black hat EUROPE 2021

2021: A Titan M Odyssey

Damiano Melotti
Maxime Rossi Bellom
Philippe Teuwen

Quarkslab

Bringing Security in System Design

3 ways to improve security through specialized hardware:

- Virtual Processor (ARM TrustZone)
- On-chip Processor (Apple SEP)
- External security chip (**Google Titan M**)

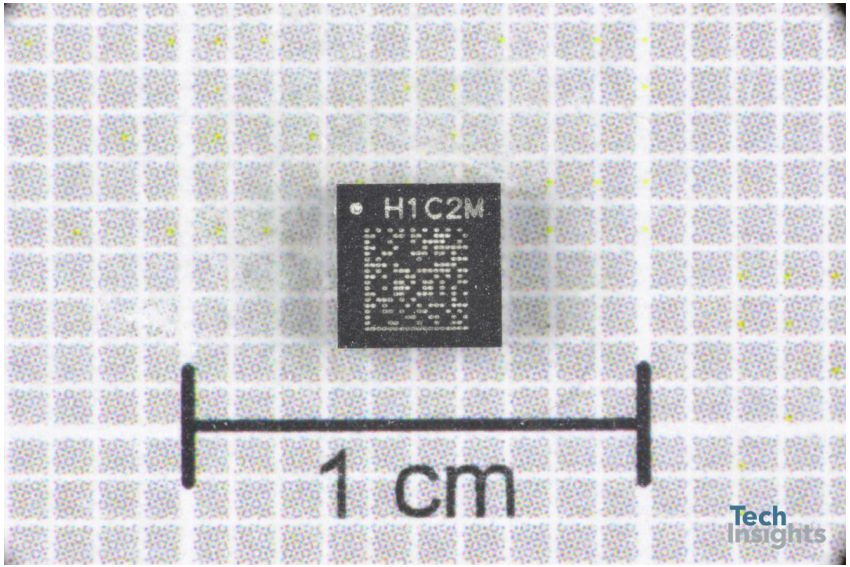
What is Titan M?

Source: https://www.ifixit.com/Guide/Google+Pixel+3+Battery+Replacement/124702

What is Titan M?

- Security chip made by Google,

  for Pixel devices
- Implements critical security features
  - StrongBox
  - AVB (Android Verified Boot)
  - Weaver
  - ...



Research Status and Goals

Lack of publicly available knowledge

- Closed source, the vendor claimed intention to publish the sources, but never did
- No existing research/presentation/blogpost
- Only one CVE write-up (CVE-2019-9465)

➜ Understand internals, extract hidden information and find vulnerabilities

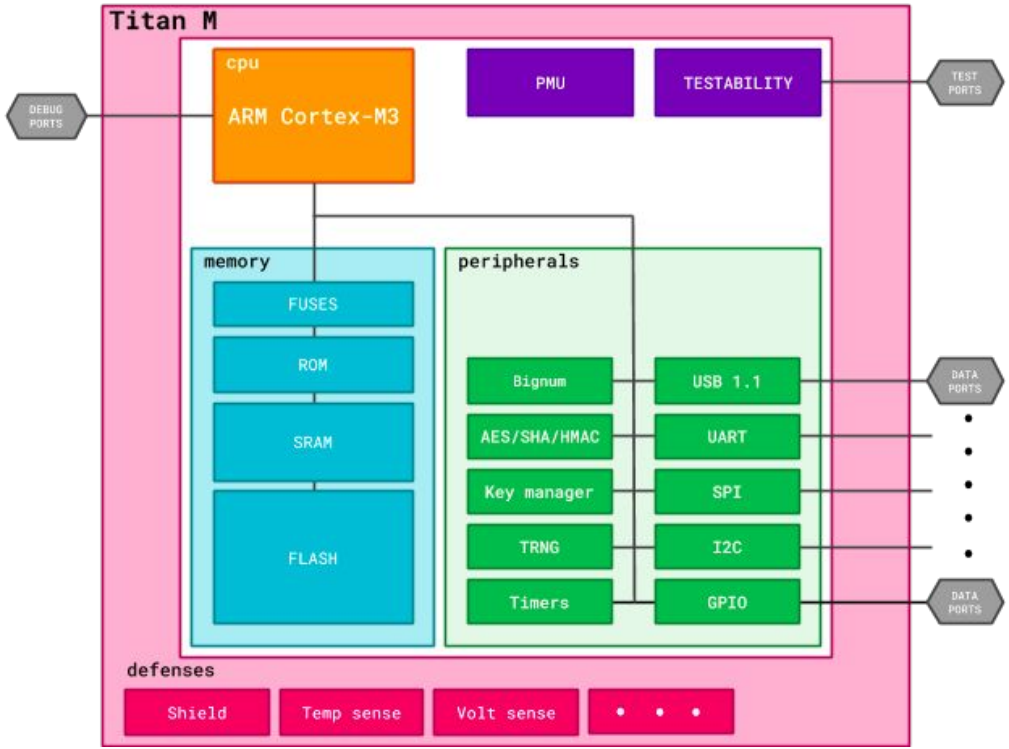
Architecture and Internals

Specification

Hardened SoC based on ARM Cortex-M3

- Anti-tampering defenses
- Cryptographic accelerators &

  True Random Number Generator
- UART for logs and console
- SPI to communicate with Android

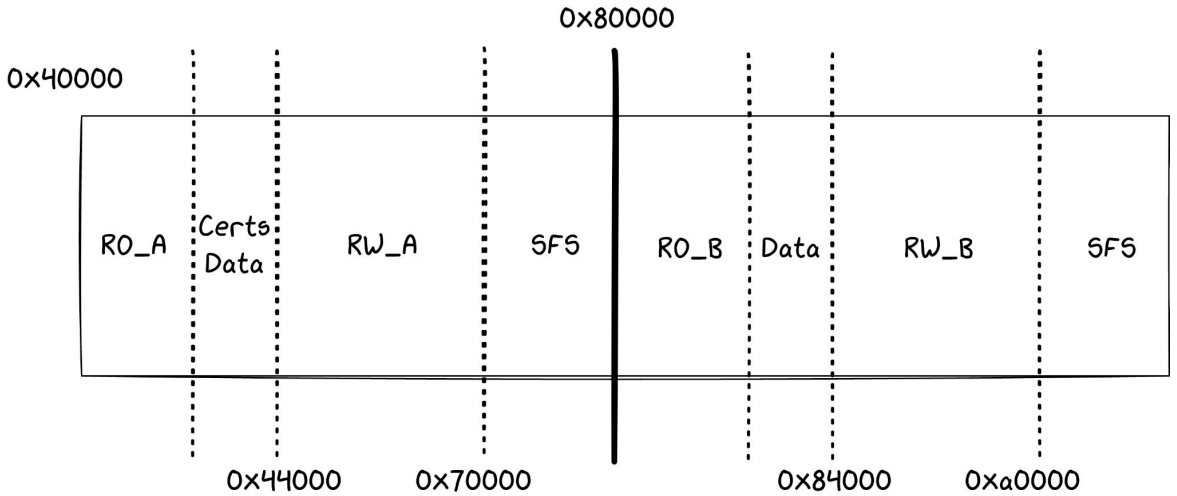


Source: https://android-developers.googleblog.com/2018/10/building-titan-better-security-through.html

Firmware

Present in the Pixel file system

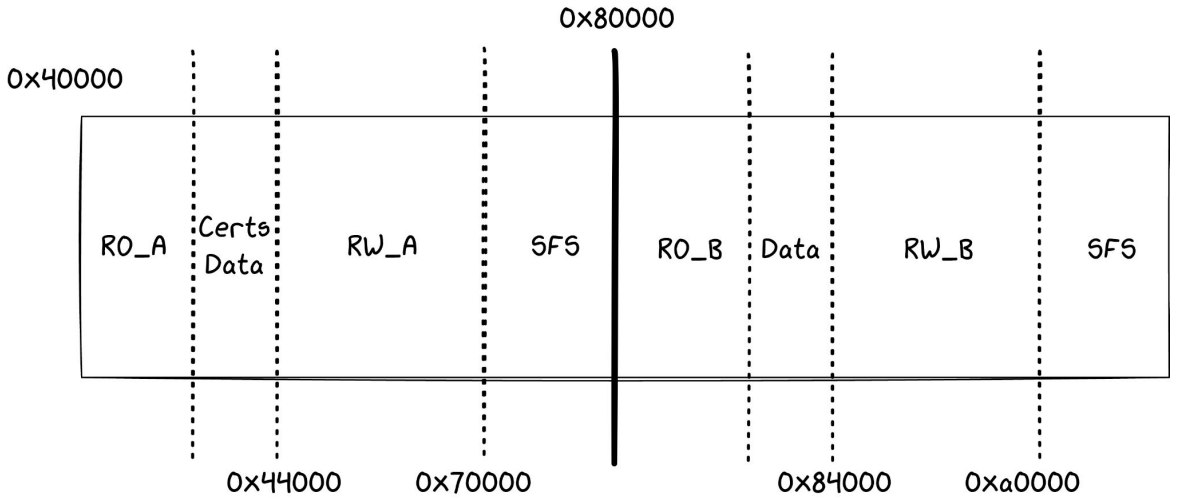
- `/vendor/firmware/citadel/ec.bin`
- No encryption, no obfuscation
- Debug strings



Firmware

A/B update mechanism

RO section is the loader, RW the main OS

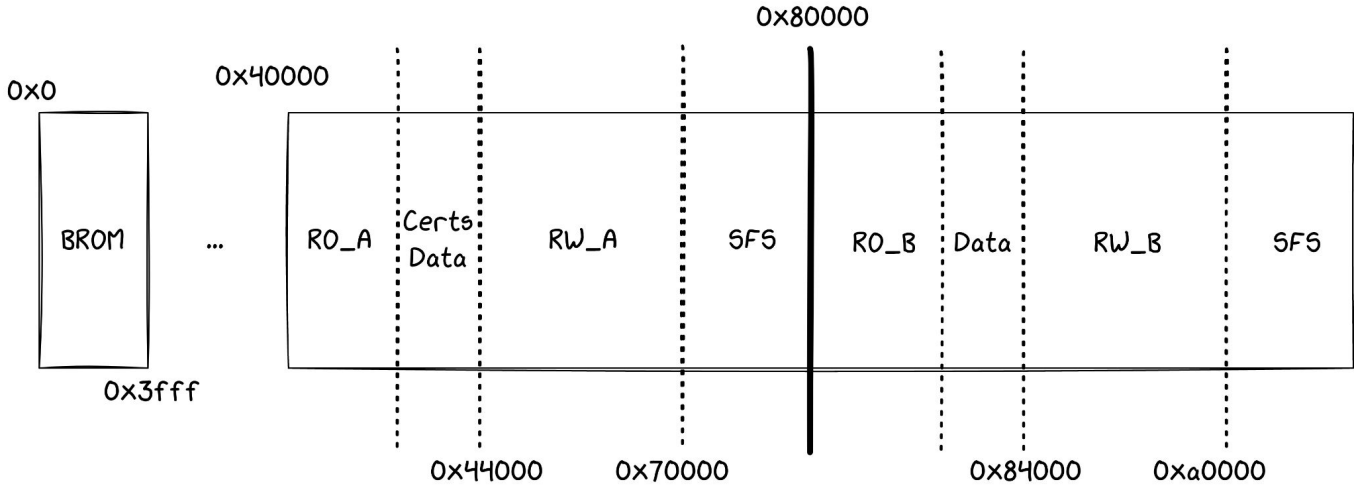


Memory Layout

Boot ROM mapped at address 0

Dedicated flash regions for persistent data

Memory mapped registers



Titan M Operating System

EC: Embedded Controller

- Open Source OS developed by Google
- Written in C

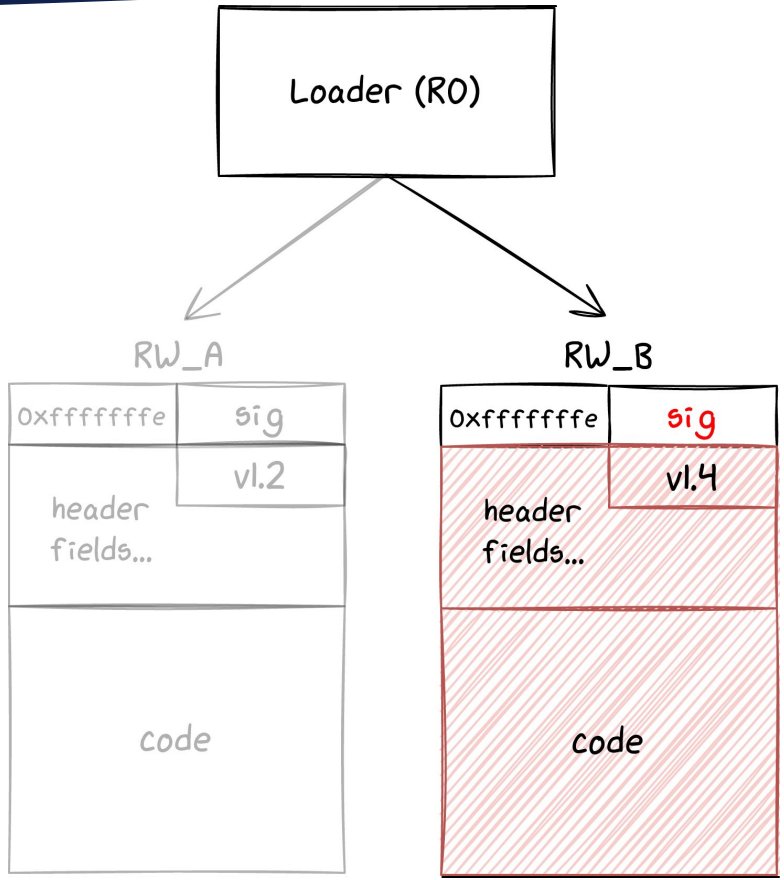
Conceptually simple

- No dynamic allocation
- Based on *tasks* with pre-allocated stack
- Driven by interrupts

EC Tasks

`idle` hook	➜ system events, timers
`nugget`	➜ system control task
`AVB`	➜ secure boot management
`faceauth`	➜ biometric data
`identity`	➜ identity documents support
`keymaster`	➜ key generation and cryptographic operations
`weaver`	➜ storage of secret tokens
`console`	➜ debug terminal and logs

Firmware Boot



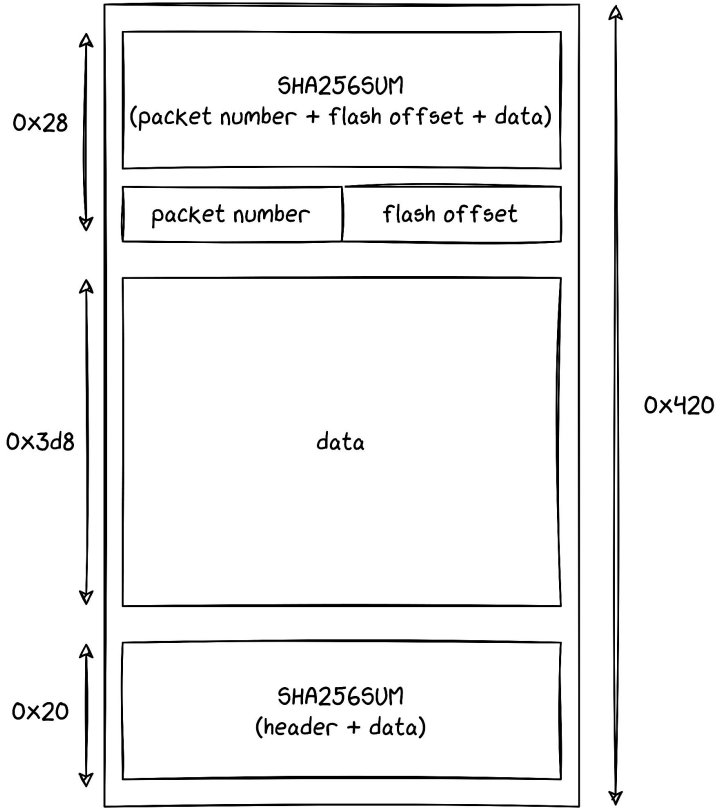
Firmware Update

Regular updated with Nugget task

- One command to write data in the flash
  - Overwrites unused RO/RW images
  - Invalidates associated magic number
- Second command to activate the new image
  - Requires a hash derived from user password
  - Changes back the magic number

Firmware Rescue

- Implemented in the Titan M loader
- Allows to flash RW_A image
- No need for user password
  - But userdata and RW_B image are erased
- Requires image to be in a specific format called .rec
- Can be triggered through fastboot



Firmware Security Measures

- Secure boot (images are signed and verified at boot)
- No MMU, but MPU to give permissions to the memory partitions
- Only software protection: hardcoded stack canary checked in the SVC handler

```
if (*CURRENT_TASK->stack != 0xdeadd00d) {
  next = (int)&CURRENT_TASK[-0x411].MPU_RASR_value >> 6;
  log("\n\nStack overflow in %s task!\n",(&TASK_NAMES)[next]);
  software_panic(0xdead6661,next);
}
```

Communication with Android

```
package nugget.app.keymaster;
// ...
service Keymaster {
  // ...
  rpc AddRngEntropy (AddRngEntropyRequest) returns (AddRngEntropyResponse);
  rpc GenerateKey (GenerateKeyRequest) returns (GenerateKeyResponse);
  // ...

message AddRngEntropyRequest {
  bytes data = 1;
}
message AddRngEntropyResponse {
  ErrorCode error_code = 1;
}

message GenerateKeyRequest {
  KeyParameters params = 1;
  uint64 creation_time_ms = 2;
}
```

- Protobuf-based
  - Serialization framework by Google
  - Language agnostic
  - Titan M uses the nanopb library
  - Limited risk of input validation bugs
- Protobuf definitions are part of the AOSP

StrongBox

- `StrongBox`: hardware-backed version of `Keystore`
  - The highest security level for keys
  - Generate, use and encrypt cryptographic material
- Titan M does not store keys
  - Key blobs encrypted with a Key Encryption Key
  - Sent to the chip to perform crypto operations
  - `root` can *use* any key, but not *extract* it

StrongBox and Root of Trust

- StrongBox builds the KEK with several components. Among them:
  - Root of Trust: SHA256 digest sent once by the bootloader
  - Salt: generated from random when a new RoT is provided
- Stored in a memory area called SFS

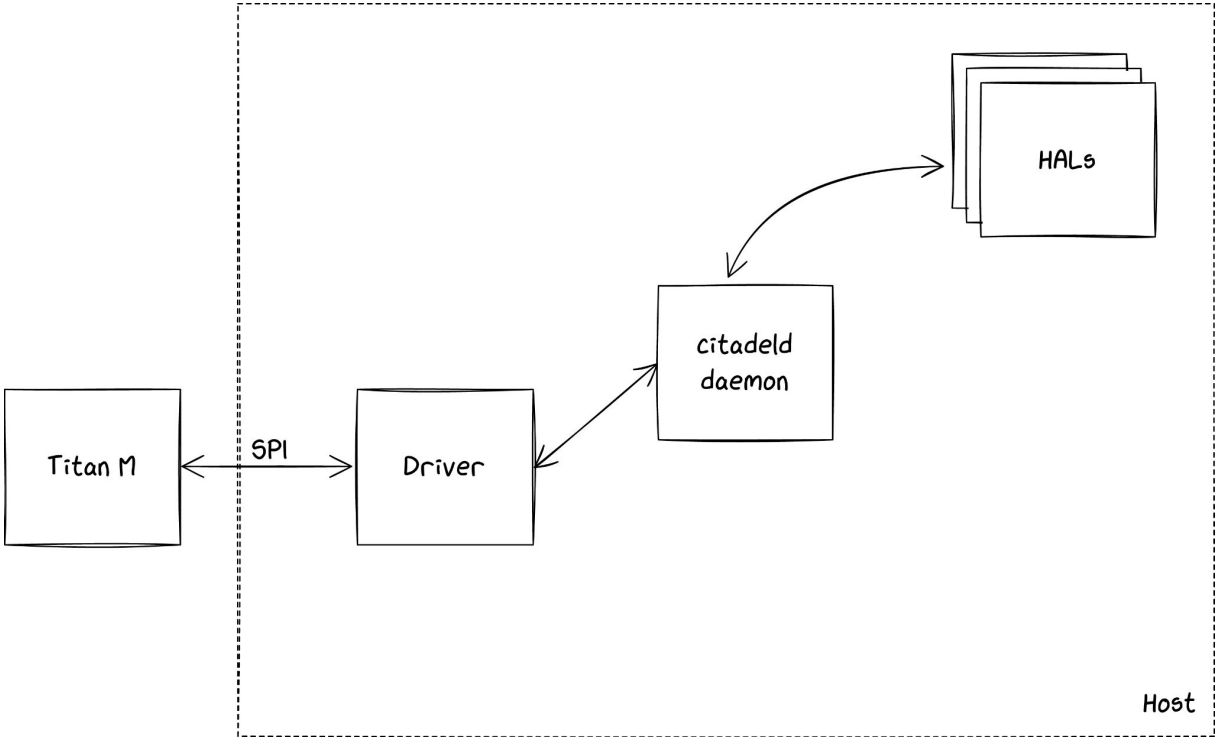
Tools

Static Analysis: Ghidra Loader

We implemented a loader to help static reversing

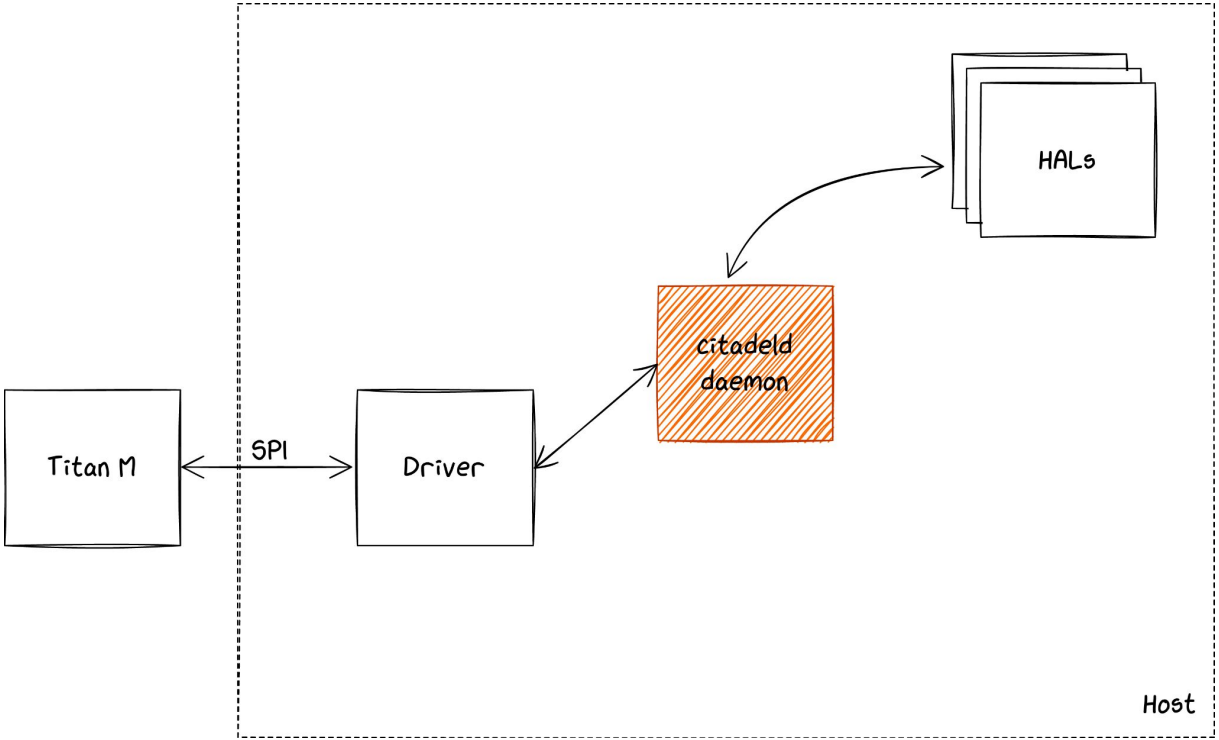
- Loading images to the right addresses
- Creating memory regions (registers, ram, etc)

Dynamic Analysis: Sniffing Communication



Where to hook?

Dynamic Analysis: Sniffing Communication



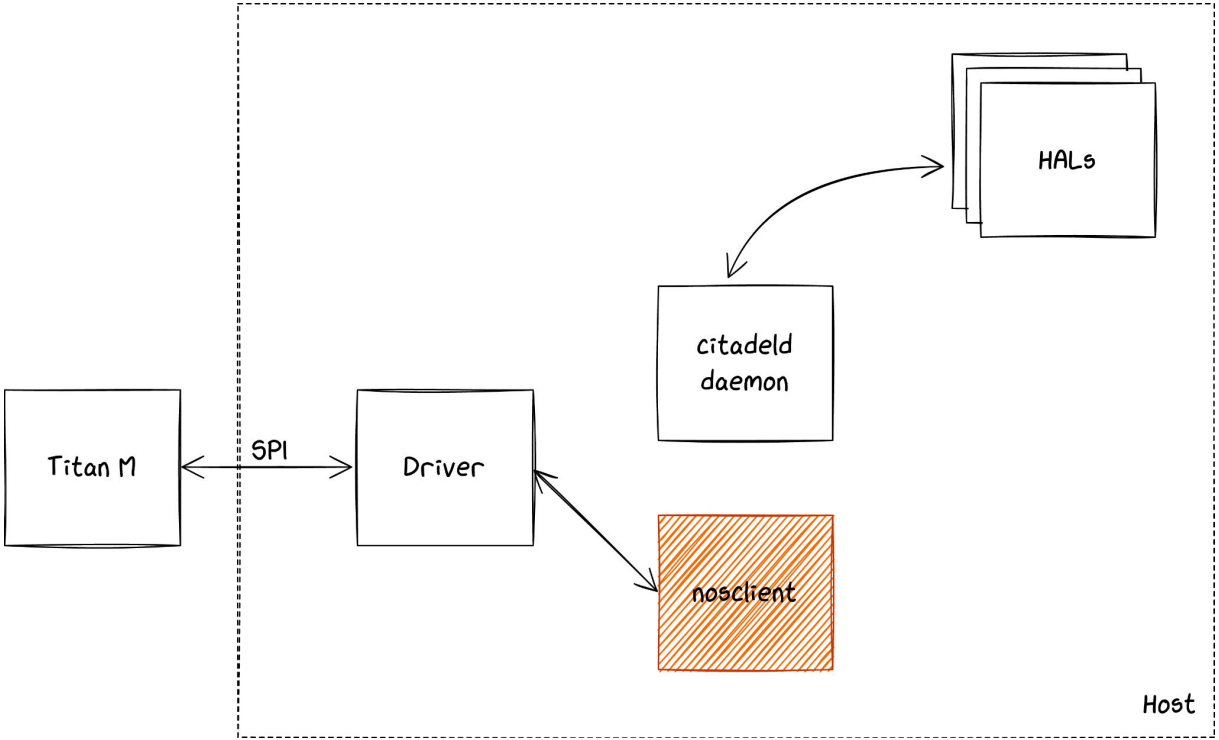
Using Frida,
hook the **citadeld** daemon
(*nos_call_application*)

Sniffing Communication: Command Parsing

```
qb_parser: appID: 0x2, param: 0x0
qb_parser: request: 0x731ee559f0, request_size: 0x12
qb_parser: reply: 0x741ee64070, reply_size_addr: 0x7fe922ea64
qb_parser: Request:
qb_parser:            0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
qb_parser: 00000000  0a 10 7a e3 18 0e 27 42 18 d6 89 58 65 c9 58 e0  ..z...'B...Xe.X.
qb_parser: 00000010  00 f4                                           ..
qb_parser: Reply size:
qb_parser:            0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
qb_parser: 00000000  00 00 00 00                                     ....
qb_parser: Reply size is 0
qb_parser: AddRngEntropyRequest
```

One of the steps of key generation, sniffed with Frida

Dynamic Analysis: Sending Commands



Implementing a client bypassing *citadeld*: **nosclient**

Communicates directly with the driver

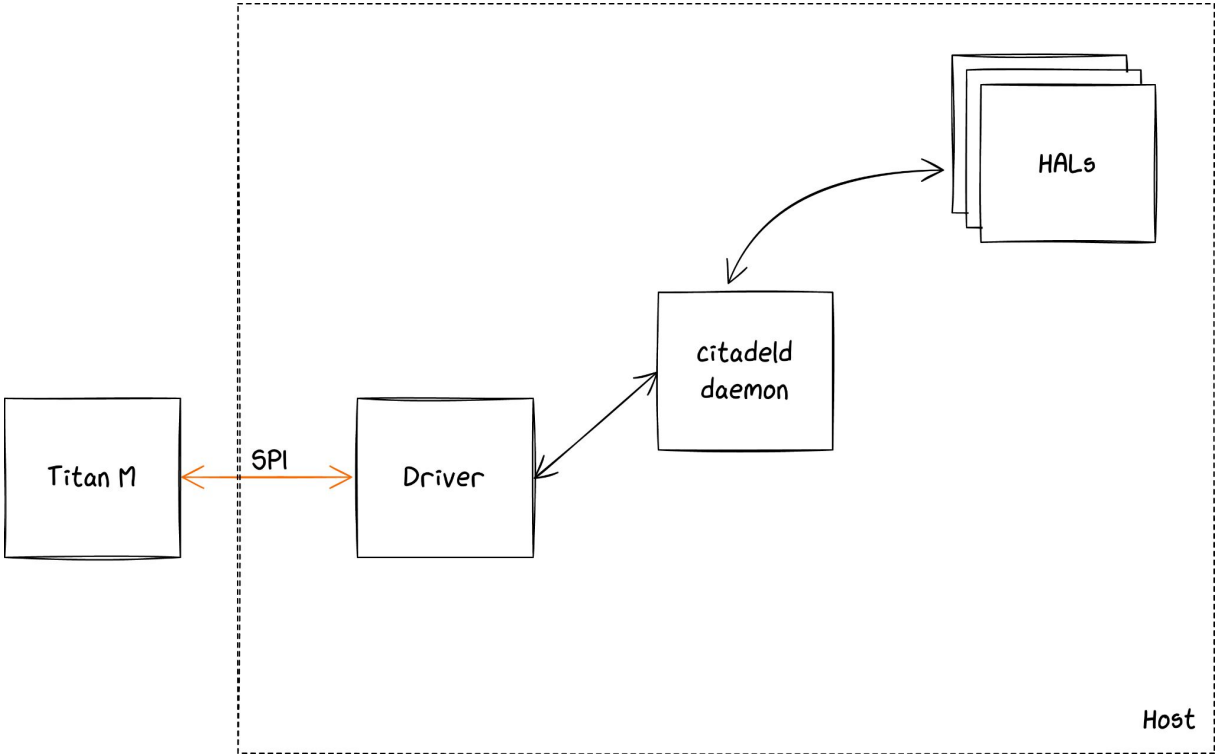
Dynamic Analysis: Sending Custom Commands

Our client, **nosclient** leverages protobuf definitions

- To generate command data
- To display the result sent by the chip

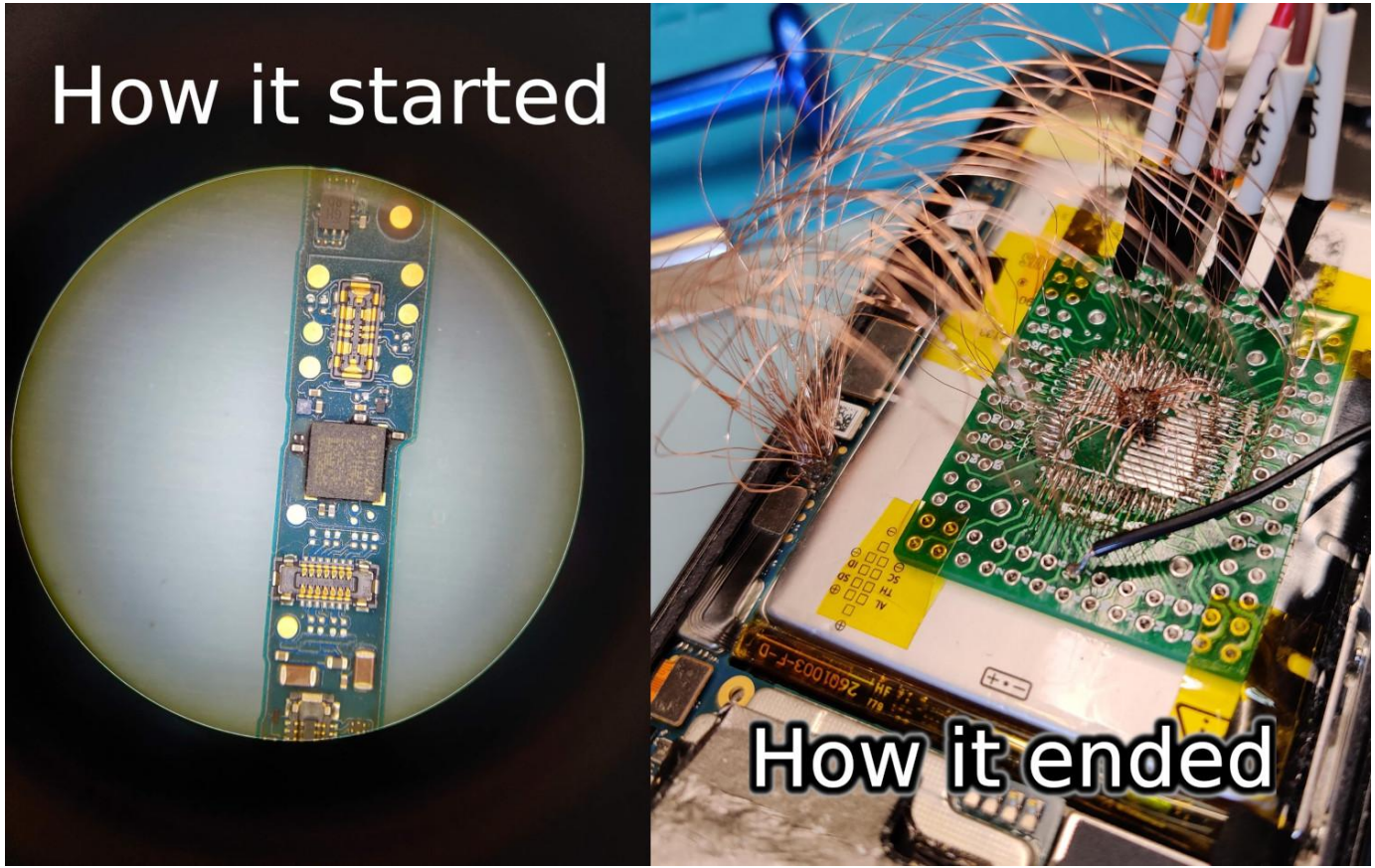
```
# ./nosclient Keymaster GetBootInfo
is_unlocked: true
boot_color: BOOT_UNVERIFIED_ORANGE
```

Dynamic Analysis: Sniffing Communication



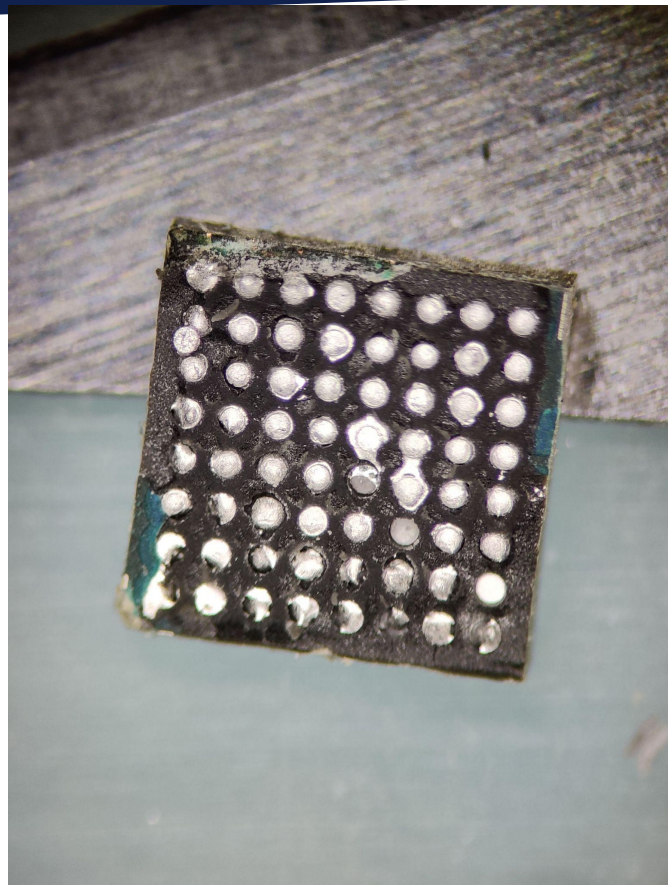
Physically sniffing on the SPI bus

Hardware Reverse: Finding SPI



Hardware Reverse: Guessing Pinout

	1	2	3	4	5	6	7	8	
A				Vcc	Button6 Vol Up	P8 N-reset2 ? Output ?	P9 UART RX	h cpu N-reset ? Input ?	(1)
B		USB CC1	Button1 Vol Down	h	h	h	h	P10 UART TX	(2)
C		USB CC2	GND	h	h	GND	ButtonSide	h	(3)
D	Vcc	*output regul?*	*active ?*	GND	GND		MISO	P3 Power On Button	(4)
E	H → self?	h	*VCO output ?*	GND	GND		CS	P7	(5)
F	GNDed pin	h	GND				SCK	MOSI	(6)
G	h		C						(7)
H	Vcc		P7			h			(8)
	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)	



Hardware Reverse: Tracing SPI

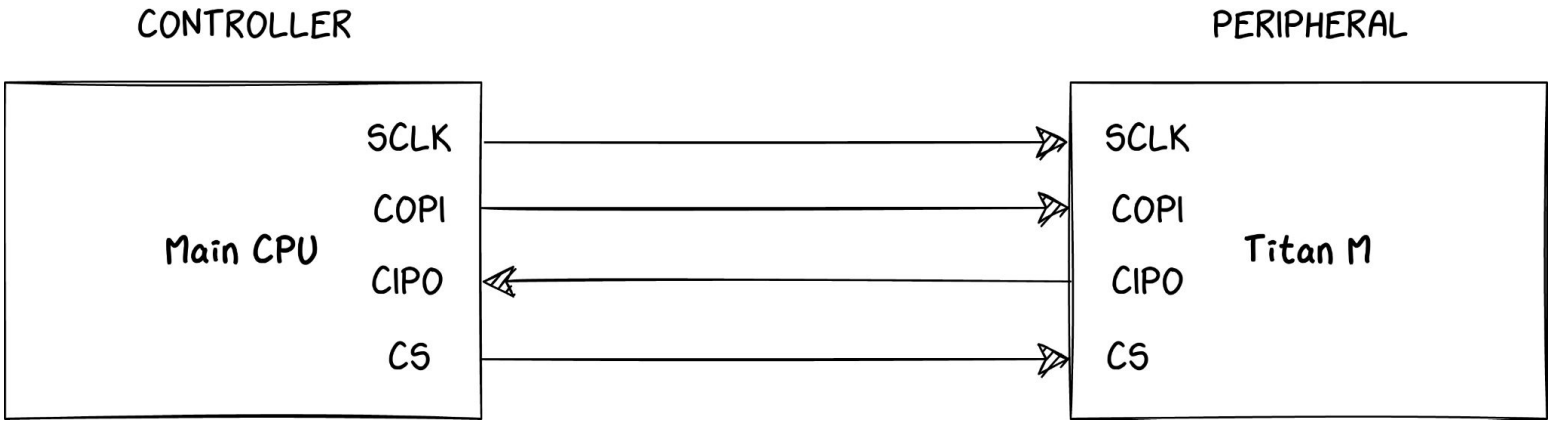
```
$ LD_PRELOAD=./libparser.so python parse_sigrok-csv.py reboot_after_spi_rescue.csv
...
AVB: GetLock
{
    IN { lock: BOOT }
    OUT {}
}
Keymaster: SetRootOfTrust
{
    IN { digest: "4bf5122f344554c53bde2ebb8cd2b7e3d1600ad631c385a5d7cce23c7785459a" }
    OUT {}
}
Keymaster: SetBootState
{
    IN
    {
        is_unlocked: true
        public_key: "0000000000000000000000000000000000000000000000000000000000000000"
        color: BOOT_UNVERIFIED_ORANGE
        system_version: 163840
        system_security_level: 10568
        boot_hash: "00dfccb48f331975a1390d5133ce5321e65123bc1f1f76b6ffb9deb61f5d6be8"
    }
    OUT {}
}
...
```

Taking Control of SPI

Now, how to send commands?

Taking Control of SPI

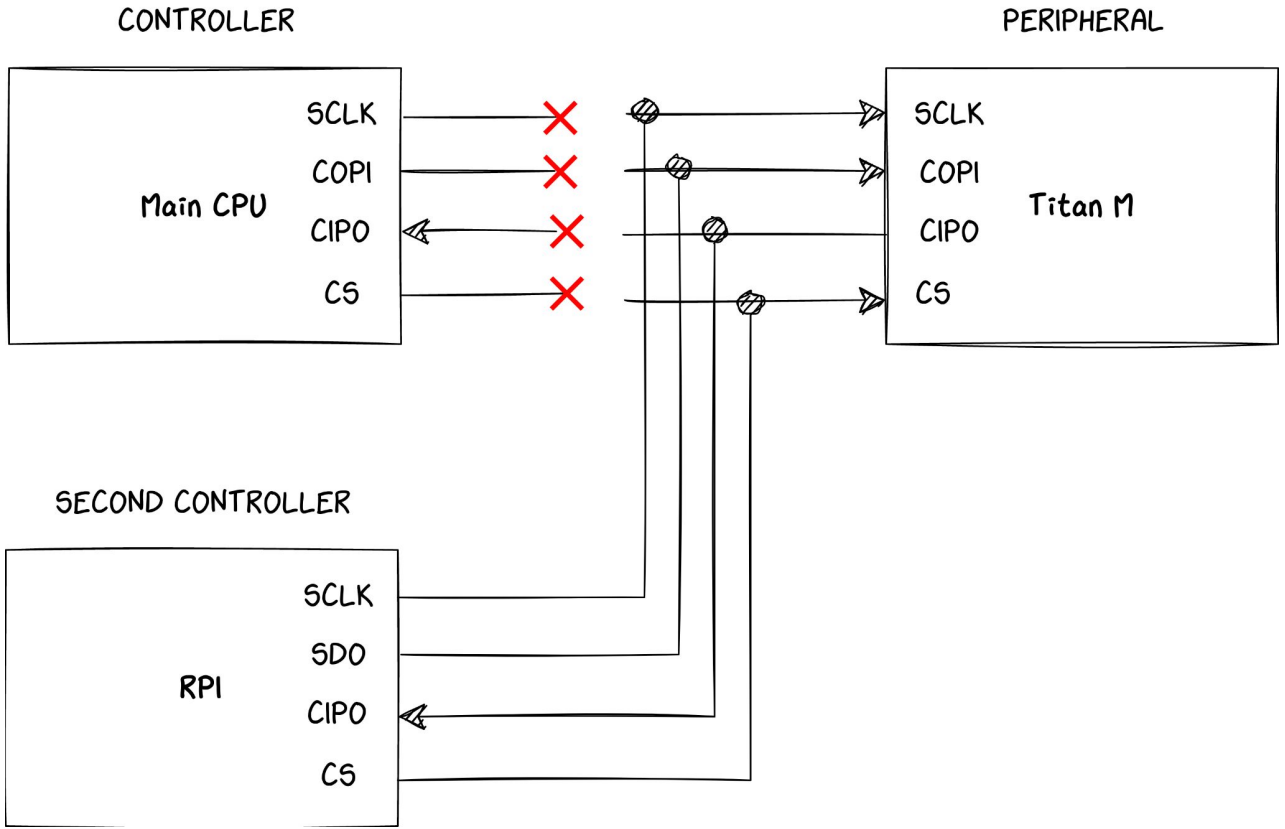
Now how to send commands?



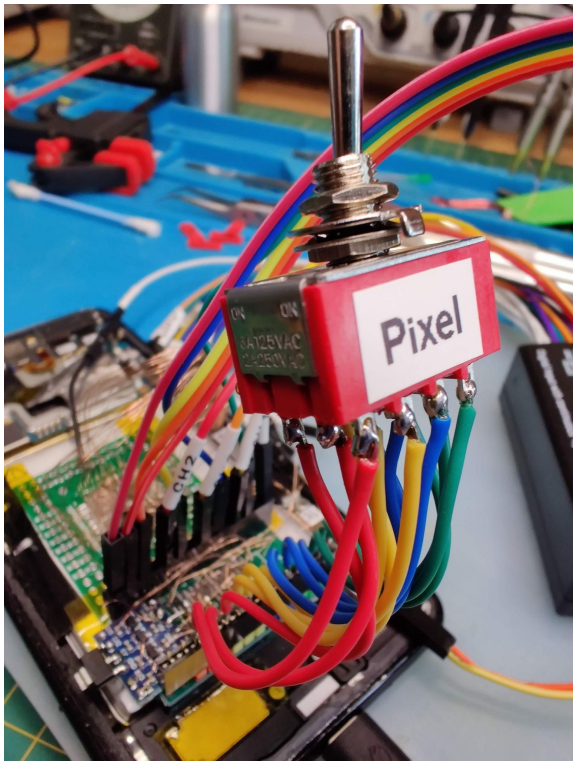
SPI is not multi-controller

➜ Need to multiplex the bus for a second controller

Taking Control of SPI



Taking Control of SPI



Manual switch to choose between SPI controller:

- Phone Application Processor
- Raspberry PI

➜ Now we can send commands to Titan M

even when the main CPU is in bootloader mode

Vulnerabilities and Exploits

First 0-day: Out of Bounds Read

```
void nugget_ap_uart_passthru(uint index)
{
  if (PASSTHRU != index) {
    cprint(4,"passthru %s",(&string_array)[index]);
  }
```

- *index* is provided through SPI command
- Its value isn't checked
- Can only be called when AP in bootloader

```
string_array = {
      0x65c00, // -> "off"
      0x68594, // -> "usb"
      0x68598, // -> "ap"
      0x6859c, // -> "ssc"
      0x685a0, // -> "citadel"
      0x4004002c, // some hw register
      0x0,        // address 0?
      0x40040030
      …
```

Second 0-day: Downgrade Issue

Anti-downgrade mechanism seems to be implemented

… but not used

➜ Use SPI Rescue to flash any firmware version

```
$ fastboot stage <any rec file>
$ fastboot oem citadel rescue
```

➜ Can we downgrade and exploit a known vulnerability?

Looking for a Known Vulnerability

- CVE-2021-0454 or CVE-2021-0455 or CVE-2021-0456
- `Identity` task, `ICpushReaderCert` command

```
uVar1 = (uint)ic_struct;
 if (*(int *)(uVar1 + 0xbc) == 0) {
LAB_00062822:
   if (pubkey_size != 0) {
     *(uint *)(uVar1 + 0xbc) = pubkey_size;
     memcpy((void *)(uVar1 + x78),pubkey_addr,pubkey_size);
     pubkey_size = 1;
   }
 }
```

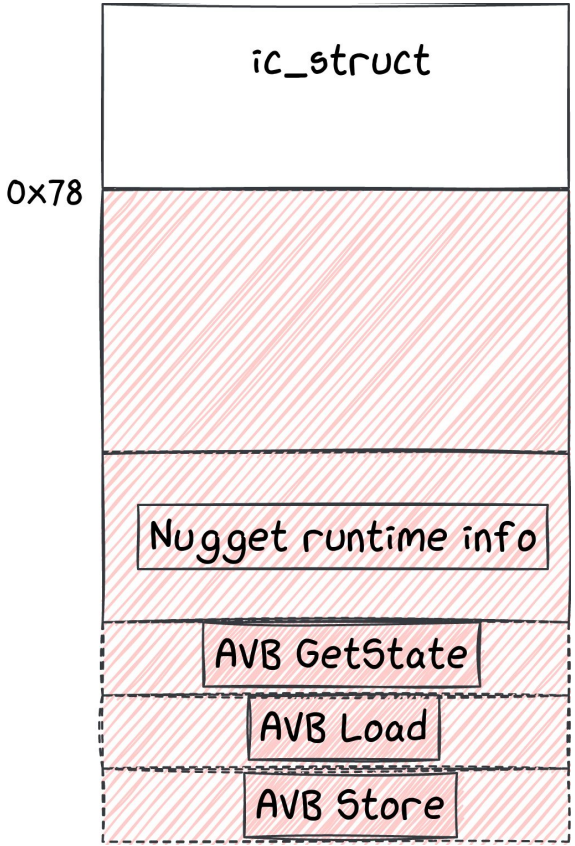
What can we do with the exploit?

Vulnerable buffer placed just before

- runtime data of the chip...
- ... and the list of command handler pointers

➜ overwrite command handler addresses

to gain code execution!



Post Exploitation

We modified our **nosclient** to exploit this vulnerability

- Could not find a way to re-configure MPU
  - Only code reuse attack possible (ROP)
- Still, we can use this vulnerability to leak data from the memory
  - Helpful for debugging
  - Allowing to dump Boot Rom
  - Allowing to leak the Root of Trust

Fuzzing for More Vulnerabilities

Fuzzing Titan M

Blackbox approach based on `libprotobuf-mutator`

- On old firmware (2020-09-25)
  - 2 known buffer overflows (including the exploited one)
  - 7 other vulnerabilities leading to device hanging or rebooting
- 2 remaining bugs on latest firmware
  - Chip crash, same underlying function performing a null pointer dereference
  - Not severe enough to be considered as vulnerabilities by Google

Remarks

- All bugs found after few seconds of fuzzing
  - No additional results afterwards
  - No coverage ⇒ only shallow states exercised
- Possible improvements
  - Analyze the actual response
  - Parse the UART log
  - Open the emulation Pandora’s box
  - Grammar aware ➜ Protocol aware

Conclusion

Conclusion

- Interesting findings about the firmware
  - Simple design, but debatable security measures
- Quite effective tooling developed to interact with the chip
  - Future work can be done also on the hardware side
- Exploited a known vulnerability and leaked the boot rom
  - First code-execution exploit known on Titan M
- Fuzzing can bring even more interesting results

Tools & resources:
https://github.com/quarkslab/titanm

Thank you!

@max_r_b
@DamianoMelotti
@doegox

contact@quarkslab.com

Quarkslab

Command Handling Example on Titan M

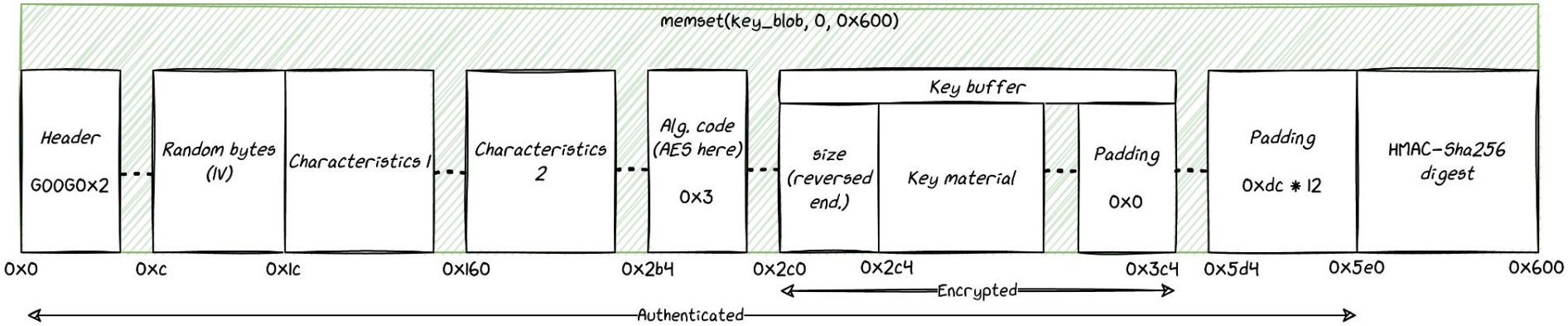
```
uint32_t keymaster_AddRngEntropy (...,
    keymaster_AddRngEntropyRequest *request, ...,
    keymaster_AddRngEntropyResponse *response) {

  // ...

  iVar1 = pb_decode_ex(param_1,param_2,request,(uint)param_4);
  if (iVar1 == 0)
    return 1;

  km_add_entropy(request,response);
  iVar1 = pb_encode(param_4,param_5,response);

  return iVar1 == 0 ? 2 : 0;
}
```

Firmware Boot

At boot, the loader (RO image)

- Chooses the most recent candidate (RW image) based on version numbers
- Checks if a magic number in the header is present, then verifies the image signature
- If something goes wrong with a candidate, the other one is chosen

Key Blob Structure



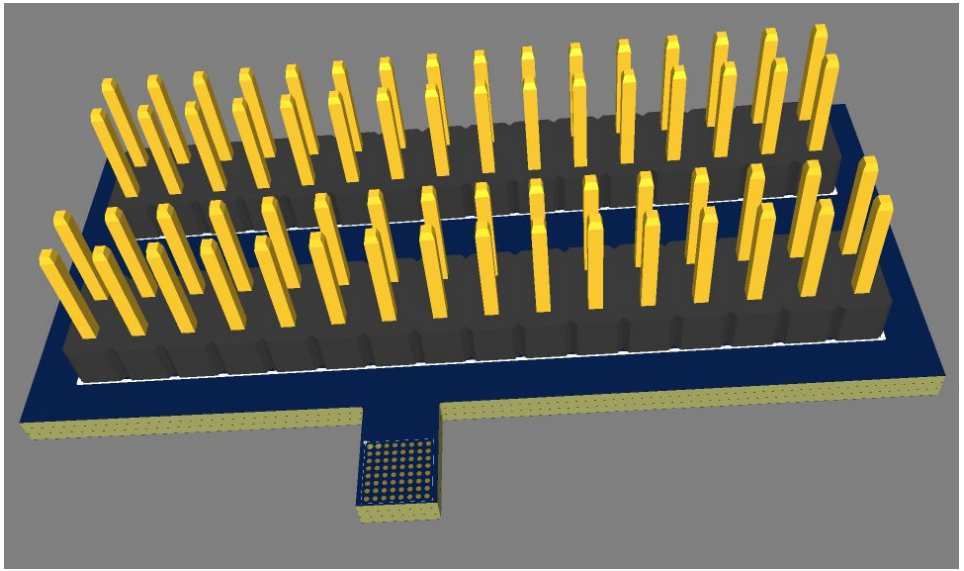
KEK: SHA256(Root of Trust || salt || req1 || req2 || flash_bytes)

HMAC KEY: SHA256(Root of Trust || salt || flash_bytes)

Hardware Reverse: Finding SPI

First attempt:

- design a flex PCB exposing all 64 pins
- flex PCB allows really small tracks
- should fit in the small space between vias



Cost: $1500 !!!

Fuzzing Titan M

- Black-box approach
  - Cannot recompile and instrument the firmware
  - Almost no useful debugging information
- Rely on return value from library call
- Mutation-based (using `libprotobuf-mutator` natively on Android)
  - Mutate messages respecting Protobuf definitions
  - Random operators to trigger typical vulnerabilities