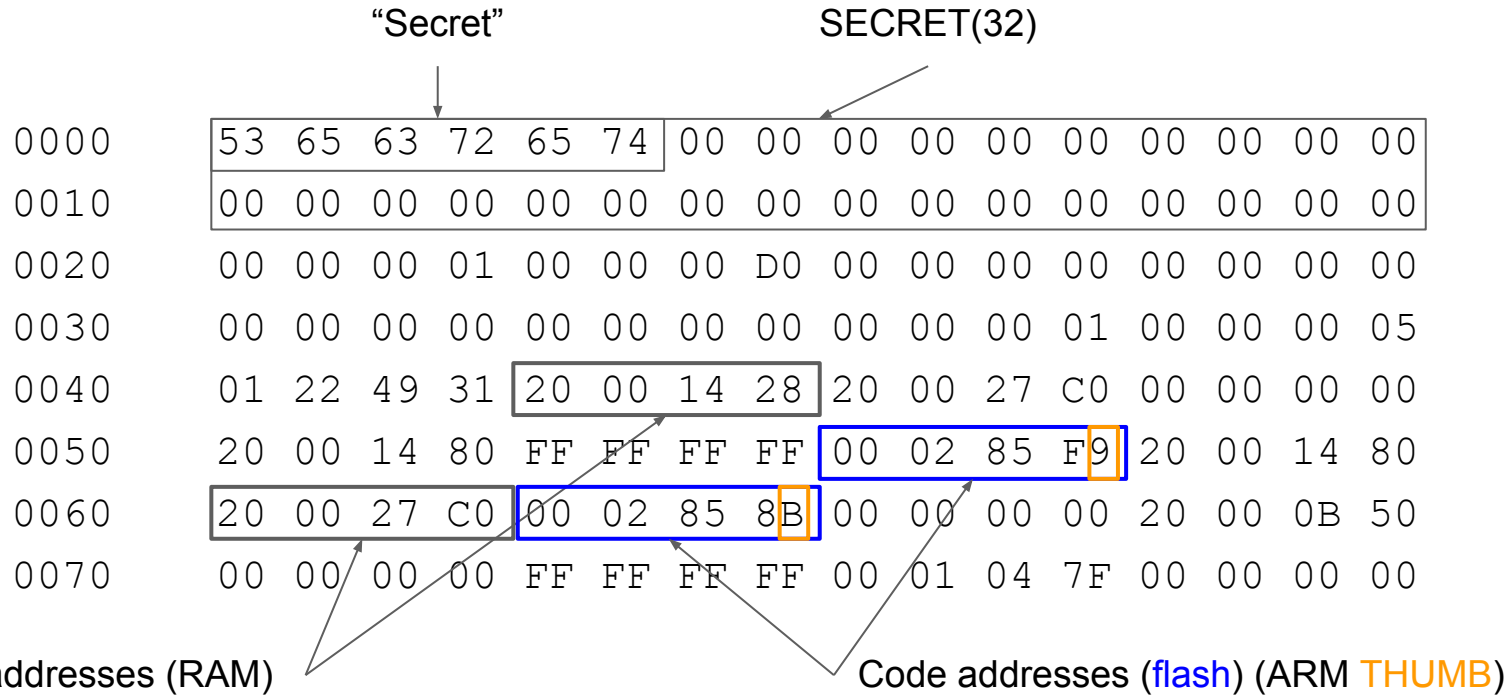


Oracle 2

- `APDU_readWeaver`

  Second: Send CHALLENGE

Oracle 2 - Stack leak!



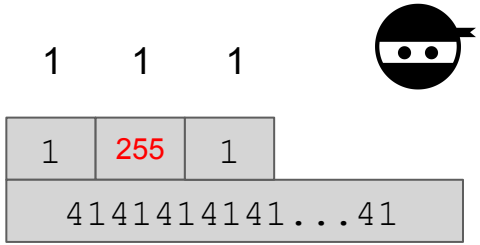
Oracle 2 features

- Leak RAM address range + pointers
- Leak CODE (flash) address range + pointers
- Stack layout of `APDU_readWeaver`
- Enable dynamic reverse engineering
- Further experimenting different APDU handlers
- BlindROP / DarkROP like testing

From Oracle to 0-day

- `APDU_writeWeaver`

  Set CHALLENGE / SECRET



255

What if?

==>

Oracle 1 hit!

`APDU_writeWeaver` crashed?

S3K250AF Attack so far

- Have stack leak, but only for `APDU_readWeaver`
- `APDU_writeWeaver` triggers *Oracle 1* on $n_s > 84$
- Back to skool:
  - “Smashing the stack for fun and profit” (Aleph One,1996)

- Next move, **alternative 1**:
  - `secret[84:88]` assumed code pointer?
  - Brute force => hit ROP gadget w/ *no* Oracle 1 trigger

- Next move, **alternative 2**:
  - Assume stack `APDU_readWeaver` **~=** `APDU_writeWeaver`
  - Manual stack guesstimating

Alternative 2: stack guesstimating

- Partial S3K250AF FW found on Galaxy S20 filesystem
  - Most of FW is encrypted :(
- Contains unencrypted “dev” version of IWEA code
- IWEA is short for IWEAVER
  - `APDU_readWeaver_dev` disassembly possible
  - `APDU_writeWeaver_dev` disassembly possible

- We can “simulate” stack use, and hope it fits “prod” code on chip
  - <trial and error>

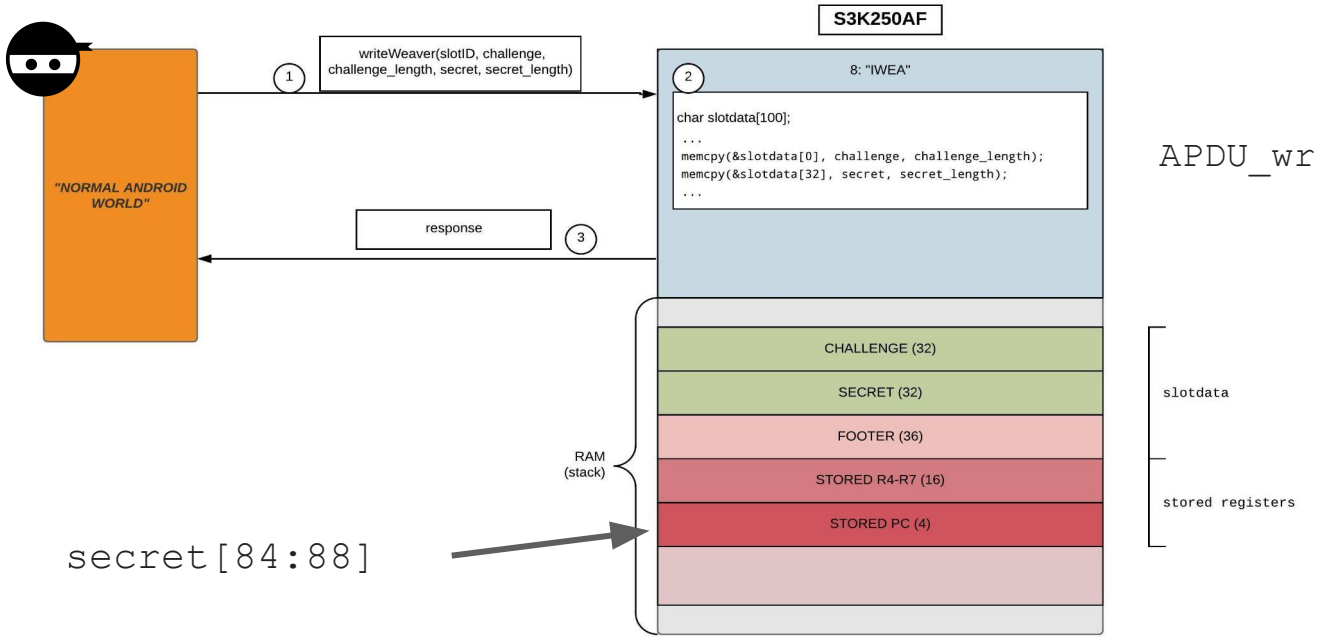
Stack layout found

Victory!

- Stack layout of `APDU_writeWeaver` guessed!
- Know position of return address (PC) `POP`'ed from stack!
- We can set R4-R7 and PC to return properly!

- Can now overflow stack and control execution on S3K250AF eSE!
- Pwned!

APDU_writeWeaver **Stack smash!**



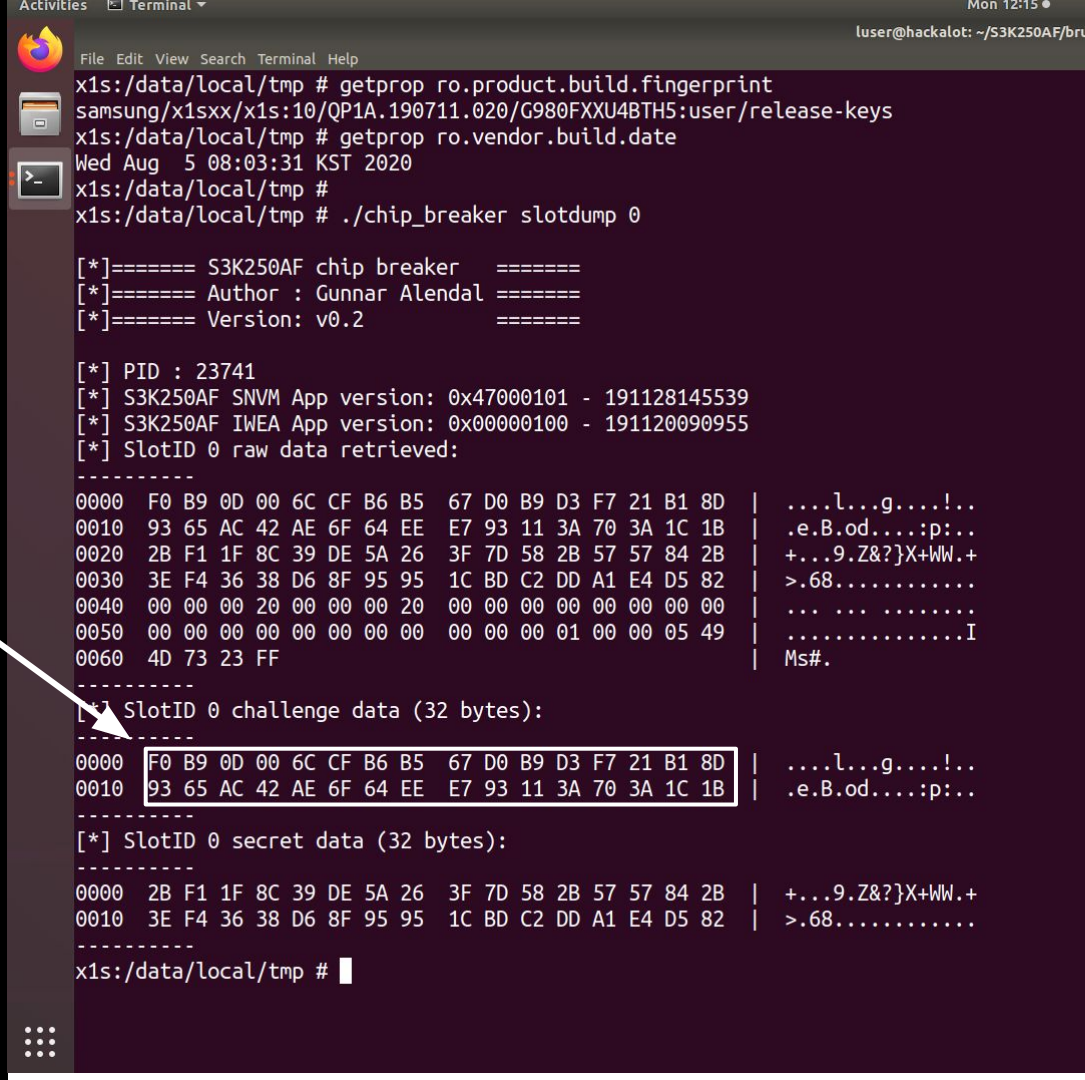
Next goal: Execute something useful

- One ROP to rule them all
  - Dumps 16 bytes from arbitrary address

```
MOVS    R0, #0x10   ; size to read. Fixed size 0x10.
STR     R7, [R4]    ; R7 is address to read => We control R7!
STR     R0, [R4,#4] ; Store size
MOVS    R0, #0x90   ; SW1 => SW is just return code (Status Word). 0x90 == "Success"
STRB    R0, [R4,#8] ; Store SW1
MOV     R0, R5      ; SW2
STRB    R5, [R4,#9] ; Store SW2
POP     {R1-R7,PC}  ; pop and return =>We get 0x10 bytes from arbitrary address!
```

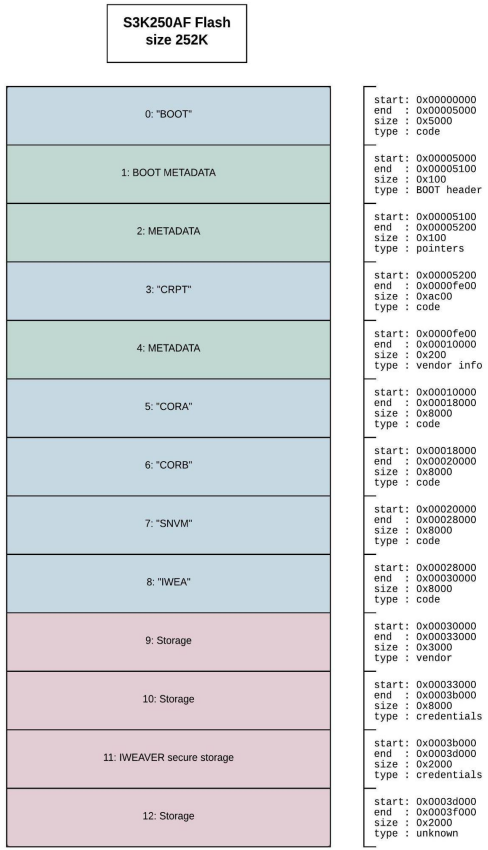
chip_breaker

- Dump CHALLENGE



Full eSE flash dump

- We dump all code + metadata
- We dump all sensitive data
  - “11: IWEAVER”:
    CHALLENGE + SECRET

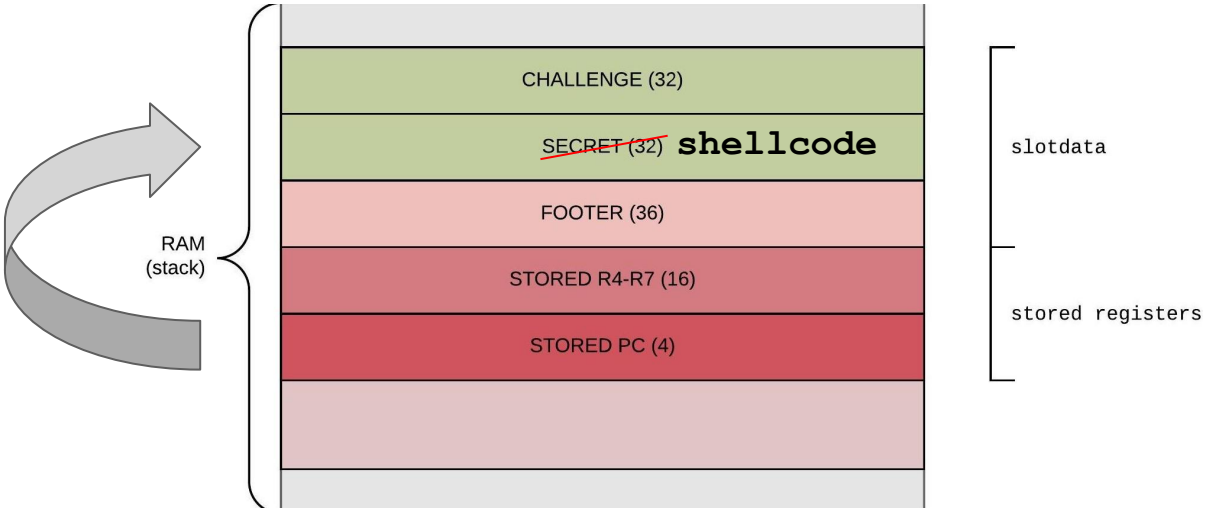


- Off-device brute force: **Check**!
- Digital Forensic Acquisition: **Check!**


Mission
accomplished!

But wait! Can we do more?

- We can achieve *arbitrary code execution (ACE)*
  - RAM/Stack is executable!
  - Return-to-APDU-buffer => ACE / (RCE)



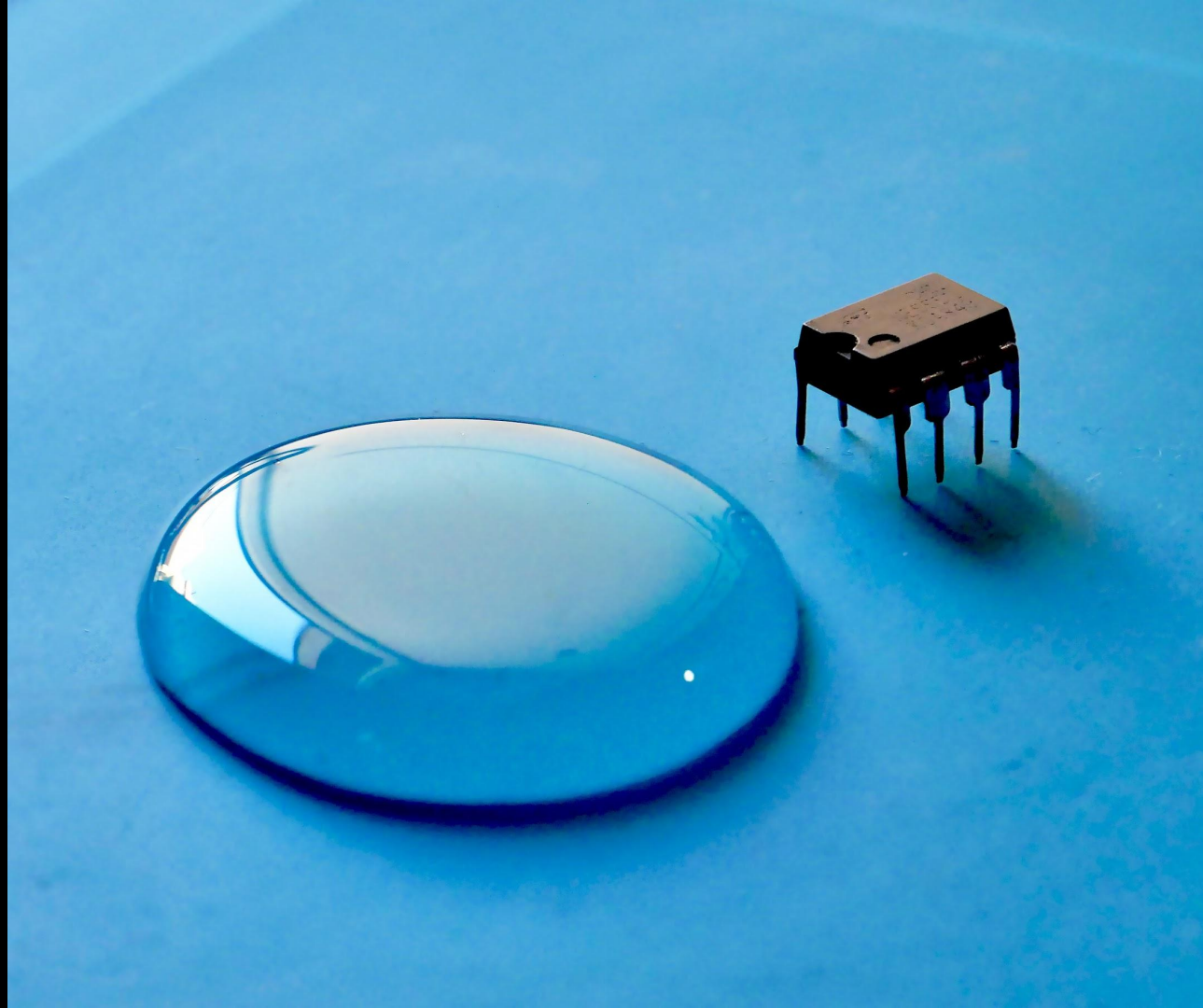
Arbitrary code execution

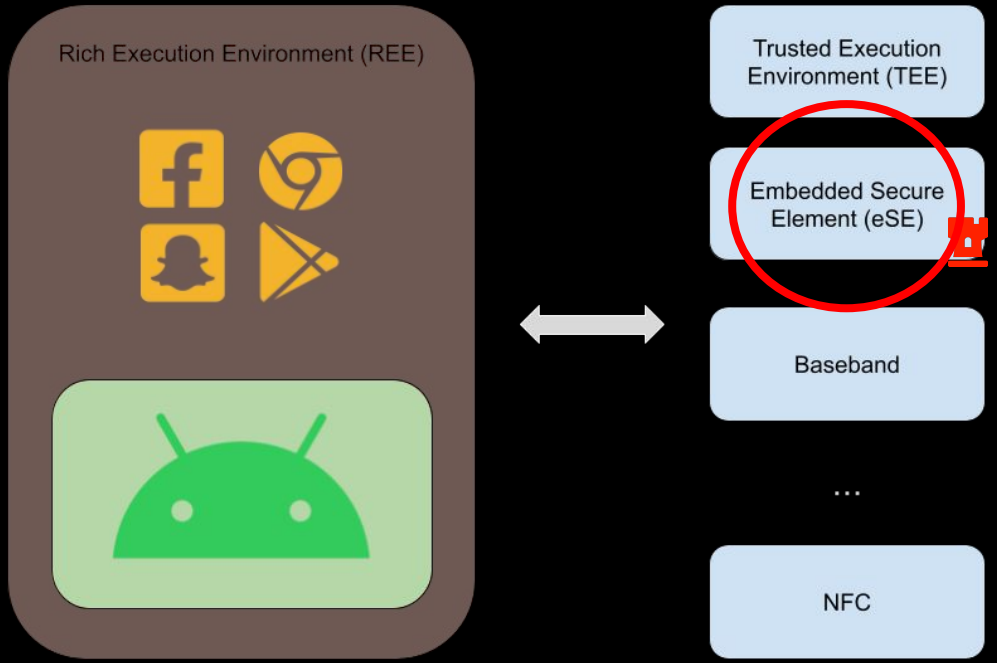
- We can **read** flash + RAM
  - Dump hardcoded AES key => Used for FW encryption
  - No more encrypted FW updates
  - No FW code or sensitive data safe

- We can **write** flash + RAM
  - No eSE Secure Boot!
  - Persistent(!) changes to any eSE feature
  - Set up **C** build env.
    - “Breaking Samsung firmware, or turning your S8/S9/S10 into a DIY Proxmark” - Christopher Wade

Write persistent changes => New attack variant?

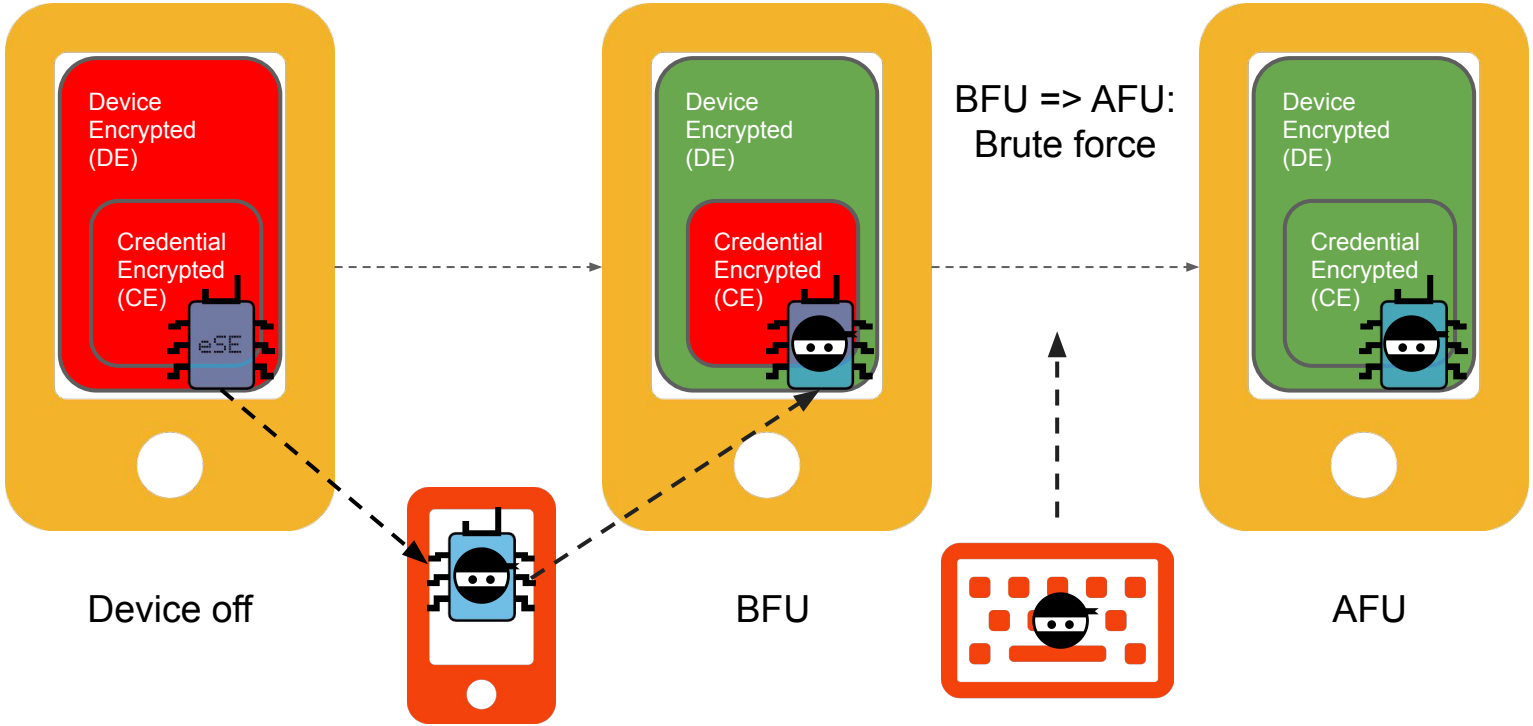
- eSE only attack
- Remove “root” REE requirement





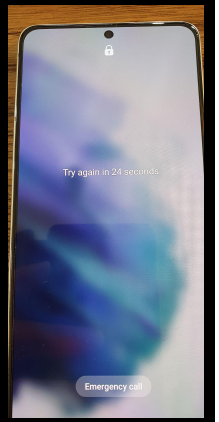
Towers preventing DFA >= Galaxy S20

Potential “HW Trojan” attack

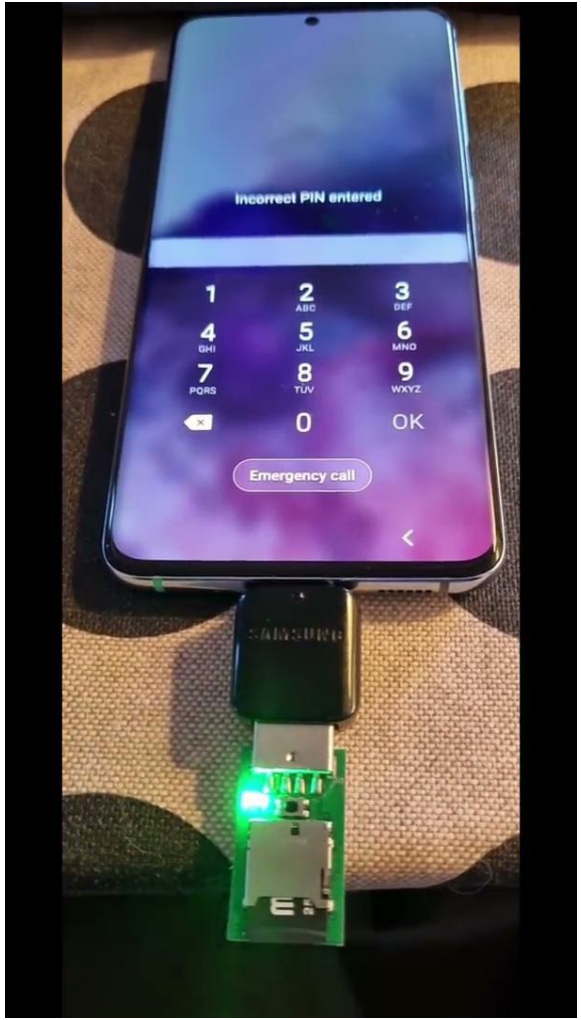


“HW Trojan” attack PoC demo

- Rubber Ducky HID simulation
- Send all PINs
- No timeouts!

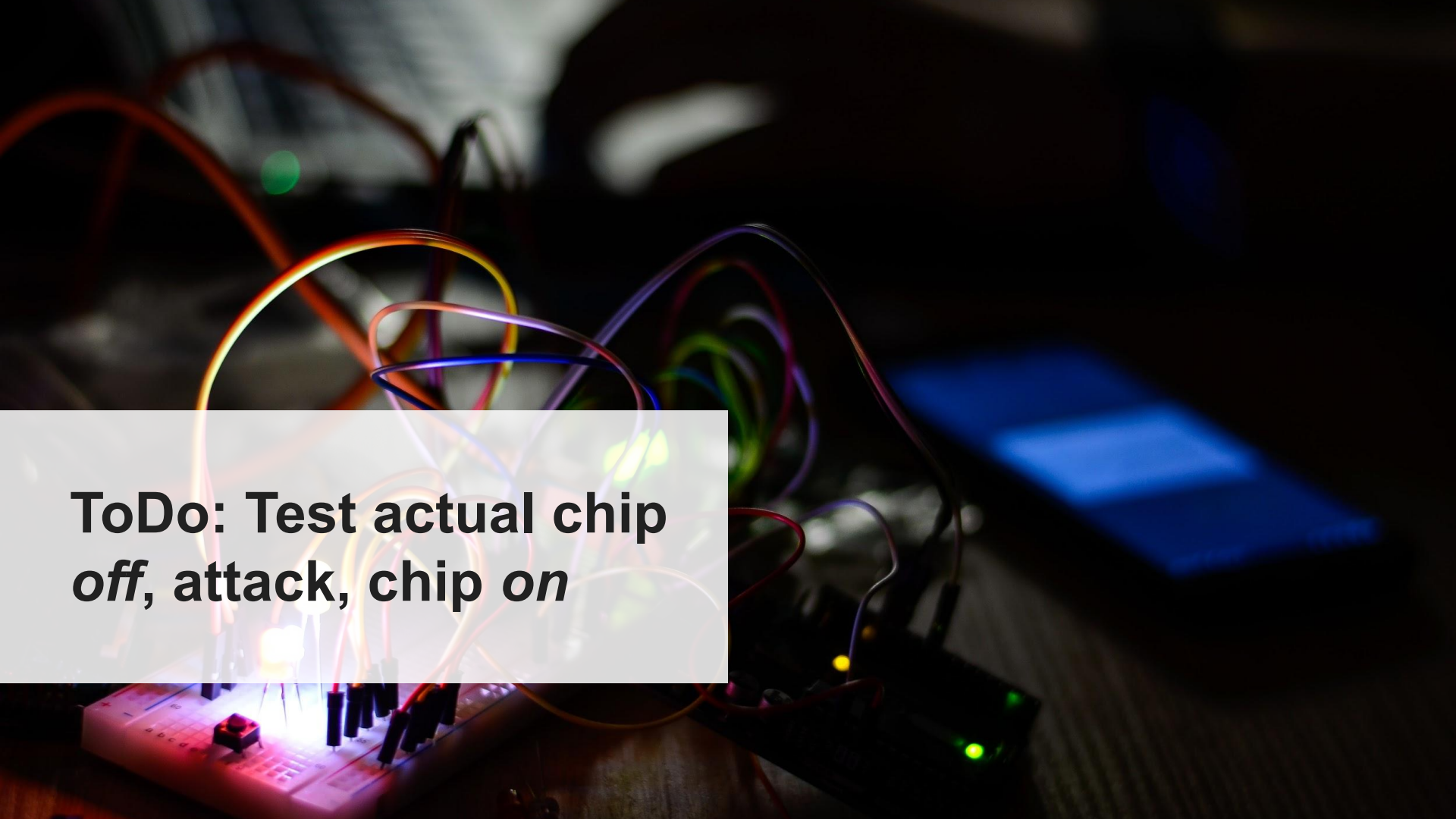


Unpatched



Patched:
eSE brute force protection removed

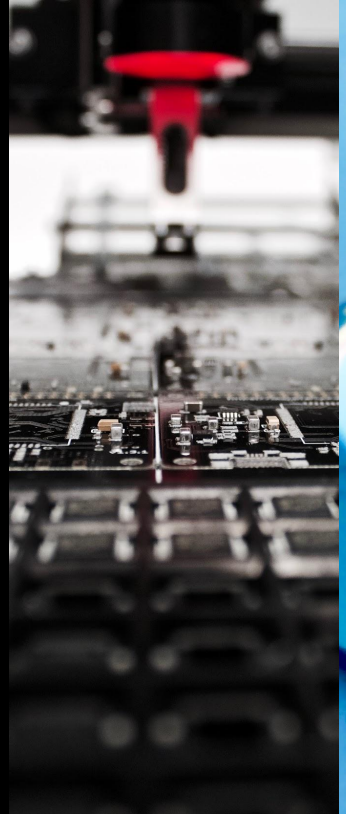
Music: @dubmood

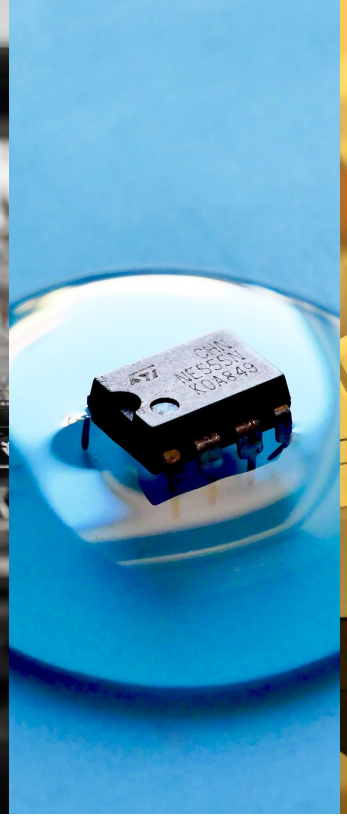

ToDo: Test actual chip
off, attack, chip *on*

Certification

⇓

Security?





"In theory, there is no difference between theory and practice, while in practice, there is"

- Benjamin Brewster

CC EAL 5+ AVA_VAN.5

- Security Goals in “Security Target”:
  - SG1 => Integrity of user data
  - SG2 => Confidentiality of user data
  - SG3 => Correct operation

Broken by our attack

- AVA_VAN.5:
  - “A methodical vulnerability analysis is performed by the evaluator to ascertain the presence of potential vulnerabilities”
  - A certified stack smashing buffer overflow?

Intended vs. *achieved* security

- S3K250AF meant to protect against state level actors
  - Broken by 1 researcher, no special tools, ~1 month
- FW encryption AES key revealed
  - No encrypted OTA possible for fielded devices
- Can fielded S3K250AF devices regain trust?
  - Can we create *undetectable / unremovable* eSE FW modifications?

Black Hat Sound Bytes

- **One** old skool stack buffer overflow to break the S3K250AF eSE
  - Patched by Samsung (CVE-2020-28341 / SVE-2020-18632)

- CC EAL 5+ AVA_VAN.5 gives no guarantees of *achieved* security

- Digital Forensic Acquisition in 2021: Finding and exploiting 0-days

NTNU
Norwegian University of Science and Technology

Thank you

(see full paper for details)

Gunnar Alendal
@gradoisageek

Thanks:
Geir Olav Dyrkolbotn, Stefan Axelsson, @zutle, @dubmood (music) and Samsung