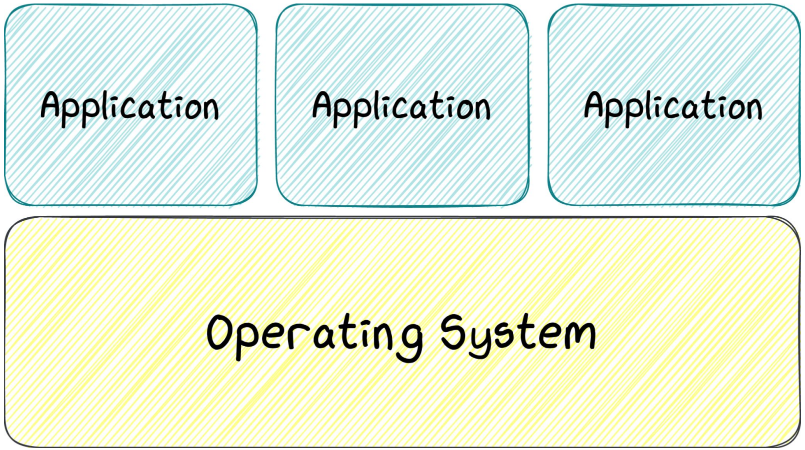
Reversing and Fuzzing the Google Titan M chip

Damiano Melotti
Maxime Rossi Bellom
Andrea Continella

UNIVERSITY OF TWENTE.
Quarkslab

Introduction - system design

Once upon a time...



- Kernel running in privileged mode
- Small code base

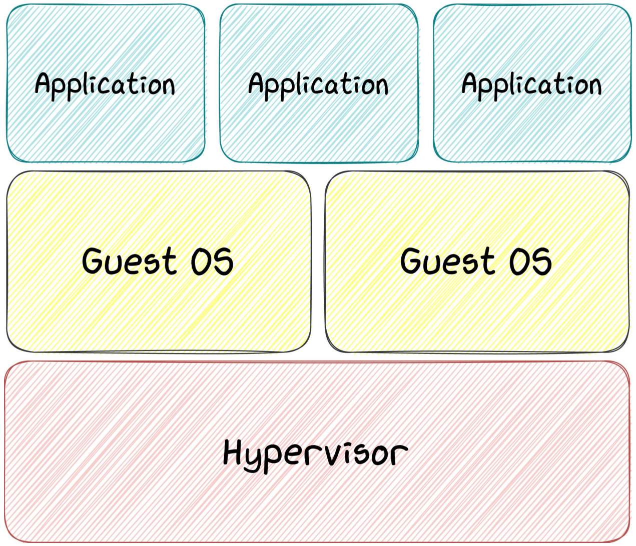
Problem: what if the kernel *could* be compromised?

Source: Maxime Peterlin, Joffrey Guilbon, and Alexandre Adamski. Breaking Samsung's ARM TrustZone.
https://www.blackhat.com/us-19/briefings/schedule/#breaking-samsungs-arm-trustzone-14932, August 2019

Introduction - system design

UNIVERSITY OF TWENTE.

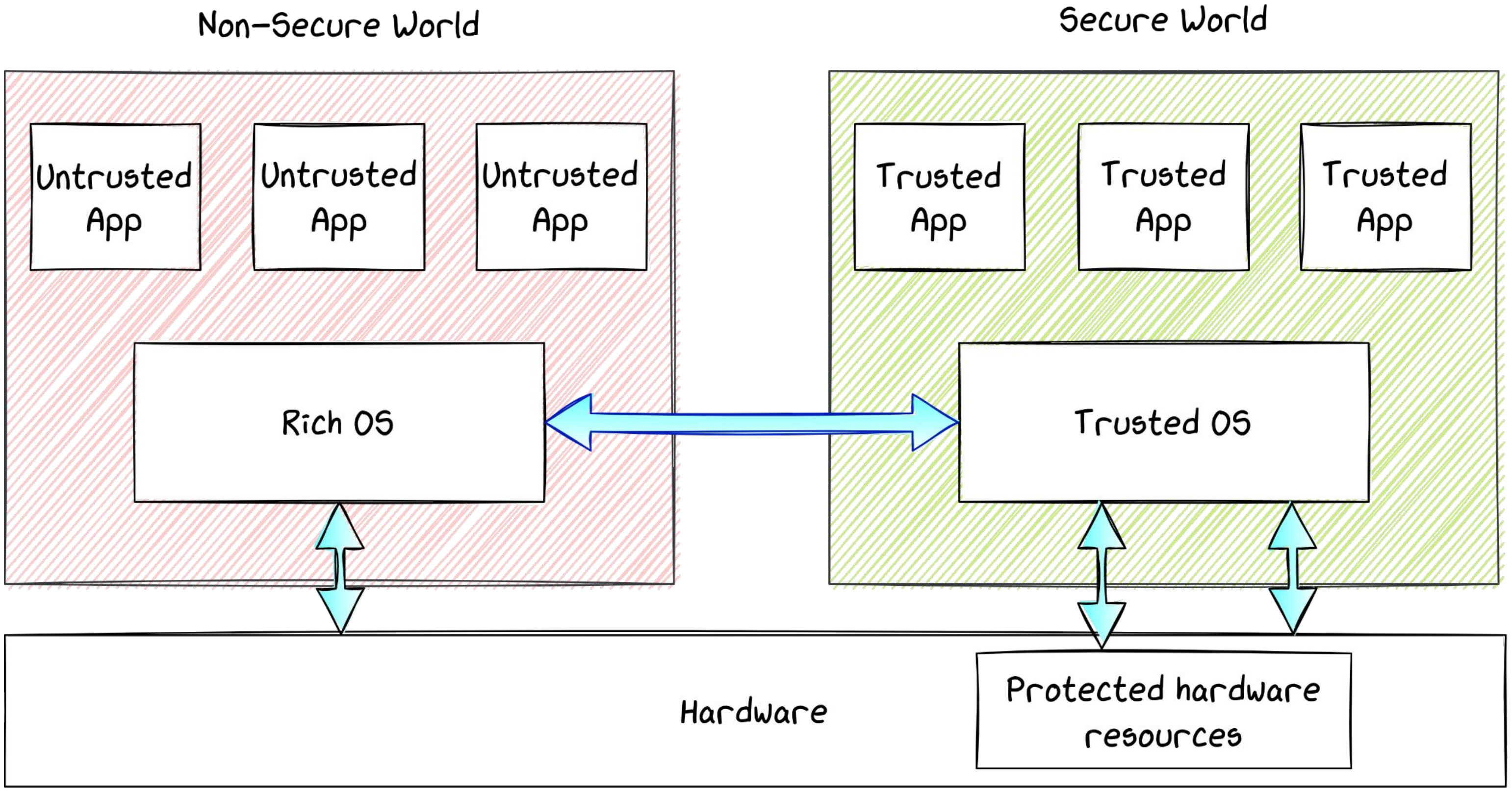
- Hypervisors
  - Additional secure layer
  - Still software! VM-escapes and other attacks



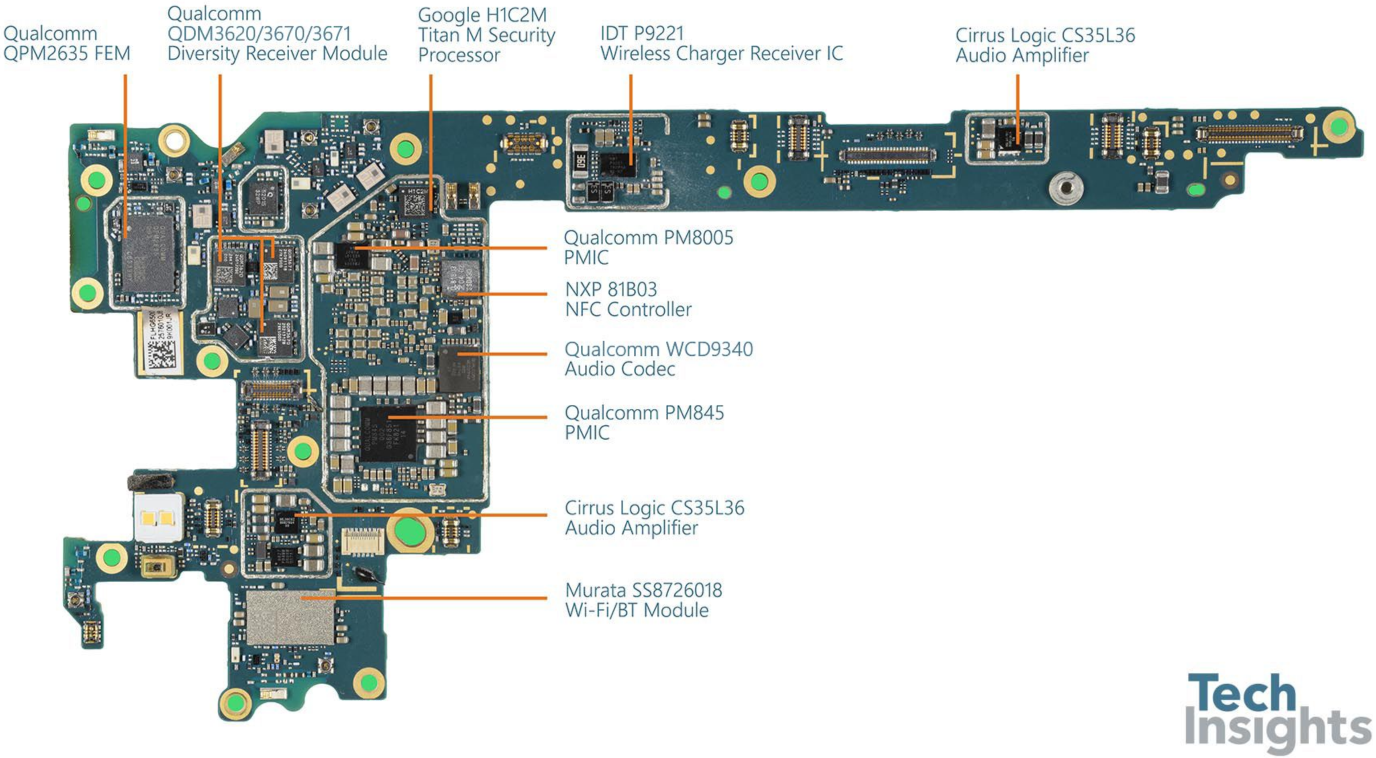
Introduction - system design

- Rely on specialized hardware to improve security
- Three alternatives:
  - Virtual Processor (ARM TrustZone)
  - On-chip Processor (Apple SEP)
  - External Coprocessor (**Google Titan M**)
- All are types of Trusted Execution Environment (TEE)

Secure World vs Normal World

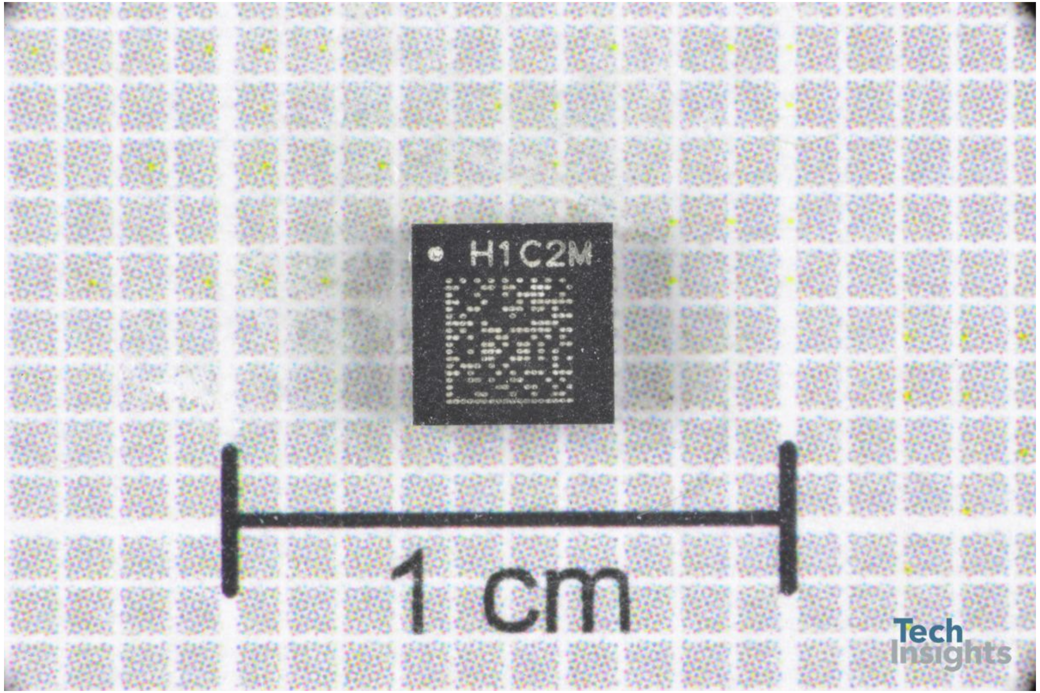


What is Titan M?



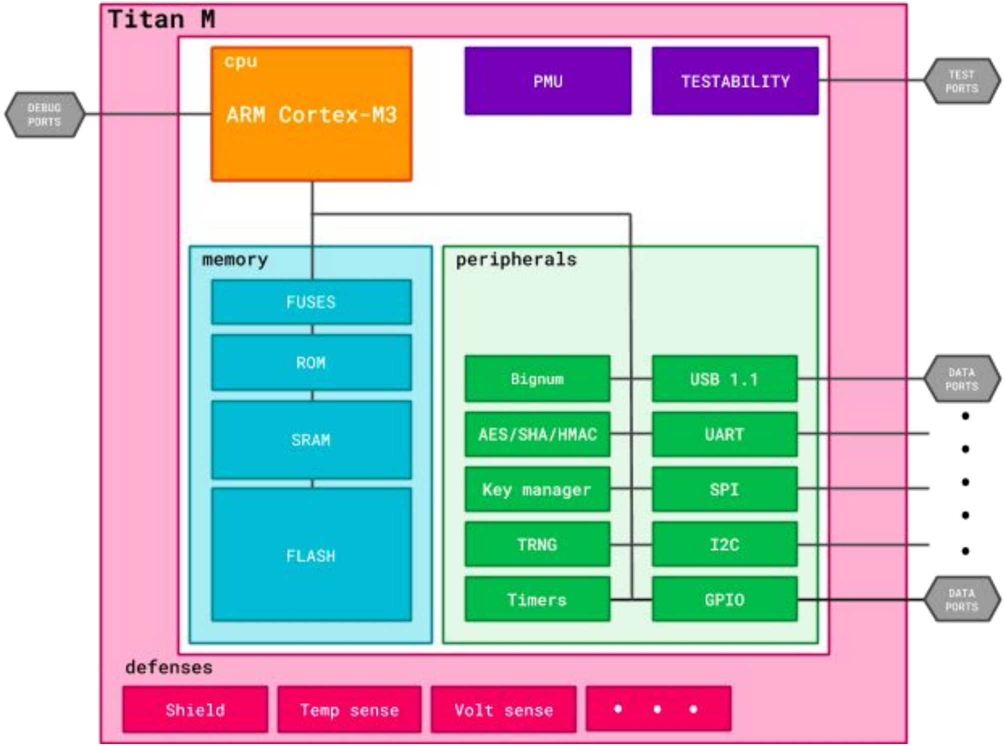
Source: https://www.techinsights.com/blog/google-pixel-3-xl-teardown

What is Titan M?



Specification

- SoC based on ARM Cortex-M3
- 64 kB RAM, internal flash memory
- HW accelerators for common crypto algorithms
- Key manager module and True Random Number Generator
- Common busses
  - UART for logs and console
  - SPI to communicate with Android



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

What is it used for?

UNIVERSITY OF TWENTE.

Android hardware-backed security APIs

- Android Verified Boot (AVB)
- Strongbox
- Weaver
- Identity/Faceauth
- ...

In general, management of secrets and critical information for the security of the device

Why is it interesting?

- First physically separated HSM in Android smartphones
- *Root of Trust* of the device
  - Has to be inviolable and tamper-proof
- Lack of publicly available knowledge about it:
  - The vendor claimed to publish the sources, never did
  - No existing research/presentation/blogpost
  - Only one CVE write-up

Approach

- Study the Titan M chip to understand its security features
- Firmware analysis
  - Extraction and loading
  - Reverse engineering
  - AOSP review for communication with Android
- Vulnerability analysis and exploitation
  - Dynamic perspective
  - Explore protections as an attacker would
- Black box fuzzing for more vulnerabilities

Firmware

- Raw binary on Android FS at `/vendor/firmware/citadel`
- EC: Embedded Controller
  - Base of the Titan firmware
  - Open Source OS, also developed by Google
  - Written in C
- Features:
  - No dynamic allocation
  - Designed around the concept of *task*
  - Driven by interrupts

Firmware 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 Updates

- Regular updates in Nugget task
  - First command writes the image on the flash
  - Second command activates it (requires user password)
- SPI rescue in Titan M loader
  - Feature accessible from fastboot mode
  - Wipes all user data
  - No need for user password

Firmware

- Firmware security?
  - Conceptually simple
  - No MMU, MPU to give permissions to the memory partitions
  - Secure boot
  - No particular software protections, apart from…

```
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);
}
```

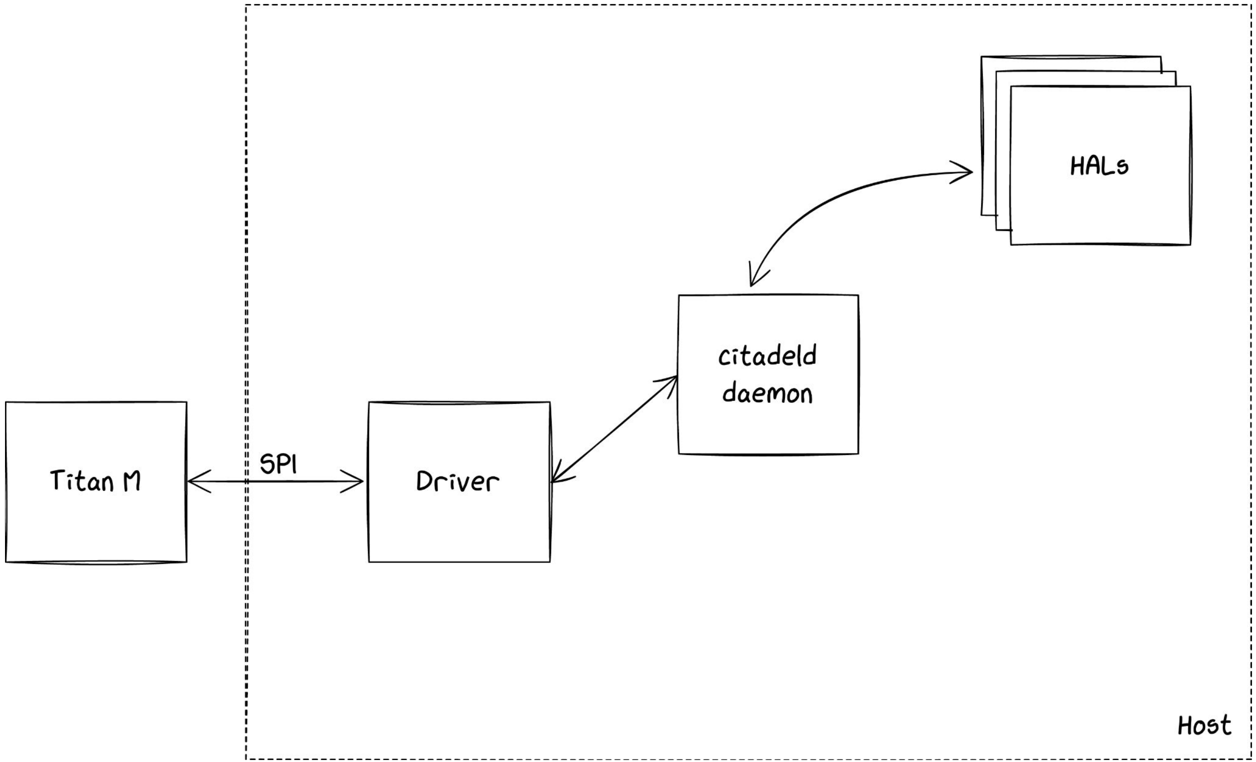
  - Hardcoded stack canary checked in the SVC handler

Communication with Android

UNIVERSITY OF TWENTE.

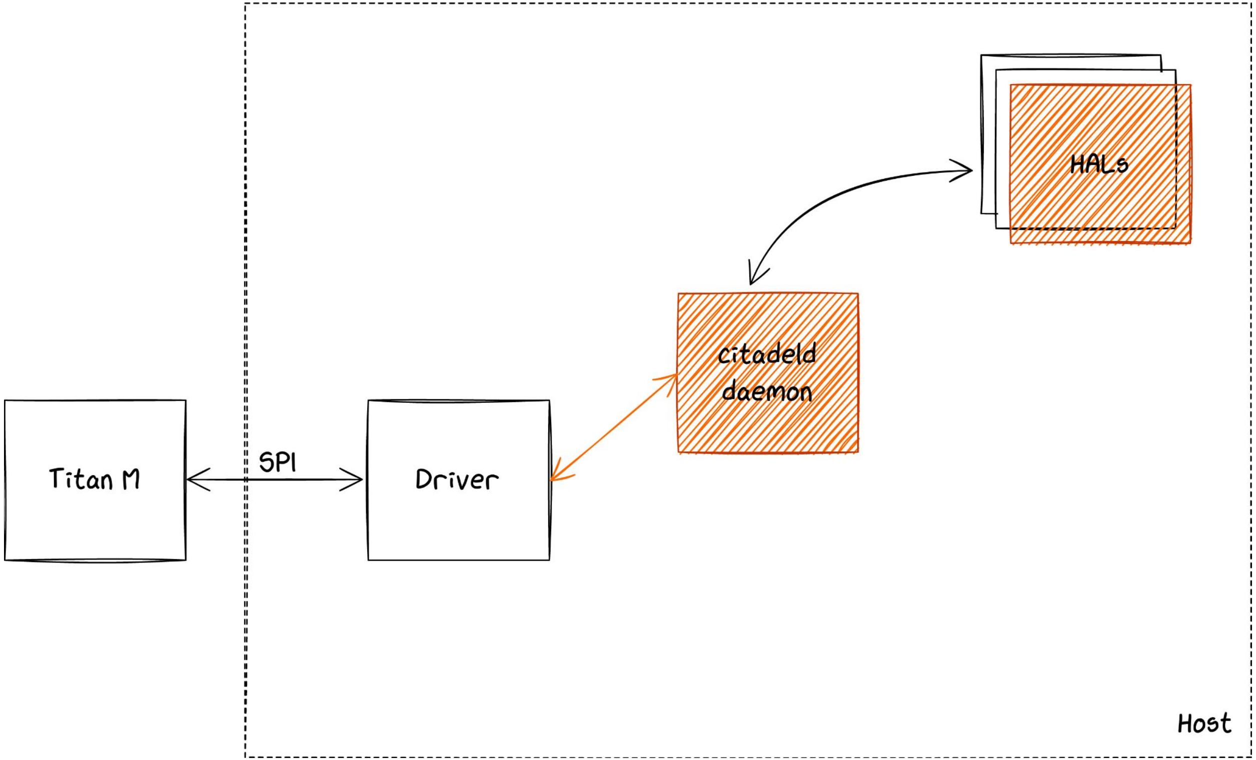
- Protobuf-based
  - Serialization framework by Google
  - Language agnostic
  - Titan M uses the nanopb project
  - Limits the risk of input validation bugs
- Automatically generated primitives to encode/decode messages
- Each task interacting with the main OS has its own `.proto` file

Communication with Android



Where to hook?

Communication with Android



- Using a debugger, the HAL, starting from an Android API
- Using Frida, the citadel daemon (hook `nos_call_application`)
- With a custom client, communicate with the driver directly

`nosclient`

- The daemon uses `libnos_transport` and `libnos_datagram` to communicate with Titan M
- We developed a custom client to use those libraries directly
- Using their function, we can send any message to the chip
- `nosclient` is the main tool we used
- Open sourced at: https://github.com/quarkslab/titanm

After reverse engineering

- Still unknown parts of the firmware (e.g. bootrom)
- To gain more knowledge, exploit a vulnerability
  - Leak interesting memory, or...
  - Obtain code execution
- Goals
  - Improve understanding of the firmware internals
  - Instrument the firmware and test it
  - Load newer versions and search for other vulnerabilities

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?

Challenges

- Vulnerabilities are reported on a monthly basis in the Android security bulletin
- Very few involve Titan M
- Details are very poor
  - Need to manually diff firmware versions to find the patches
- Given a vulnerability:
  - Is it exploitable?
  - How can we reach the vulnerable code?
  - How can we debug a proof-of-concept?

The chosen one

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

```
message ICpushReaderCertRequest{
  bytes x509Cert = 1;
  uint32 tbsCertificateOffset = 2;
  uint32 tbsCertificateSize = 3;
  uint32 signatureOffset = 4;
  uint32 signatureSize = 5;
  uint32 publicKeyOffset = 6;
  uint32 publicKeySize = 7;
  uint32 signAlg = 8;
}
```

The chosen one

Structure in RAM for runtime information of the Identity task

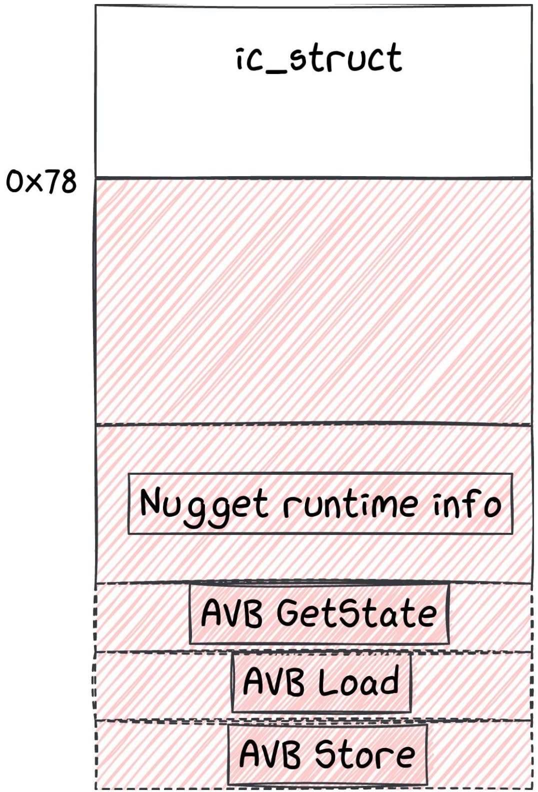
This is always true before calling this command

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

Taken from the message
➜ attacker controlled

What can we do with the exploit?

- Vulnerable buffer not allocated on the stack of the function
  - Cannot simply overwrite the saved return address
- After the buffer we have other runtime data of the chip...
- ... and the list of pointers to the command handlers



Exploitation

Using `nosclient`:

- Reset the chip
- Send an `Identity ICpushReaderCert` command
  - Overwrite `ic_struct`
  - Overwrite the Nugget structure, writing back initialization values
  - Overwrite the first command handler with the first function/gadget
- Send an `AVB GetState` command
- Code execution!

Exploitation

- Code-reuse attack
  - Return Oriented Programming (ROP)
- Cannot fetch arbitrary instructions on writable memory
- Still, we can leak the content of any memory address
  - Leaked the boot ROM
  - Read primitive to debug other vulnerabilities

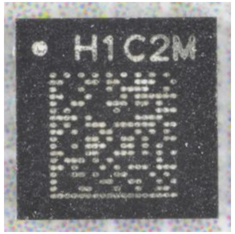
Improving vulnerability research

- We know what messages can be sent to the Titan M
- We have an idea of the responses we expect
  - `nos_call_application` returns a meaningful return code

➜ Design and develop a *structure-aware black-box fuzzer*

Designing a fuzzer for Titan M

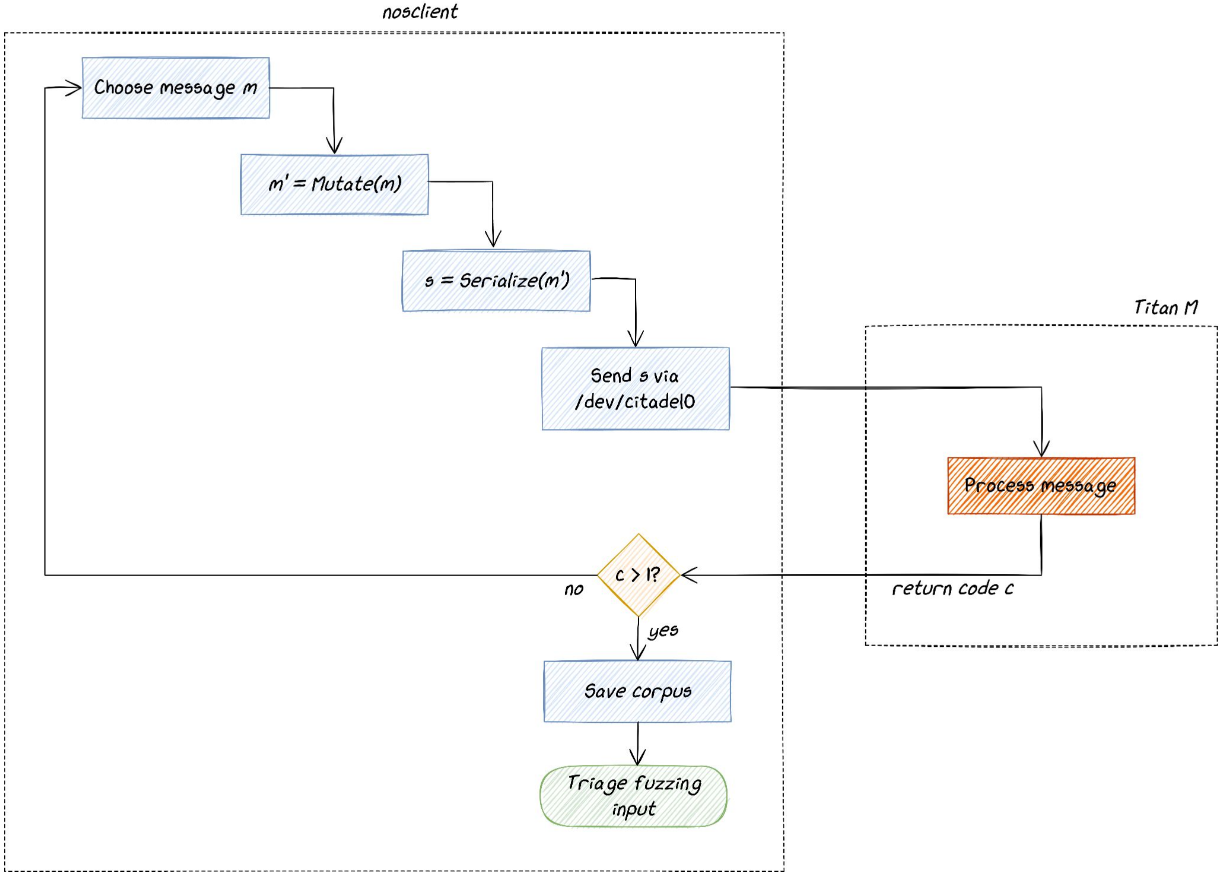
- Fully black-box approach
  - Cannot recompile and instrument the firmware
  - Cannot use DBI
  - Almost no useful debugging information
  - No coverage
- Rely on return value from library call
  - If greater than 1, something went wrong
- Mutation-based
  - Mutate messages respecting Protobuf definitions
  - Random operators to trigger typical vulnerabilities



Implementing a fuzzer for Titan M

- Use `nosclient`
  - Sends custom messages to Titan M
  - Relies on library functions of the Android OS
- Mutate messages with `libprotobuf-mutator`
- Check return code
- Store and triage inputs generating faulty states

Fuzzer workflow



Experiment setup

- Google Pixel 3
  - Android 11
  - Rooting required to communicate with SPI driver
- `nosclient` running natively
- Mutate messages from `Keymaster`, `Identity`, and `Weaver` tasks
  - `AVB` excluded because of secure boot commands

Results

Firmware version: 2020-09-25, `0.0.3/brick_v0.0.8232-b1e3ea340`

- Buffer overflow in `Identity ICpushReaderCert`
- Buffer overflow in `Identity ICsetAuthToken`
- `Identity {WICbeginAddEntry, WICaddAccessControlProfile, WICfinishAddingEntries, ICstartRetrieveEntryValue}` make the chip crash (nullptr-deref)
- `Keymaster {FinishAttestKey, IdentityFinishAttestKey}` make the chip reboot

Demo

Results

Firmware version: latest, `0.0.3/brick_v0.0.8292-b3875afe2`

- `Identity {WICfinishAddingEntries, ICstartRetrieveEntryValue}` still make the chip crash
- Same function, dereferencing values from uninitialized structures
- Bug report sent to Google
- Not severe enough to be considered as a vulnerability

Some comments

- Throughput around 74 msg/sec
- All these results come from the first minutes of fuzzing
  - Some of them even after 1-2 seconds
  - Approach seems promising
- State space probably explored quickly
  - No further results in the subsequent hours
- Return code $> 1 \not\Rightarrow$ vulnerability found
  - Some commands require previous configuration and should be ignored
  - I/O, application-specific or timeout errors happen, but rarely

Limitations and possible improvements

- No visibility on coverage
- Explore different sources
  - Analyze the actual response
  - Parse the UART log (problem here is accessing that from Android)
- Open the emulation Pandora's box
  - Completely different approach, with other challenges
- Anyway, hard to reproduce sequences of messages

Conclusion

- Titan M is a “first-of-its-kind”
- Interesting findings about the firmware
  - Simple design, but some debatable security measures
- Effective tooling developed to interact with the chip
- Exploited a known vulnerability and leaked the boot rom
  - First code-execution exploit known on Titan M
- Fuzzing can bring even more interesting results

All the tools we developed are available at: https://github.com/quarkslab/titanm