TA$ZK

WE FIND NEEDLES

TROOPERS 2025

MOBILE VRP LESSONS FOR BUG HUNTERS

EASTERN PROMISES

TASZK
WE FIND NEEDLES

#whoami

- Daniel Komaromy, Laszlo Szapula, Laszlo Radnai @ TASZK Security Labs
- Various Mobile VRPs in the last 5 years
  - some successes (low 7 figures in total rewards, 75+ CVEs, vendor HoFs and other accolades)
  - many failures!
- The following stories of failure come from our participation in Bug Bounty programs of various Asian Android OEMs


Part 1: Anger is dangerous. It makes people do stupid things.

TASZK
WE FIND NEEDLES

Part 1: Fail Early, Fail Often

- Things going wrong when:
  - choosing what devices to find bugs in
  - deciding what/when to submit once you are finding bugs
  - planning ahead with publication following disclosure

TASZK
WE FIND NEEDLES

Wrong Targets

- Huawei: invitation to Bug Bounty
  - we read the rules to see what models' basebands are included
- Ended up submitting
  - 2G RCE bugs in Helio chipsets
  - 2G RCE bugs in Kirin chipsets (in 3rd party library)
- Responses
  - Helio: Mediatek chipsets are excluded from the bounty
  - Kirin: the models you picked are excluded from the bounty, newer models use a different 3rd party library version

TASZK
WE FIND NEEDLES

Wrong Targets

- Happy End
  - vendor took our feedback, heavily modified the description to address these gaps
  - gave rewards for both anyway “due to high quality of submission”
    - tyvm <3
- Technical Details
  - see: How To Tame Your Unicorn (Black Hat 2021)

TASZK
WE FIND NEEDLES

Wrong Targets

- Samsung: public Mobile VRP
  - covered Shannon baseband RCEs for 6+ years
- Ended up submitting
  - 2G RCE bugs in Shannon chipsets
- Response
  - chipset PSIRT has been split out into “Samsung Semiconductor” unit
  - baseband no longer covered by Mobile VRP, new unit does not offer bounty

TASZK
WE FIND NEEDLES

Wrong Targets

- Happy End
  - vendor took our feedback, Samsung Semiconductor started a separate bug bounty, we got an “after the fact” reward 7 months later
  - Mobile VRP ended up factoring in the impact of our baseband bugs into Android bugs we submitted simultaneously (for a baseband RCE + Android Pivot chain) and gave higher reward there; also awarded “#1 Researcher” accolade to TASZK Security Labs for our 2023 submissions
    - again: tyvm!
- Technical Details
  - see: There Will Be Bugs (CanSecWest 2024)

TASZK
WE FIND NEEDLES

Wrong Targets

- Lesson: develop firmware scraping/testing automation
  - building that infra becomes quite important vs pure bug finding over time!
- Lesson: you may need trial-and-error to figure out the *actual* model/component coverage intent of a VRP
  - aim always for newest model?
    - drawback: often hardest RE target / has different attack surfaces, meanwhile other (older) models may count too
  - aim only for “most established” components?
    - drawbacks of that approach are obvious

TASZK
WE FIND NEEDLES

Wrong Time To Submit

- Theory: “report -> wait reward -> find more of the same if good reward”
- Lesson: “novelty” is regularly considered in reward amount
  - “report -> wait reward -> find more of the same if good reward -> much lower reward”
- Lesson: racing the PSIRT on attack surfaces
  - “report -> wait reward -> find more of the same if good reward -> vendor variant analysis DUP’ed it”
- Food for thought: these phenomenons can push bug hunters to hoard findings

TASZK
WE FIND NEEDLES

Disclosure TL Misconceptions

- Submitted baseband full chain vulns to Samsung Mobile VRP
  - RCE vulns in GPRS + Android Pivot vulns in RemoteFileSystem
- Submitted at the start of April 2023
- First asked for fix status after 2 months
  - asked more times over 2 more months
- Reply end of August: “will be disclosed on the 6th, Nov”

TASZK
WE FIND NEEDLES

Disclosure TL Misconceptions

- In comparison: at same point in 2023, Samsung Semiconductor Bulletins have released 23 baseband bugs, Low to High severity
  - 21 released within 3 months of report submission, 2 within 4
- Theory: we targeted a Fall conference publication
- Reality: CVE/Bulletin released one week after the conference
- Lesson: don’t assume ~90 days is an “industry standard” disclosure timeline

TASZK
WE FIND NEEDLES

Don’t Look Back In Anger

- These annoyances sometimes lead to overthinking VRPs
- In some cases decided to skip/postpone target idea altogether …
  - and then later saw others publish great successes on it
- Be careful about “skipping” a target out of sheer annoyance with a VRP!
  - they actually *do* reflect/change/improve, saw quite a bit of that too

Part 2: Sometimes, if things are closed, you just, open them up.

TASZK
WE FIND NEEDLES

Part 2: Huawei Hypervisor

- Complete talk: Don’t Believe The Hype(rvisor) @ OffByOne 2025
- Why attack a Hypervisor?
  - a separate privilege level beneath the kernel in ARM processors
  - intended to thwart/mitigate kernel LPEs
- Researching an exploit mitigation bypass
  - Assumption: kernel r/w bug primitive or nothing (a complete kernel LPE)
  - Goal: privileged shell (i.e. completed LPE)

TASZK
WE FIND NEEDLES

Huawei HKIP As A Target

- Huawei VRP listed “exploit mitigation bypass” as a high reward category
- HKIP is Huawei Kernel Integrity Protection, built on the ARM Hypervisor feature
- Huawei’s own white paper on HKIP defines it as an exploit mitigation
- our idea was: let’s find a standalone HKIP bypass and get a reward

TASZK
WE FIND NEEDLES

HWPSIRT-2021-80829: A Sob Story

- Late 2021: we found, exploited, and reported such a vulnerability
- Logic bug: didn’t need code exec in or even corruption of Hypervisor memory
- Submitted poc of exploit mitigation bypass of all HKIP defenses

TASZK
WE FIND NEEDLES

HWPSIRT-2021-80829: A Sob Story

- Spring 2022: built emulation, started fuzzing
- May 2022: Huawei PSIRT response
  - outside VRP scope for mitigation bypass
  - HKIP is to "make attack more difficult”, it is not an "exploit mitigation”
- Sidenote: we also submitted several kernel vulns triggerable by `untrusted_app/isolated_app` around the same time (Jan 2022), and one UAF did get a Critical rating and a reward in Mar 2022
  - so that was nice, at least

TASZK
WE FIND NEEDLES

HWPSIRT-2021-80829: A Sob Story

- Lesson: be careful when interpreting VRP category definitions
  - sometimes exploit mitigations morph into “more a hardware mechanism rather than a security feature”
- Vendors can be susceptible to “thinking inside the box”
  - we were also told that a vuln that gets code exec in Hypervisor would qualify
  - slightly contradicts with “higher reward amounts only for novelty”?
- We are still not aware of a fix
  - the vendor saw talk preview with the “we are still not aware of a fix” line
  - they didn’t communicate to us anything different about that

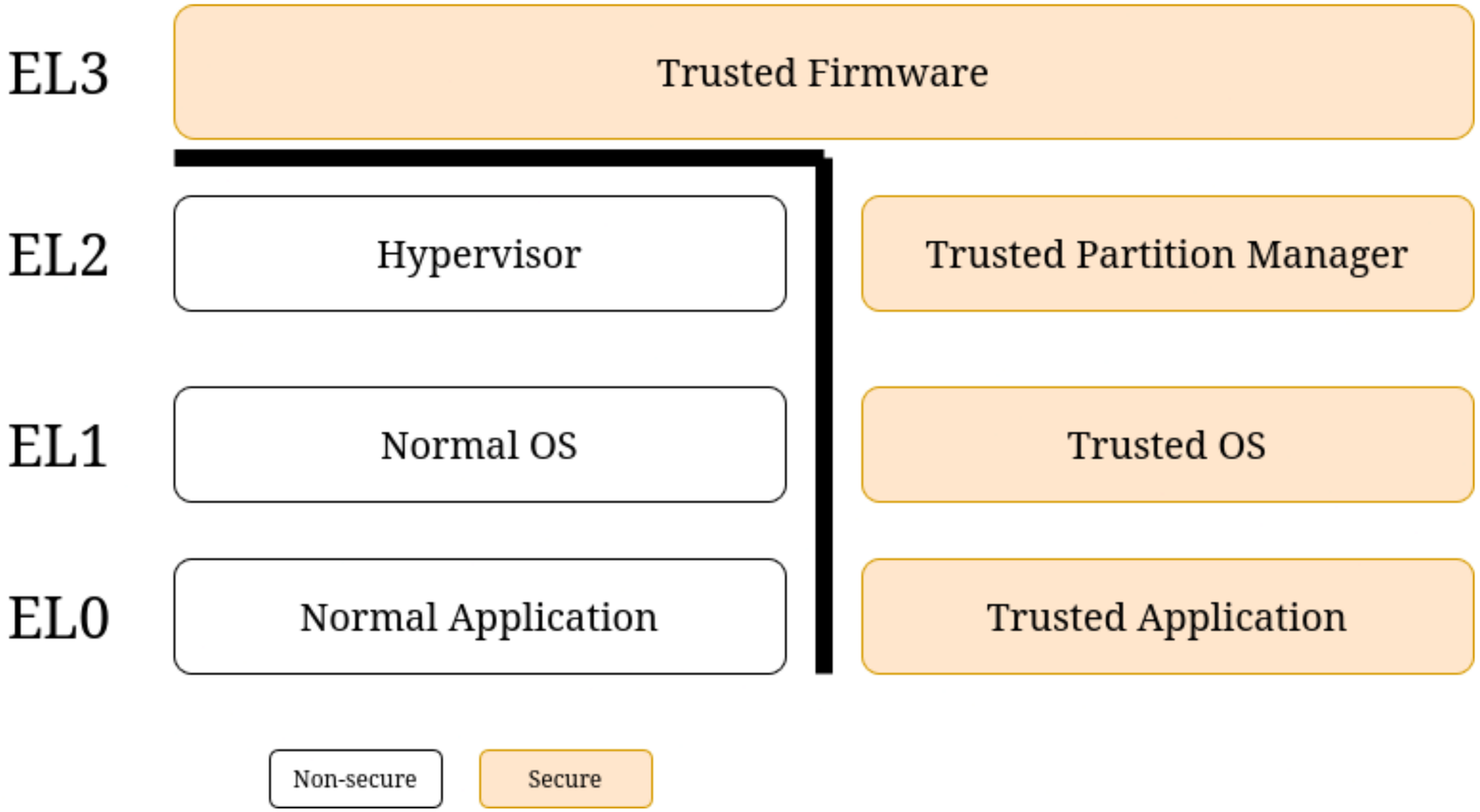
TASZK
WE FIND NEEDLES

(Prior) Art

- Prior Art Then
  - Lifting the (Hyper) Visor - Gal Beniamini
  - A Samsung RKP Compendium - Alexandre Adamski
  - **Emulating Hypervisors: A Samsung RKP case study - Aristeidis Thallas**
- Since
  - Reversing Samsung's H-Arx Hypervisor Framework - DAYZEROSEC
  - **Shedding Light on Huawei's Security Hypervisor - Alexandre Adamski, Maxime Peterlin**
  - Qualcomm Hypervisor Reverse Engineering - Sharad Khann

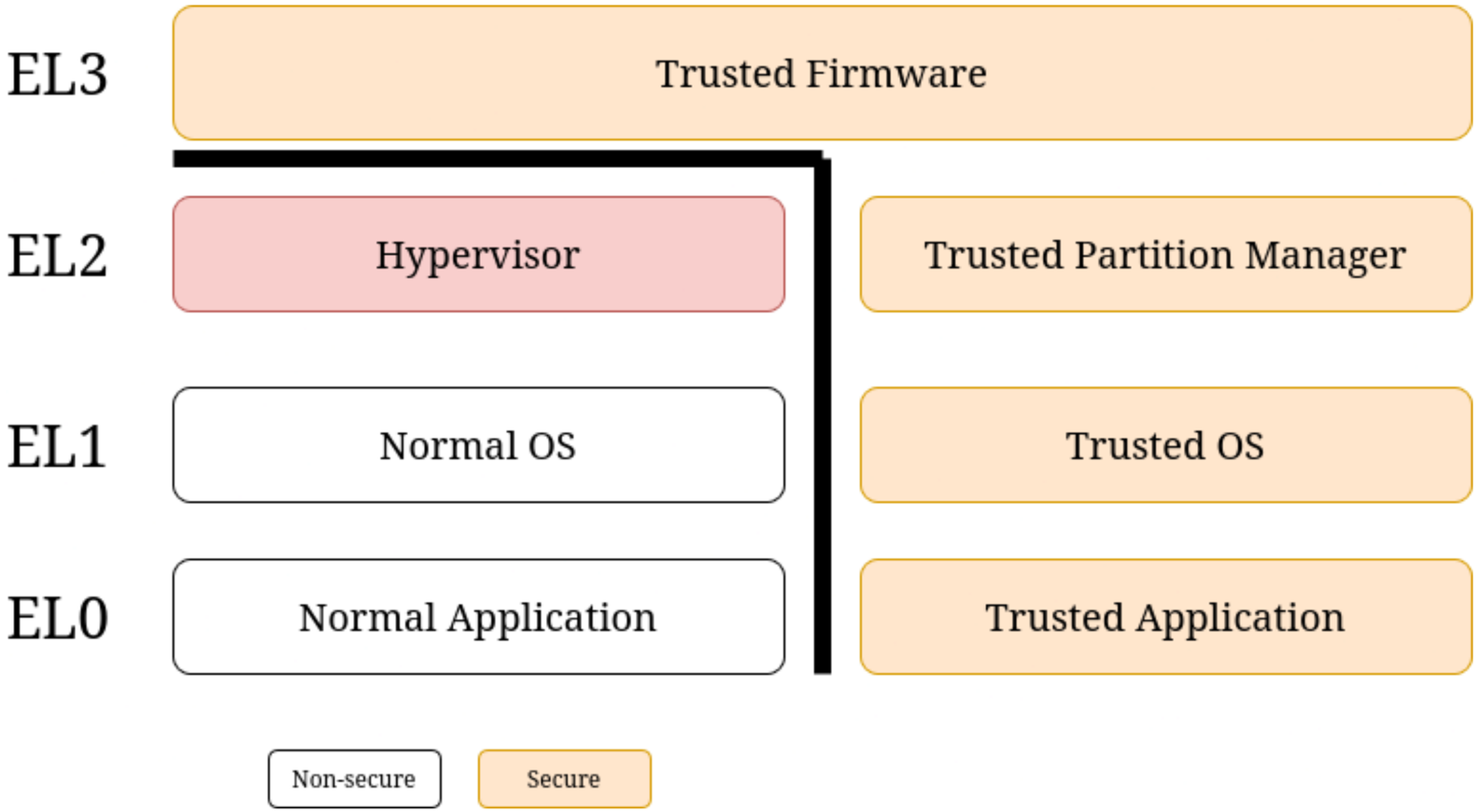
TASZK
WE FIND NEEDLES

AArch64 Exception Levels



TASZK
WE FIND NEEDLES

AArch64 Exception Levels

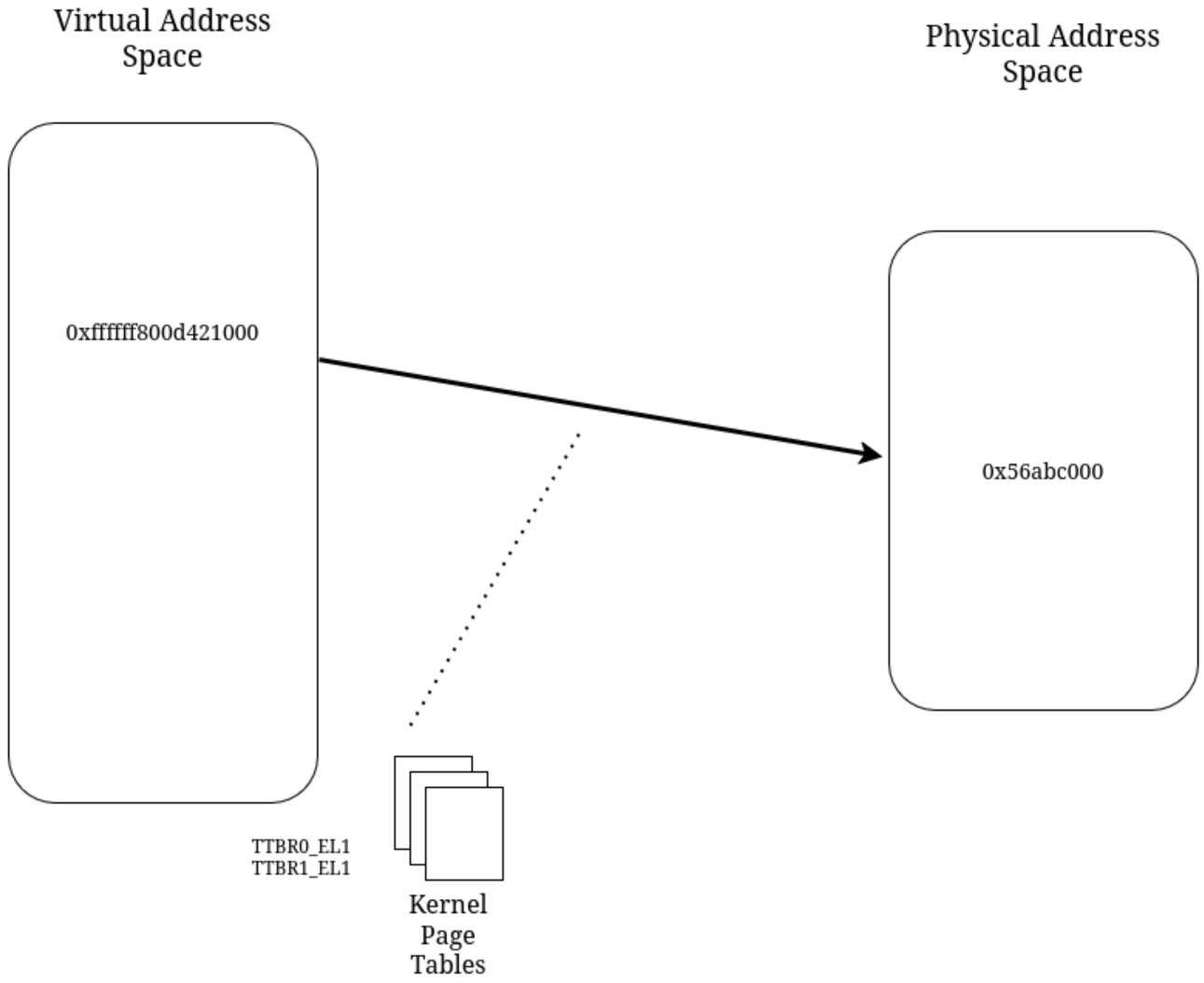


TASZK
WE FIND NEEDLES

Accessing The Hypervisor

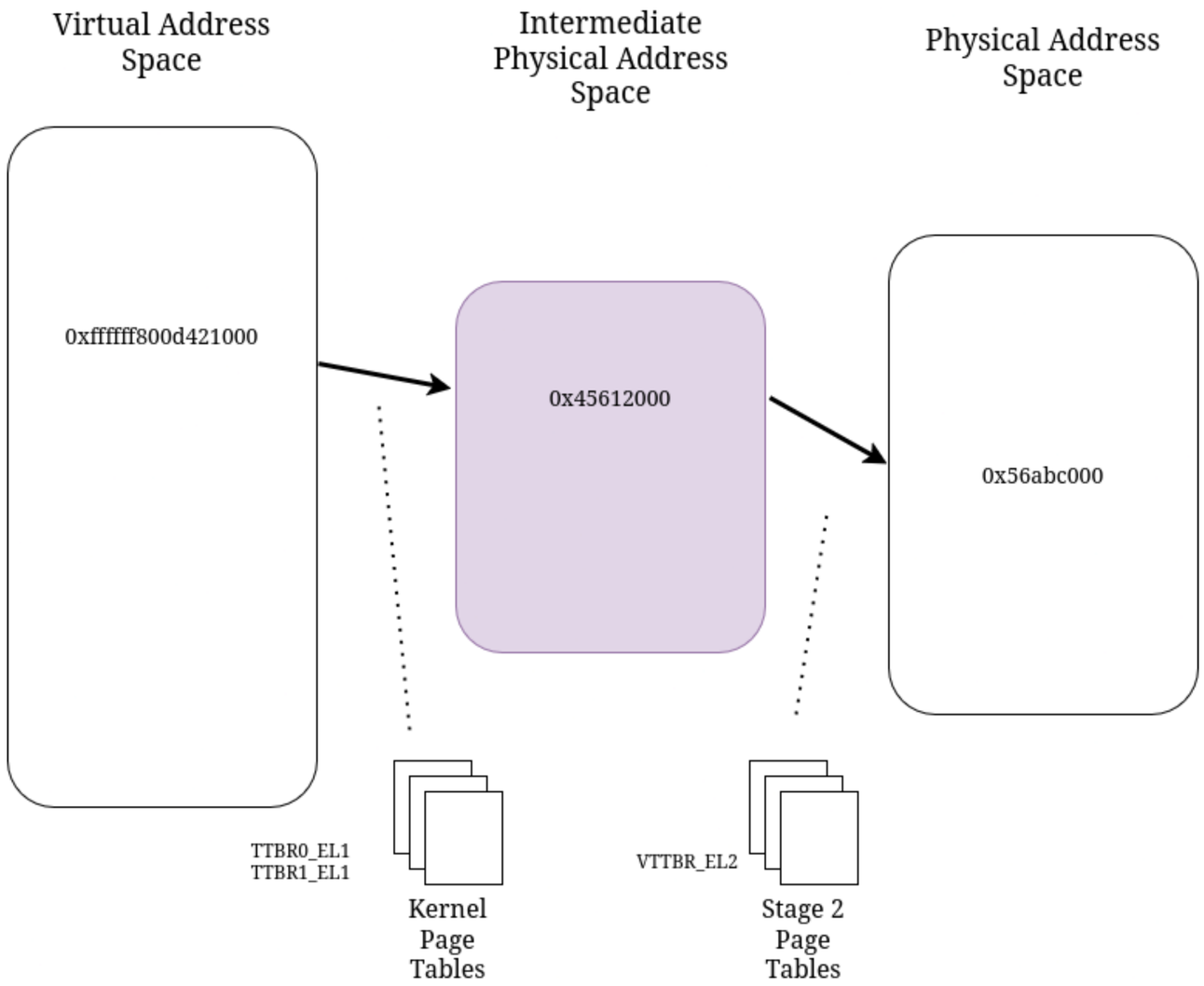
- Hypervisor Call (`HVC`)
- Trapping instructions
  - `MSR` and `MRS` instructions
    - e.g. `TTBR1_EL1`
  - Secure Monitor calls
- Shared memory

Stage 2 Translation



TASZK
WE FIND NEEDLES

Stage 2 Translation



TASZK
WE FIND NEEDLES

The Huawei Way

- Using default Stage-2 protection bits
  - AP bits in page entry
- Using 4 bits of unassigned bits in the PTE
- These bits show what kind of protection

TASZK
WE FIND NEEDLES

Prmem bits

Protection	Value	AP Bits
Unprotected	0b1111	RW
ROWM	0b1010	RO
ROWM reclaimable	0b1011	RO
RO	0b1000	RO
RO reclaimable	0b1001	RO
XO	0b1100	RO
ROX	0b1000	RO
KO	0b1101	RO

TASZK
WE FIND NEEDLES

Example: Huawei Checkroot

- Overwriting cred structure: all good and fine until opening shell

```
$ /data/local/tmp/exp.elf
[+] Exploit task found
[+] Creds overwritten, opening shell...
UID root escalation!
$
```

- OffByOne 25 (`Dont Believe The Hypervisor`) for kernel src details
- tl;dr: kernel traps into Hypervisor with HVC to make decisions, because the relevant structures are RO protected

TASZK
WE FIND NEEDLES

Emulation

- No access to HW elements
  - no need for snapshot
- Requires minimal kernel and Secure Monitor
- Init heavily platform dependent
  - Samsung: started by kernel
  - Huawei: started by Secure Monitor

TASZK
WE FIND NEEDLES

Inspiration

- Thallas: On emulating hypervisors; a Samsung RKP case study
- QEMU system mode emulator and fuzzer
- Starting point for HHEE fuzzer

TASZK
WE FIND NEEDLES

Challenges

- Memory layout
- Initial register contents

TASZK
WE FIND NEEDLES

Challenges

- HHEE loaded to `0x114c0000`
- When launching QEMU, code is not there
- Unlike on higher addresses
- Solution: board definition

TASZK
WE FIND NEEDLES

Challenges

- HHEE loaded to `0x114c0000`
- When launching QEMU, code is not there
- Unlike on higher addresses
- Solution: board definition

TASZK
WE FIND NEEDLES

Board definition

```
// /hw/arm/virt.c
static const MemMapEntry base_memmap[] = {
    // ...
    [VIRT_PCIE_MMIO] =      { 0x10000000, 0x2eff0000 },
    [VIRT_PCIE_PIO] =       { 0x3eff0000, 0x00010000 },
    [VIRT_PCIE_ECAM] =      { 0x3f000000, 0x01000000 },
    /* Actual RAM size depends on initial RAM and device memory
settings */
    [VIRT_MEM] =            { GiB, LEGACY_RAMLIMIT_BYTES },
};
```

TASZK
WE FIND NEEDLES

Board definition

```
// /hw/arm/virt.c
static const MemMapEntry base_memmap[] = {
    // ...
    //[VIRT_PCIE_MMIO] =      { 0x10000000, 0x2eff0000 },
    //[VIRT_PCIE_PIO] =       { 0x3eff0000, 0x00010000 },
    //[VIRT_PCIE_ECAM] =      { 0x3f000000, 0x01000000 },
    /* Actual RAM size depends on initial RAM and device memory
settings */
    [VIRT_MEM] =            { 0x00000000, LEGACY_RAMLIMIT_BYTES },
};
```

TASZK
WE FIND NEEDLES

Initial register contents

- Required for best emulation
- Set by trusted firmware (EL3)
- Tool: SMC read-write primitive
- Patched EL2 to save register contents

TASZK
WE FIND NEEDLES

It’s alive!

```
american fuzzy lop 2.56b-athallas (qemu-system-aarch64)

┌─ process timing ─────────────────────────────────┬─ overall results ─┐
│        run time : 0 days, 0 hrs, 14 min, 42 sec  │  cycles done : 15 │
│   last new path : 0 days, 0 hrs, 8 min, 58 sec   │  total paths : 82 │
│ last uniq crash : none seen yet                  │ uniq crashes : 0  │
│  last uniq hang : 0 days, 0 hrs, 14 min, 1 sec   │   uniq hangs : 11 │
├─ cycle progress ─────────────────┬─ map coverage ─┴───────────────────┤
│  now processing : 26 (31.71%)    │    map density : 0.09% / 1.47%    │
│ paths timed out : 0 (0.00%)      │ count coverage : 1.05 bits/tuple  │
├─ stage progress ─────────────────┼─ findings in depth ───────────────┤
│  now trying : splice 9           │ favored paths : 81 (98.78%)       │
│ stage execs : 26/96 (27.08%)     │  new edges on : 82 (100.00%)      │
│ total execs : 1.32M              │ total crashes : 0 (0 unique)      │
│  exec speed : 1352/sec           │  total tmouts : 1994 (11 unique)  │
├─ fuzzing strategy yields ────────┴─ path geometry ──────────────────┤
│   bit flips : 12/2624, 15/2542, 2/2378           │    levels : 6     │
│  byte flips : 1/328, 0/246, 0/82                 │   pending : 0     │
│ arithmetics : 34/18.3k, 0/5674, 0/646            │  pend fav : 0     │
│  known ints : 1/1753, 2/5742, 1/3537             │ own finds : 80    │
│  dictionary : 0/0, 0/0, 0/109                    │  imported : n/a   │
│       havoc : 12/713k, 0/566k                    │ stability : 100.00% │
│        trim : 60.00%/2, 0.00%                    ├───────────────────┘
└──────────────────────────────────────────────────┘
                                                    [cpu000: 20%]
```

TASZK
WE FIND NEEDLES

Fuzzing results

- 3 days on i5 8350u
- Many false-positives
- No new vulnerabilities found

TASZK
WE FIND NEEDLES

Old Fashioned Code Review

- Checkroot bits are managed via HVC call, but NO VALIDATION
  - again, for code flow details, see `Don’t Believe The Hypervisor`
- Code exec in kernel context -> access to Hypervisor via HVC API
- Modify ROWM protected bits via API call -> bypass checkroot

TASZK
WE FIND NEEDLES

Exploitation: Goals

- Use kernel r/w primitives
  - patched into the kernel for the research purpose
  - emulates kernel memory corruption exploit
- Achieve code execution via ROP
- Use Hypervisor API to bypass checkroot
- Overwrite credentials in `task_struct`
- Pop shell

TASZK
WE FIND NEEDLES

Done (no)

- Making arbitrary HVC calls via ROP had some challenges
  - again see the OffByOne talk for details, including some more OEM "mitigation" we bypassed for ROP
- Now root shell pops and won't get killed
- But this is just DAC bypass
- There is still SELinux
  - ... for now!

TASZK
WE FIND NEEDLES

Prmem allocator

- SELinux policy structures allocated on `selinux_pool`
- After init, pool protected with `HKIP_HVC_RO_MOD_REGISTER`
- This can be removed with `HKIP_HVC_RO_MOD_UNREGISTER`
- Now SELinux policy overwriteable in memory
- But there is a catch ...

TASZK
WE FIND NEEDLES

SELinux Bypass Plan

- RO protection removed in Stage-2 tables
- But not from kernel page tables
- Entries are protected by the Hypervisor
- But there is an easy bypass

TASZK
WE FIND NEEDLES

SELinux Bypass Plan

- `lock_range` sets the target pages to RO
- But not their linear mapping equivalents
- To can get the linear address:
  - get the physical address
  - subtract `memstart_addr` from it
  - add `0xfffffc0000000000`

```
HWANA:/ $
```

Part 3: Sentimental value? I’ve heard of that.

TASZK
WE FIND NEEDLES

Part 3: Unisoc TrustZone

- Why attack a Trusted Execution Environment?
  - ARM security concept: a separate privilege level beneath the kernel (again)
  - intended to ~~thwart/mitigate~~ devalue kernel LPEs by moving secrets/ privileged computations into a lower level
- Researching a TEE LPE
  - Assumption: code exec with kernel privileges
  - Goal: arbitrary code execution in TEE

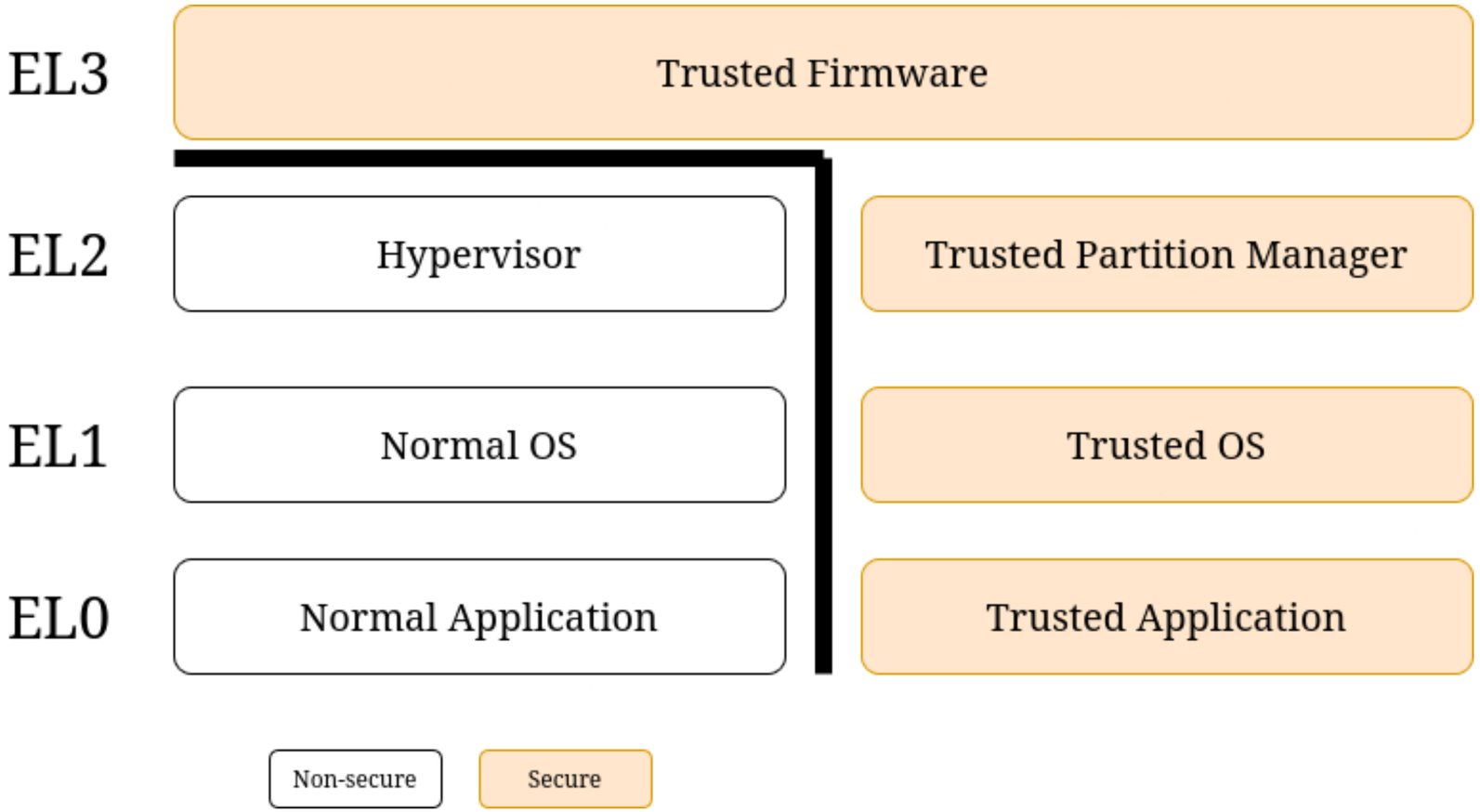
TASZK
WE FIND NEEDLES

(Prior) Art

- Prior Art Then
  - Reflections on Trusting TrustZone - Dan Rosenberg
  - QSEE TrustZone Integer Signedness Bug - Frederic Basse
  - Exploiting Trustzone on Android - Di Shen
  - Trust Issues: Exploiting TrustZone TEEs - Gal Beniamini
  - Unbox Your Phone - Daniel Komaromy
  - A Deep Dive Into Samsung's TrustZone - Quarkslab
  - Breaking TEE Security - Riscure
- Since
  - ARM TrustZone: pivoting to the secure world - Thalium
  - Hara-Kirin - Impalabs

TASZK
WE FIND NEEDLES

TrustZone 101



TASZK
WE FIND NEEDLES

Our Target (Unisoc TEE)

- Unisoc (formerly Spreadtrum)
  - Mostly low/mid-range devices
- Not straightforward to buy in Europe
    - a few accessible e.g. Samsung Galaxy Tab A8
- TrustZone not a full black box, uses common building blocks
  - OS: built on Trusty
  - Trustlet APIs: Global Platform

TASZK
WE FIND NEEDLES

Firmware Acquisition

- Firmware can be downloaded from internet
  - Trusted OS and TrustFirmware in `BL` package
  - `tos-sign.bin`
- Signature and DHTB header
- Trustlets included in this binary

TASZK
WE FIND NEEDLES

TOS Binary

TASZK
WE FIND NEEDLES

Trustlets

Trustlet Name	Image Offset (0x)
trusty-gatekeeper	BB000
ipc-unittest-srv	CD000
crypto-ipc	D8000
storage-proxy	E1000
trusty-keymaster	113000
trusty-kernelbootcp	1BD000
trusty-production	1C7000
spreadtrum-storage-proxy	227000
trusty-oemcrypto	27E000

TASZK
WE FIND NEEDLES

Trustlets

Trustlet Name	Image Offset (0x)
trusty-gatekeeper	BB000
ipc-unittest-srv	CD000
crypto-ipc	D8000
storage-proxy	E1000
trusty-keymaster	113000
trusty-kernelbootcp	1BD000
trusty-production	1C7000
spreadtrum-storage-proxy	227000
trusty-oemcrypto	**27E000**

TASZK
WE FIND NEEDLES

Trusty OEMCrypto

- Performs DRM related crypto procedures
- Two modes:
  - Widevine
  - Unisoc OEMCrypto

TASZK
WE FIND NEEDLES

Unisoc OEMCrypto

- Crypto API accessible from the kernel
- A lot of services
  - `UNISOC_OEMCrypto_InstallKeyboxOrOEMCert`
  - `UNISOC_OEMCrypto_LoadDRMPrivateKey`
  - `UNISOC_OEMCrypto_CopyBuffer`
  - `UNISOC_OEMCrypto_GenerateRSASignature`
  - `UNISOC_OEMCrypto_DecryptCENC`

TASZK
WE FIND NEEDLES

DecryptCENC

```
int UNISOC_OEMCrypto_DecryptCENC(decrypt_cenc_msg* msg, uint len,
                                 char **out_buff, int *out_len){
    char data1[24];
    char data2[16];
    char data3[8];

    uint size1;
    uint size2;
    uint size3;

    if (len == 0 || msg == NULL){
        print_log(&logbuff, "%s: [%s]%d: UNISOC_OEMCrypto_DecryptCENC():
            wrong parameters\n", "trusty_unisoc_oemcrypto",
            "UNISOC_OEMCrypto_DecryptCENC");
        ret = -8;
    } else {
        OPENSSL_memcpy(&input_0, msg, 4);
        OPENSSL_memcpy(&size1, &msg->msg_size, 4);
        OPENSSL_memcpy(data1, &msg->msg, size1);

        OPENSSL_memcpy(&size2, &msg->msg + size1, 4);
        OPENSSL_memcpy(data2, &msg->msg + size1 + 4, size2);

        OPENSSL_memcpy(&size3, &msg->msg + size1 + size2 + 4, 4);
        // ...
        OPENSSL_memcpy(data3, &msg->msg + size1 + size2 + size4 + 8, 8);

        // size validations and rest of the function
    }

    return ret;
}
```

TASZK
WE FIND NEEDLES

DecryptCENC

```
if (len == 0 || msg == NULL){
    print_log(&logbuff, "%s: [%s]%d: UNISOC_OEMCrypto_DecryptCENC():
        wrong parameters\n", "trusty_unisoc_oemcrypto",
        "UNISOC_OEMCrypto_DecryptCENC");
    ret = -8;
} else {
    OPENSSL_memcpy(&input_0, msg, 4);
    OPENSSL_memcpy(&size1, &msg->msg_size, 4);
    OPENSSL_memcpy(data1, &msg->msg, size1);
```

TASZK
WE FIND NEEDLES

DecryptCENC

```
if (len == 0 || msg == NULL){
    print_log(&logbuff, "%s: [%s]%d: UNISOC_OEMCrypto_DecryptCENC():
        wrong parameters\n", "trusty_unisoc_oemcrypto",
        "UNISOC_OEMCrypto_DecryptCENC");
    ret = -8;
} else {
    OPENSSL_memcpy(&input_0, msg, 4);
    OPENSSL_memcpy(&size1, &msg->msg_size, 4);
    OPENSSL_memcpy(data1, &msg->msg, size1);
```

Time travel, nice :)

TASZK
WE FIND NEEDLES

Triggering The Bug

- Through `/dev/trusty-ipc-dev0`
- Requires `teetz_device` context and system user/group
- … or a rooted device
- Use ioctl on the driver
- Connect:

```
ioctl(fd, TIPC_IOC_CONNECT, "com.android.trusty.oemcrypto");
```

TASZK
WE FIND NEEDLES

Triggering The Bug

- Connect to the trustlet
- OEMCrypto Init
- OEMCrypto DecryptCENC with large size
- Win

TASZK
WE FIND NEEDLES

Exploitation

```
$ checksec trusty_oemcrypto.elf
    Arch: arm-32-little
    RELRO: No RELRO
    Stack: No canary found
    NX: NX disabled
    PIE: No PIE (0x8000)
    RWX: Has RWX segments
```

More time travel, very nice :)

TASZK
WE FIND NEEDLES

Exploitation (POC)

- ROP chain
  - Put string in BSS
  - Call log function
    - Visible in dmesg
  - Return to message handler loop

```
$ adb shell su -c "/data/local/tmp/expl.elf"
```

```
$ adb shell su -c "dmesg -w | grep trusty"
```

TA$ZK
WE FIND NEEDLES

Disclosure Outcome 1

- Severity merry-go-round
  - first response: “CVSS is Medium CVSS:3.1/…”
  - we point out that full-on TEE code execution LPE tends to be higher than Medium
  - second response: “High”
  - very very nice
- submission: May 17th 2023, bulletin with CVE: Aug 2023
  - very very very nice

TASZK
WE FIND NEEDLES

Disclosure Outcome 2

- submission: **May** 17th 2023
- after bulletin, we ask about eligibility for the publicized Unisoc Chipset Security Reward Program

"We used to have a Unisoc Chipset Security Reward Program which was affiliated with the Google Chipset Security Reward Program on HackerOne. However, Google terminated the project in **May**, *resulting in the shutdown of our program as well"*

- ... not very nice :'(

TASZK
WE FIND NEEDLES

Disclosure Lessons

- Some OEMs use CVSS scoring
  - applied rigidly it's often a terrible fit for mobile LPE/RCE/SBX/etc bugs
  - can't assume that different OEMs' assessment of "same" bug is identical
- Sometimes "pick a random device model" *is* the more effective strategy
- Bug Bounty programs really do change on-a-dime … must pay attention
  - it was very fun to have this happen with Unisoc in May 23, after the same happening with Samsung in April 23

Part 4: Why are you doing this? Why are you helping us?

TASZK
WE FIND NEEDLES

Part 4: Mediatek Baseband

- Why attack a baseband?
  - separate processors of System-on-Chips handle connectivity stacks
  - exposes remote attack surfaces, often 0-click, some cases end-to-end
- Researching a baseband RCE
  - Assumption: can range from “access/mitm of an MNO” to “nothing”
  - Goal: compromise baseband runtime, find highest impact bugs (access vector-wise)

TASZK
WE FIND NEEDLES

(Prior) Art

- VoLTE
  - Marco Grassi and Kira: Over-The-Air Baseband Exploit
  - Natashenka et al: How To Hack Shannon Baseband
- Emulated Baseband Fuzzing
  - Prior: BaseSAFE, FirmWire
  - Since: BaseBridge, Securing The Airwaves

TASZK
WE FIND NEEDLES

Voices from the Internet

- Classic cellular networks (2G/3G): calls over control plane
- Modern networks (4G/5G): adopt VoIP for calls over user plane
- VoIP: voice call over IP-network
  - SIP: session management
  - SDP: stream definitions
  - RTP: actual data

TASZK
WE FIND NEEDLES

IMS

- IMS (IP Multimedia Subsystem)
  - IP-based network
  - separate from (regular) data traffic
- VoLTE (Voice-over-LTE)/VoNR(Voice-over-NewRadio)/VoWifi
  - VoIP over IMS

TASZK
WE FIND NEEDLES

IMS As Attack Surface

- IMS control protocols (SIP, SDP etc), underlying protocols (TCP/IP etc), codecs of voice/video calls
- Code location in mobiles
  - typically in the baseband
  - sometimes (e.g. iPhone) in the application processor
- Reachability
  - malicious or compromised IMS core network
  - between UEs end-to-end

IMS Attack Surface: End-to-End?

- SIP end-to-end on paper
- But IMS nodes can filter/re-encode
- Are IMS networks perfect stop gaps against malformed SIP packets in practice?
- (Untrustworthy) operator inter-operability loopholes?

IMS Attack Surface: End-to-End?

- SIP end-to-end on paper
- But IMS nodes can filter/re-encode
- Are IMS networks perfect stop gaps against malformed SIP packets in practice?
- (Untrustworthy) operator inter-operability loopholes?

Leaky location data bug fixed by O2 UK

News By Sead Fadilpašić published May 20, 2025

A bug that was introduced in early 2023 was recently fixed

TASZK
WE FIND NEEDLES

SIP

- HTTP-looking communication-protocol
- over TCP
  - REGISTER to network (CSCF)
  - SUBSCRIBE/NOTIFY
  - Call (3-way): INVITE, 200 OK, ACK

TASZK
WE FIND NEEDLES

SIP INVITE

```
INVITE sip:+10123456789@ims.sharetechnote.com SIP/2.0
...
From: <sip:+11234567890@ims.sharetechnote.com>;tag=659
To: <sip:+10123456789@ims.sharetechnote.com>
Call-ID: 131949458@192.168.1.15
CSeq: 1 INVITE
Max-Forwards: 70
Content-Length: 387
```

TASZK
WE FIND NEEDLES

SIP INVITE - SDP

```
INVITE sip:+10123456789@ims.sharetechnote.com SIP/2.0
...
Content-Length: 387

v=0
o=mhandley 2890844526 2890842807 IN IP4 126.16.64.4
s=SDP Seminar
i=A Seminar on the session description protocol
...
a=recvonly
m=audio 49170 RTP/AVP 0
m=video 51372 RTP/AVP 31
m=application 32416 udp wb
a=orient:portrait
```

TASZK
WE FIND NEEDLES

Methodology

- Fuzz it with libafl
  - harness: emulate VoLTE SIP+SDP stack using qemu
  - mutation: grammar fuzzing
  - detection: custom ASAN-lite hooks for better crash determinism
  - goal: coverage

TASZK
WE FIND NEEDLES

Custom-Built Harness

- Mediatek baseband is quite RE friendly (ton of debug symbols in fw imgs)
  - NanoMIPS ISA architecture needed tool customization: prior work
- Emulation challenge: lot of state machines on the one hand, complex hardware-related code paths on the other
  - our approach: similar to BaseBridge (OffensiveCon 2025)
  - leverage ramdump of the baseband runtime’s memory to get correctly initialized state machine values

TASZK
WE FIND NEEDLES

Grammar Fuzzing

- Similar to… any research related to fuzzing text-based protocols
- Keywords
- "Well-formed" (Context-free grammar)
- “Valid”, "almost valid", "almost well-formed"

TASZK
WE FIND NEEDLES

Start Fuzzing

TASZK
WE FIND NEEDLES

Effective Fuzzing vs Findings

- Fuzzing will always find already-found, shallow bugs :(
- waiting for bugs to be fixed takes long
  - patch the bugs for ourselves!
  - maybe kill a whole feature-set until fixed

TASZK
WE FIND NEEDLES

Example: CVE-2023-32889

- BOF when decoding AMR/AMR-WB codec mode-set parameter

```
m=audio 49170 RTP/AVP 99
a=rtpmap:99 AMR-WB/16000/1
a=fmtp:99 mode-change-capability=1; octet-align=1;\
    mode-set=0,3,5,6
```

Example: CVE-2023-32889

- BOF when decoding AMR/AMR-WB codec mode-set parameter

```
m=audio 49170 RTP/AVP 99
a=rtpmap:99 AMR-WB/16000/1
a=fmtp:99 mode-change-capability=1; octet-align=1;\
    mode-set=0,3,5,6
```

TASZK
WE FIND NEEDLES

Example: CVE-2023-32889

- Mediatek code had 16 mode-set slots in the output structure
  - hence, input length was limited to 32 characters
- But the parsing loop itself had no bounds checking …
- And no error handling for failed integer conversion
- Can you see the problem? :)

```
int cc_call_sdp_get_info(...)
  iterator = strchr(iterator,',');
  // ... bail out early iterator==-1
  idx = 1;
  lVar9 = (int8) strtol(iterator + -1, NULL, 10);
  (bw_ind->media_config).mode_set[0] = lVar9;
  do {
    // [1] Each token: convert to decimal
    lVar9 = (int8) strtol(iterator + 1, NULL, 10);
    // [2] write into mode_set array w/o bounds check
    (bw_ind->media_config).mode_set[idx] = lVar9;
    idx = idx + 1;
    iterator = strchr(iterator + 1,',');
  } while (iterator != NULL);
```

TASZK
WE FIND NEEDLES

Example: CVE-2023-32889

- Mediatek code had 16 mode-set slots in the output structure
  - hence, input length was limited to 32 characters
- But the parsing loop itself had no bounds checking …
- And no error handling for failed integer conversion

```
int cc_call_sdp_get_info(...)
  iterator = strchr(iterator,',');
  // ... bail out early iterator==-1
  idx = 1;
  lVar9 = (int8) strtol(iterator + -1, NULL, 10);
  (bw_ind->media_config).mode_set[0] = lVar9;
  do {
    // [1] Each token: convert to decimal
    lVar9 = (int8) strtol(iterator + 1, NULL, 10);
    // [2] write into mode_set array w/o bounds check
    (bw_ind->media_config).mode_set[idx] = lVar9;
    idx = idx + 1;
    iterator = strchr(iterator + 1,',');
  } while (iterator != NULL);
```

- “empty” commas lead to more than 16 iterations!

```
,,,,,,,,,,,,,,,,1,2,3,4,5,6
```

TASZK
WE FIND NEEDLES

Findings

Bug	Trigger	Effect	CVE

multipat/mixed content	`Content-Length: 32`
	
	`300*"A"`
		Heap BOF	CVE-2023-32886

SDP extract codec info	too many rtmap entries	Stack BOF	CVE-2023-32874
SDP mode set	`,,,,,,,,,,,,,,,,1,2,3,4`	Intra-struct OF	CVE-2023-32889
SIP Asserted Identity	`"""""""""""""*`	Heap BOF	CVE-2023-32888
SIP comment recursion	`Via: ((((((((((`	Stack OF	CVE-2023-32887

TASZK
WE FIND NEEDLES

Disclosure Outcome

- Severity: Heap OFs set to Critical first, then lowered to Medium
- Vendor response 1*: heap overflows can’t be exploited due to a “mitigation mechanism” that is “similar to sanitizers”
  - we then provided detailed analysis of the heap implementation, describing that we see no sign of such a thing in it
- Vendor response 2*: next slide

**for legal reasons, we are not claiming these quotes are actual responses and this part is creative story telling strictly for entertainment purposes only*

TASZK
WE FIND NEEDLES

Disclosure Outcome

"(...) after the buffer is OOB written in modem, there are two possible consequences:

1. Overwrite to code-segment: Because the code segment is configured as read only , the overwritten in code-segment will cause system reset immediately.

2. Overwrite to data-segment: Because the data-segment is configured as non-executable, the overwritten in data-segment will not be executed even they are instructions. Although the overwritten in data-segment may not be detected and reset immediately, the impact is temporary and will be recovered after reboot."

TASZK
WE FIND NEEDLES

Ongoing Disclosure

- We then found that we lacked the resources for it, so we stopped the complex work of analyzing ~20.000 crashes
- Finally we got around to them 1y+ later
- RCA'd, found new bugs, verified liveness (~Q4 2024 so a while ago) … then later found time to dedup and write reports … then finally:

vuln report 1	→ MediaTek Security Team	6/23/25, 9:29 PM
vuln report 2	→ MediaTek Security Team	6/23/25, 9:30 PM
vuln report 3	→ MediaTek Security Team	6/23/25, 9:31 PM
vuln report 4	→ MediaTek Security Team	6/23/25, 9:31 PM
vuln report 5	→ MediaTek Security Team	6/23/25, 9:33 PM
vuln report 6	→ MediaTek Security Team	6/23/25, 9:34 PM
vuln report 7	→ MediaTek Security Team	6/23/25, 9:36 PM
vuln report 8	→ MediaTek Security Team	6/23/25, 9:37 PM
vuln report 9	→ MediaTek Security Team	6/23/25, 9:38 PM
vuln report 10	→ MediaTek Security Team	6/23/25, 9:41 PM
vuln report 11	→ MediaTek Security Team	6/23/25, 9:45 PM

TASZK
WE FIND NEEDLES

Ongoing Disclosure

- 11 additional SIP/SDP bugs, most potentially reachable end-to-end
- 3 heap OF, 1 stack OF, 1 memleak, 6 DoS
  - worth noting the impact of end-to-end reachable DoS vulns (let alone RCE)
- We didn’t use Mediatek’s Bug Bounty Program this time
  - reports still included RCA and easy to replicate poc instructions
  - we intend to publish in <=90 days (see https://taszk.io/disclosure)
- Time lag between liveness checks and reporting
  - we truly don't know whether any / how many are 0ds as of today

Disclosure Lessons

- A VRP (to a vendor) is not always “coordinated disclosure with a reward”
- Is it necessary to re-learn lessons from the past? We certainly can’t express them any more eloquently than these trailblazers

No more free bugs for software vendors

Author:
Dennis Fisher

March 23, 2009
/ 2:56 pm

2:30 minute read

Share this article:

Thank you!
Questions?