Advanced Android Archaeology: Battling Bloated Complexity

Mathias Payer

EPFL hexhive

Android Complexity is Beyond Imagination

Over 3 billion users across 190 countries

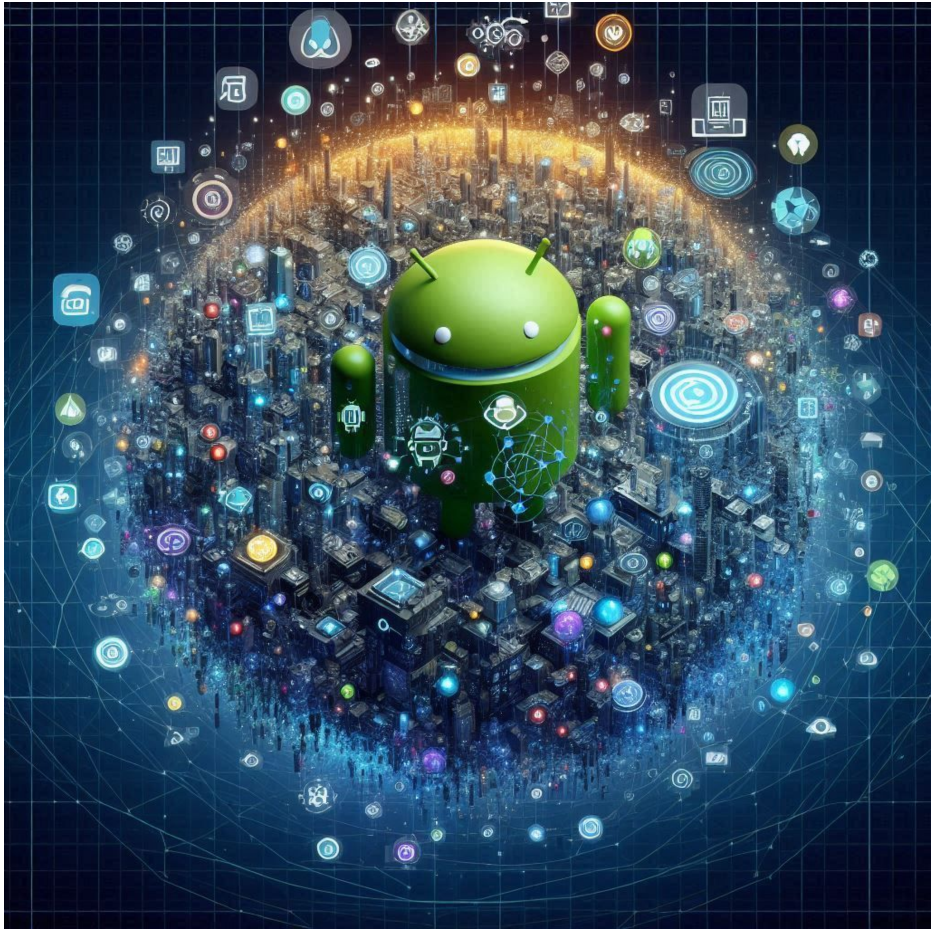
Almost ¾ market share for mobile phones

2.6mio apps in the App store

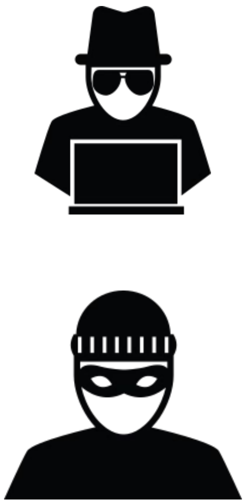
~1.5 billion devices sold per year

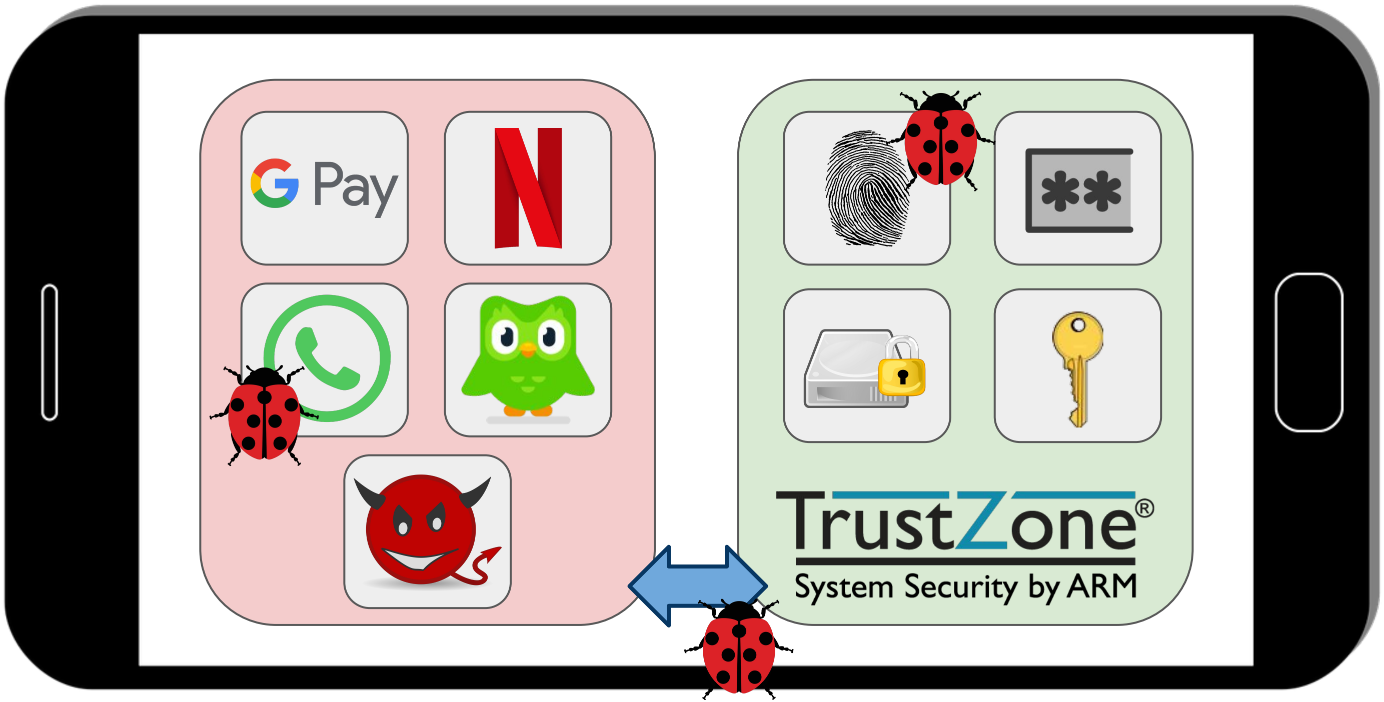
Several TB of system images

Roughly 11 TB of apps



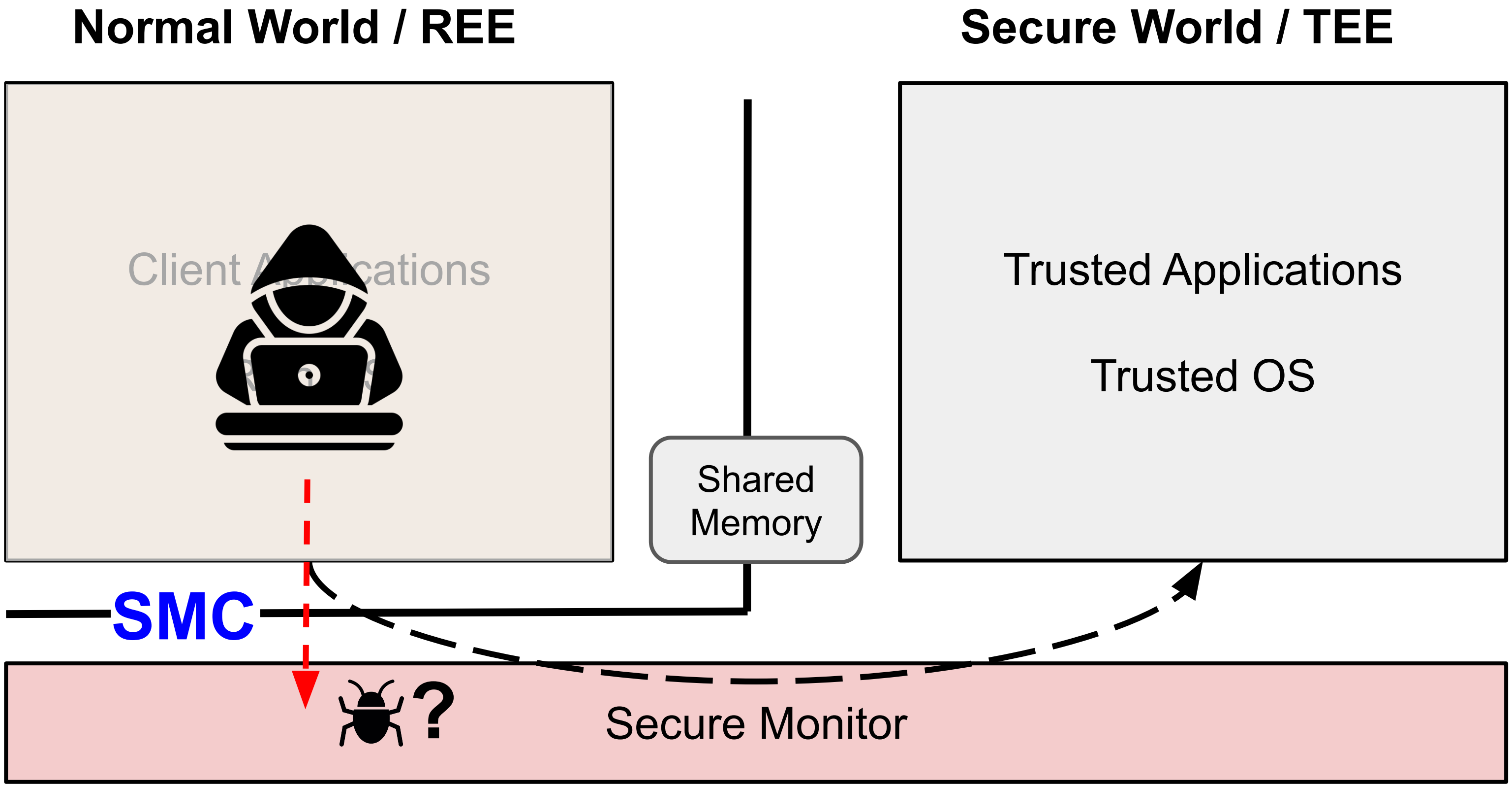
Android Architecture Overview





EL3XIR: 🧙 Be Greedy and Dig Deep

ARMv8-A TrustZone



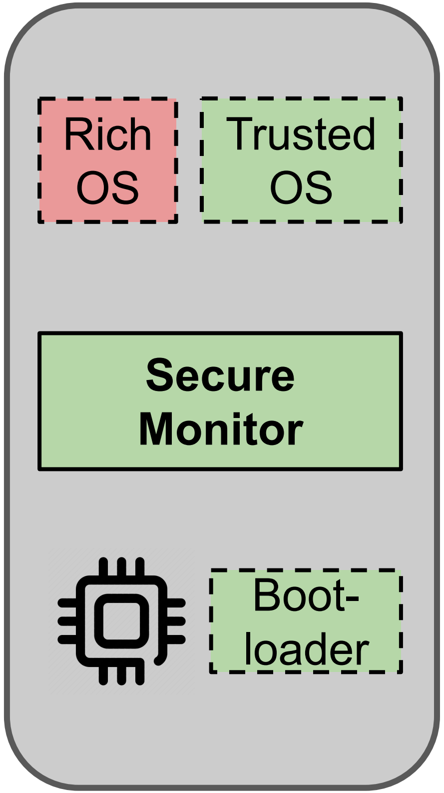
Fuzzing Secure Monitors - Challenges

C1 Limited Introspection

Rehosting: Execute firmware in an emulated environment mimicking (parts of) the original device

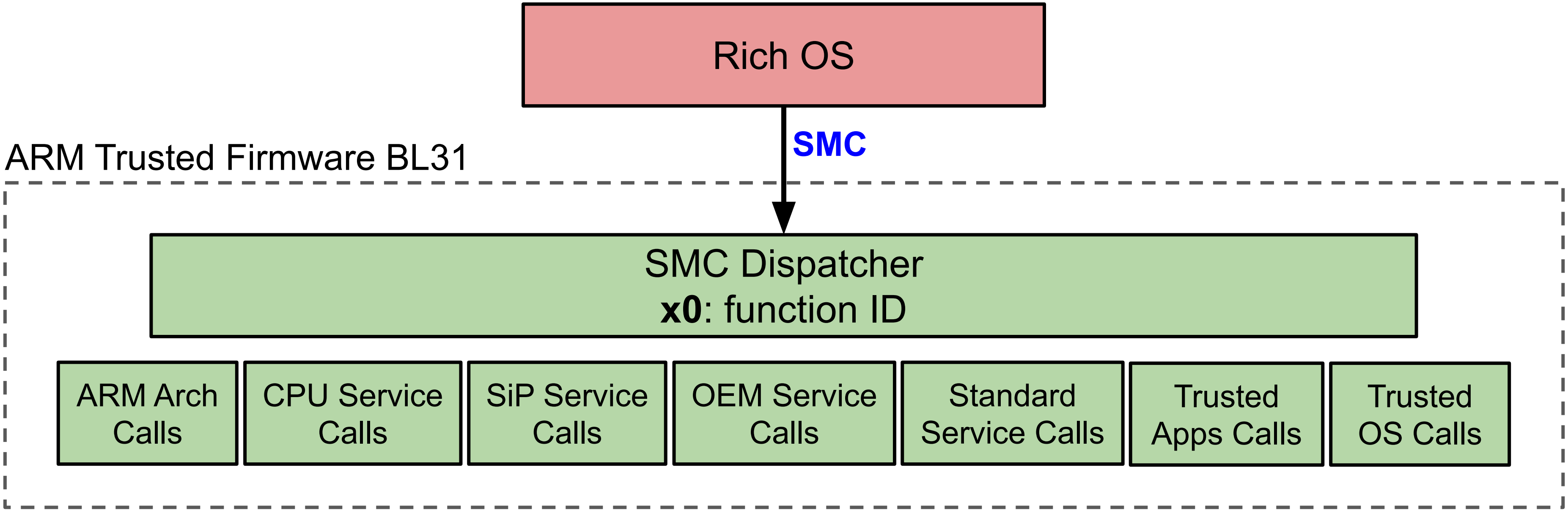
C1.1 Dependency on Software Components

C1.2 Infeasibility of Manual Peripheral Modeling



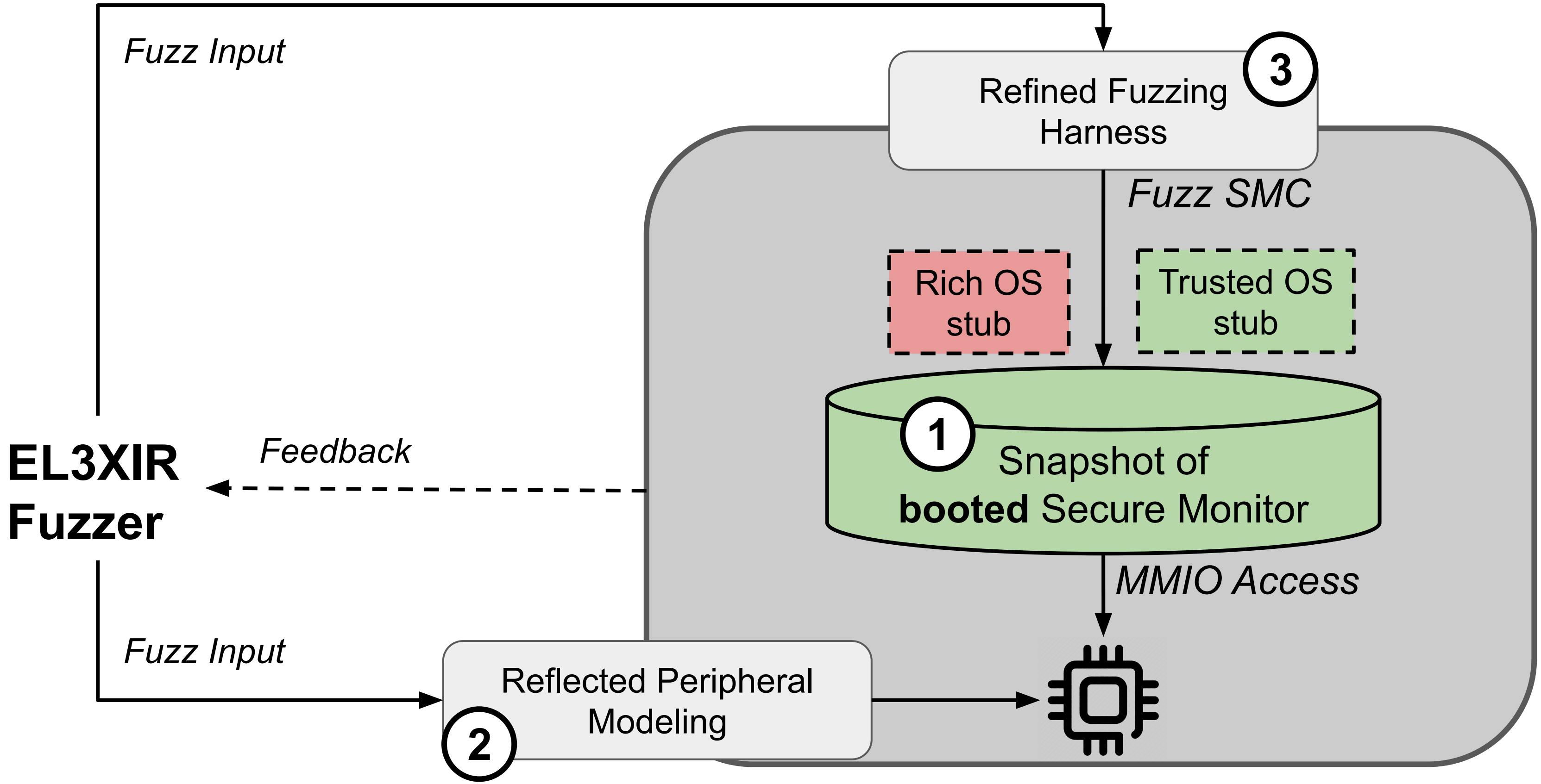
Fuzzing Secure Monitors - Challenges

C2 Complex Input Space



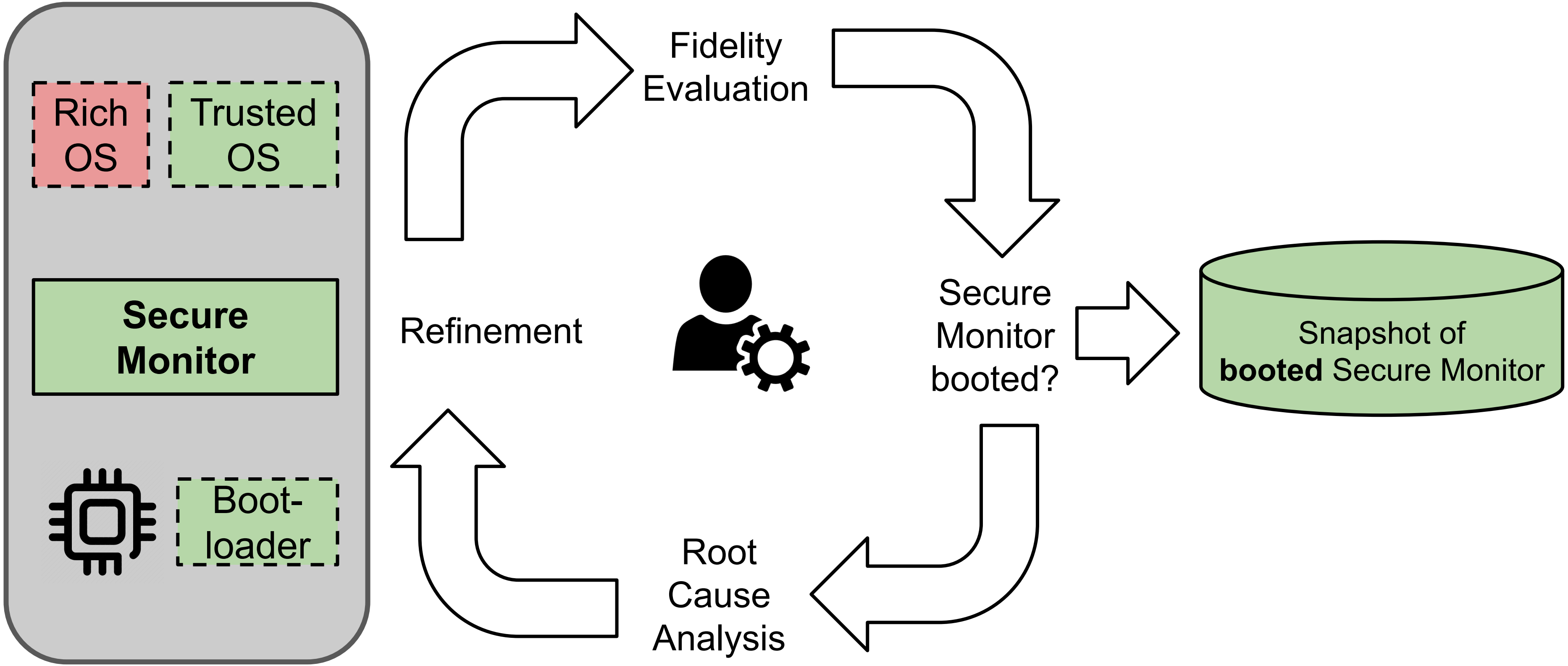
Several tens of runtime services with unique APIs…

EL3XIR's Approach - Overview



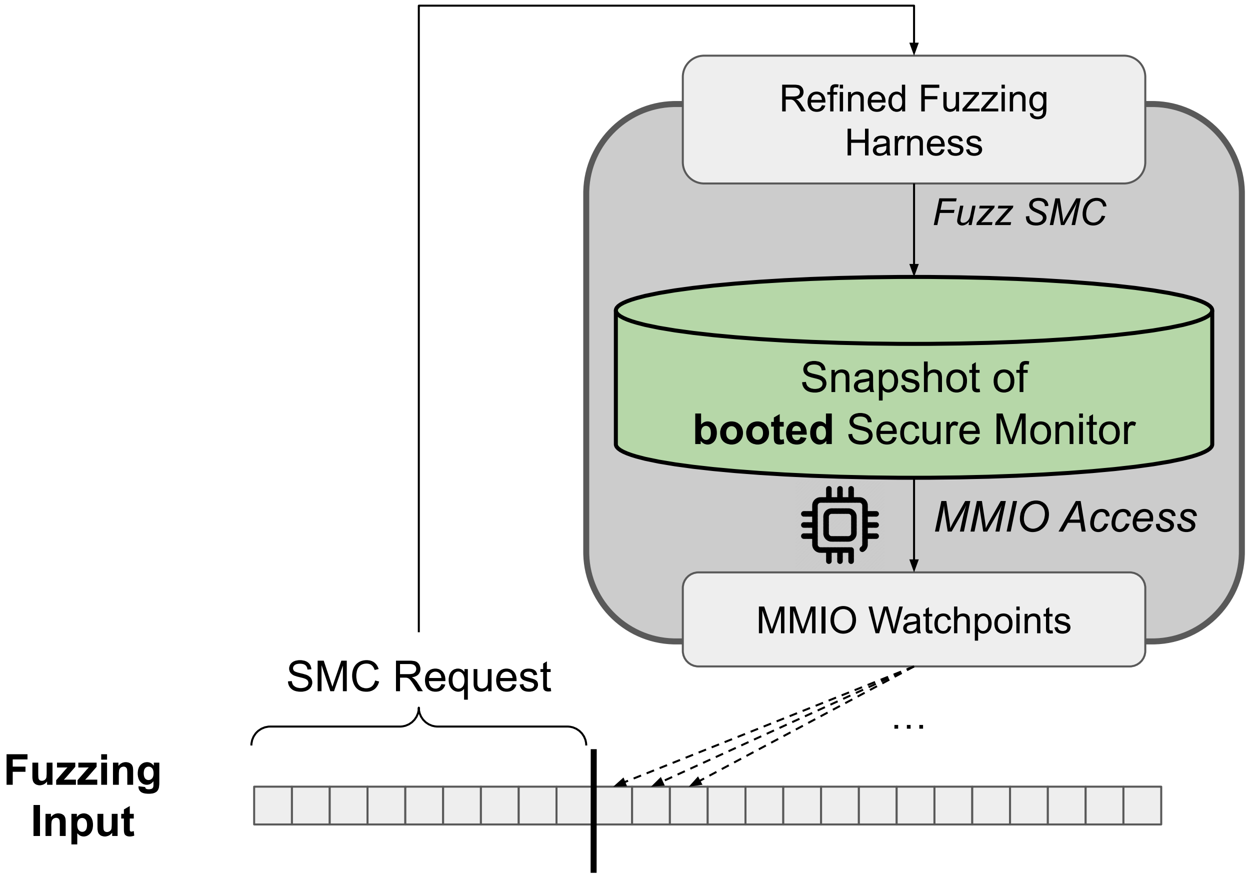
Contribution 1 Partial-Rehosting of Secure Monitors

C1.1 Dependency on Software Components



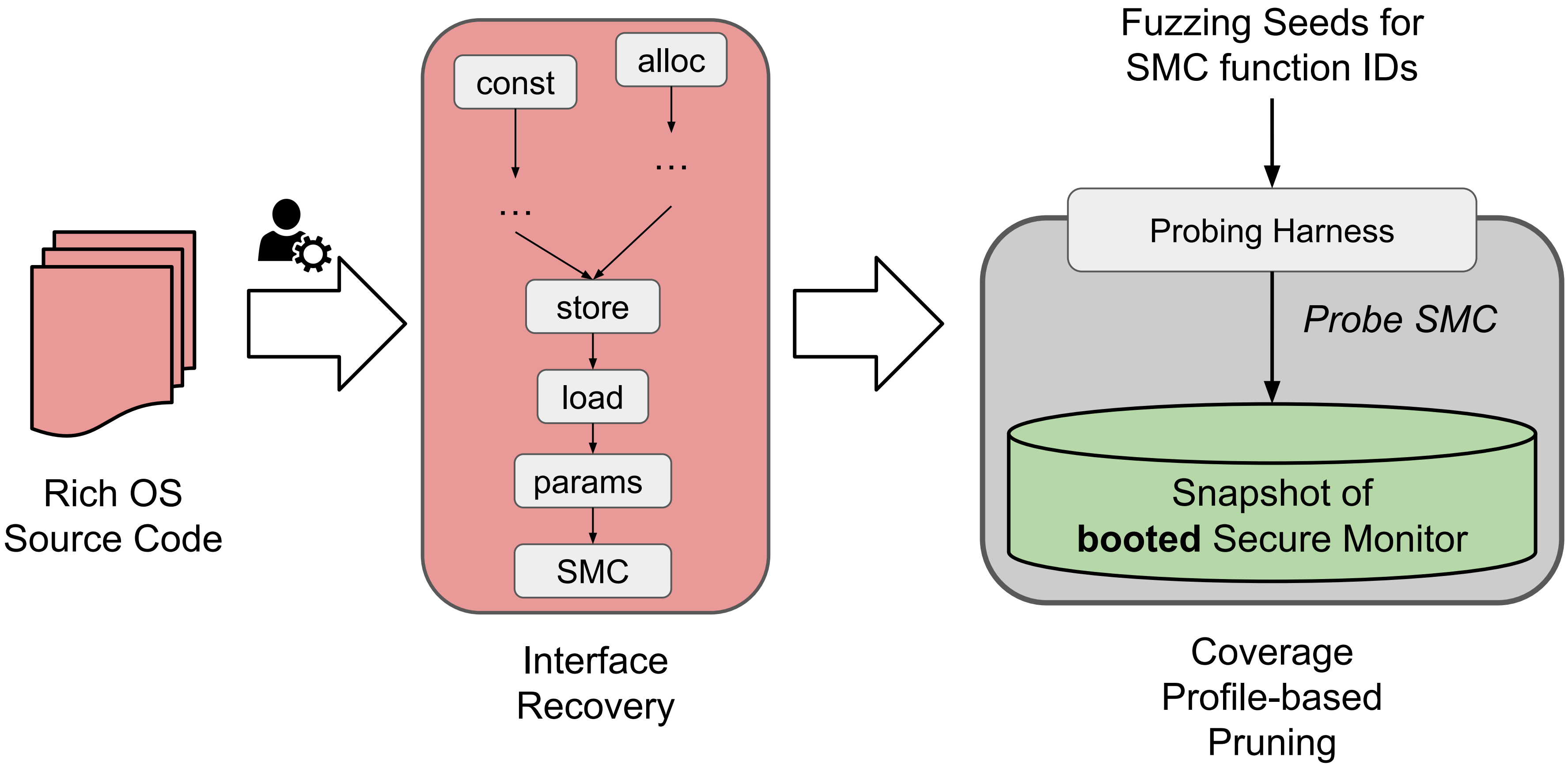
Contribution 2 Reflected Peripheral Modeling

C1.2 Infeasibility of Manual Peripheral Modeling



Contribution ③ Harness Synthesis

C2 Complex Input Space



Evaluation - Bugs and CVEs

7 targets from 6 different vendors

- 4 open-source, 3 closed-source

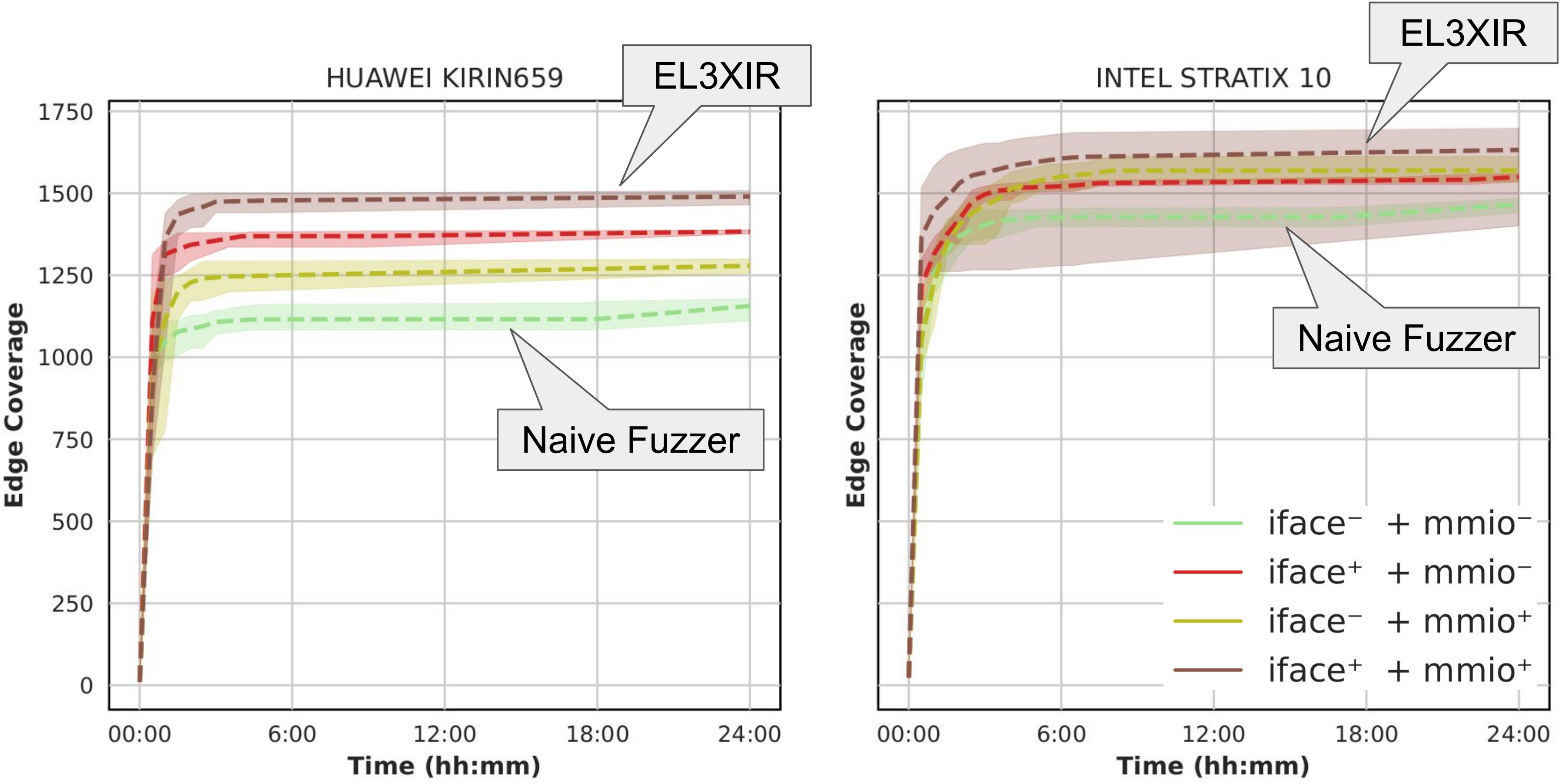
EL3XIR triggered 34 bugs (**17** security relevant) in 5 targets

- Naive baseline comparison triggered 19 bugs (**10** security relevant)

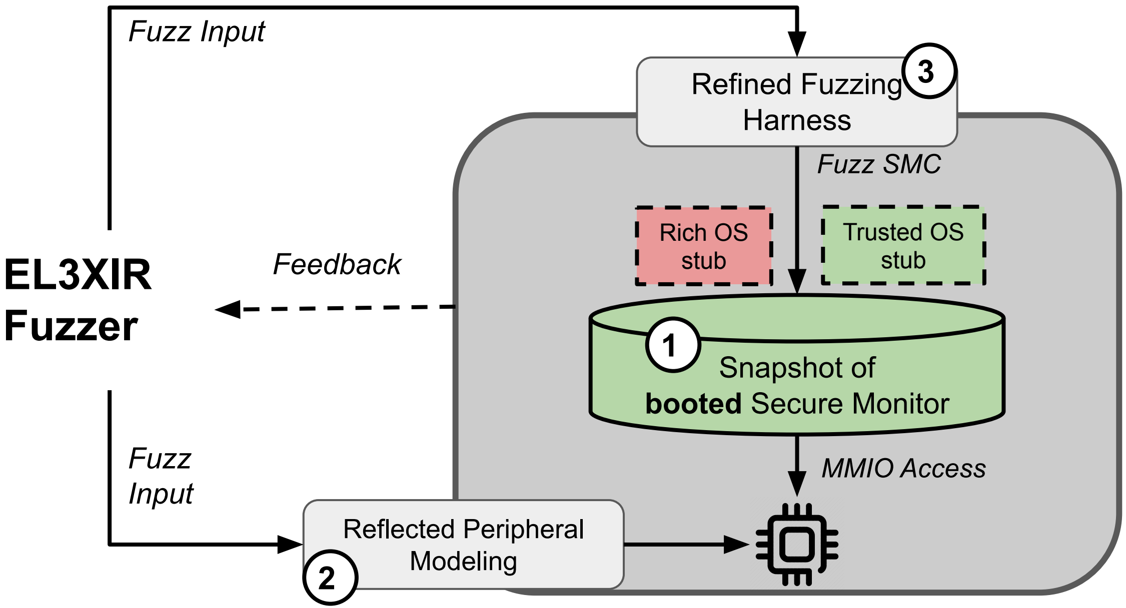
Responsible disclosure resulted in 6 CVEs plus 11 confirmed bugs

CVE-2022-38787, CVE-2023-22327 (5 different bugs), CVE-2023-49614, CVE-2024-22390, CVE-2023-31339, CVE-2023-49100

Evaluation - Coverage



EL3XIR: Fuzzing COTS Secure Monitors



Rehosting Framework for proprietary TrustZone Firmware

Highly automated Fuzzing Pipeline including Harness Synthesis and Peripheral Modeling

Fuzz your own Secure Monitor

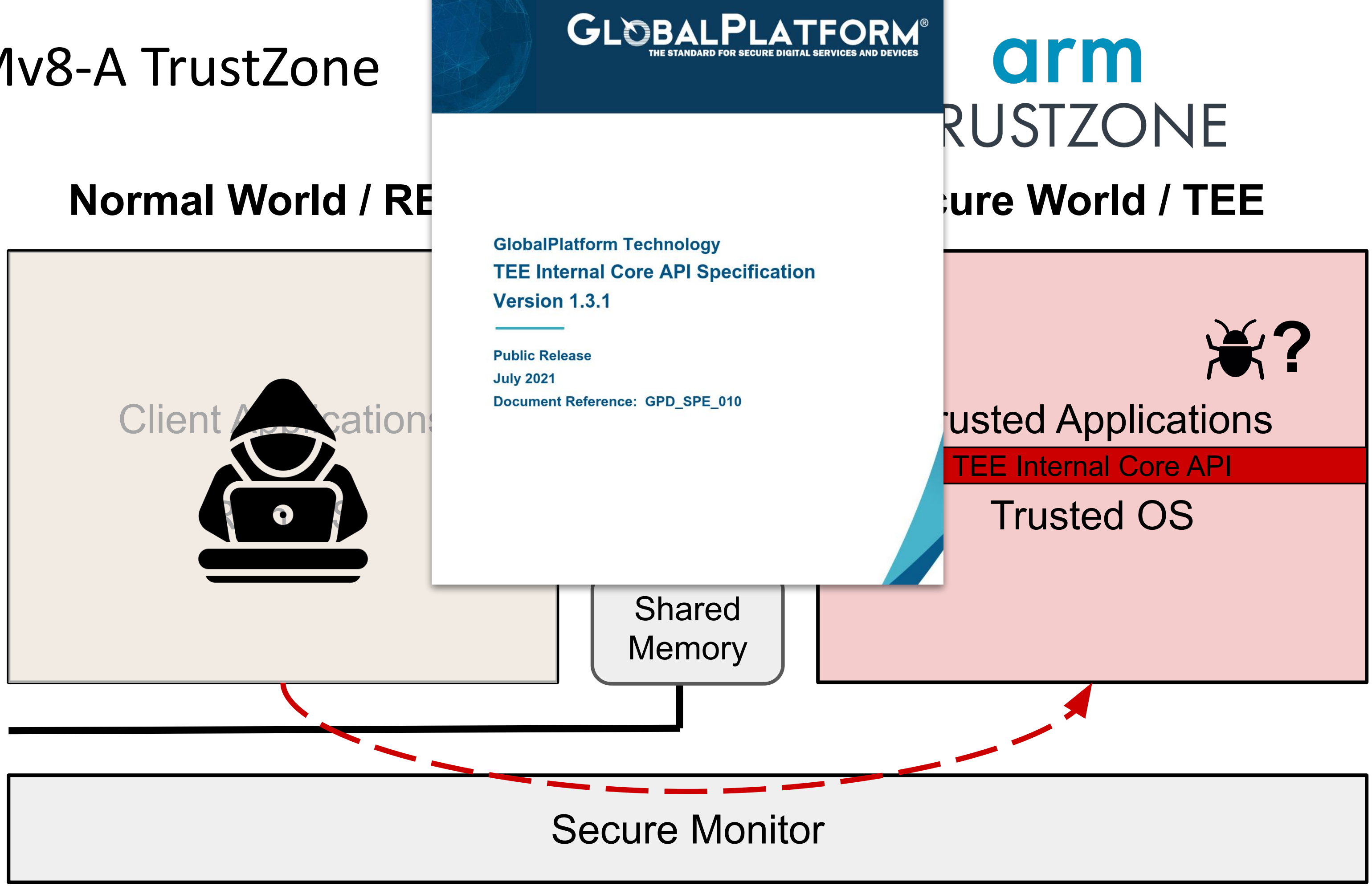
github.com/HexHive/EL3XIR

EL3XIR: Fuzzing COTS Secure Monitors.
Christian Lindenmeier, Mathias Payer, and Marcel Busch. In SEC'24

🌍😵‍💫 GlobalConfusion

Test Android Trusted Apps

ARMv8-A TrustZone



```c
TEE_Result TA_InvokeCommandEntryPoint(void *sessCtx, uint32_t cmdId,
                                      uint32_t paramTypes, TEE_Param params[4])
{
```

Stores session state

Chooses TA cmd handler

Determines types of params

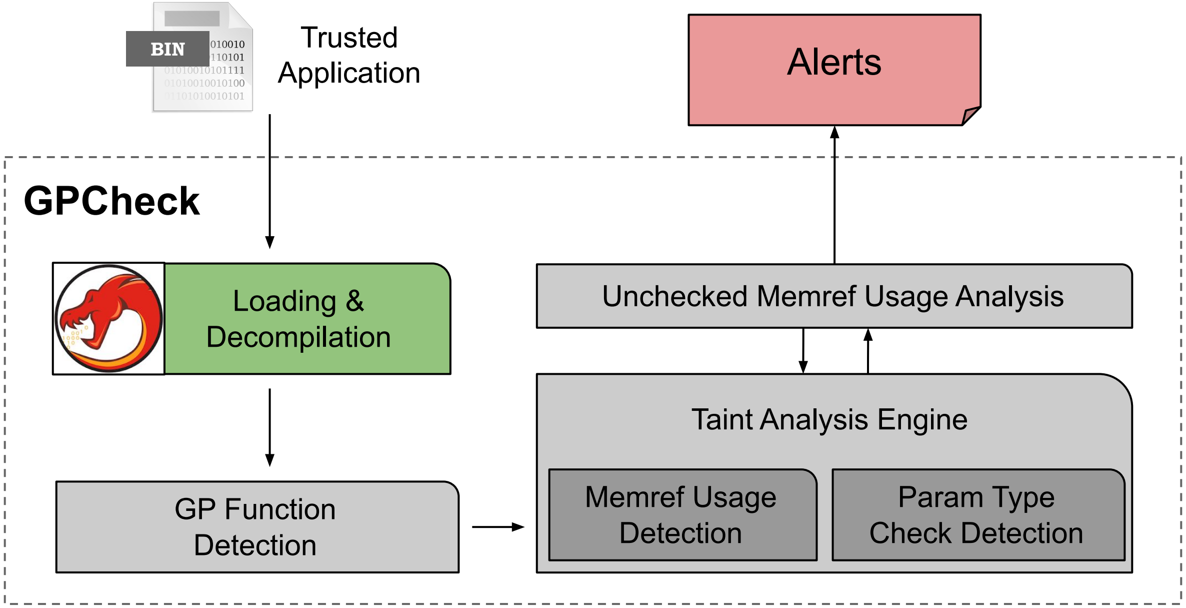
Four TEE_Param parameters

OPTIONAL

```c
typedef union {
    struct {
        void *buffer;
        uint32_t size;
    } memref;
    struct {
        uint32_t a;
        uint32_t b;
    } value;
} TEE_Param;
```

GPCheck

- Ghidra-based
- Post-production binary analysis/check
- Open-Source



https://github.com/HexHive/GlobalConfusion

```c
TEE_Result vuln(TEE_Param params[4] ,uint32_t param_types) {

  uint32_t a;
  uint32_t b;

  a = params[0].value.a;
```

```c
dEntryPoint(void __maybe_unused *sess_ctx,
nt32_t cmd_id,
nt32_t param_types, TEE_Param params[4])

* Unused parameter */

D_CMD_INC_VALUE:
n(params, param_types);

_ERROR_BAD_PARAMETERS;

;
```

Not checked, but not interesting!

```c
TEE_Result vuln(TEE_Param params[4] ,uint32_t param_types) {

  uint32_t a;

  char* buf = params[0].memref.buffer;

  a = ((uint32_t*) buf)[0];
  ((uint32_t*)buf)[1] = a;
  return TEE_SUCCESS;
}

TEE_Result TA_InvokeCommandEntryPoint(void __maybe_unused *sess_ctx,
                        uint32_t cmd_id,
                        uint32_t param_types, TEE_Param params[4])
{
        (void)&sess_ctx; /* Unused parameter */

        switch (cmd_id) {
        case TA_HELLO_WORLD_CMD_INC_VALUE:
                return vuln(params, param_types);
        default:
                return TEE_ERROR_BAD_PARAMETERS;
        }
        return TEE_SUCCESS;
}
```

Not checked, interesting!

Let’s Scan All Apps in the TA Ecosystem!

~6,900 TAs are GP-compliant (~131 unique TAs)

850 vulnerable TAs (33 unique vulnerable TAs)

9 publicly known

10 silently patched

14 0-days

CVE-2023-32835, CVE-2023-32834, CVE-2023-32848, CVE-2024-20078, …

> $ 12k bug bounty

GlobalConfusion: Mitigation

GlobalPlatform is changing their API, making checks explicit

Change fail-open to fail-close design

- Mandatory type check
- Fail-safe abort without proper check

Sent proposal to GP; Draft for API update in progress

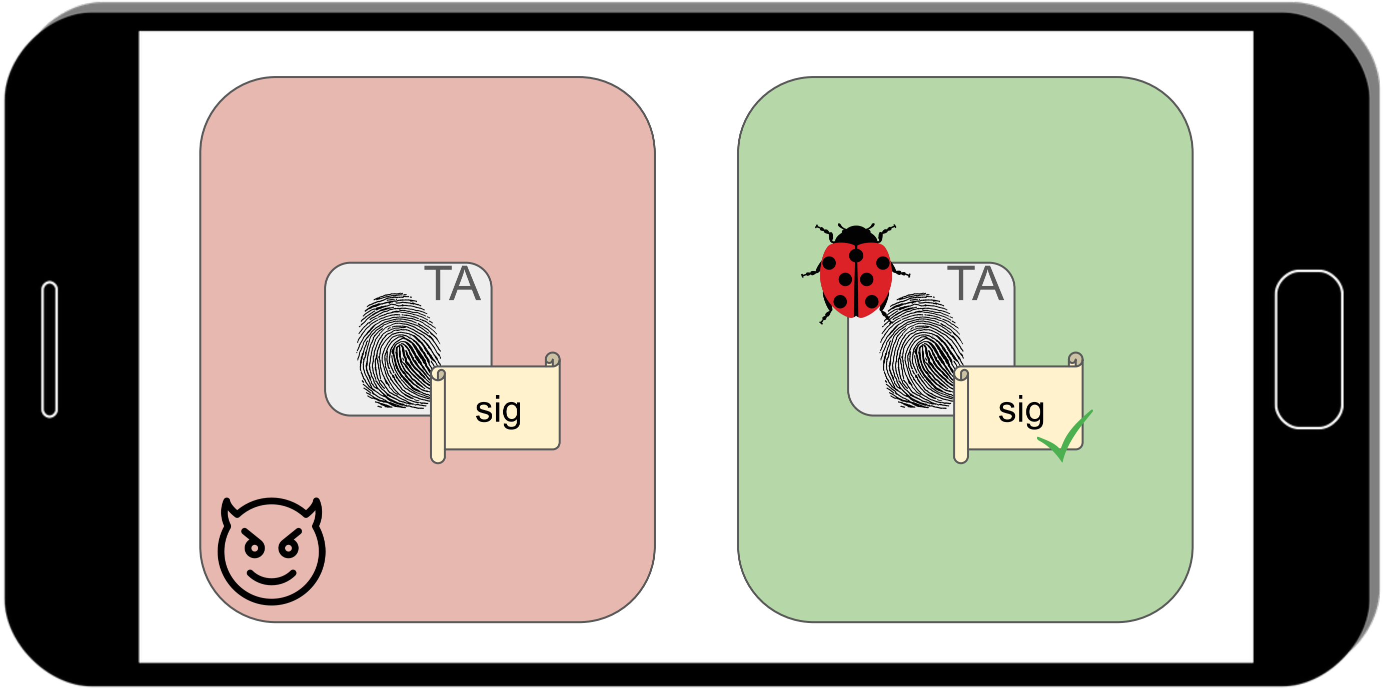
No changes to external API (backwards compatible)

Open-source and based on OPTEE

Cheesing Android Trusted Applications

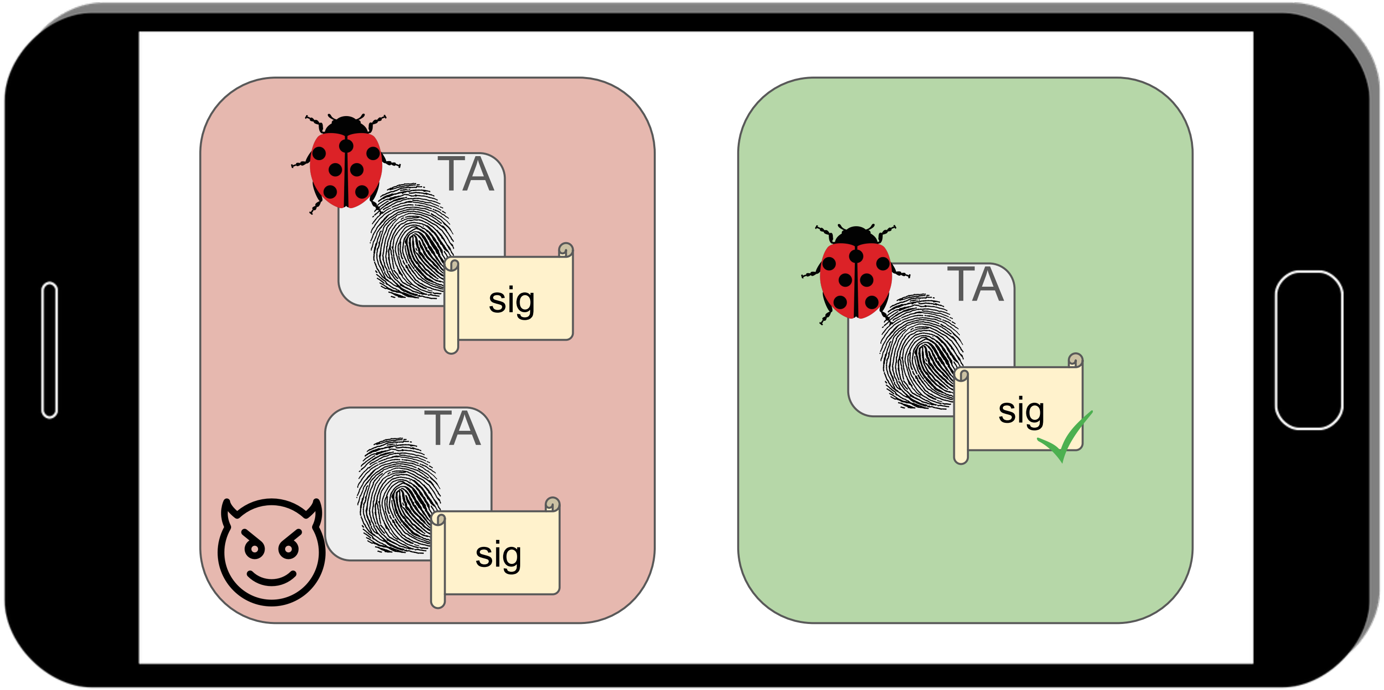
🫖

Trusted Applications



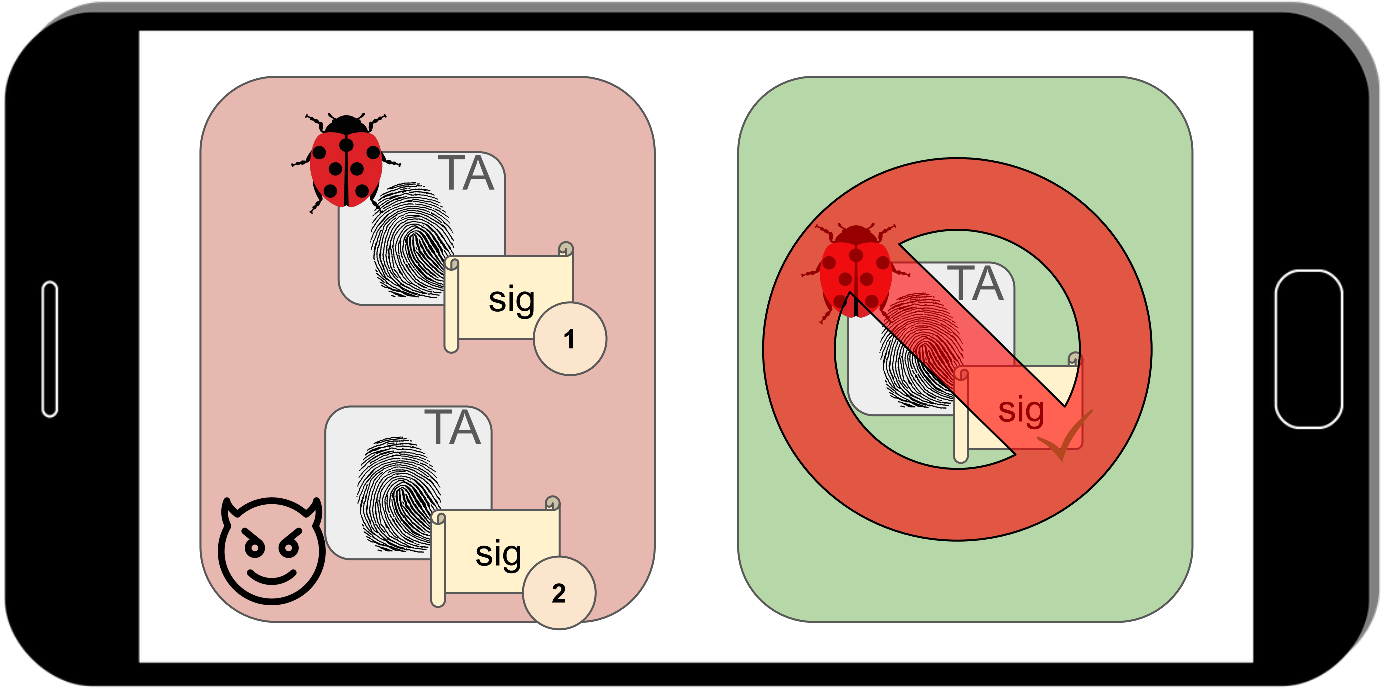
Trusted Applications are authentic dynamically-loadable modules

TA Rollback Attacks



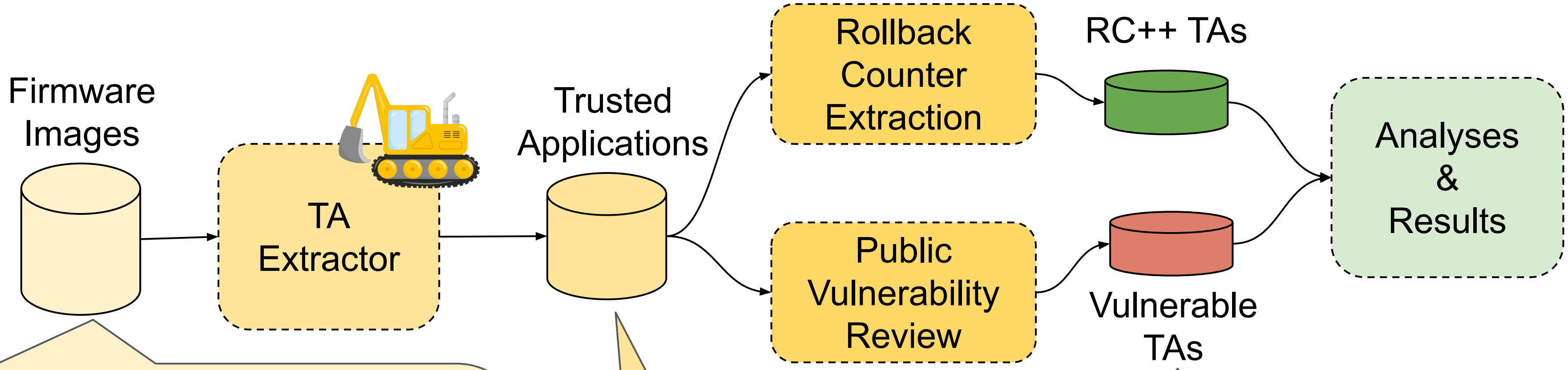
TA Rollback Attacks exploit the authenticity of old and vulnerable TAs

TA Rollback Prevention is Essential for Security



TA Rollback Counters allow TEEs to enforce latest known TA version

Spill the TeA: Analysis of Correct Rollback Prevention



- 5 OEMs >65% market share over the last 4y
- firmware from < 4y
- group phones by ODM, pick at least 2 phones per group
- at least 5 firmware images per phone (over at least 2 years)

- 35,541 TAs (293 unique)
- 4 TEE implementations

- 2,582 vulnerable TAs
- 190 rollback counter usages

`keyinstall` Parameter Type Confusion

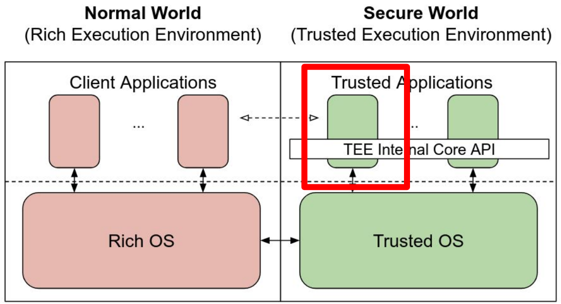
Decompile: TA_InvokeCommandEntryPoint - (08110000000000000000000000000000.ta)

```

undefined8
TA_InvokeCommandEntryPoint(int *session_id,undefined4 command_id,undefined4 param_3,int *parameters)

{
  char *pcVar1;
  undefined4 uVar2;
  undefined4 uVar3;
```

param_3 = parameters_types (cannot rename unused arguments in ghidra)



keyinstall Parameter Type Confusion

```
case 1:
  TEE_LogPrintf("[KI_TA] INFO:");
  TEE_LogPrintf("TZCMD_DRMKEY_QUERY start");
  TEE_LogPrintf("\n");
  tee_param_0 = *parameters;
  tee_param_2 = parameters[2];
  tee_param_1 = parameters[1];
  TEE_LogPrintf("[KI_TA] INFO:");
  TEE_LogPrintf("pInput      = %p",tee_param_0);
  TEE_LogPrintf("\n");
  TEE_LogPrintf("[KI_TA] INFO:");
  TEE_LogPrintf("nInputSize = %d",tee_param_1);
  TEE_LogPrintf("\n");
  TEE_LogPrintf("[KI_TA] INFO:");
  TEE_LogPrintf("pOutput     = %p",tee_param_2);
  TEE_LogPrintf("\n");
  if ((tee_param_1 != 0) && (tee_param_0 != 0)) {
    local_54 = TEE_CheckMemoryAccessRights(5,tee_param_0,tee_param_1);
    if (local_54 != 0) {
      TEE_LogPrintf("[KI_TA] ERROR:");
      TEE_LogPrintf("wrong input access rights!");
      TEE_LogPrintf("\n");
      goto LAB_0000952c;
    }
    local_54 = FUN_000097ec(tee_param_0,&local_58,tee_param_2);
```

Params[0] and params[1] used.

Assumed to be pointers? (no check so we can pass arbitrary integers)

Unchecked parameters Passed to this function

keyinstall Parameter Type Confusion

```
local_54 = FUN_000097ec(tee_param_0,&local_58,tee_param_2);
```

```
DRMKEY_QUERY(void* keyblock, int* count, void* pOutput){
  int keycount = *(int*)(keyblock + 0x44);
  void* src = (void*)(keyblock + 0x48);
    int buffer[0x16];
    if(keycount < 0x201){
        pOutput[0] = keycount;
        int ct = 0;
        while(ct != keycount){
            memcpy(buffer, src, 0x58);
            int encDrmKeySize = buffer[3];
            int keyblockLeng = encDrmKeySize + 0x60;
            keyid = buffer[0];
            *(void*)(pOutput + 4*ct) = keyid; //arbitrary write
            src = src + keyblockLeng;
            ct++;
        }
    ...
    }
```

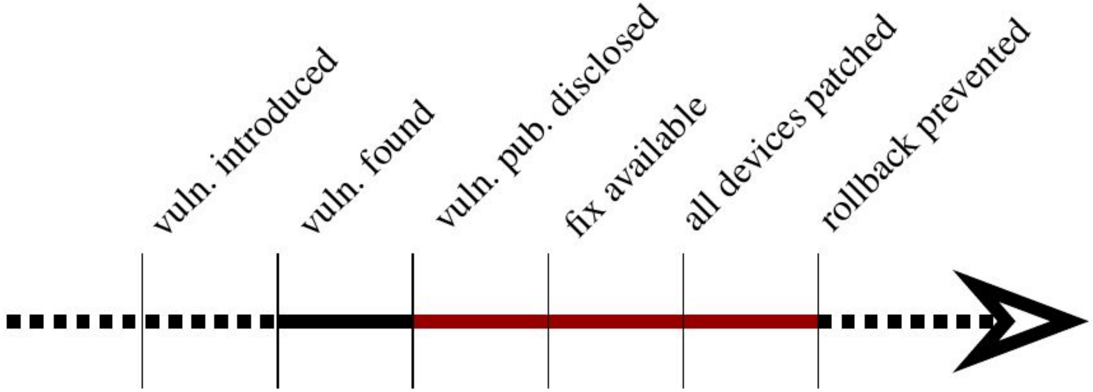
Write 4 bytes from our input buffer to a pointer we control..

Spill the TeA: Summary

TA rollback prevention is incomplete with questionable TA vulnerability practices

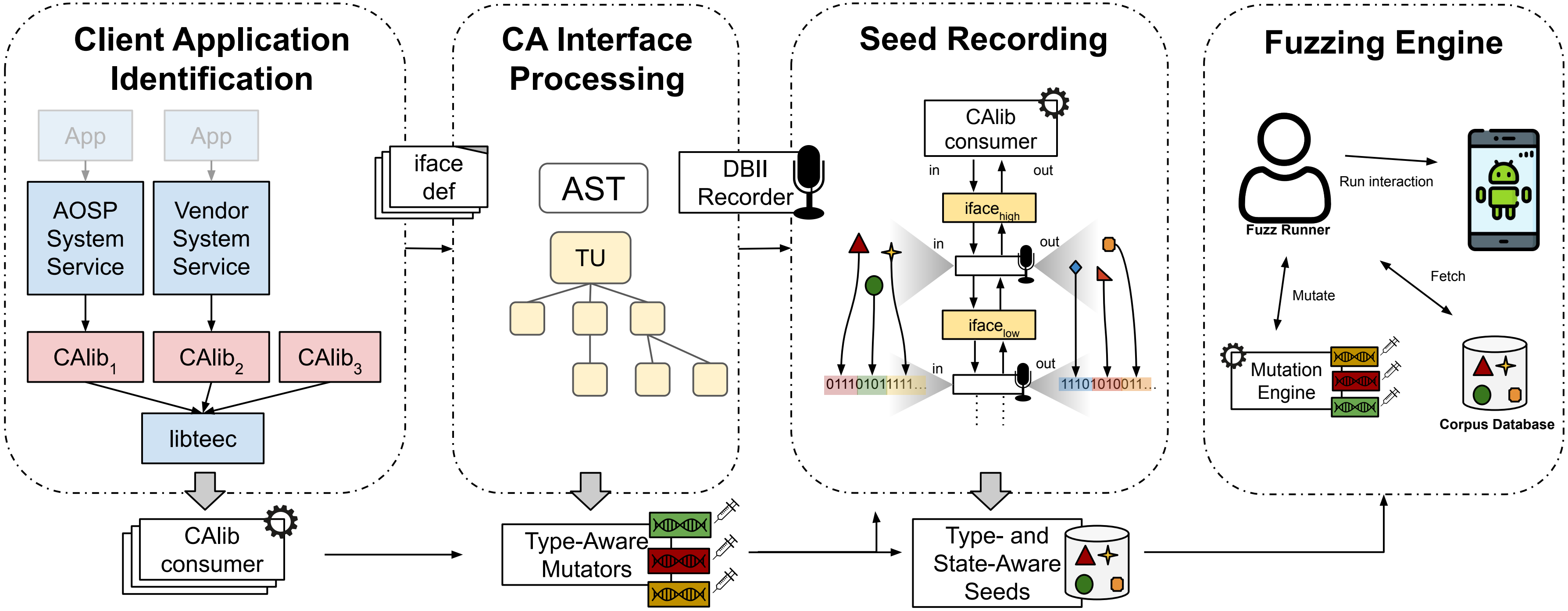
- Internally patched TAs (without disclosure/rollback prevention)
- Security patches limited to one product, not shared across targets

Lack of transparency regarding TA rollback prevention



Spill the TeA: An Empirical Study of Trusted Application Rollback Prevention on Android
Marcel Busch, Philipp Mao, and Mathias Payer. In SEC'24

TEEzz Fuzzing Pipeline: Stateful Interface Fuzzing



TEEzz: Fuzzing Trusted Applications on COTS Android Devices.

Marcel Busch, Mathias Payer, Aravind Machiry, Christopher Kruegel, Giovanni Vigna, and Chad Spensky. In Oakland'23

📚🔥 Just Slap a Secure Allocator On It

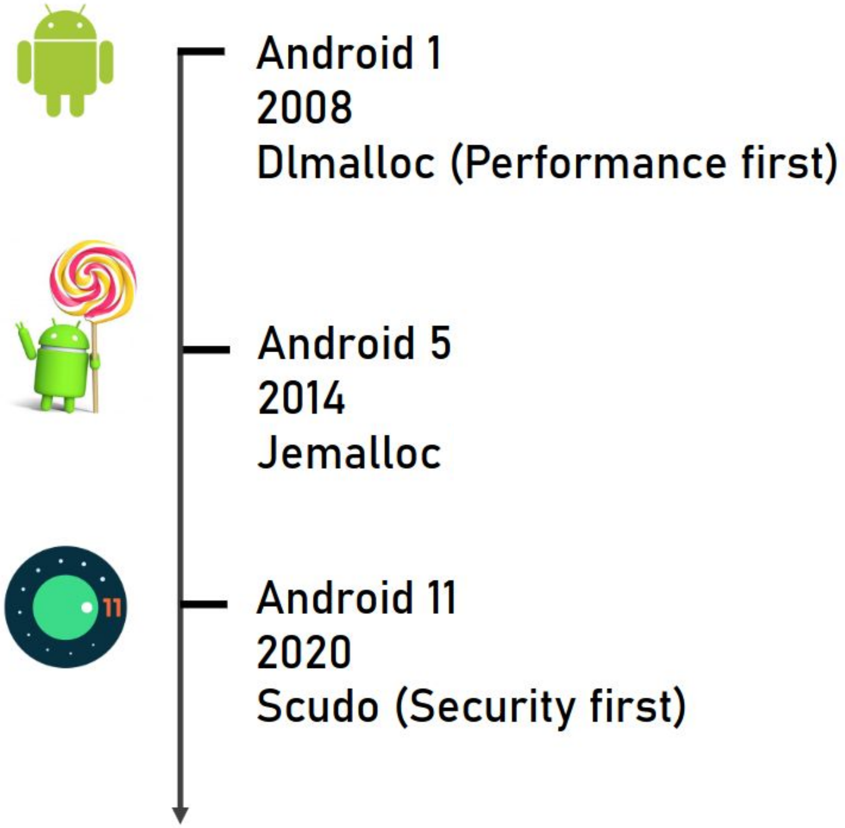
Scudo: the Hardened Memory Allocator

Scudo is..

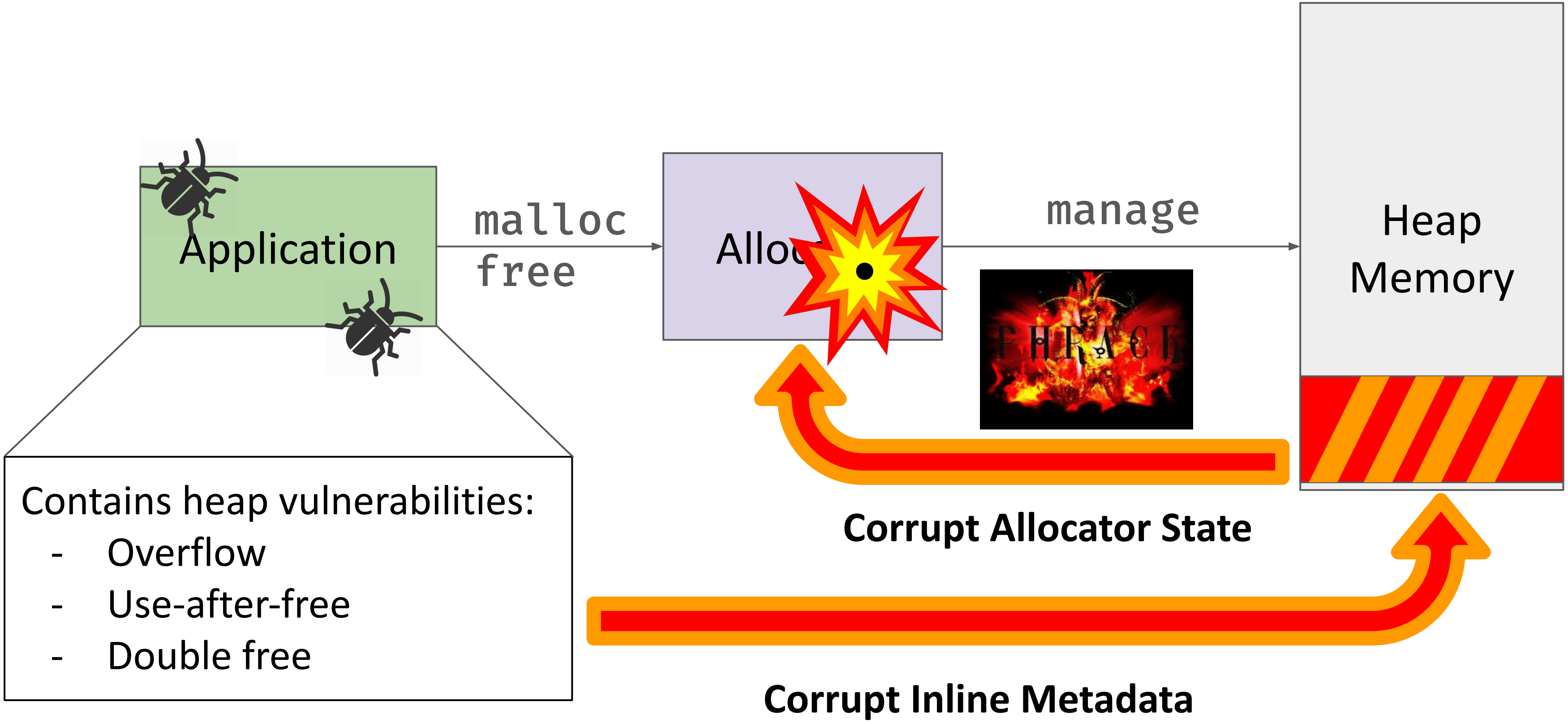
… a userspace memory allocator

… designed to prevent exploitation of heap-based memory corruption vulnerabilities

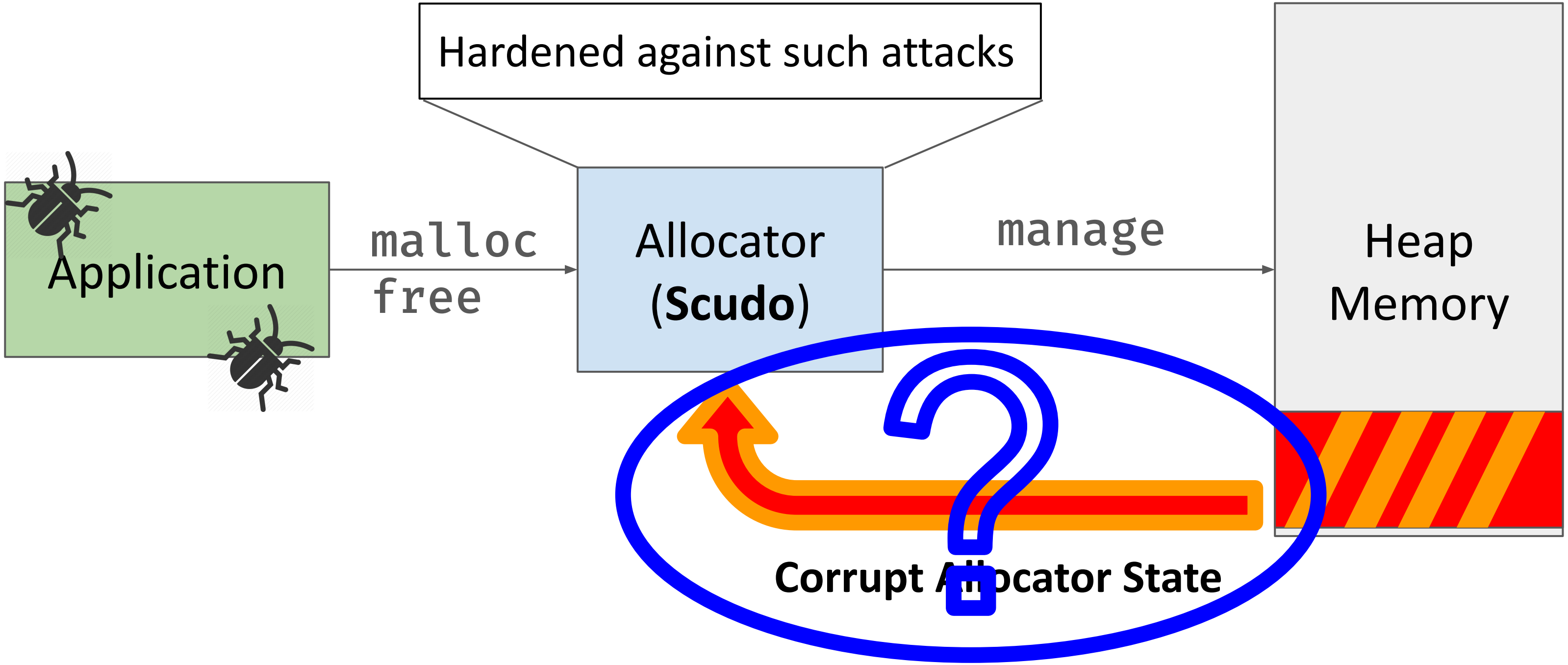
But is it secure?



Exploiting the Allocator



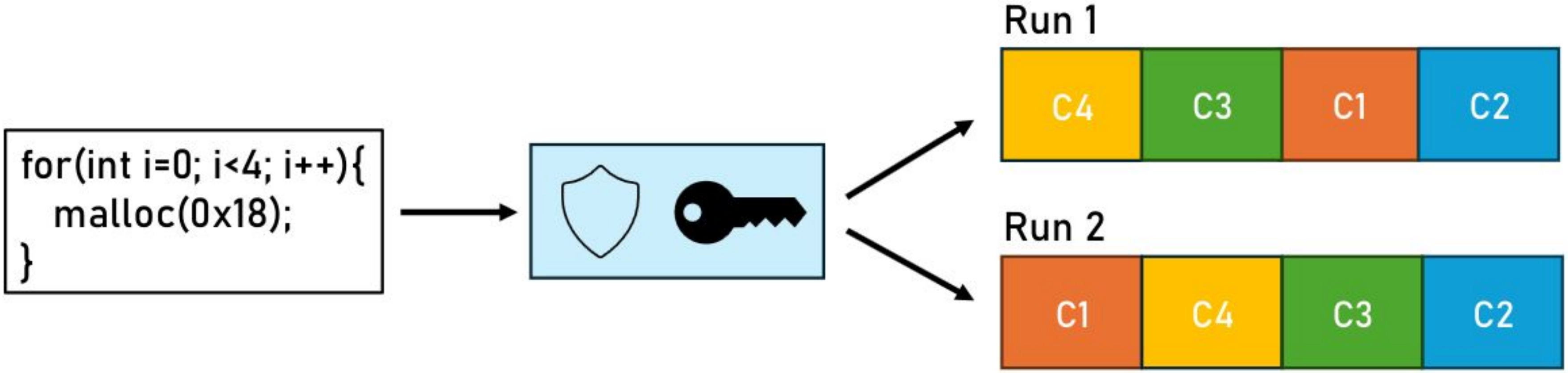
Is Exploiting the Allocator still possible for Scudo?



Threat Model: Able to corrupt heap memory

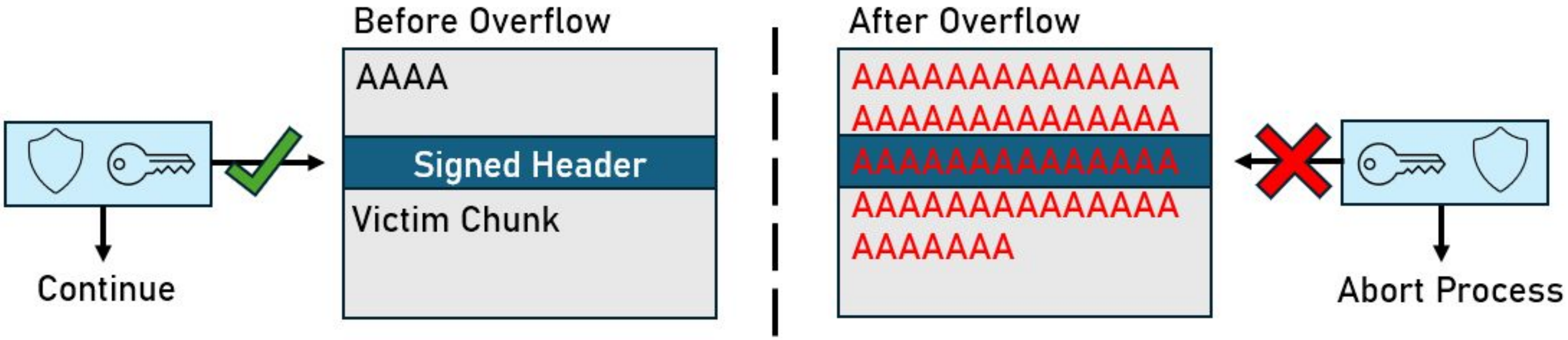
Randomization: Scudo Randomizes the Address of Allocations

Prevent attackers from arranging the heap in a particular layout.

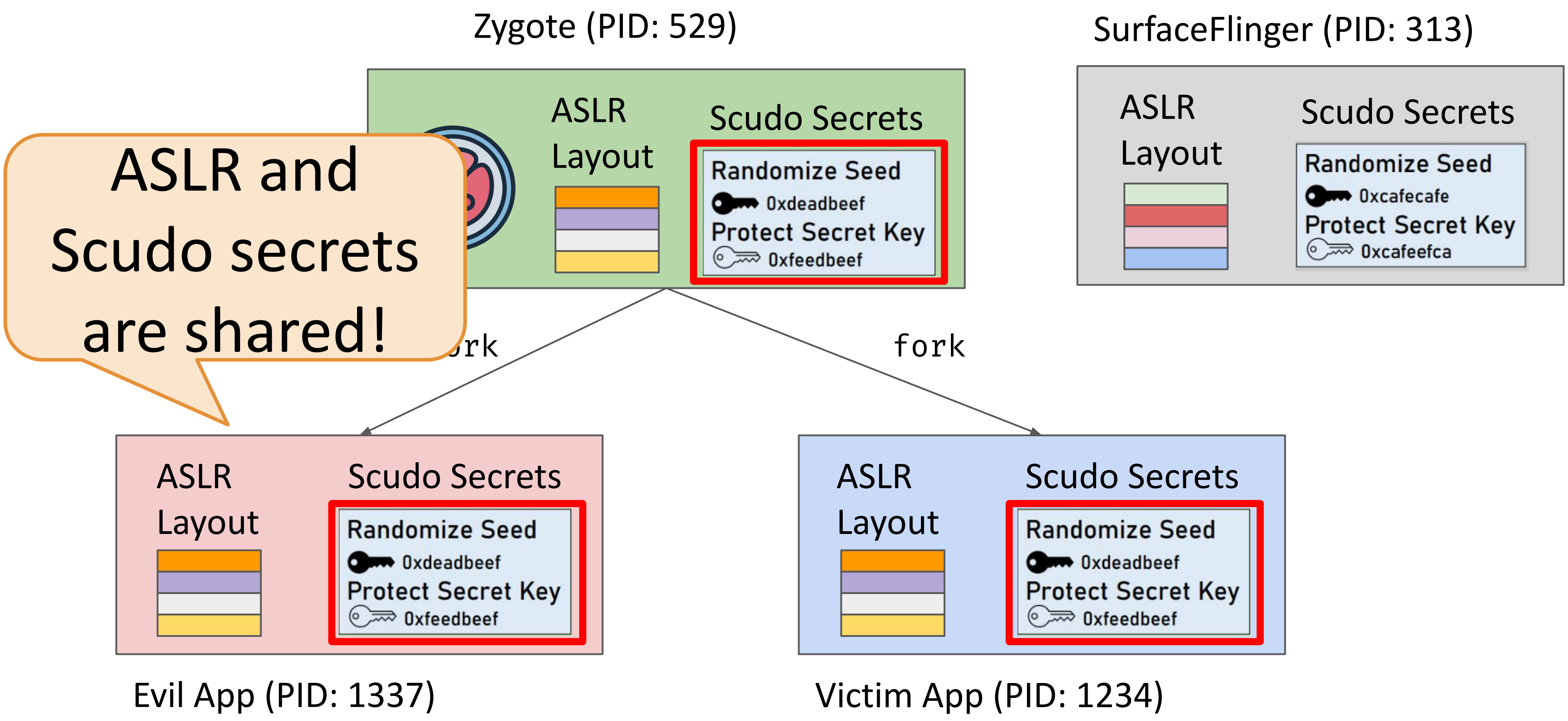


Protection: Scudo protects inline Heap Metadata

Chunk headers are signed, Scudo verifies the signature before parsing the metadata



Android's Performance Optimization Weakens Scudo



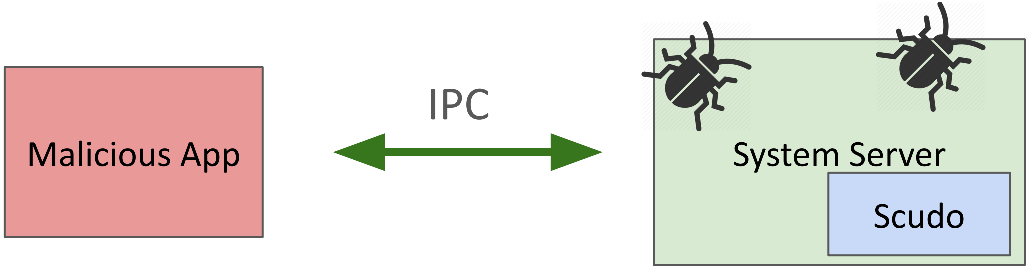
Feasible? Exploiting a Heap Underflow in the System Server

System Server is a highly privileged process, hosting multiple system services.

Apps interact with the system server over Binder IPC.

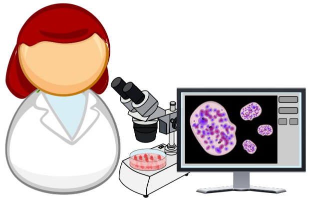
Backport CVE-2015-1528 to Android 14 (Heap overflow & underflow)

Use Forged Commitbase technique to allocate a chunk on the stack and hijack the PC (ROP)



Exploiting Android's Hardened Memory Allocator.
Philipp Mao, Elias Valentin Boschung, Marcel Busch, and Mathias Payer. In WOOT'24 (best paper)

hexhive



Software Testing
- Goal: prune bugs
- A tool for developers

Mitigation
- Goal: stop exploitation
- Last line of defense

Compartments
- Goal: least privilege
- Divide & conquer security


hexhive
EPFL
Join us on this research journey!

Android Security: A Moving Target

Android developed into a complex ecosystem 🤯

- Secure: per-app compartmentalization 👍
- Private: Sensitive data remains in the trusted world 👍
- Expected: Bugs in the hypervisor 🧙
- Unnecessary: Vulnerable communication APIs 🌍😵‍💫
- Terrible: forgetting rollback 🫖
- Naive: Unsafe allocators that create new attack surfaces 📚🔥

Lots of opportunities for research across the software stack!

Join us: https://hexhive.epfl.ch