Advanced Android Archaeology: Baffled By Bloated Complexity

Mathias Payer

EPFL hexhive

Android Complexity is Beyond Imagination

Over 3 billion users across 190 countries

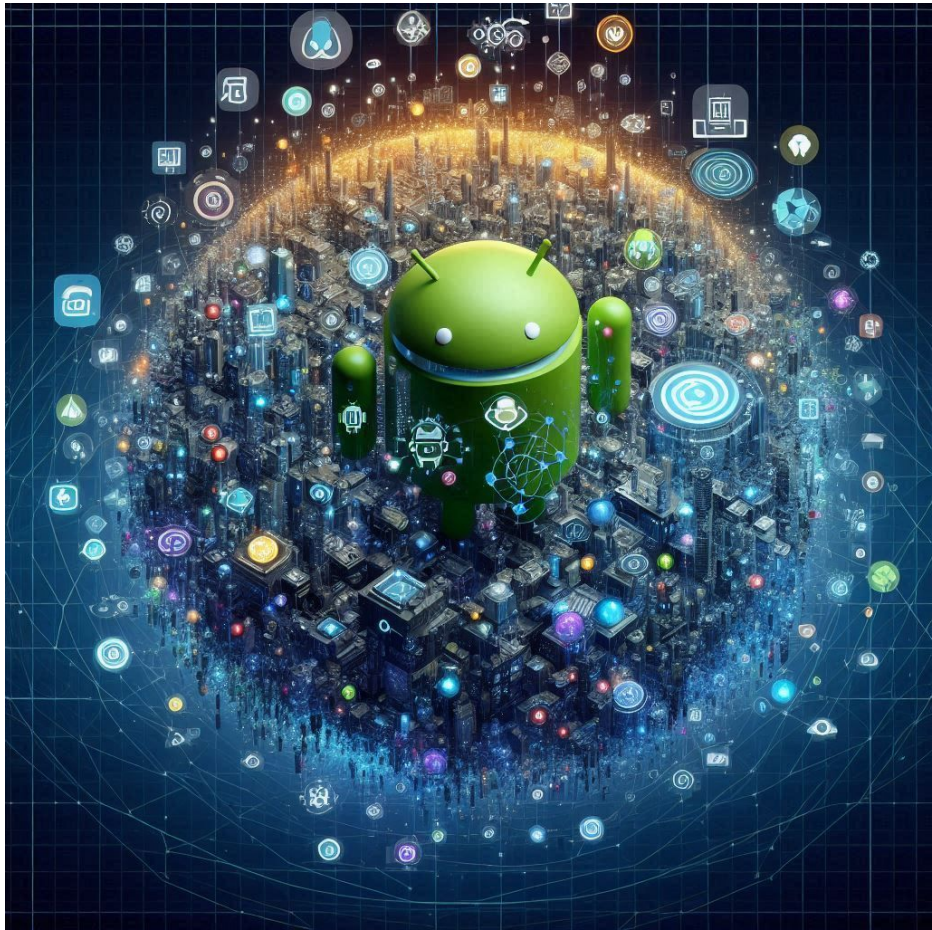
Almost ¾ market share for mobile phones

2.6mio apps in the App store

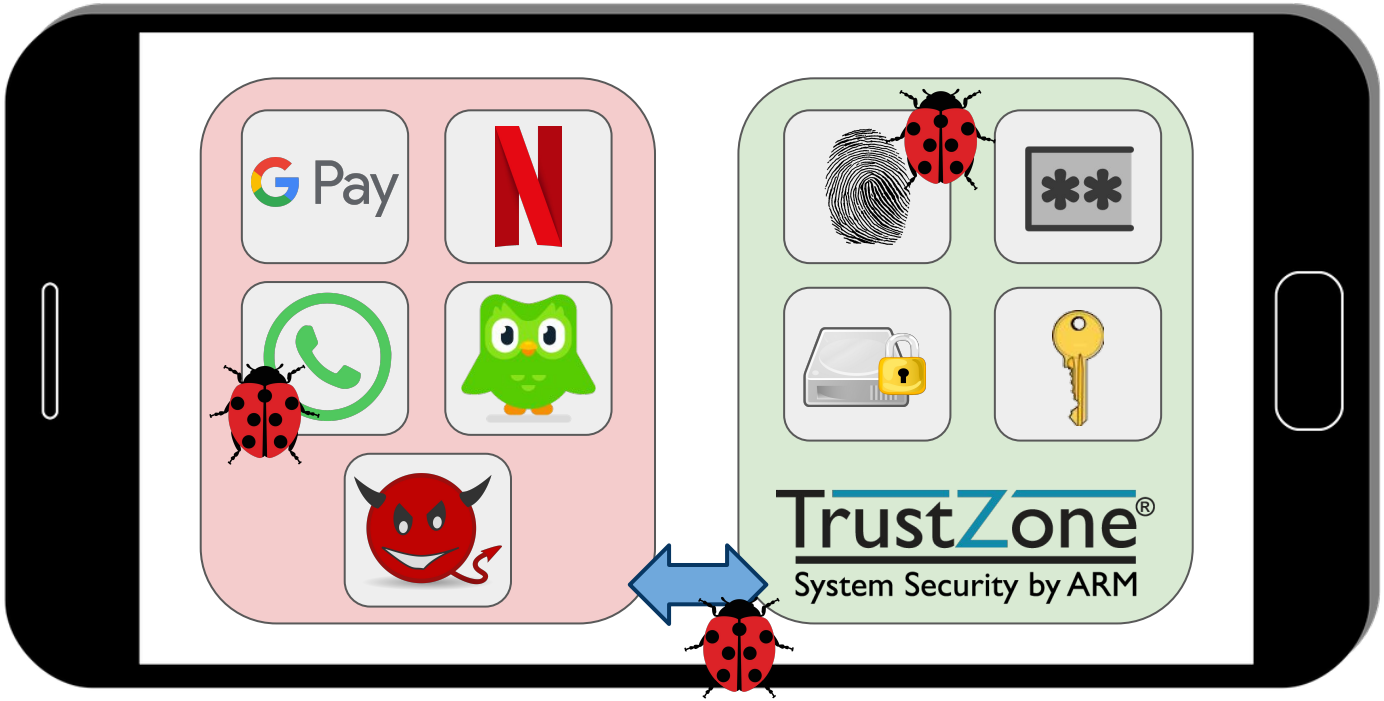
~1.5 billion devices sold per year

Several TB of system images

Roughly 11 TB of apps



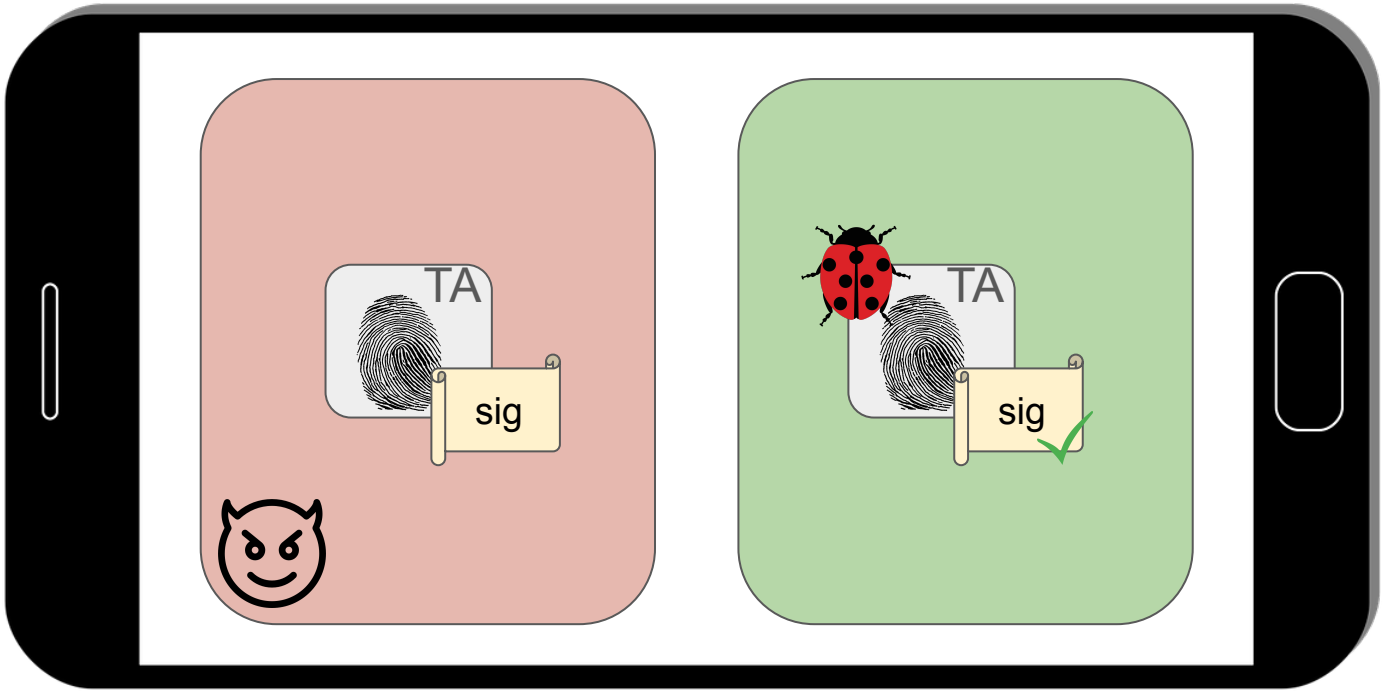
Android Architecture Overview




Physical Map Design by Chris Taylor
Political Designations by Chris Guarette
Based Upon the Works of J.R.R. Tolkien
ICE BAY of FOROCHEL
FORODWAITH
ANGMAR
ERED MITHRIN (Grey Mountains)
IRON HILLS
ERIADOR
RHUDAUR
ARNOR
CARDOLAN
FORLINDON
HARLINDON
ERED LUIN
HOLLIN
LORIEN
RHÛN
SEA of RHÛN
ROHAN
FANGORN
EMYN MUIL
ERED LITHUI (Ash Mountains)
ANFALAS (Langstrand)
ANDRAST (Ras Morthil)
BELFALAS
BAY OF BELFALAS
TOLFALAS
LEBENNIN
SOUTH ITHILIEN
SOUTH GONDOR
KHAND
NEAR HARAD
BELEGAER (The Great Sea)
P.
M.
MIDDLE–EARTH
in the Third Age

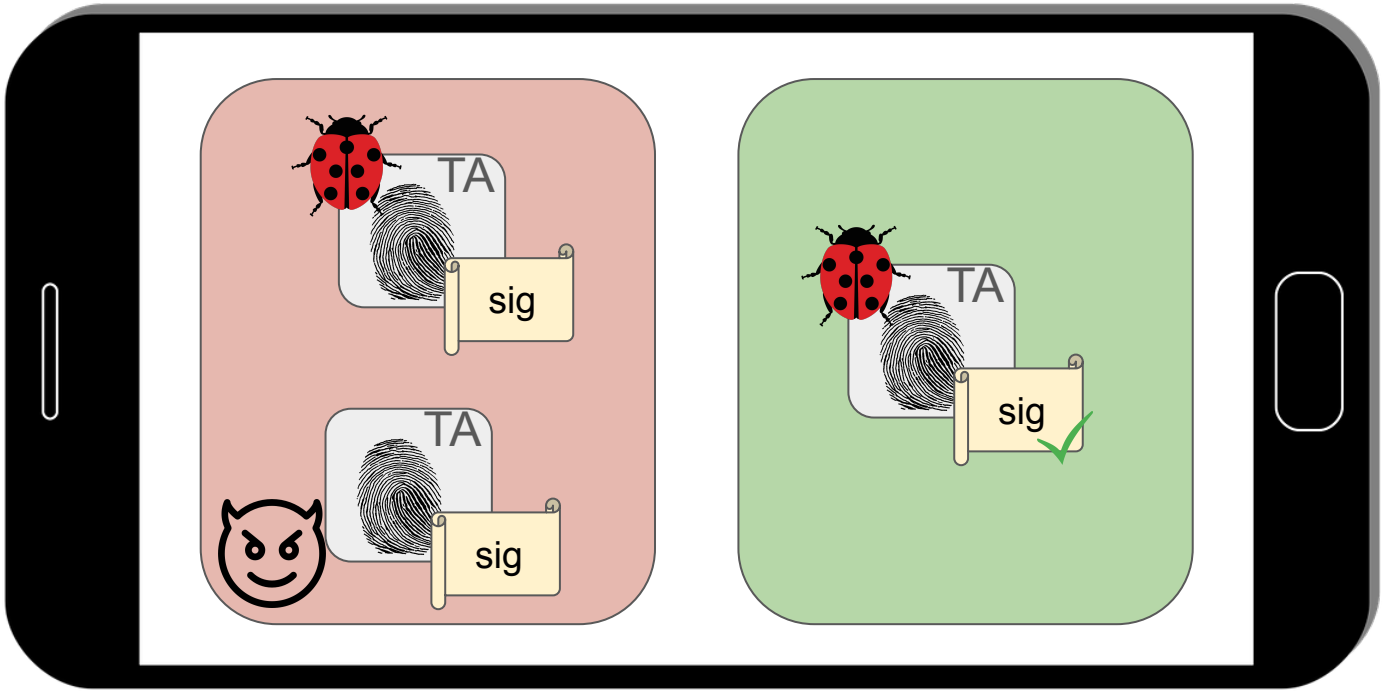
Cheesing Android Trusted Applications

Trusted Applications



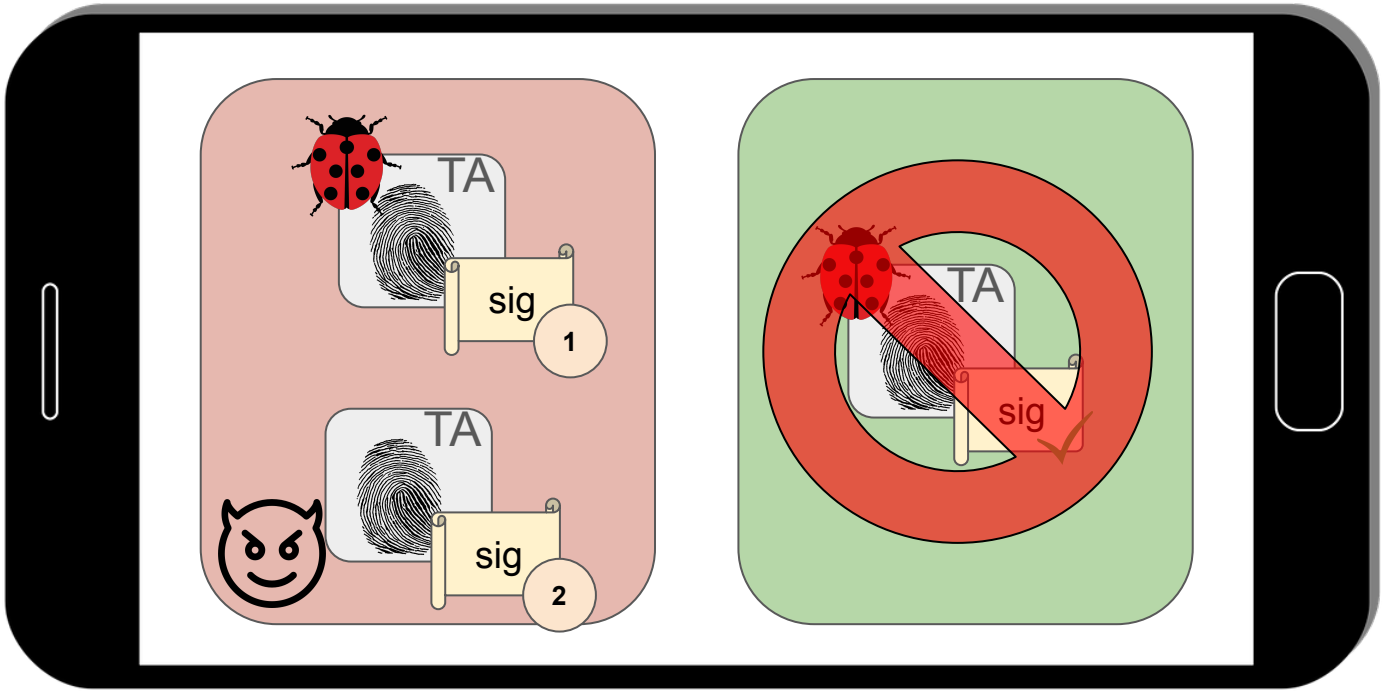
Trusted Applications are authentic dynamically-loadable modules

TA Rollback Attacks



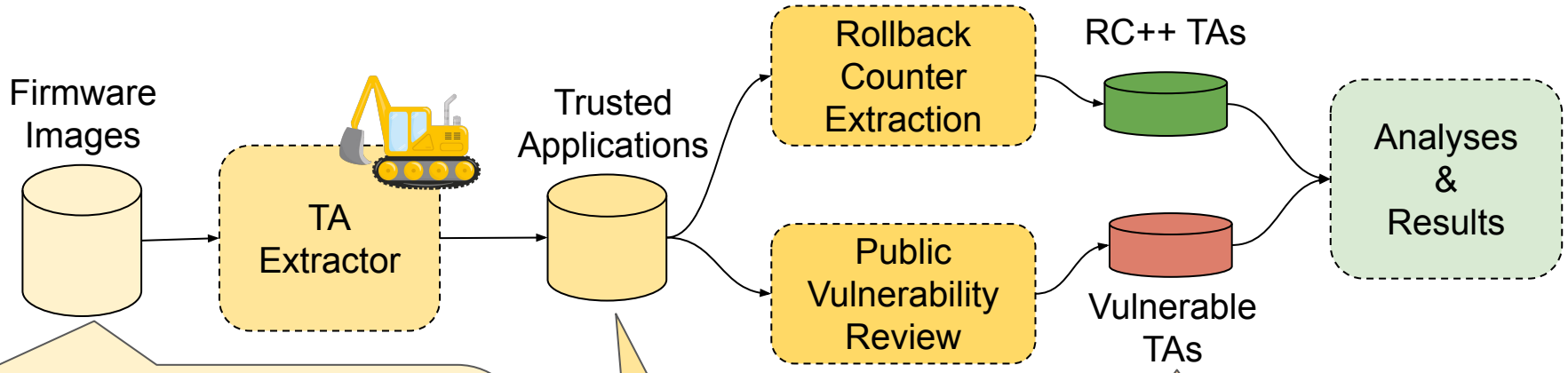
TA Rollback Attacks exploit the authenticity of old and vulnerable TAs

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

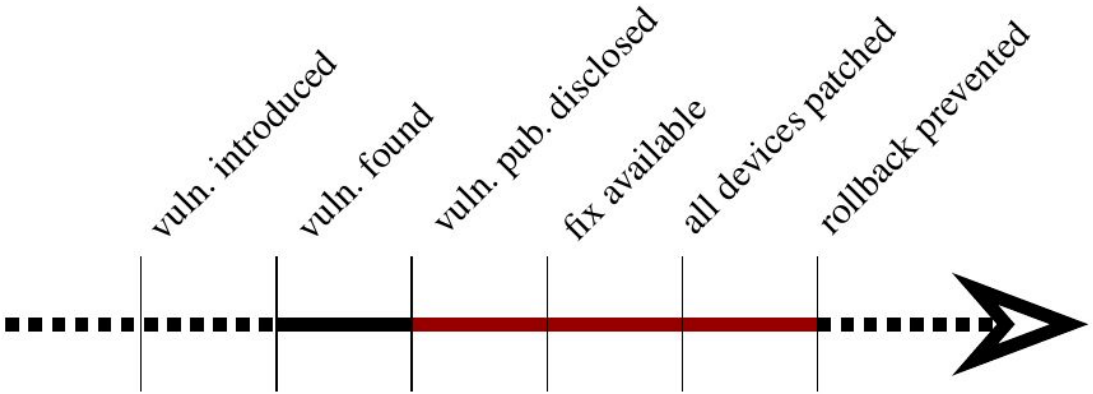
- 2,582 vulnerable TAs
- 190 rollback counter usages

Spill the TeA: Summary

TA rollback prevention is incomplete with questionable TA vulnerability practices

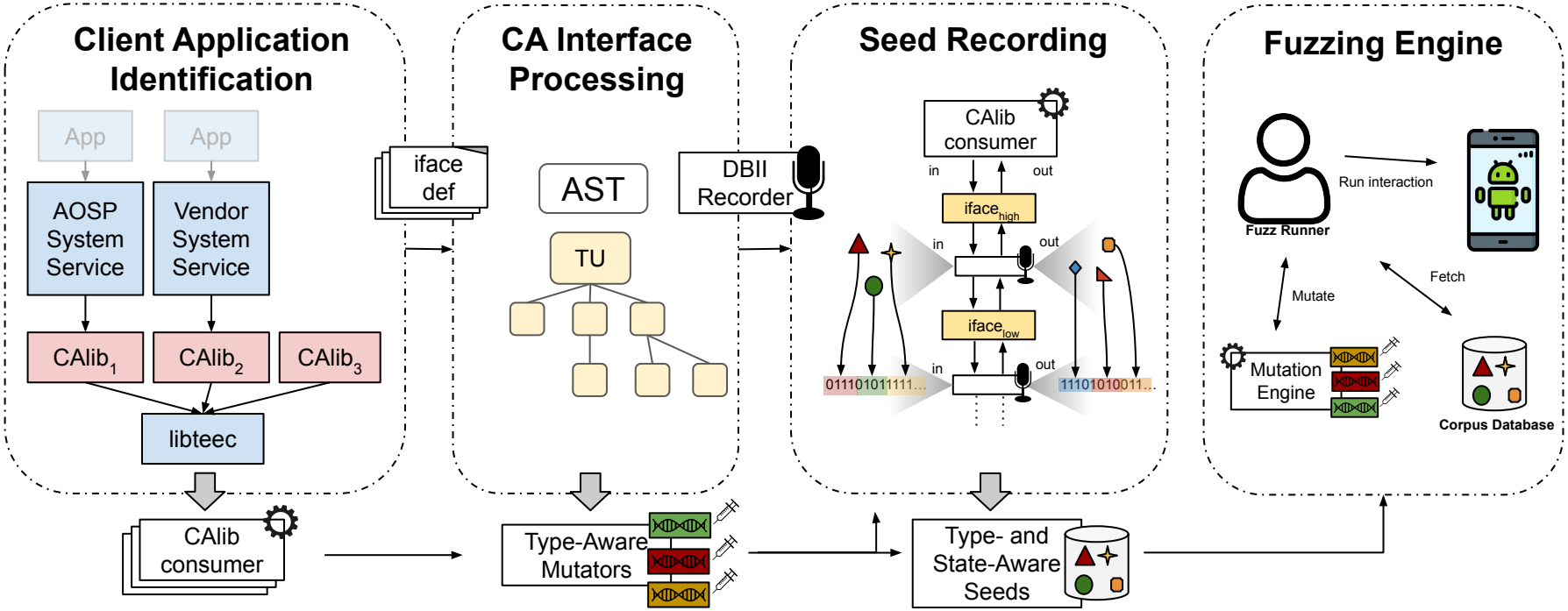
- Internally patched TAs (without disclosure/rollback prevention)
- Security patches limited to one product, not shared across targets

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

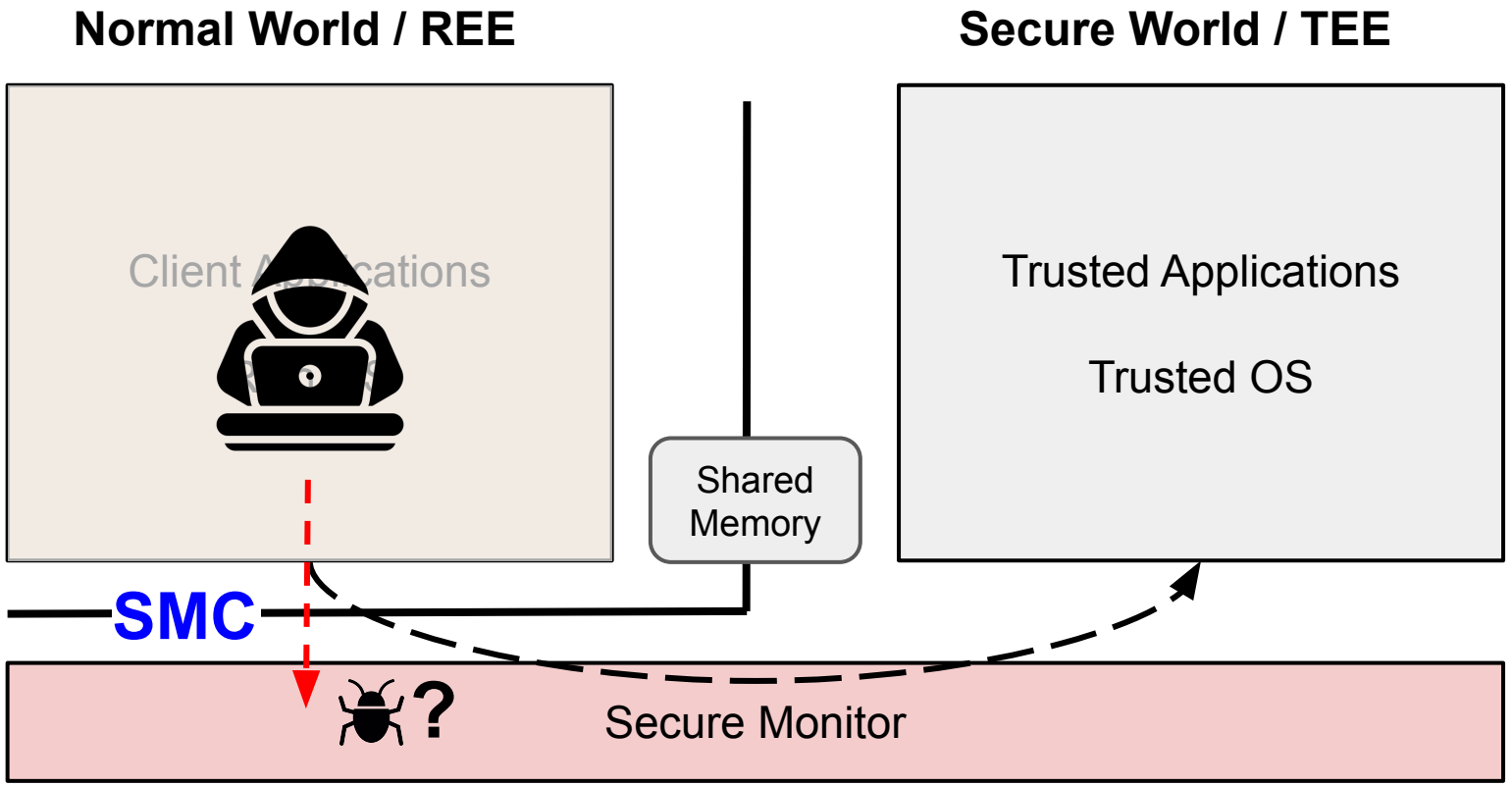


TEEzz: Fuzzing Trusted Applications on COTS Android Devices.
Marcel Busch, Mathias Payer, Aravind Machiry, Christopher Kruegel, Giovanni Vigna, and Chad Spensky. In Oakland'23


The dwarves delved too greedily and too deep. You know what they awoke in the darkness of Khazad-dum... shadow and flame.
EL3XIR: 🧙
Be Greedy and Dig Deep

ARMv8-A TrustZone

arm TRUSTZONE



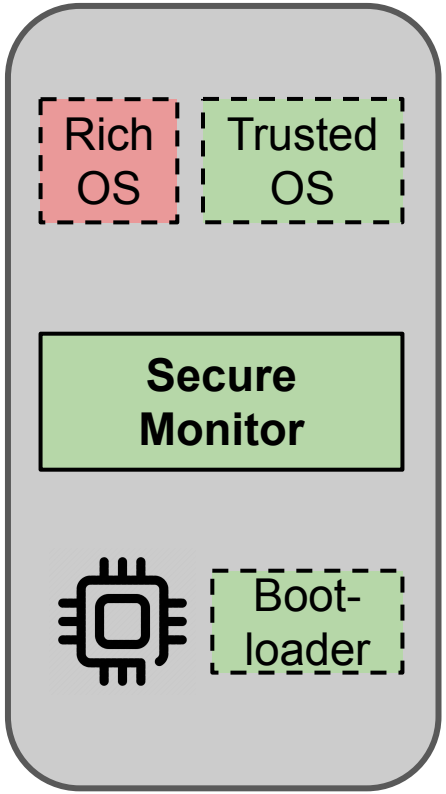
Fuzzing Secure Monitors - Challenges

C1 Limited Introspection

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

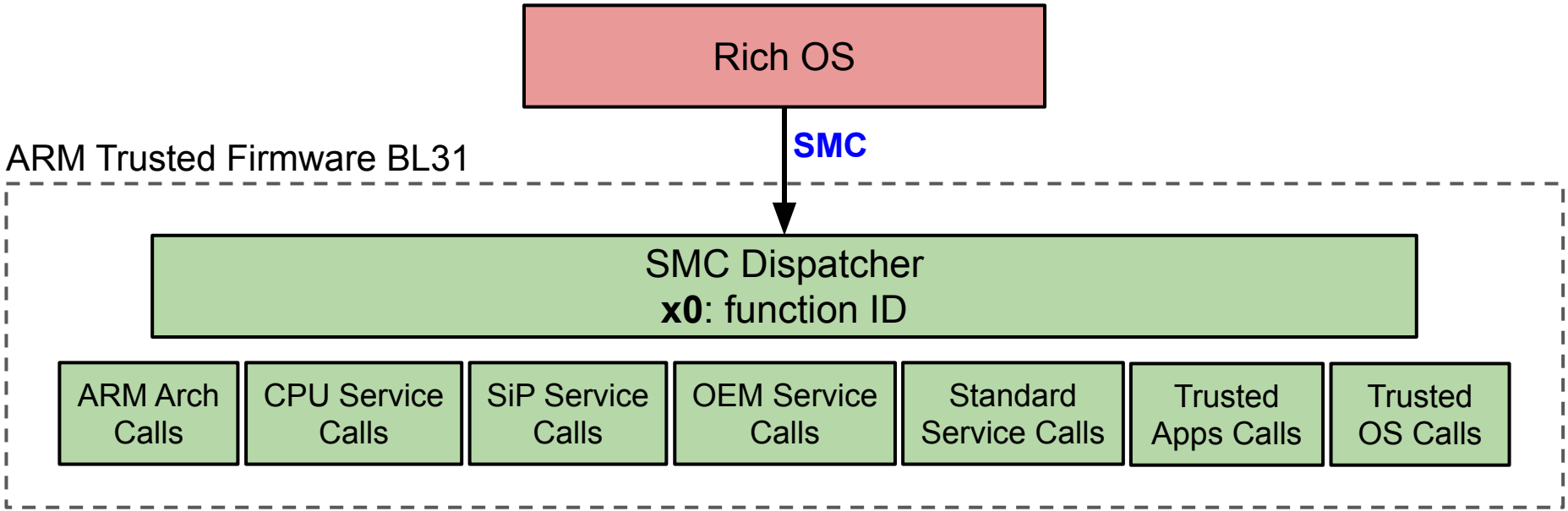
C1.1 Dependency on Software Components

C1.2 Infeasibility of Manual Peripheral Modeling



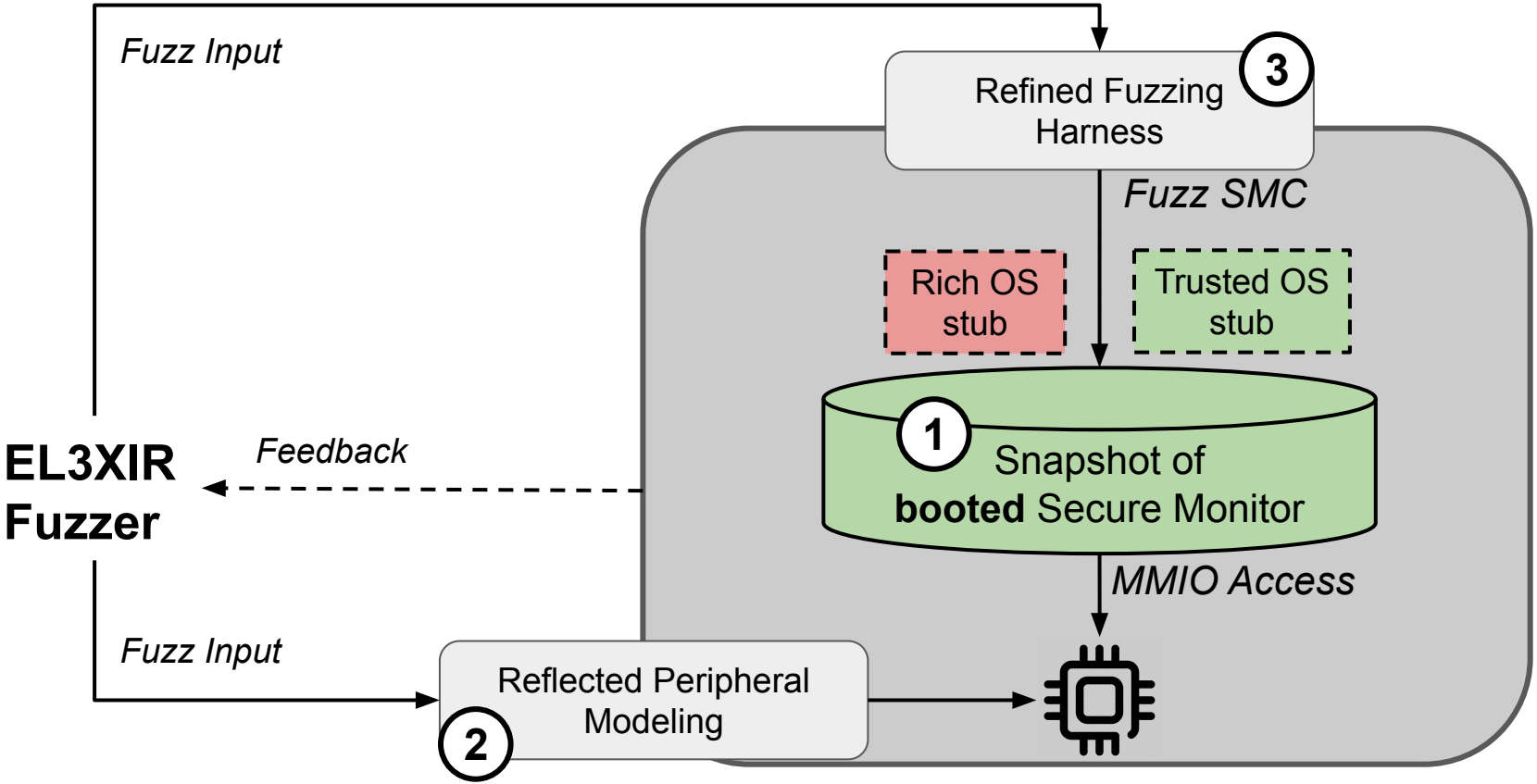
Fuzzing Secure Monitors - Challenges

C2 Complex Input Space



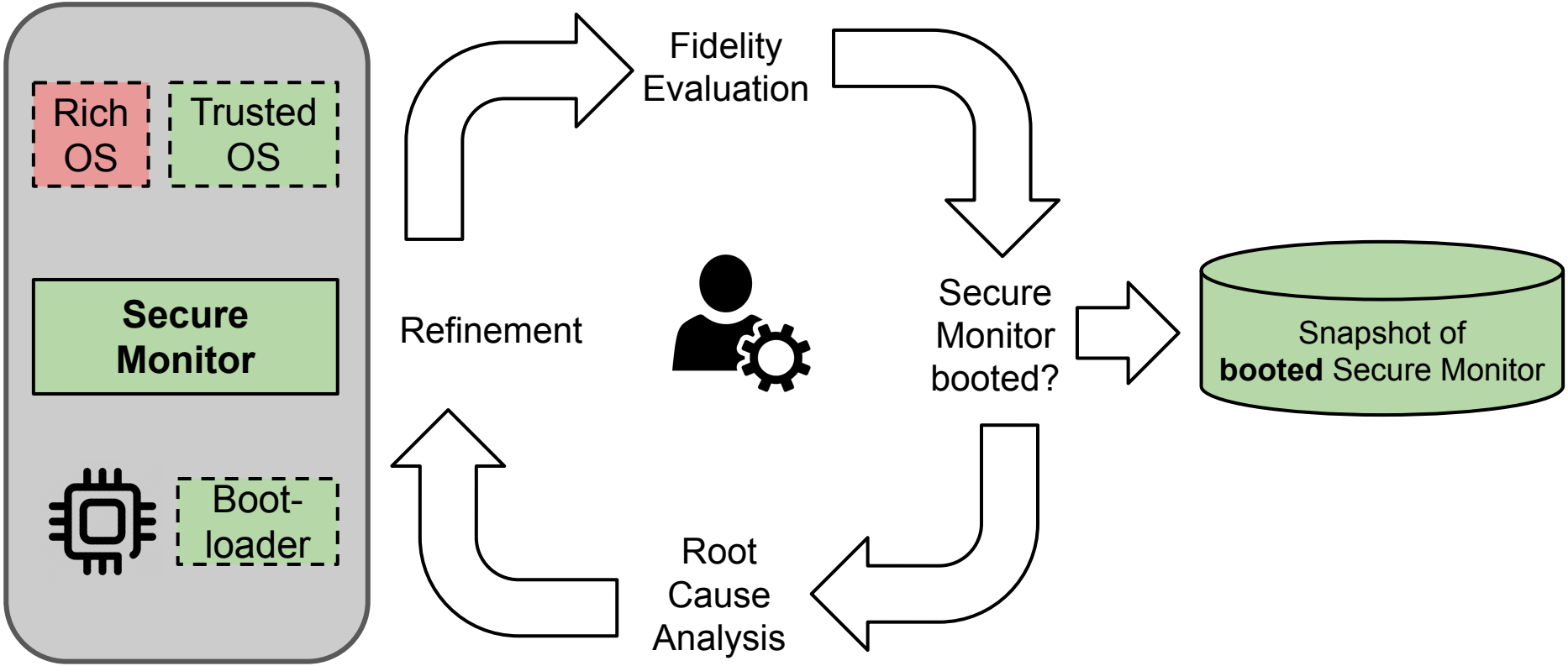
Several tens of runtime services with unique APIs…

EL3XIR's Approach - Overview



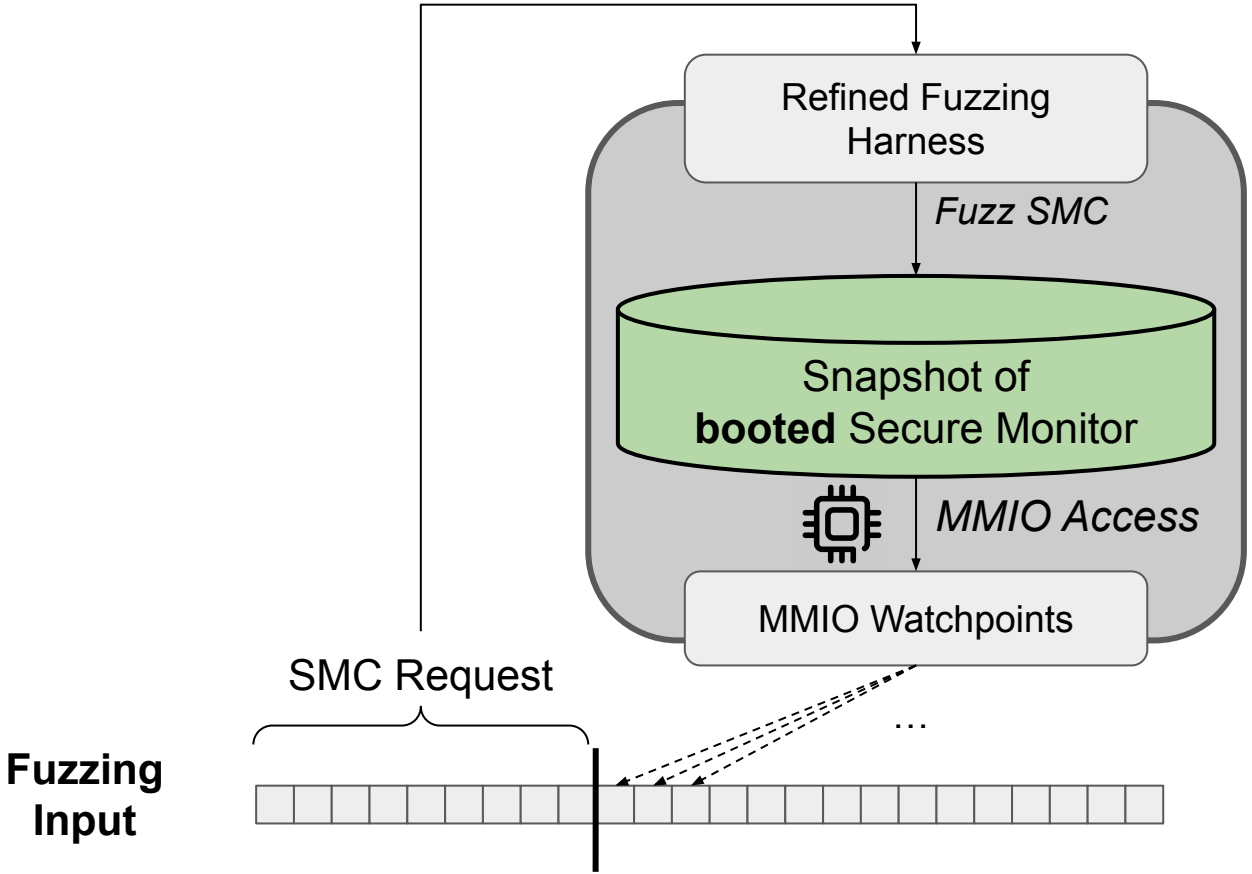
Contribution ① Partial-Rehosting of Secure Monitors

C1.1 Dependency on Software Components



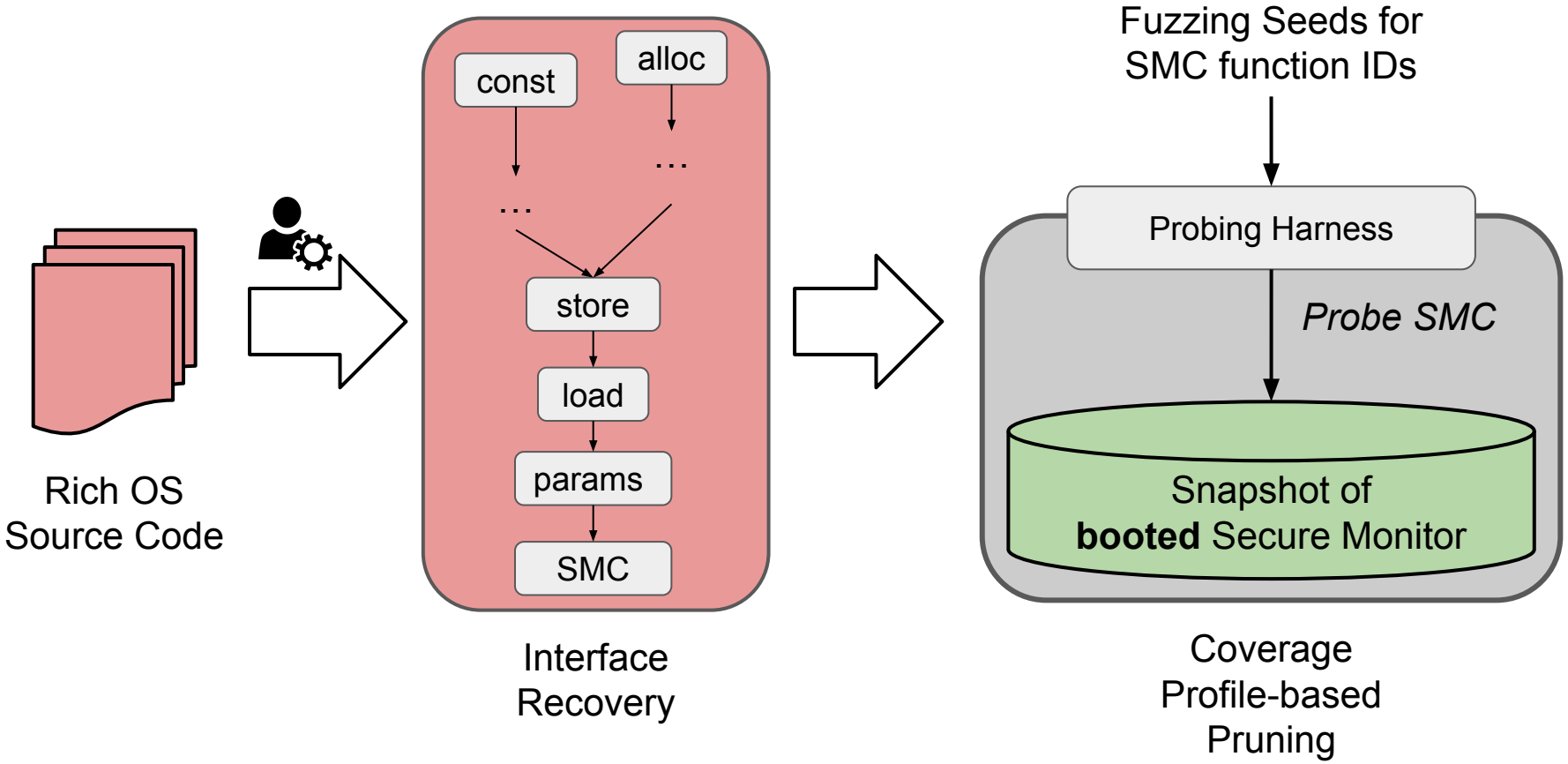
Contribution ② Reflected Peripheral Modeling

C1.2 Infeasibility of Manual Peripheral Modeling



Contribution ③ Harness Synthesis

C2 Complex Input Space



Evaluation - Bugs and CVEs

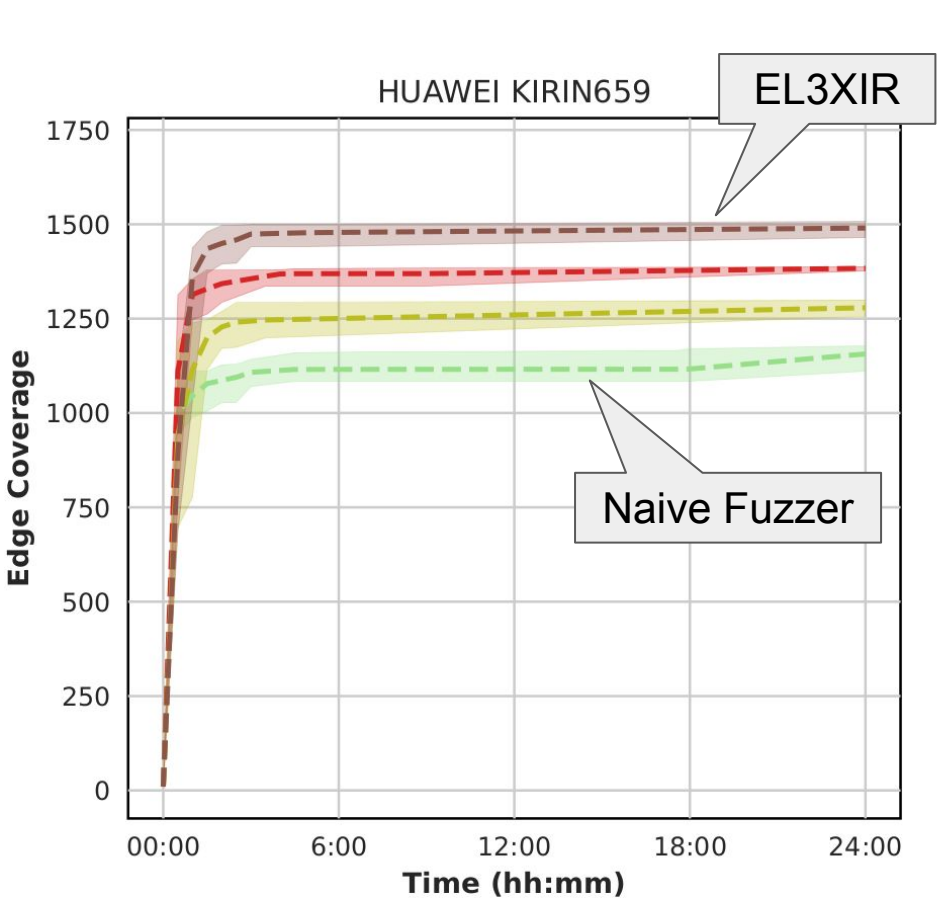
7 targets from 6 different vendors
- 4 open-source, 3 closed-source

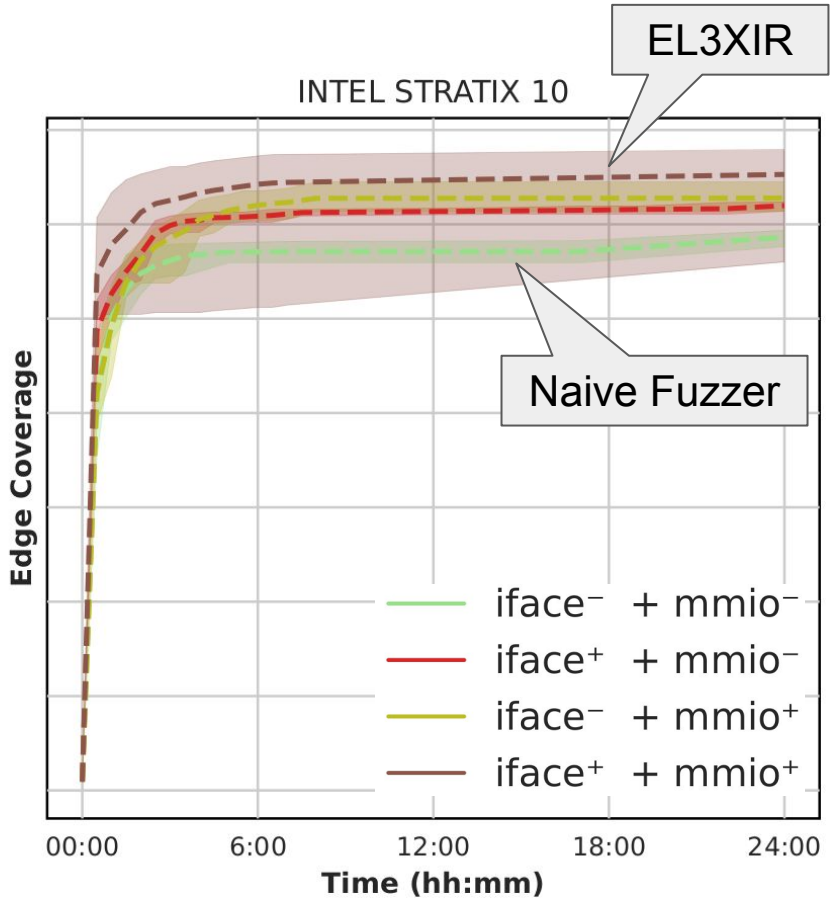
EL3XIR triggered 34 bugs (**17** security relevant) in 5 targets
- Naive baseline comparison triggered 19 bugs (**10** security relevant)

Responsible disclosure resulted in 6 CVEs plus 11 confirmed bugs

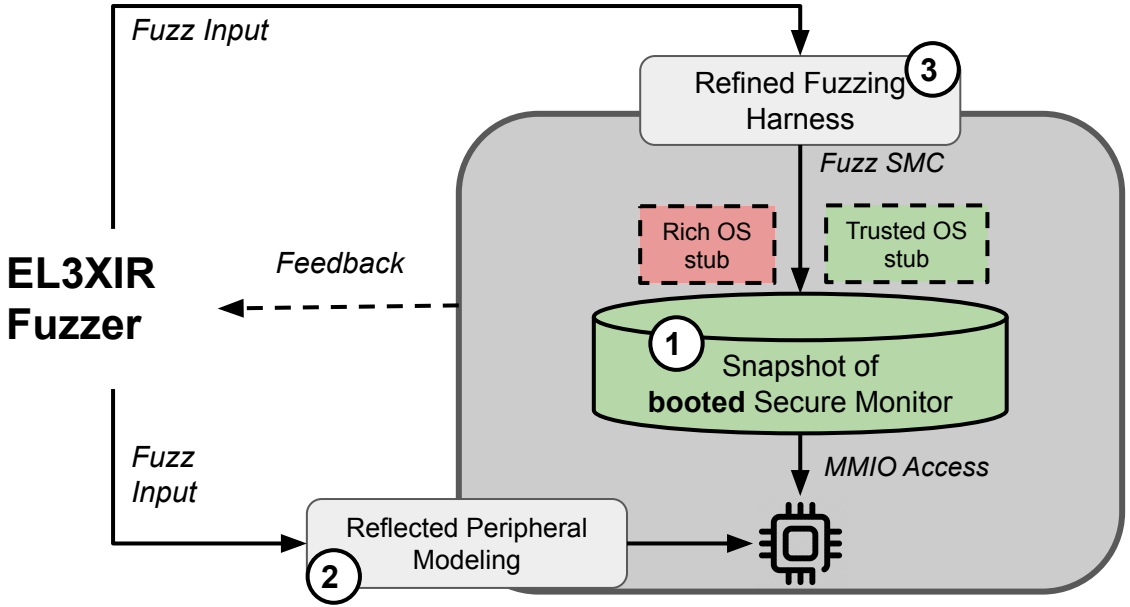
CVE-2022-38787, CVE-2023-22327 (5 different bugs), CVE-2023-49614, CVE-2024-22390, CVE-2023-31339, CVE-2023-49100

Evaluation - Coverage





EL3XIR: Fuzzing COTS Secure Monitors



Rehosting Framework for proprietary TrustZone Firmware

Highly automated Fuzzing Pipeline including Harness Synthesis and Peripheral Modeling

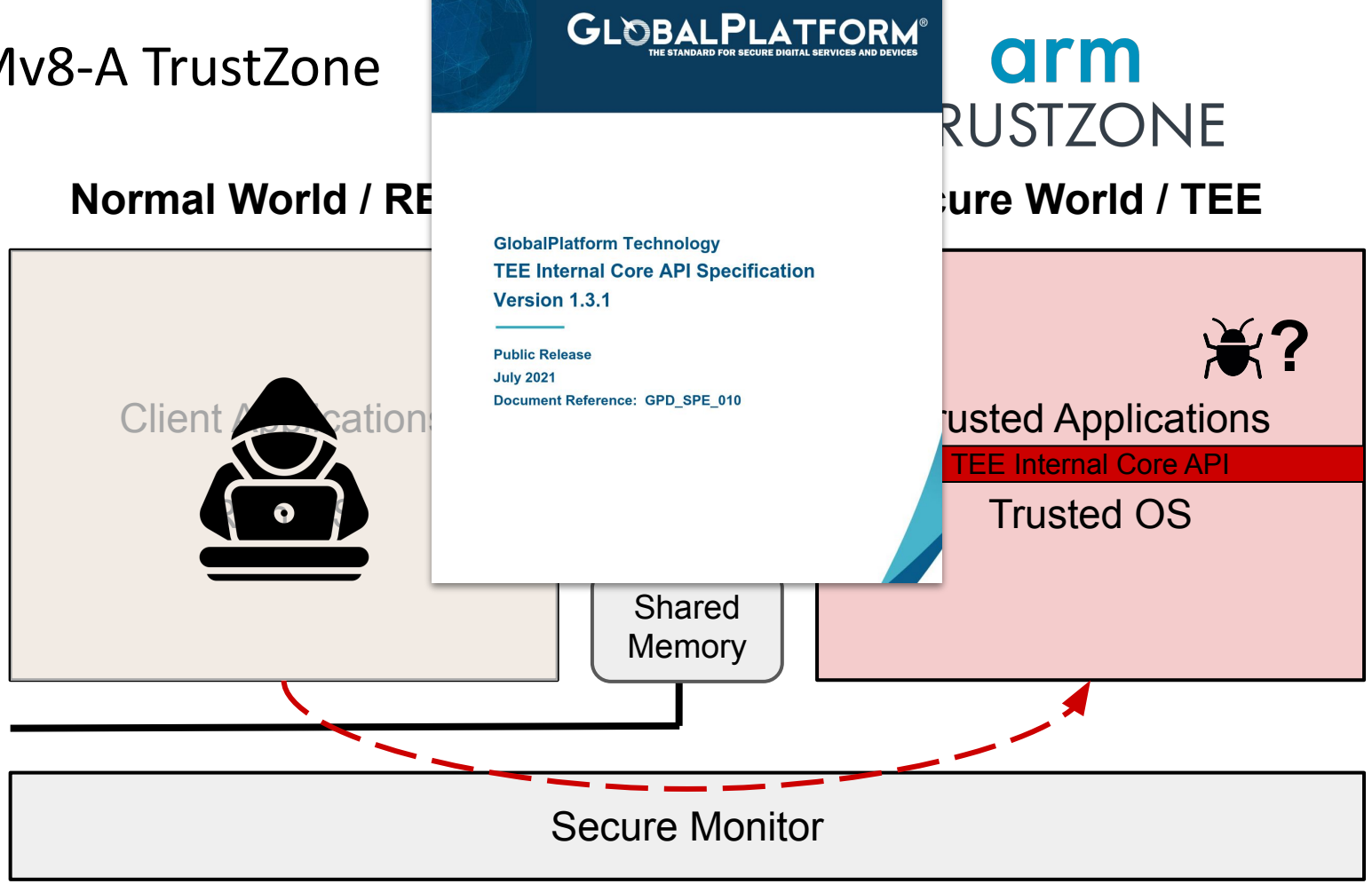
Fuzz your own Secure Monitor

github.com/HexHive/EL3XIR

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


Gandalf: A palantir is a dangerous tool, Saruman.
Saruman: Why should we fear to use it?
Gandalf: They are not all accounted for, the lost seeing stones. We do not know who else may be watching
🌍😵 GlobalConfusion
Test Android Trusted Apps

ARMv8-A TrustZone



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

Stores session state

OPTIONAL

Chooses TA cmd handler

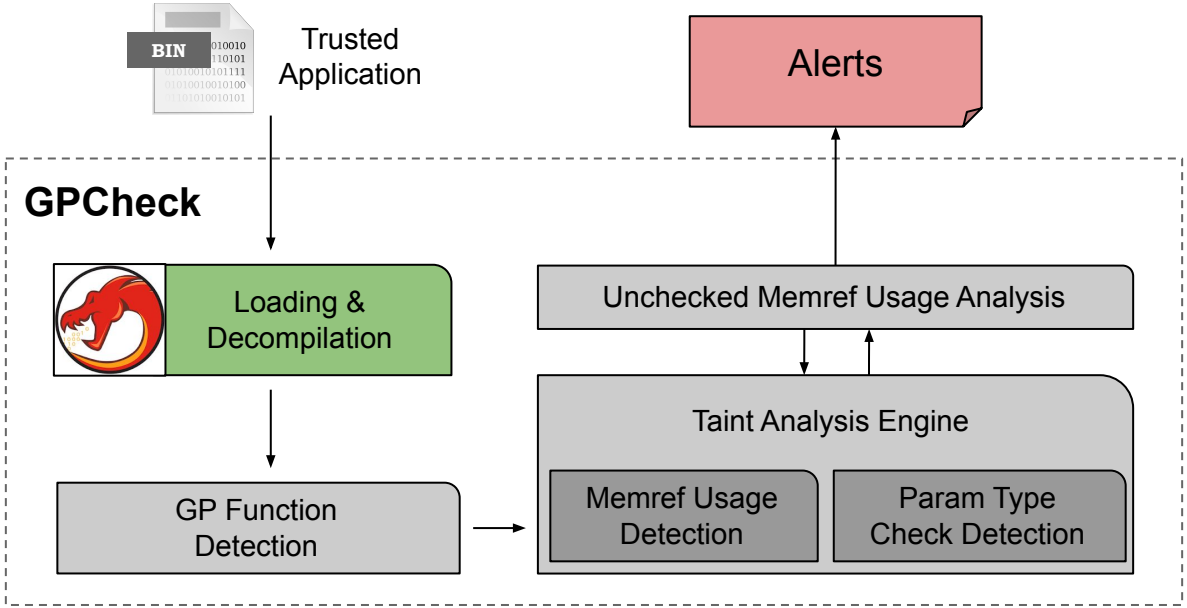
Determines types of params. If the type is a memref, then the runtime system validates the buffer location

Four TEE_Param parameters

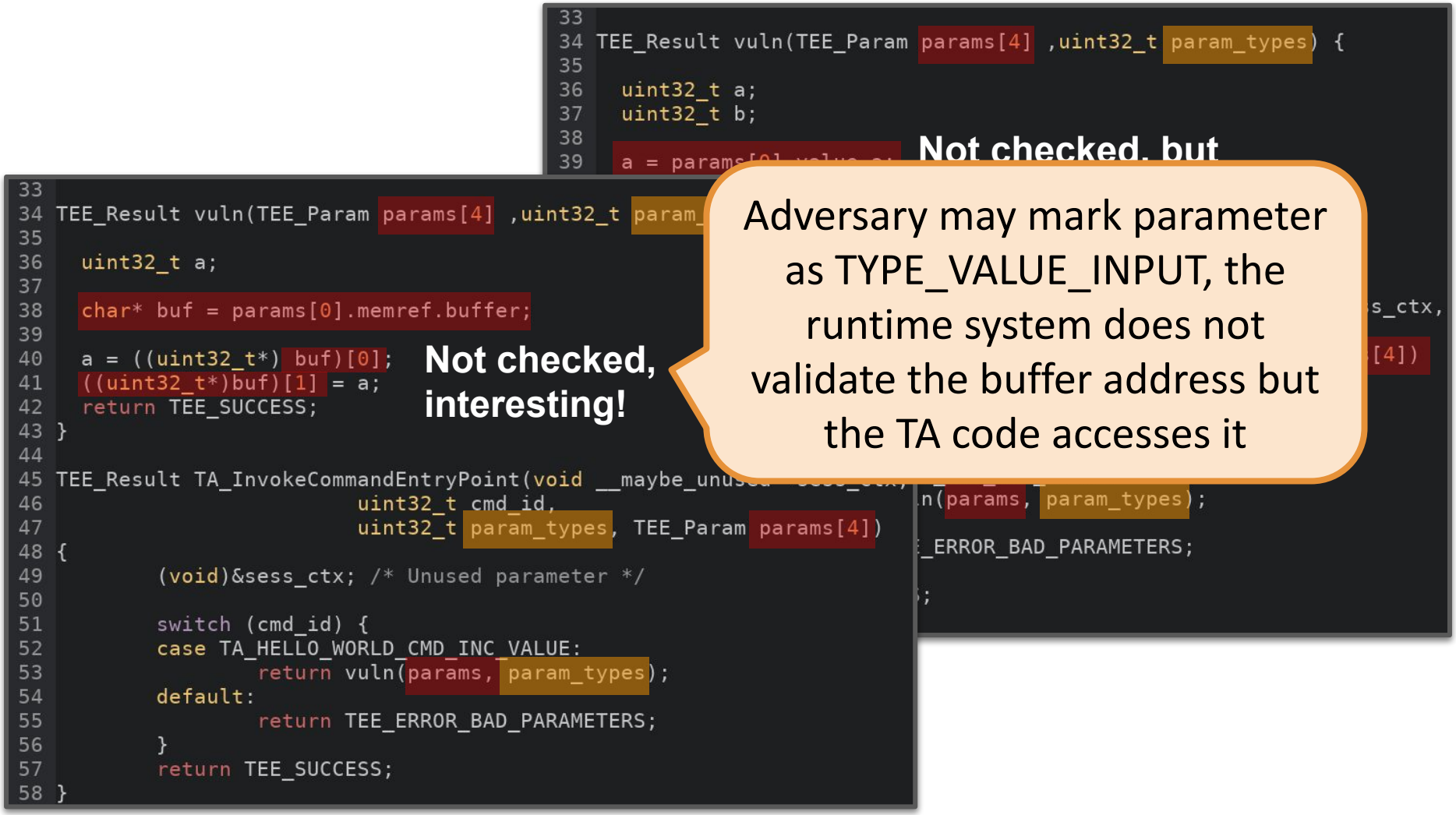
```
typedef union {
        struct {
                void *buffer;
                uint32_t size;
        } memref;
        struct {
                uint32_t a;
                uint32_t b;
        } value;
} TEE_Param;
```

GPCheck

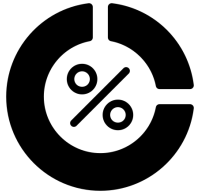
- Ghidra-based
- Post-production binary analysis/check
- Open-Source



https://github.com/HexHive/GlobalConfusion


TEE_Result vuln(TEE_Param params[4] ,uint32_t param_types) {
uint32_t a;
uint32_t b;
Not checked, but
TEE_Result vuln(TEE_Param params[4] ,uint32_t param_
uint32_t a;
char* buf = params[0].memref.buffer;
a = ((uint32_t*) buf)[0];
((uint32_t*)buf)[1] = a;
return TEE_SUCCESS;
}
Not checked, interesting!
Adversary may mark parameter as TYPE_VALUE_INPUT, the runtime system does not validate the buffer address but the TA code accesses it
TEE_Result TA_InvokeCommandEntryPoint(void __maybe_unused
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;
}

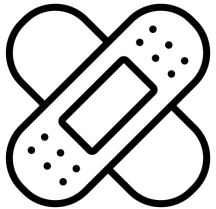
Let’s Scan All Apps in the TA Ecosystem!



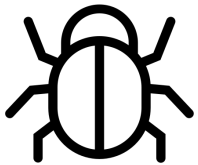
~6,900 TAs are GP-compliant (~131 unique TAs)

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

Change fail-open to fail-close design

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

GlobalPlatform is changing their API, making checks explicit

Sent proposal to GP; Draft for API update in progress

No changes to external API (backwards compatible)

Open-source and based on OPTEE

GlobalConfusion: TrustZone Trusted Application 0-Days by Design.
Marcel Busch, Philipp Mao, and Mathias Payer. In SEC'24

📚🔥 Just Slap a Secure Allocator On It

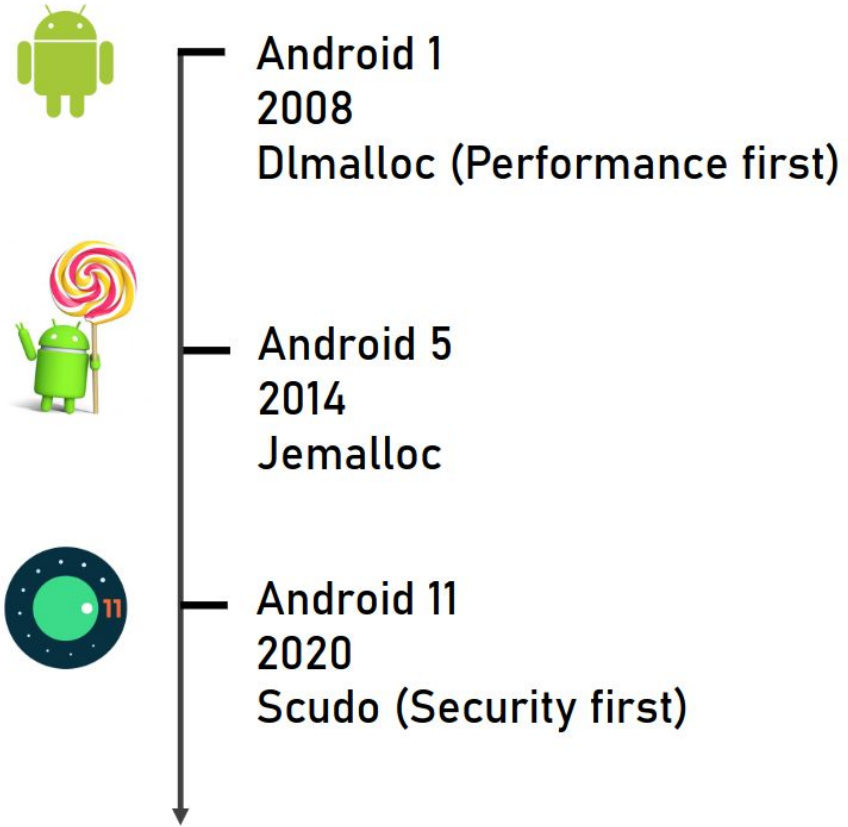
Scudo: the Hardened Memory Allocator

Scudo is..

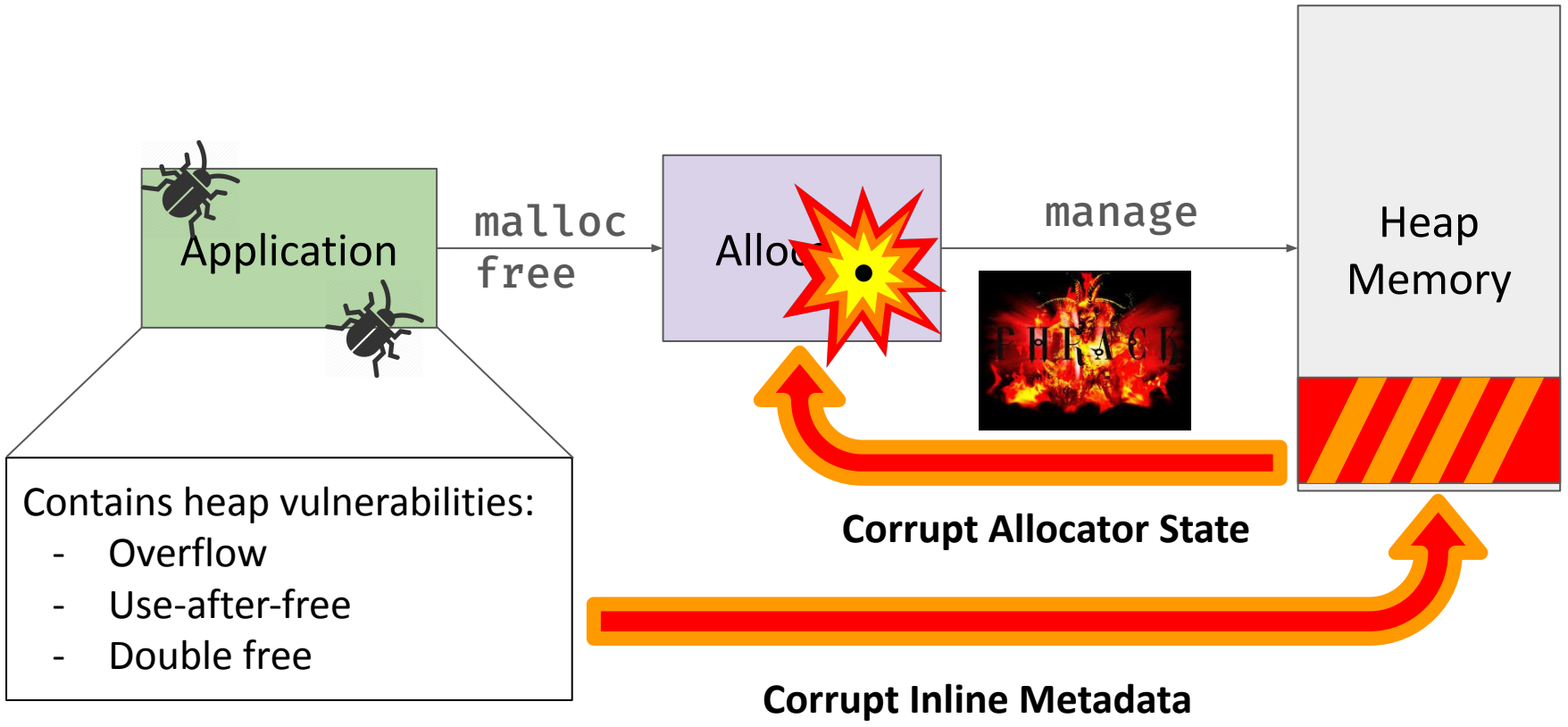
… a userspace memory allocator

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

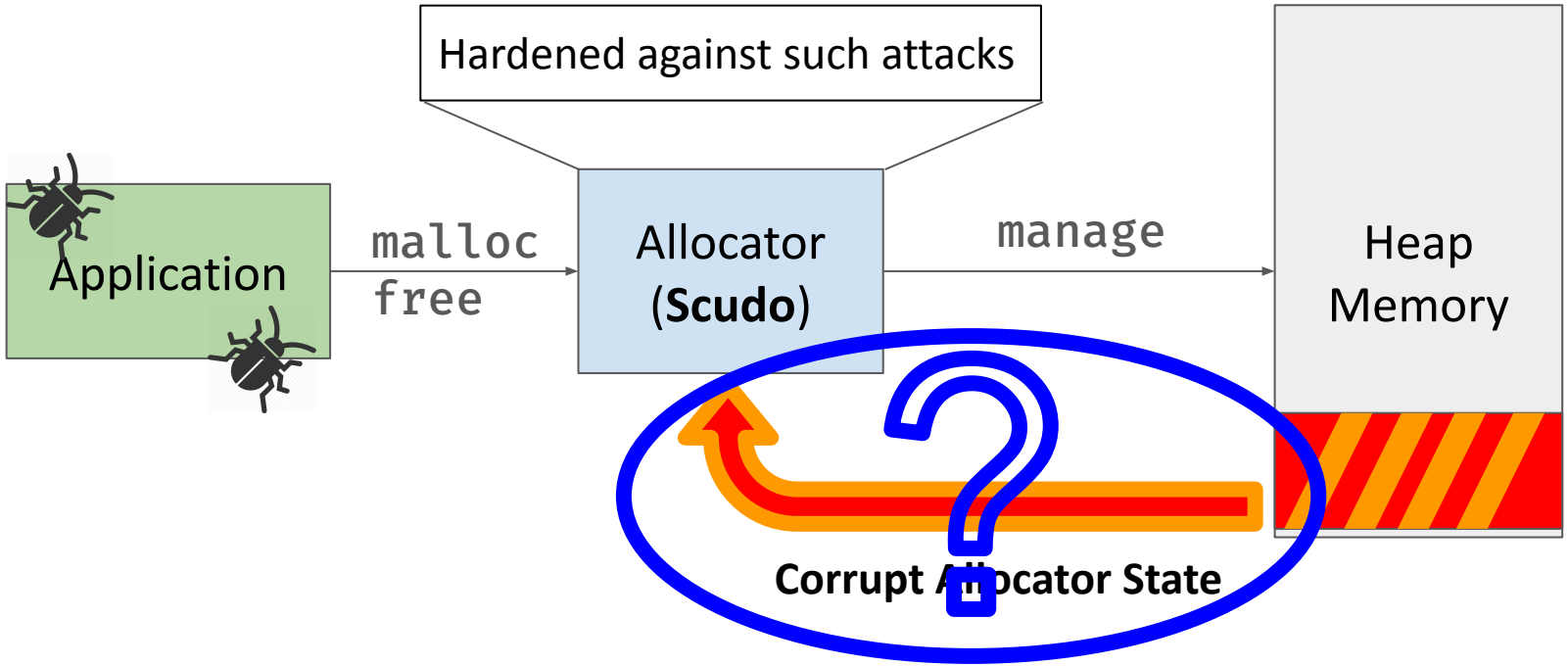
But is it secure?



Exploiting the Allocator



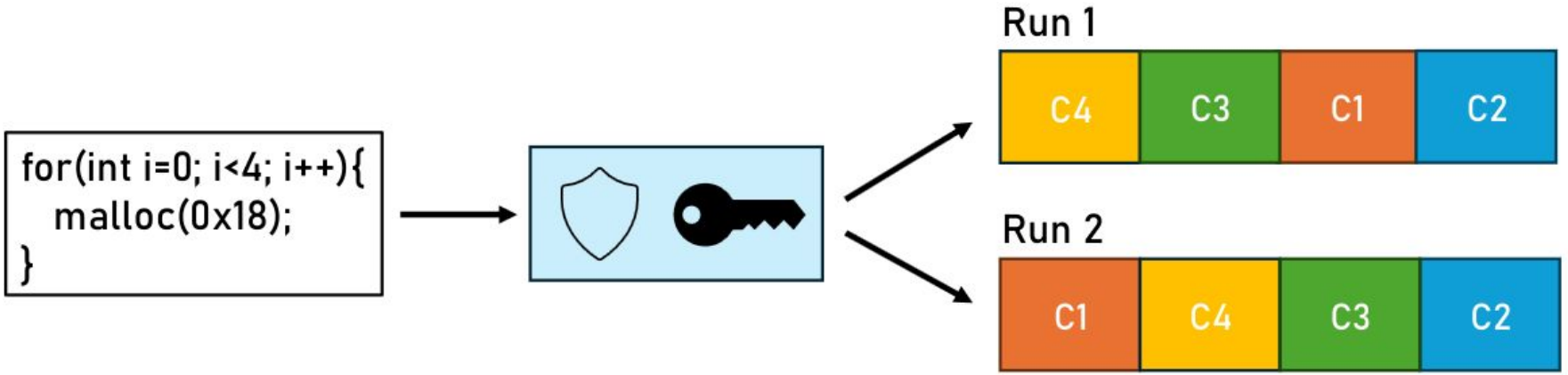
Is Exploiting the Allocator still possible for Scudo?



Threat Model: Able to corrupt heap memory

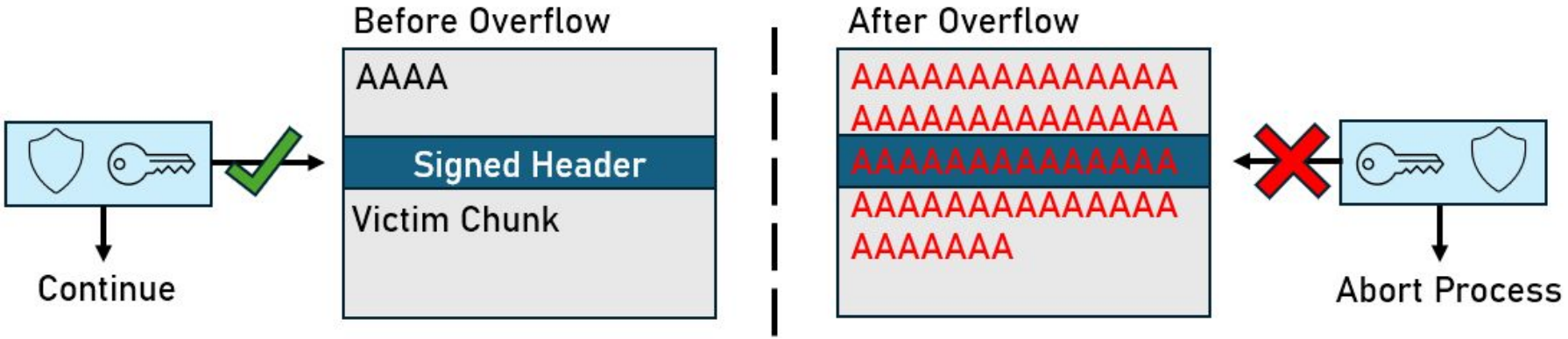
Randomization: Scudo Randomizes the Address of Allocations

Prevent attackers from arranging the heap in a particular layout.

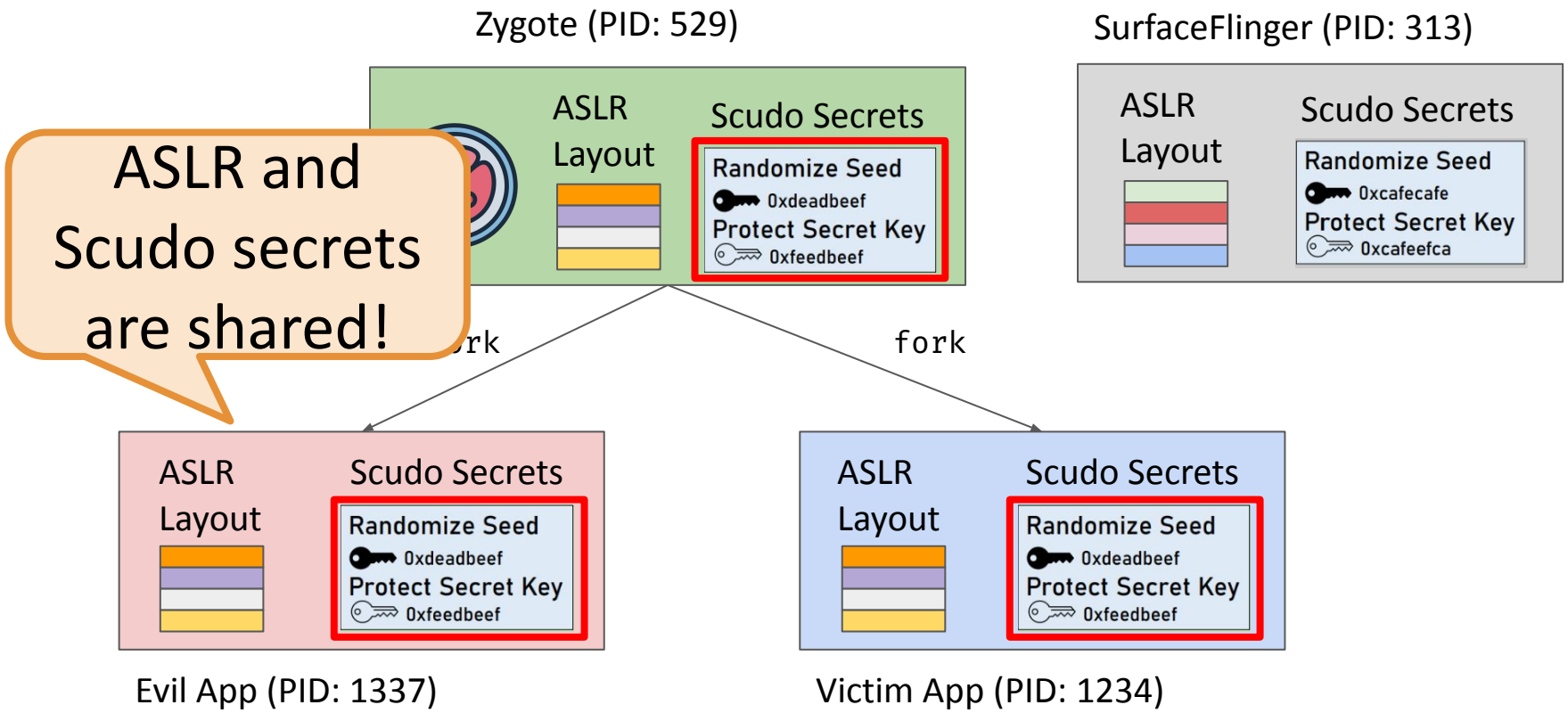


Protection: Scudo protects inline Heap Metadata

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



Android's Performance Optimization Weakens Scudo



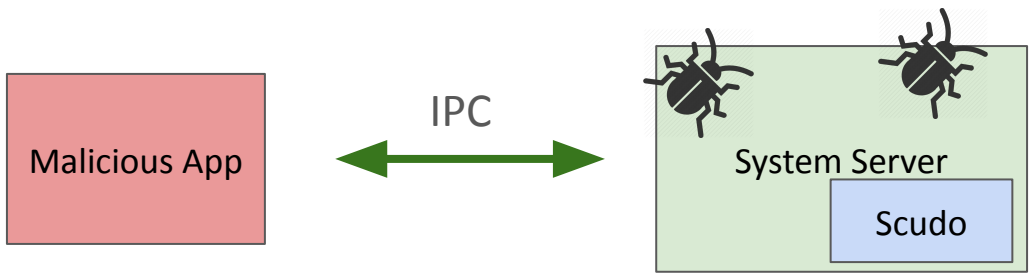
Feasible? Exploiting a Heap Underflow in the System Server

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

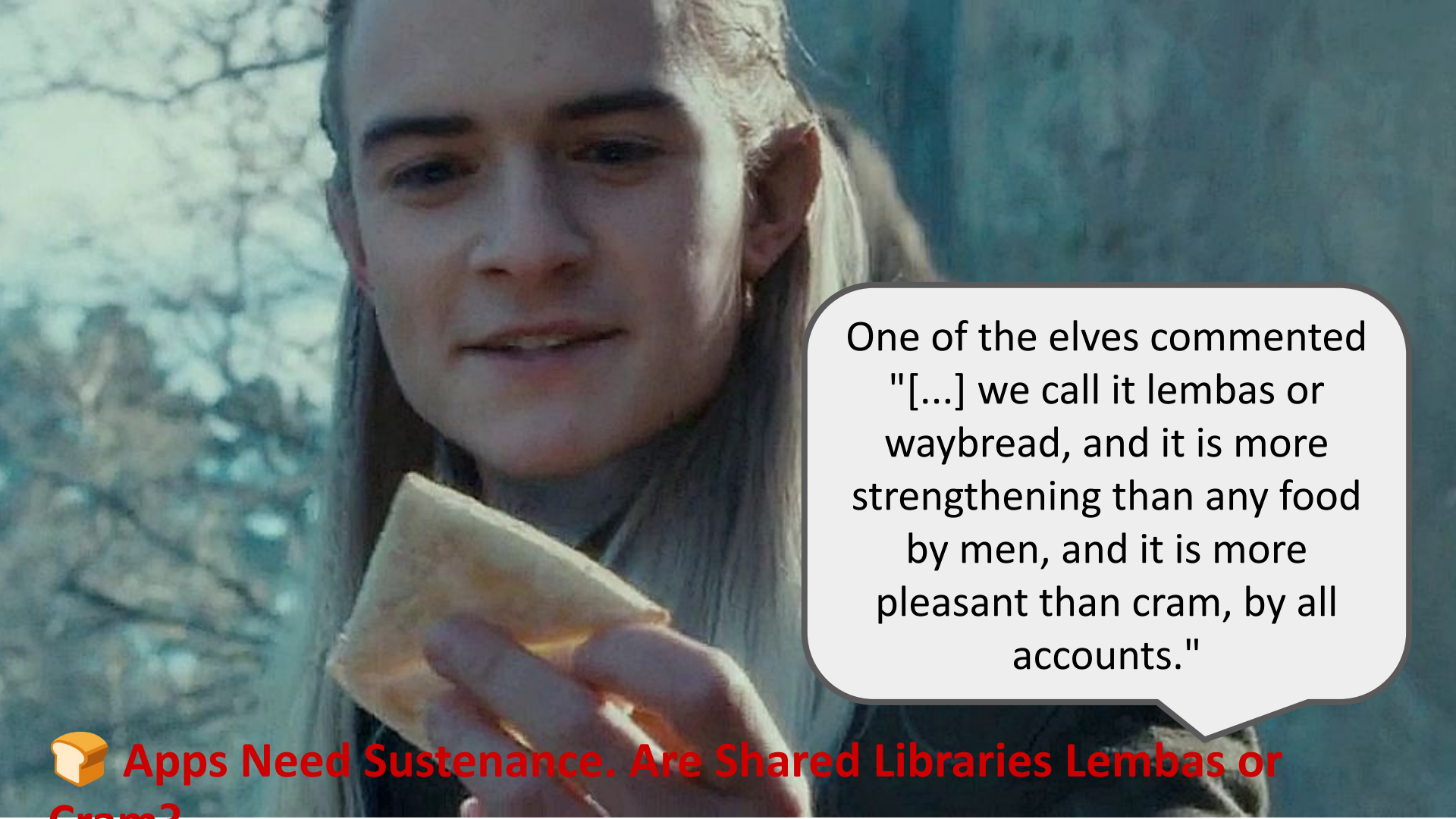
Apps interact with the system server over Binder IPC.

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)

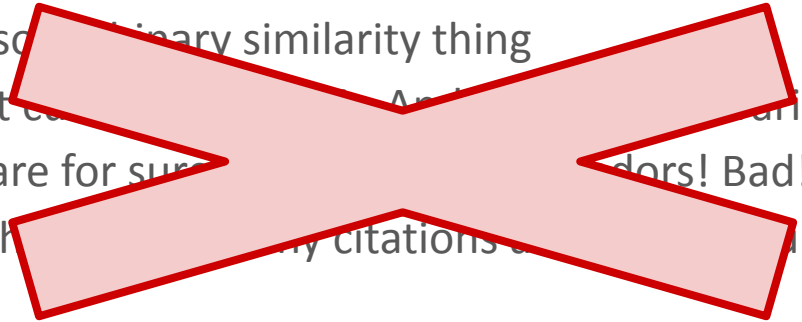


🍞 Apps Need Sustenance. Are Shared Libraries Lembas or Cram?

Goal: Assessing the Android App Ecosystem

Our amazing 5 step plan:

1. Download some APKs
2. Build so[illegible]inary similarity thing
3. Detect c[illegible] An[illegible]ries
4. They are for sur[illegible]dors! Bad!
5. Publish[illegible]y citations [illegible]nes in the future $$$



Constructing a Dataset for Ground Truth Validation

Scraping diverse markets (PlayStore, PlayDrone, AppChina, Anzhi, Mi.com, F-Droid, …)

We got 8.7 million APKs, 12.2% have native libraries, resulting in ~1mio unique libraries

First discovery: lots of code is shared (based on hashes):

- Top 50 binaries account for 14.3% of all native libraries
- Top 500 binaries account for 49.0% of all native libraries
- Top 5000 binaries account for 77.4% of all native libraries

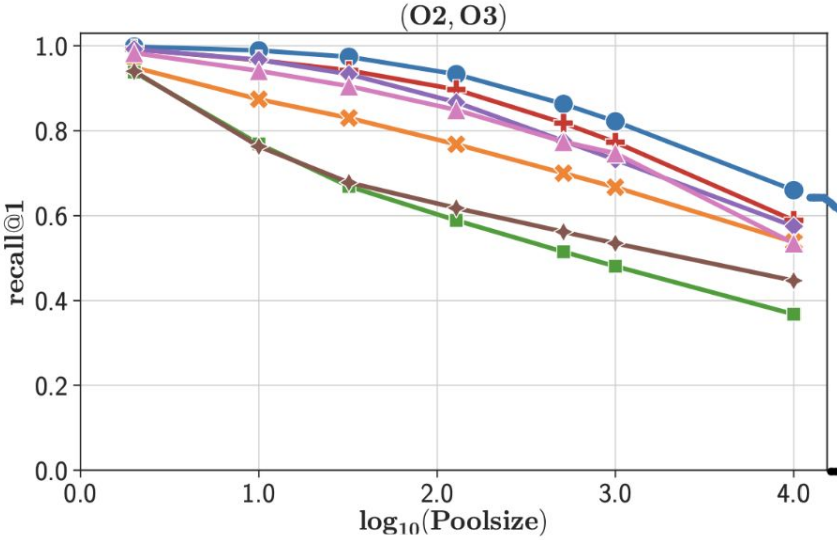
How to Infer Library Names and Versions? Machine Learning!

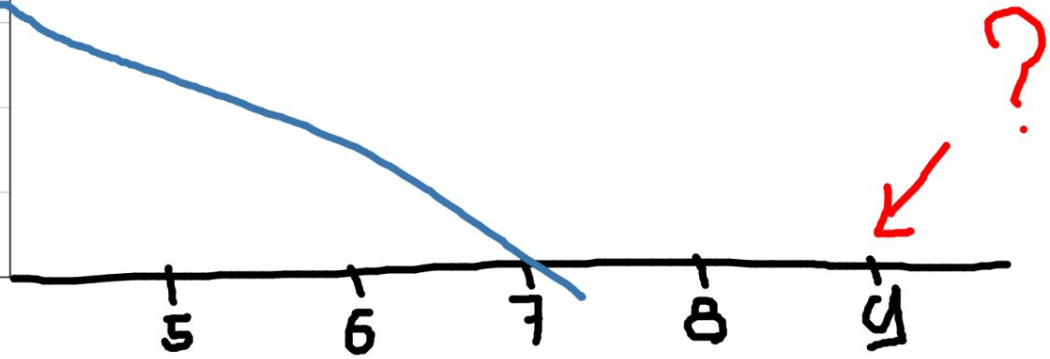
Asm2Vec: Boosting Static Representation Robustness for Binary Clone Search

jTrans Genius Gemini Asm2Vec SAFE GraphEmb OrderMatters

Steven H. H. Ding*, Benjamin C. M.
*Data Mining and Security Lab, School of Information
Emails: steven.h.ding@mail.mcgi

jTrans: Jump-Aware Transformer for Binary Code Similarity





it is the first solution that embeds control flow information of binary code into Transformer-based language models, by using a

https://doi.org/10.1145/3533767.3534367

1 INTRODUCTION

ML Works Great For Small Datasets, Breaks for Large Datasets

Toy dataset of 350 binaries:

12 hours to create embeddings

Our dataset of 1 million libraries:

~38 years 4 months 8 days

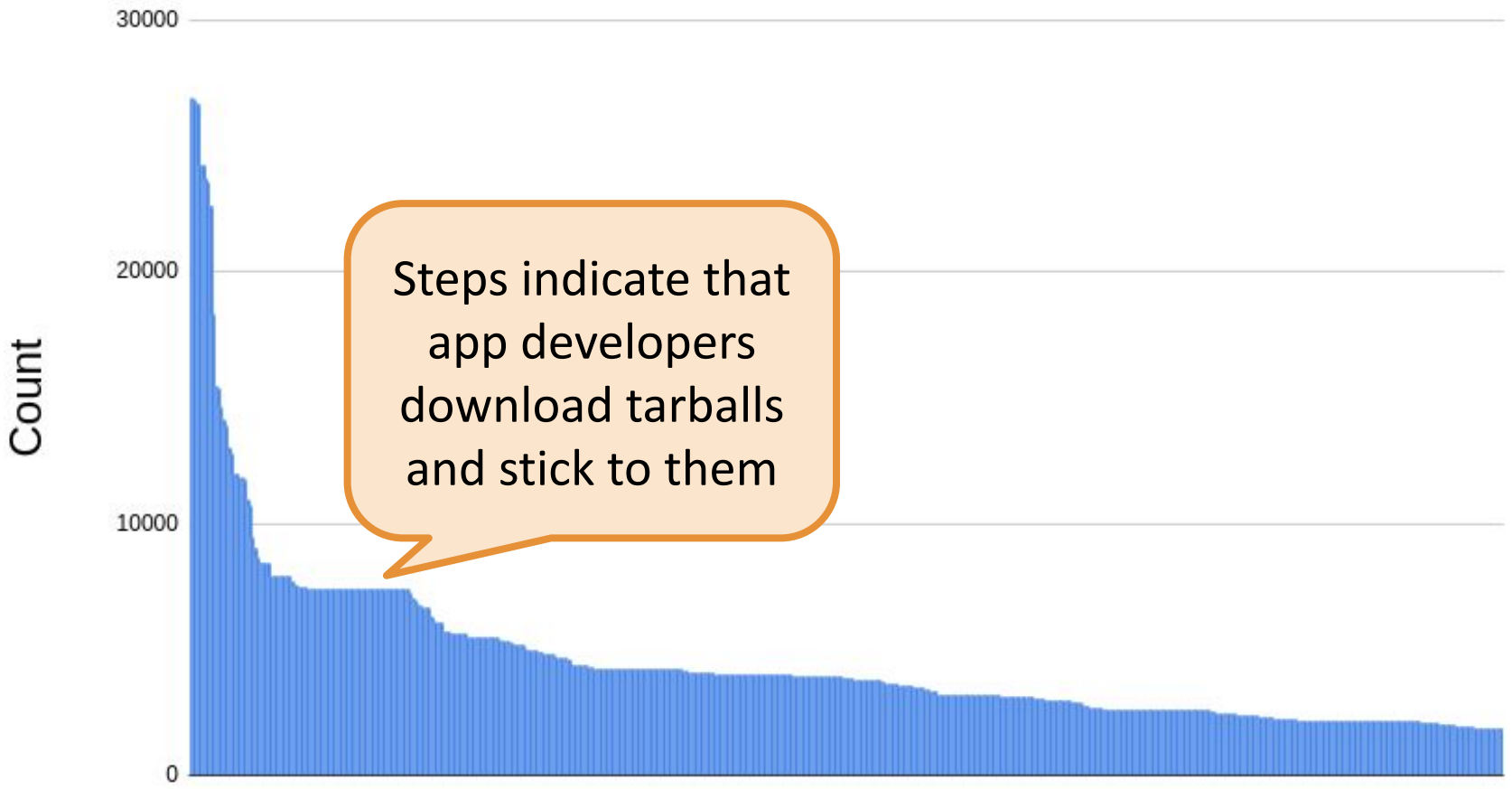
(28’000 times longer)

1,000 L

10 x

10 x 2.5 million L

Are Libraries Copy-Pasted? (1225x Zoom)



Ground Truth From Maven

Looking at Maven we match 29.2% of libraries

Adding heuristics for react native, we get to 59.5%

We can now instantly recognize 60% of all libraries, no need for any AI!

maven_artifact	count
react-native	1620891
react-android	517320
imagepipeline	185892
android-sdk	124471
pdfium-android	118407
carbon	99096
nativeimagefilters	89714
nativeimagetranscoder	83641
sentry-android-ndk	71098
fbjni	63556
animated-gif	50555
android-gif-drawable	49129
imagepipeline-native	42262
webpsupport	31765
dali	24491
conceal	15543
android-database-sqlcipher	13487
full-sdk	10686
hermes-android	10136
ucrop	9944
rootbeer-lib	9548
bugsnag-plugin-android-anr	7686
bugsnag-plugin-android-ndk	7609
mmy-core	7556
ttsdk-ttapplog	6086
androidphotofilters	5753
alcamera	5154
mapbox-android-sdk	5149
pdfium	5136

Case Study: libpl_droidsonroids_gif.so

44,696 Android apps use this library.

CVE-2019-11932 is 6 years old.

How many apps remain vulnerable?

About 78% of apps remain vulnerable. Filtering to only apps on the PlayStore, 58% of apps are vulnerable (2,968 apps)

WhatsApp Remote Code Execution Vulnerability

The vulnerability, tracked as **CVE-2019-11932**, is a double-free memory corruption bug that doesn't actually reside in the WhatsApp code itself, but in an open-source GIF image parsing library that WhatsApp uses.

Top 3 apps vulnerable to CVE-2019-11932

Screen Recorder Video Recorder

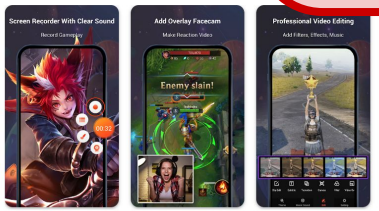
VIDEOSHOW Video Editor & Make

Contains ads · In-app purchases

4.4★ 1.63M reviews | 100M+ Downloads

Install

You don't have any devices

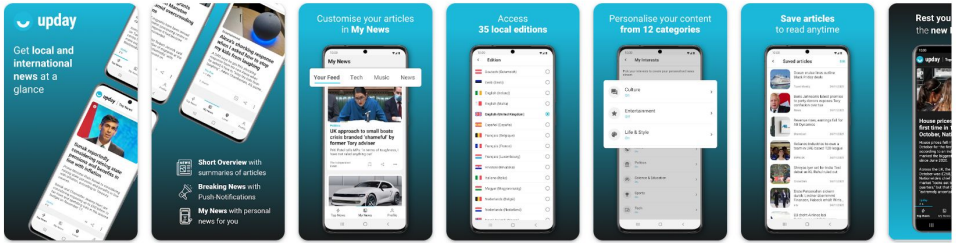


143 million dov
screenrecorde

Exploiting these apps may be hard or even impossible. We only highlight that vulnerable libraries remain used.

upday

Get local and international news at a glance



403 million downloads
de.axelspringer.yana.zeropage

Paint by Number: Coloring Game

ownloads
ber.pixel.art.coloring
zle

hexhive

Software Testing

- Goal: prune bugs
- A tool for developers

Mitigation

- Goal: stop exploitation
- Last line of defense

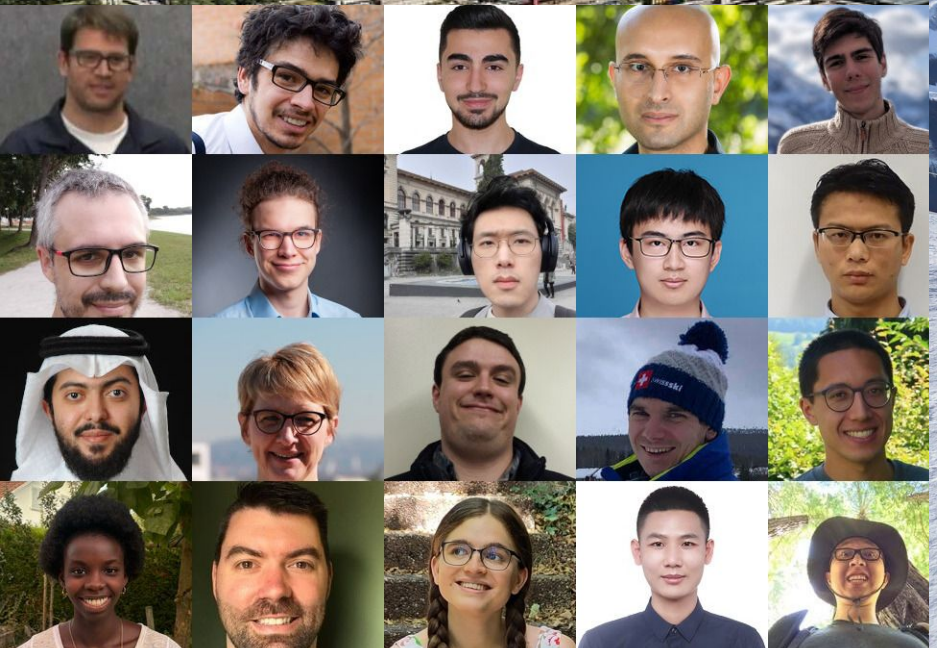
Compartments

- Goal: least privilege
- Divide & conquer security

hexhive

EPFL

Join us on this research journey!



Android Security Remains an Elusive 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 📚🔥
- Crazy: Just reusing apps without updates 🍞

Lots of opportunities for research across the software stack!

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

EPFL *Mathias Payer (@gannimo on bsky/infosec.exchange)* hexhive