Executing code in the TrustZone land

Edgar Barbosa

SyScan360 - Shanghai

2016

Me

- Edgar Barbosa
  - Senior Security Researcher
  - COSEINC - Singapore
  - https://google.com/#q=Edgar+Barbosa+COSEINC
  - twitter.com/embarbosa

Agenda

- What is TrustZone?
- TZ Applications
- TZ Architecture
- Secure Boot
- Executing TZ code
- Reverse Engineering

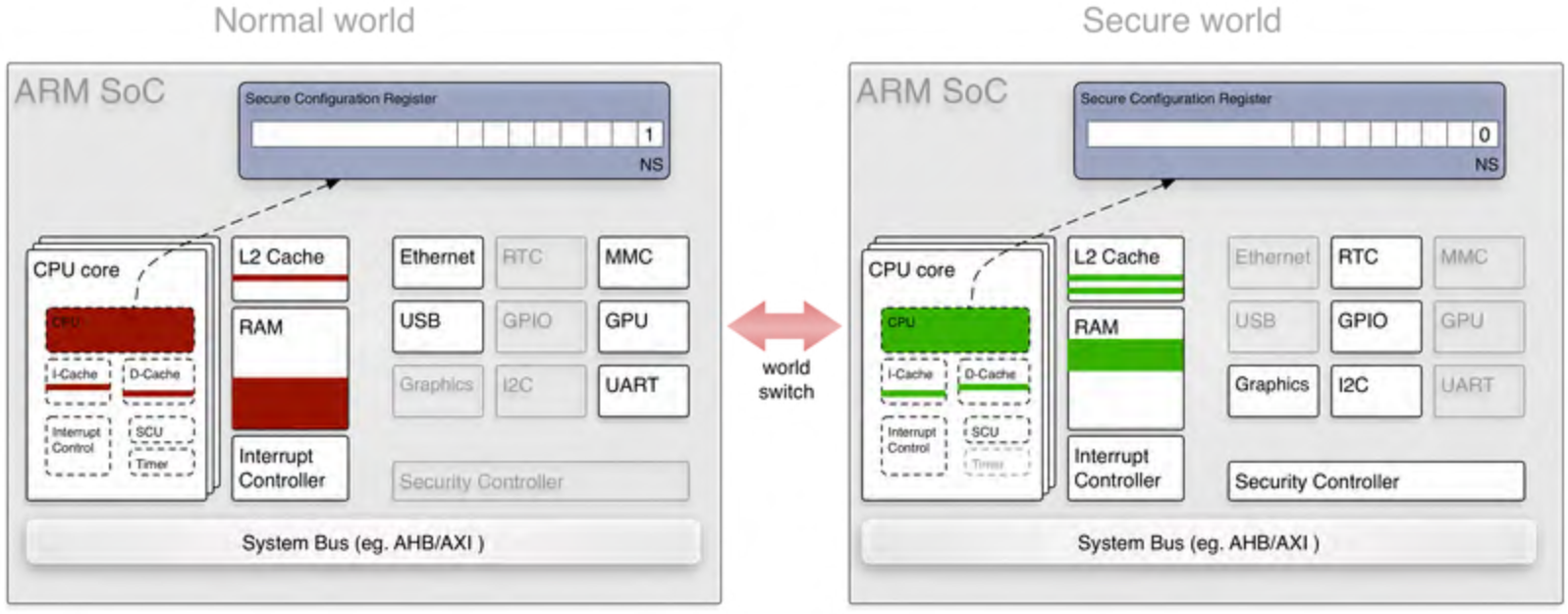
Disclaimer

- This talk provides only introductory level information about TrustZone
- There are so many **undocumented** things about TrustZone that's not even funny to talk about. Some things also requires signing NDA ಠ_ಠ
- The Android ecosystem is a huge mess!
- Btw, is Android really open source?

TrustZone (TZ)

- TZ is a set of security extensions added to ARM processors
- Can run 2 operating systems
  - secure operating system
  - normal operating system
- Hardware protection/isolation of memory and devices

2 worlds



https://genode.org/documentation/articles/trustzone

Features

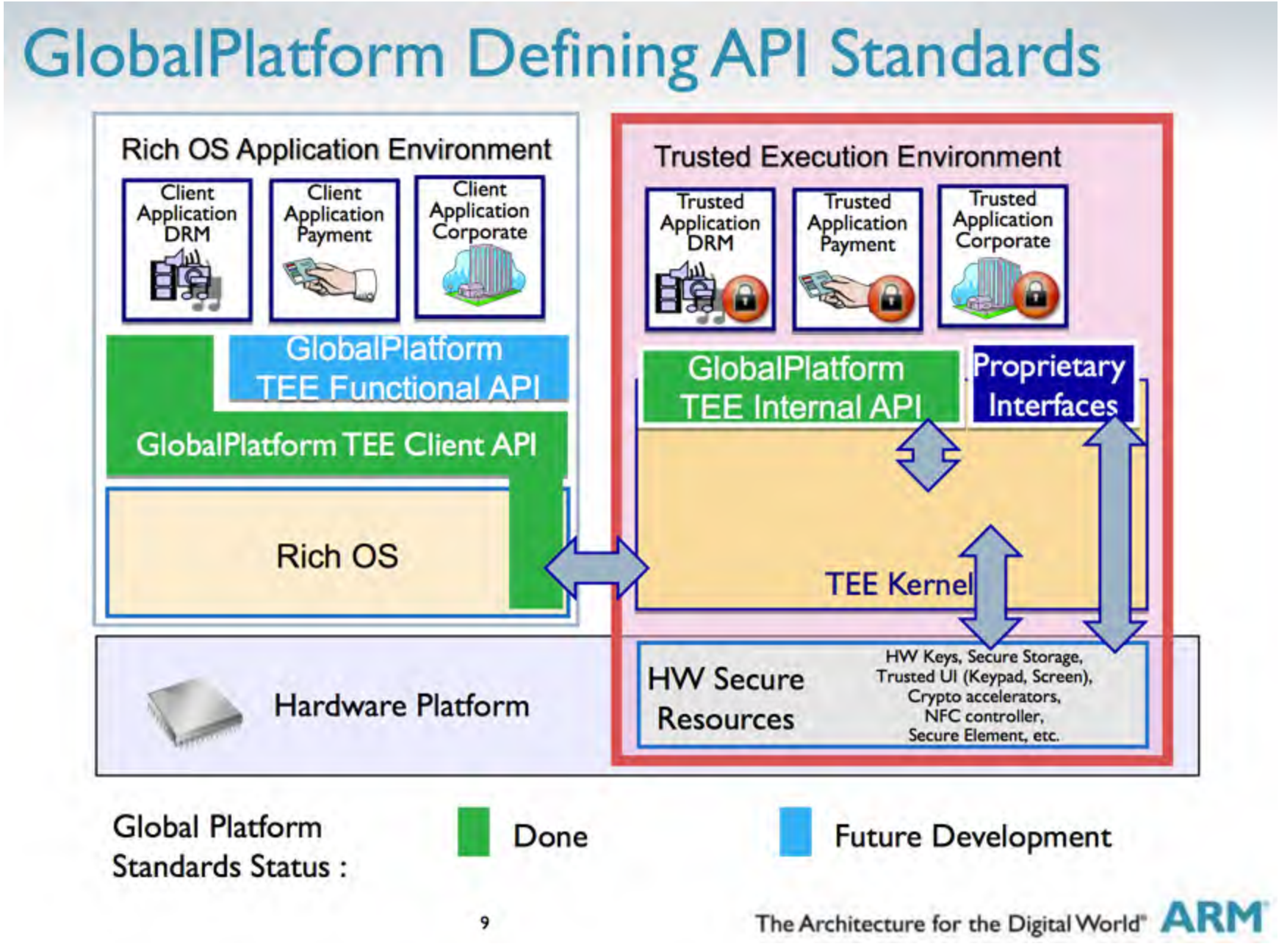
TrustZone is System Wide Security

- Complete TrustZone solution consists of:
  - TrustZone-Enabled CPU Core (eg Cortex®-A5 core)
  - TrustZone secure firmware running on the CPU core
  - TrustZone-Aware L2 cache controller (if L2 cache is used)
  - TrustZone-Aware AXI Interconnect Fabric
  - Secure-World Memory (in addition to Normal World memory)
  - TrustZone-Aware Interrupt Controller
  - On-SoC ROM protection for Trusted Boot Code
  - Off-SoC Memory Address Space Control
  - Secure Debug Control – Disable debug of Secure World

Applications

- Secure storage of crypto keys/secrets
- Trusted User Interface (keypad/screen)
- DRM (obviously!)
- Payment solutions
- ...

Applications



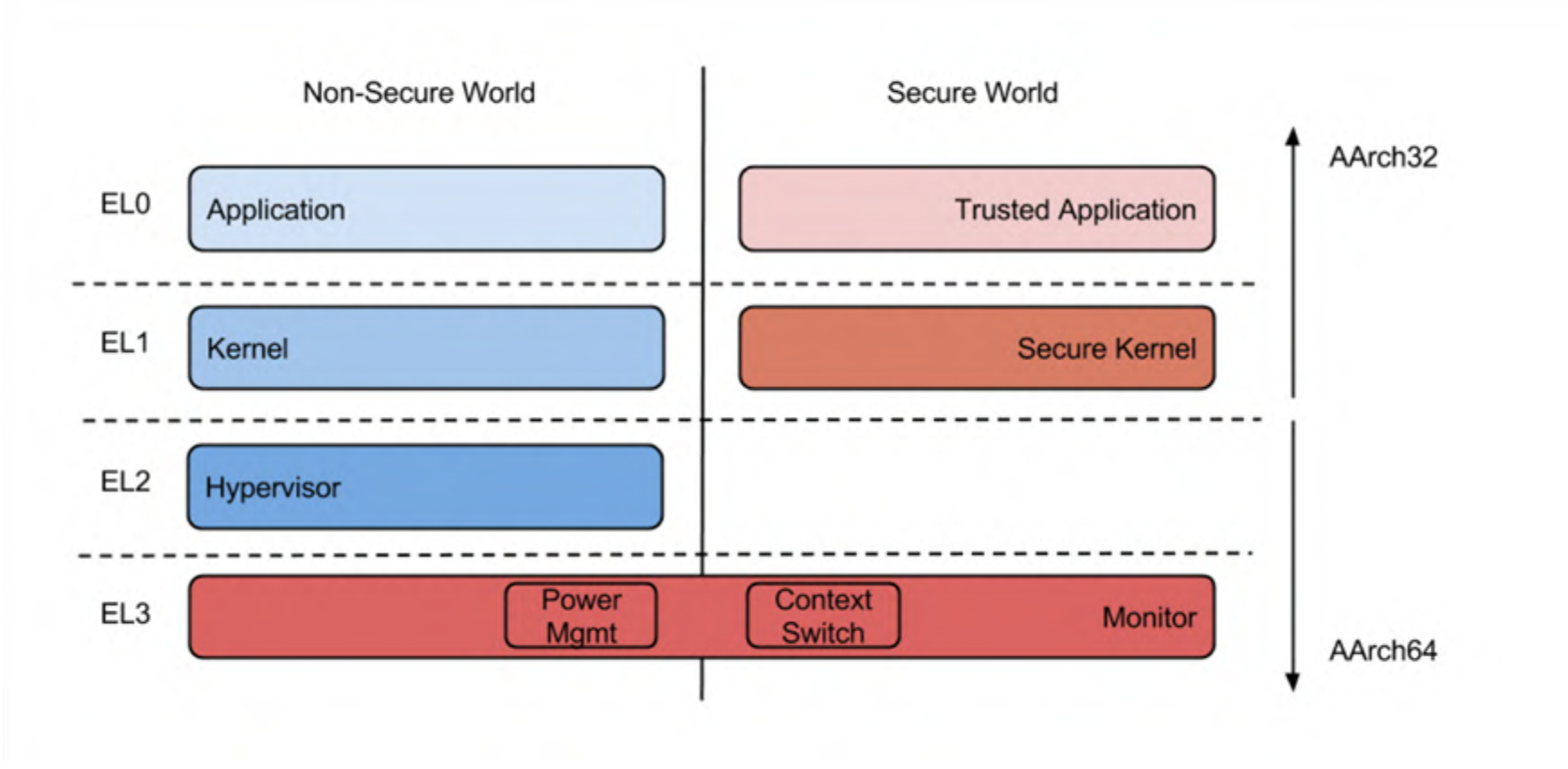
TrustZone - architecture

ARM Execution Levels (EL)

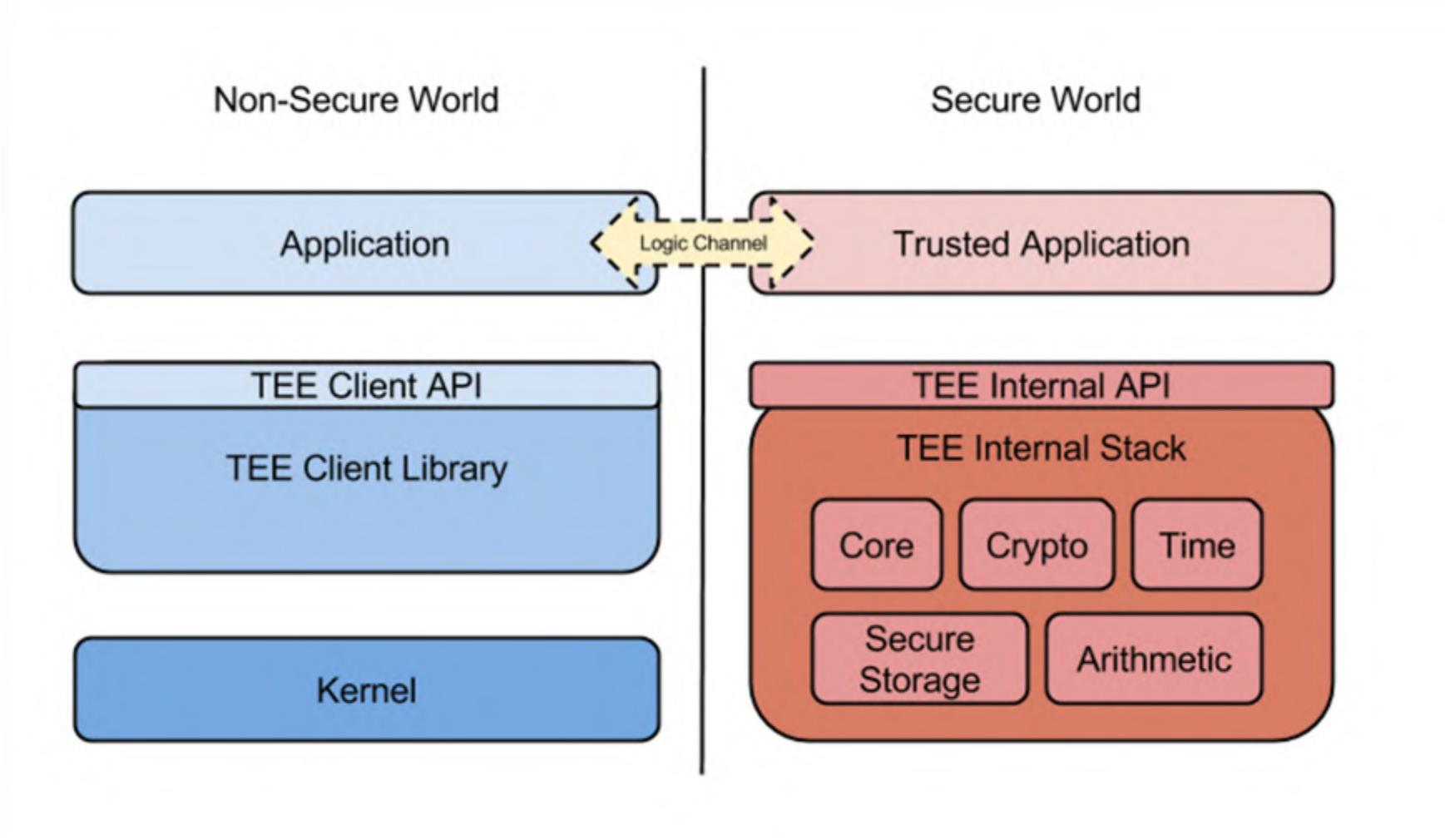
4 executions levels (EL):

- EL0 - usermode
- EL1 - kernel (normal OS)
- EL2 - hypervisor
- EL3 - highest level (secure OS) - TrustZone

TrustZone EL

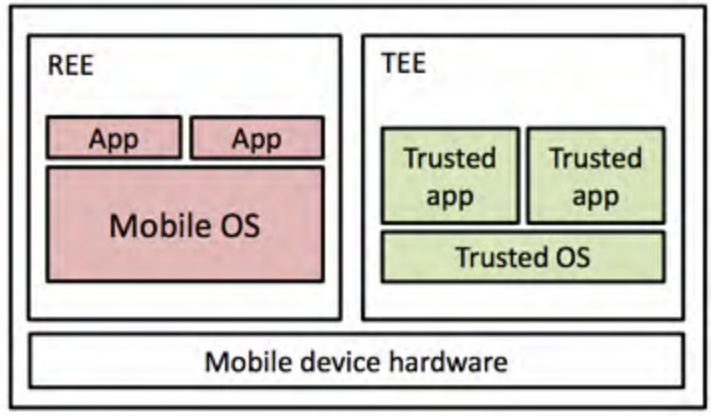


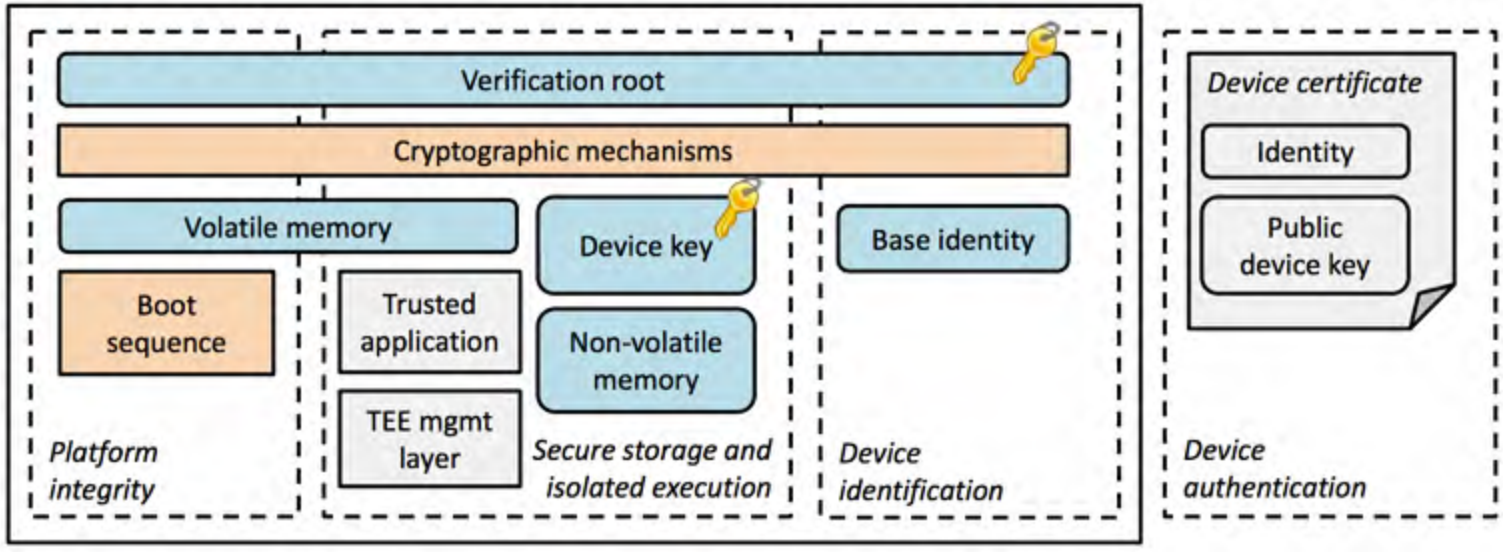
Trusted Execution Environment - TEE



TEE

1. Platform integrity
2. Secure storage
3. Isolated execution
4. Device identification
5. Device authentication





- source: https://www.cs.helsinki.fi/group/secures/CCS-tutorial/tutorial-slides.pdf

Qualcomm Secure Execution Environment - QSEE

- TEE from Qualcomm (driver is open source)

SMC

- **Secure Monitor Call** instruction
- Requires **kernel** (EL1) privilege to be executed
  - Need a device driver
  - Linux kernel provides some functions
- The bridge between the secure and normal world
- There is usually an interface between user-mode applications and TEE device drivers

SCM - Linux kernel

```
171 static u32 smc(u32 cmd_addr)
172 {
173         int context_id;
174         register u32 r0 asm("r0") = 1;
175         register u32 r1 asm("r1") = (u32)&context_id;
176         register u32 r2 asm("r2") = cmd_addr;
177         do {
178                 asm volatile(
179                         __asmeq("%0", "r0")
180                         __asmeq("%1", "r0")
181                         __asmeq("%2", "r1")
182                         __asmeq("%3", "r2")
183                         "smc    #0      @ switch to secure world\n"
184                         : "=r" (r0)
185                         : "r" (r0), "r" (r1), "r" (r2)
186                         : "r3");
187         } while (r0 == SCM_INTERRUPTED);
188
189         return r0;
190 }
```

http://lxr.free-electrons.com/source/arch/arm/mach-msm/scm.c?v=3.0#L171

Secure Configuration Register

- co-processor CP15 c1
- defines current world as Secure/Non-secure
- accessible in secure privileged modes only

Figure 3.29. Secure Configuration Register format

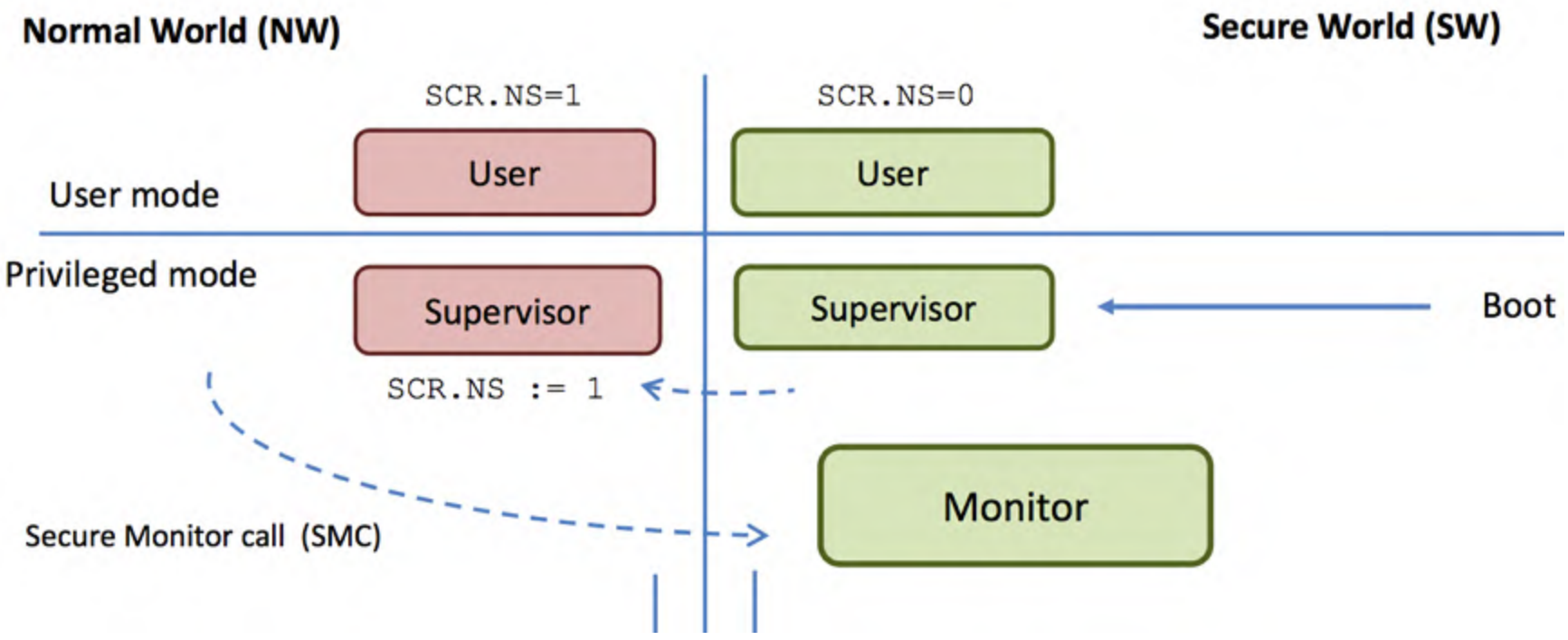
NS bit

- Non-Secure bit

[0]	NS bit	Defines the world for the processor: 0 = Secure, reset value 1 = Non-secure.

- In Secure mode the state is considered Secure regardless of the state of the NS bit

World switch



- src: https://www.cs.helsinki.fi/group/secures/CCS-tutorial/tutorial-slides.pdf

Learning TrustZone

- What options do you have if you want to learn TrustZone by creating real code to run with TZ privileges?
  - ARM Development boards
  - QEMU

Poor Mr Gigu

ARM TrustZone development

8

6

I am wondering if anyone have any information on development boards where you can utilize ARM TrustZone? I have the BeagleBoard XM which uses TI's OMAP3530 with Cortex-A8 processor that supports trust zone, however TI confirmed that they have disabled the function on the board as it is a general purpose device.

Further research got me to the panda board which uses OMAP4430 but there is no response from TI and very little information on the internet. How do you learn how to use trust zone?

Best Regards Mr Gigu

embedded arm trust-zone

share improve this question

edited May 22 '13 at 14:03

artless noise

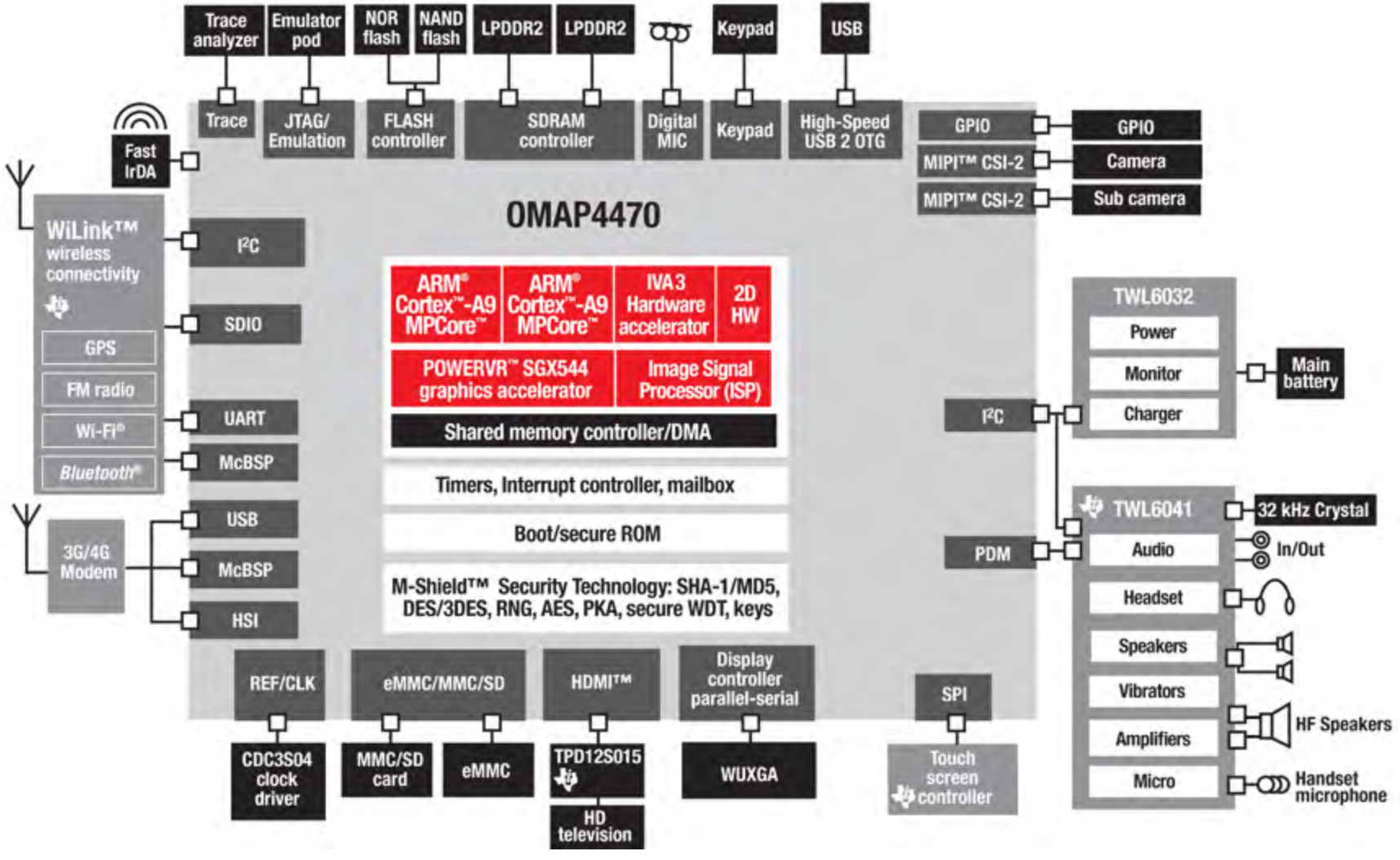
12.1k ● 4 ● 38 ● 69

asked Oct 31 '11 at 15:40

MrGigu

760 ● 2 ● 9 ● 24

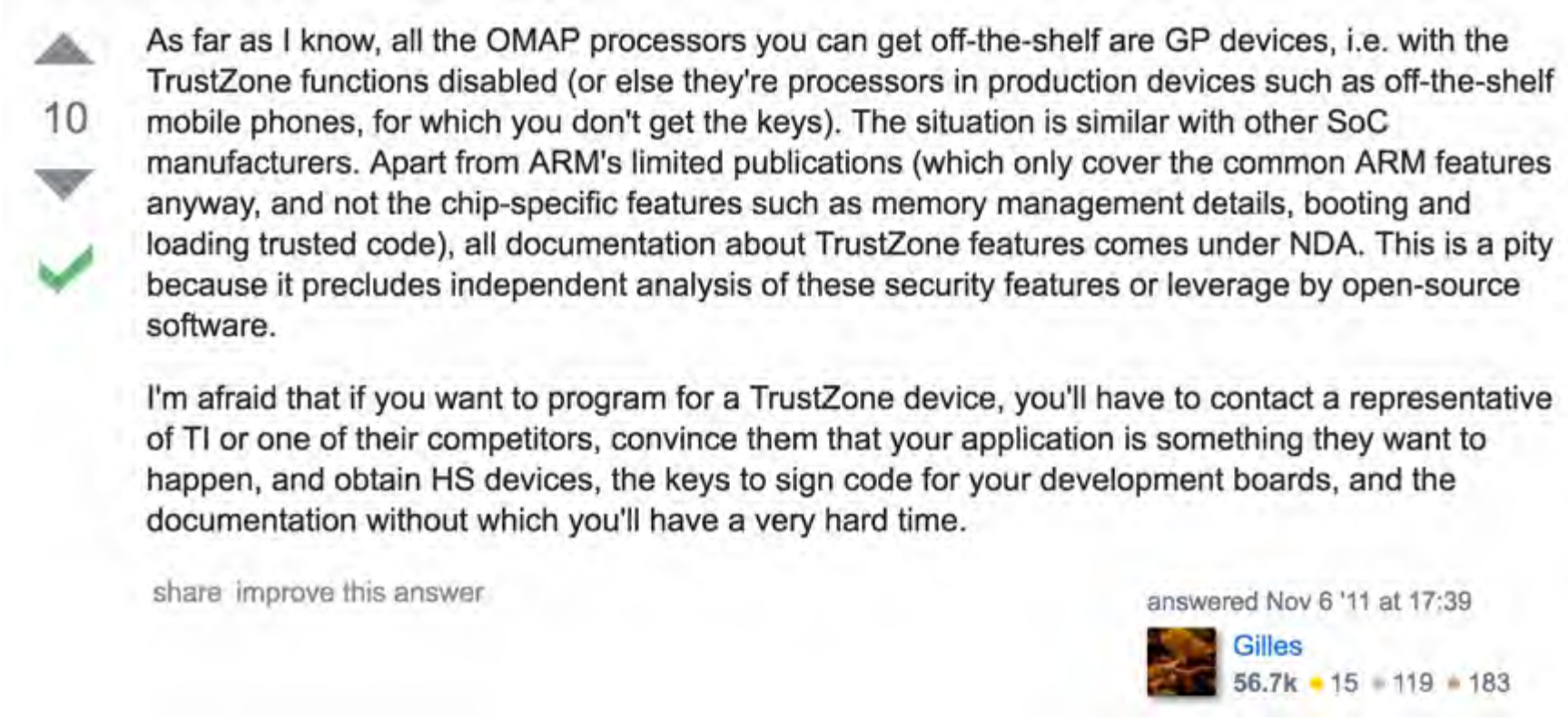
OMAP 4430 - Texas Instrument (TI)



OMAP 4430 - TrustZone support

M-Shield™ mobile security technology enhanced with ARM TrustZone® support and based on open APIs	• Content protection • Transaction security • Secure network access • Secure flashing and booting • Terminal identity protection • Network lock protection

You'll have a very hard time



10

As far as I know, all the OMAP processors you can get off-the-shelf are GP devices, i.e. with the TrustZone functions disabled (or else they're processors in production devices such as off-the-shelf mobile phones, for which you don't get the keys). The situation is similar with other SoC manufacturers. Apart from ARM's limited publications (which only cover the common ARM features anyway, and not the chip-specific features such as memory management details, booting and loading trusted code), all documentation about TrustZone features comes under NDA. This is a pity because it precludes independent analysis of these security features or leverage by open-source software.

I'm afraid that if you want to program for a TrustZone device, you'll have to contact a representative of TI or one of their competitors, convince them that your application is something they want to happen, and obtain HS devices, the keys to sign code for your development boards, and the documentation without which you'll have a very hard time.

share improve this answer

answered Nov 6 '11 at 17:39

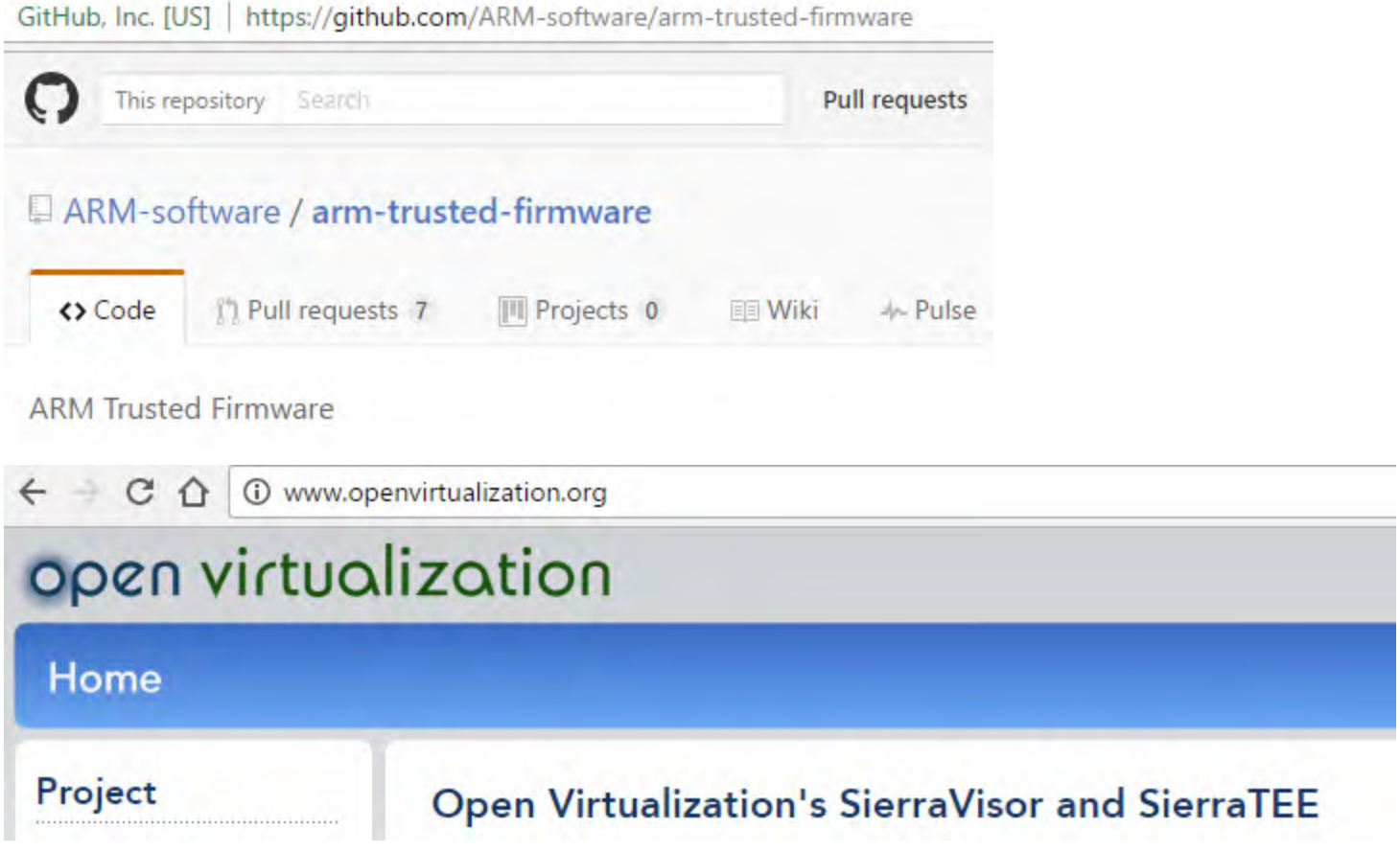
Gilles

56.7k • 15 • 119 • 183

http://stackoverflow.com/questions/7955982/arm-trustzone-development

Trustzone development

- To be fair the situation now is better.
- More information available on the web
- Open-source *reference implementations*



QEMU

- The good folks at Linaro implemented a patch to allow QEMU to run TrustZone extensions
- http://www.linaro.org/blog/core-dump/arm-trustzone-qemu/
- But I need to run TZ code on a real device!
- Let's find a way to do it! :)

TrustZone - Secure Boot

TrustZone - Secure Boot

- "SecureBoot is an on-chip, tamper resistant, ROM-based boot-up process that verifies the authenticity and integrity of critical code and data stored in flash memory."
- "The secure boot process controls the system immediately after reset by executing a known code resident in on-chip Read Only Memory (ROM). This code is the system’s root of trust, and authenticates the code used by the device."

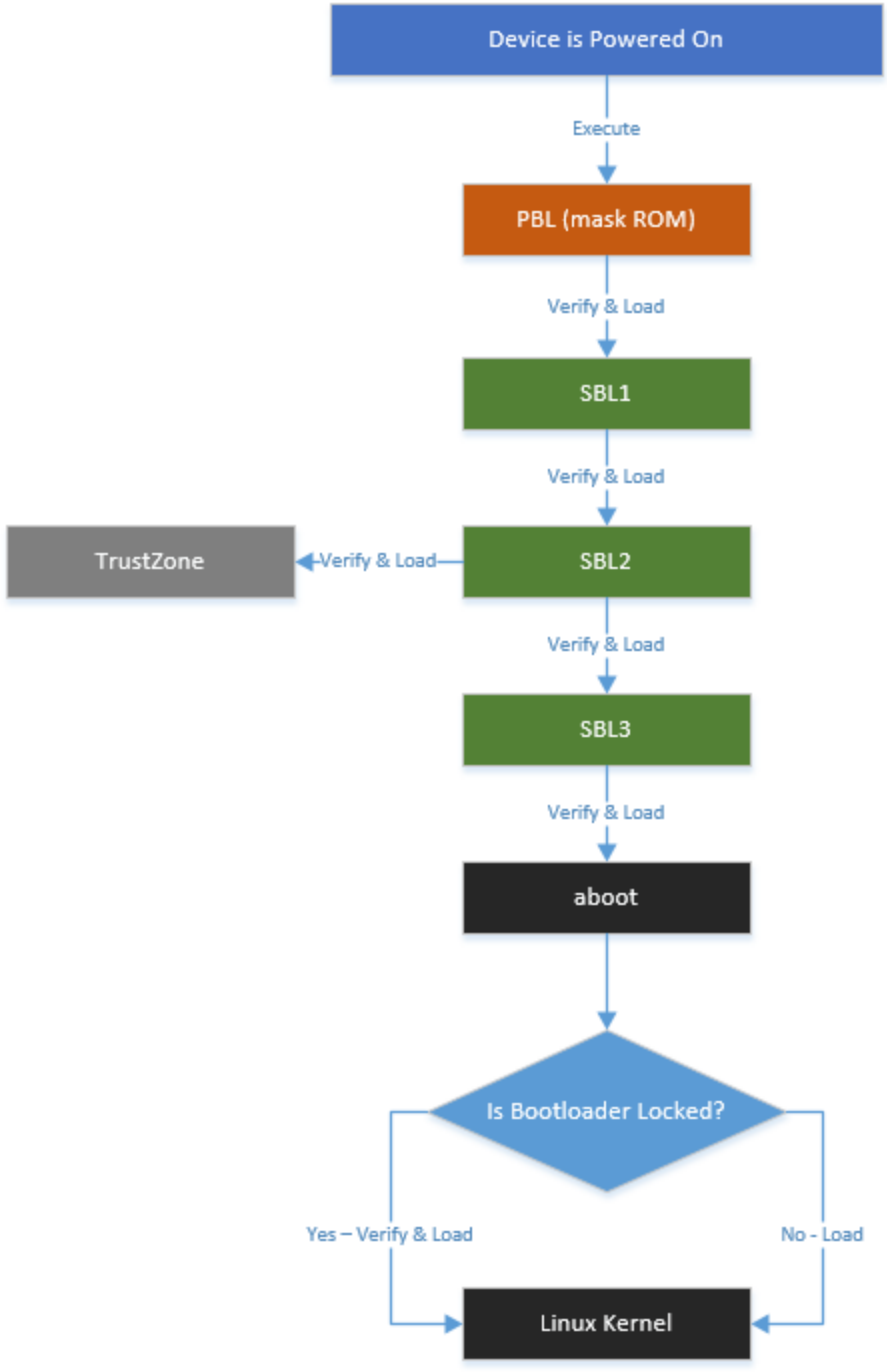
Chain of Trust (CoT) - Boot (1/2)

TrustZone **code integrity** is protected by secure boot which is based on a Chain of Trust (similar to TPM chipsets):

1. After reset the device starts executing the PBL (Primary Boot Loader)
2. The PBL is stored in read-only-memory (ROM) - it is the initial point in the chain - it is a trusted code.
3. Now each step of the boot process will **load** and **authenticate** the next step module/code **before** executing it!

CoT (2/2)

4. The PBL will load and authenticate the Secondary Boot Loader (SBL)
5. The SBL will load and authenticate the **TrustZone** code
6. SBL will then load the Android kernel (aboot partition) and execute it



[src] http://bits-please.blogspot.sg/

The target device

Xiaomi Redmi Note 2

Xiaomi Redmi

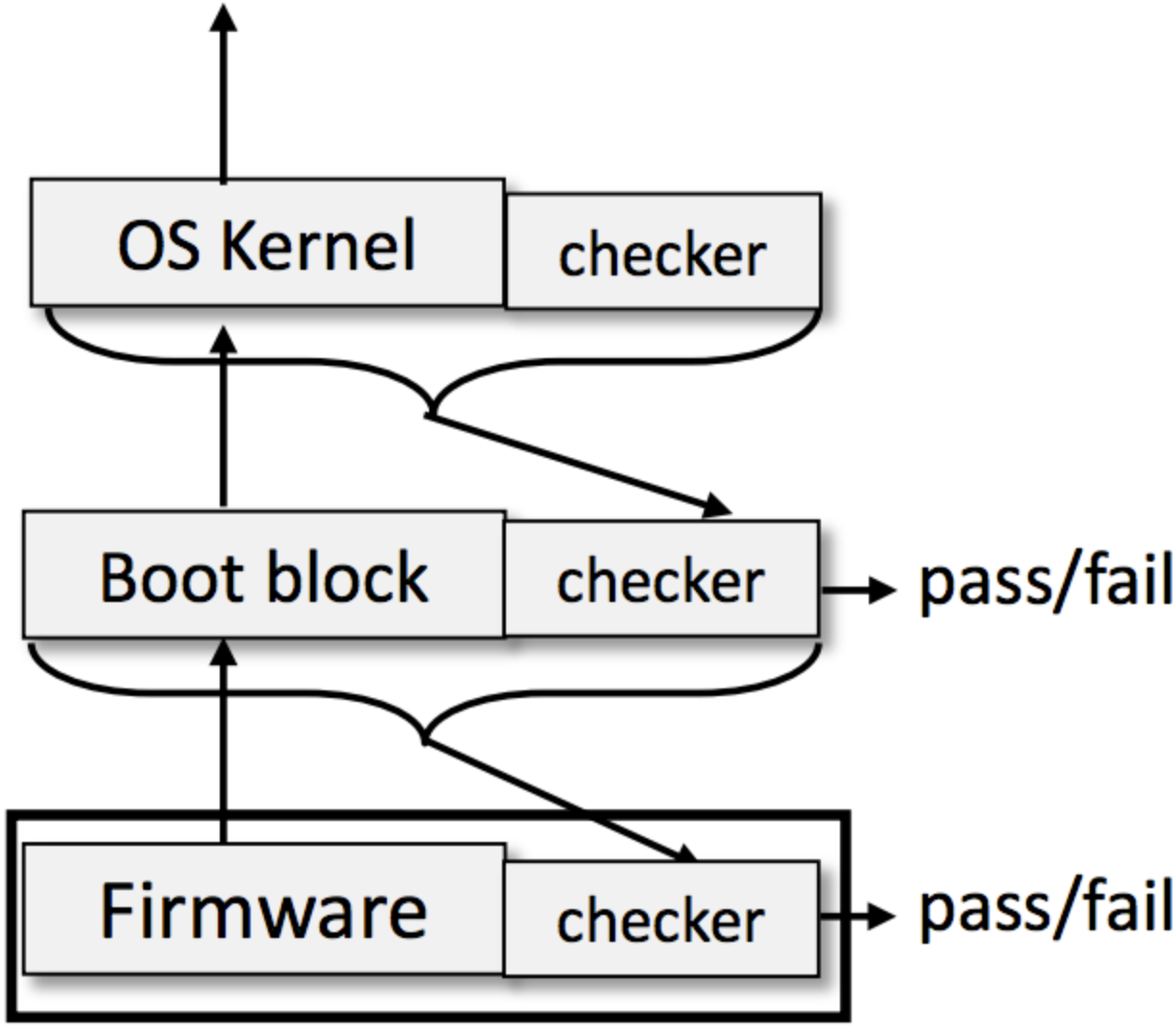
- A very nice Android phone
- Clean UI
- Comes with just a few apps
  - Different from Samsung that comes with tons of useless apps
- Cheap. Great value for the money
- Best of all: allows me to run my TrustZone code :)

Attack surfaces

- QSEE/TEE devices (ioctl)
- TrustZone system calls (accessible using SCM instruction)
  - requires priviledged access
- There is another attacker surface that has been ignored probably because it should obviously brick the device.

Remember Secure Boot?

- This is how it is supposed to work



Secure boot

and customizes its CPU cores, Qualcomm does the same with Qualcomm security. In implementing TrustZone technology, Qualcomm has designed its secure boot as independent Qualcomm proprietary technology that is independent of TrustZone. TrustZone code integrity is protected by secure boot, and is part of the chain of trust; however, it is not the secure boot itself. In the manner in which Qualcomm implements TrustZone, Qualcomm secure boot is the root of trust, and, without it, TrustZone code cannot be trusted, as TrustZone code could be easily modified by unauthorized parties or hackers. TrustZone hardware capability is licensed from ARM, but all the TrustZone support hardware and code is unique to Qualcomm.

Xiaomi Redmi

- The Xiaomi Redmi secure boot process will not **fail** if you overwrite the TrustZone partition!
- The Secondary Boot Loader will load, authenticate and execute the new TrustZone image **regardless** of the authentication result!

How?

Two methods:

- `fastboot flash patched_tz.img`
- `dd if=patched_tz.img of=/dev/block/.../tz` - root required
- It just works!

What now?

- We can run our own TZ code
- We don't need to create a secure OS from scratch
  - also, we don't have access to all the documentation we need for such a herculian task
- We can use the available TZ code and patch it
- But before, we need some reverse engineering of tz.img

Reverse Engineering TrustZone code

Reversing

Obvious first steps:

1. Locate and copy the Trustzone partition
2. Disassembling
3. Analysis
4. ARM code generation
5. Patching

TZ partition - block devices

```
root@HM2014817:/dev/block/platform/7824900.sdhci/by-name # ls -la
lrwxrwxrwx root     root              2014-01-14 00:11 DDR -> /dev/block/mmcblk0p19
lrwxrwxrwx root     root              2014-01-14 00:11 aboot -> /dev/block/mmcblk0p4
lrwxrwxrwx root     root              2014-01-14 00:11 abootbak -> /dev/block/mmcblk0p5
lrwxrwxrwx root     root              2014-01-14 00:11 boot -> /dev/block/mmcblk0p22
lrwxrwxrwx root     root              2014-01-14 00:11 cache -> /dev/block/mmcblk0p24
lrwxrwxrwx root     root              2014-01-14 00:11 config -> /dev/block/mmcblk0p28
lrwxrwxrwx root     root              2014-01-14 00:11 fsc -> /dev/block/mmcblk0p16
lrwxrwxrwx root     root              2014-01-14 00:11 fsg -> /dev/block/mmcblk0p20
lrwxrwxrwx root     root              2014-01-14 00:11 hyp -> /dev/block/mmcblk0p10
lrwxrwxrwx root     root              2014-01-14 00:11 hypbak -> /dev/block/mmcblk0p11
lrwxrwxrwx root     root              2014-01-14 00:11 keystore -> /dev/block/mmcblk0p27
lrwxrwxrwx root     root              2014-01-14 00:11 misc -> /dev/block/mmcblk0p15
lrwxrwxrwx root     root              2014-01-14 00:11 modem -> /dev/block/mmcblk0p1
lrwxrwxrwx root     root              2014-01-14 00:11 modemst1 -> /dev/block/mmcblk0p13
lrwxrwxrwx root     root              2014-01-14 00:11 modemst2 -> /dev/block/mmcblk0p14
lrwxrwxrwx root     root              2014-01-14 00:11 oem -> /dev/block/mmcblk0p29
lrwxrwxrwx root     root              2014-01-14 00:11 pad -> /dev/block/mmcblk0p12
lrwxrwxrwx root     root              2014-01-14 00:11 persist -> /dev/block/mmcblk0p25
lrwxrwxrwx root     root              2014-01-14 00:11 recovery -> /dev/block/mmcblk0p26
lrwxrwxrwx root     root              2014-01-14 00:11 rpm -> /dev/block/mmcblk0p6
lrwxrwxrwx root     root              2014-01-14 00:11 rpmbak -> /dev/block/mmcblk0p7
lrwxrwxrwx root     root              2014-01-14 00:11 sbl1 -> /dev/block/mmcblk0p2
lrwxrwxrwx root     root              2014-01-14 00:11 sbl1bak -> /dev/block/mmcblk0p3
lrwxrwxrwx root     root              2014-01-14 00:11 sec -> /dev/block/mmcblk0p21
lrwxrwxrwx root     root              2014-01-14 00:11 splash -> /dev/block/mmcblk0p18
lrwxrwxrwx root     root              2014-01-14 00:11 ssd -> /dev/block/mmcblk0p17
lrwxrwxrwx root     root              2014-01-14 00:11 system -> /dev/block/mmcblk0p23
lrwxrwxrwx root     root              2014-01-14 00:11 tz -> /dev/block/mmcblk0p8
lrwxrwxrwx root     root              2014-01-14 00:11 tzbak -> /dev/block/mmcblk0p9
lrwxrwxrwx root     root              2014-01-14 00:11 userdata -> /dev/block/mmcblk0p30
root@HM2014817:/dev/block/platform/7824900.sdhci/by-name #
```

TrustZone

- 2 TrustZone images `tz` and `tzbak`. They are the same. If `tz` is corrupted, `tzbak` is loaded instead.
- Just copy it using `dd`

```
lrwxrwxrwx root     root              2014-01-14 00:11 tz -> /dev/block/mmcblk0p8
lrwxrwxrwx root     root              2014-01-14 00:11 tzbak -> /dev/block/mmcblk0p9
```

```
opcode@ubuntu:~/src/arm-eabi-4.6/bin$ ./arm-eabi-objdump -x ~/Desktop/tz.img

/home/opcode/Desktop/tz.img:     file format elf32-littlearm
/home/opcode/Desktop/tz.img
architecture: arm, flags 0x00000102:
EXEC_P, D_PAGED
start address 0x86500000

Program Header:
    NULL off    0x00000000 vaddr 0x00000000 paddr 0x00000000 align 2**0
         filesz 0x00000194 memsz 0x00000000 flags --- 7000000
    NULL off    0x00001000 vaddr 0x86570000 paddr 0x86570000 align 2**12
         filesz 0x00001a68 memsz 0x00002000 flags --- 2200000
    LOAD off    0x00010000 vaddr 0x86500000 paddr 0x86500000 align 2**16
         filesz 0x00036000 memsz 0x00036000 flags r-x 80000000
    LOAD off    0x00046000 vaddr 0x86536000 paddr 0x86536000 align 2**12
         filesz 0x00005bb8 memsz 0x00005bb8 flags r--
    LOAD off    0x0004c000 vaddr 0x8653c000 paddr 0x8653c000 align 2**12
         filesz 0x0000b624 memsz 0x00011a60 flags rw-
    LOAD off    0x00057624 vaddr 0x8654dc00 paddr 0x8654dc00 align 2**12
         filesz 0x00004800 memsz 0x00004800 flags rw-
    LOAD off    0x00063e24 vaddr 0x86567000 paddr 0x86567000 align 2**7
         filesz 0x00000ba8 memsz 0x00000ba8 flags rw-
    LOAD off    0x0005be24 vaddr 0x86568000 paddr 0x86568000 align 2**12
         filesz 0x00003000 memsz 0x00003000 flags rw-
    LOAD off    0x0005ee24 vaddr 0x8656b000 paddr 0x8656b000 align 2**2
         filesz 0x00001000 memsz 0x00001000 flags rw-
    LOAD off    0x0005fe24 vaddr 0x8656c000 paddr 0x8656c000 align 2**14
         filesz 0x00004000 memsz 0x00004000 flags rw-
    LOAD off    0x00065000 vaddr 0x86574000 paddr 0x86574000 align 2**12
         filesz 0x00004000 memsz 0x00004000 flags rwx 80000000
private flags = 5000002: [Version5 EABI] [has entry point]
```

Strings Paradise!

```
opcode@ubuntu:~/Desktop$ strings tz.img | egrep "fail|fali"
ICB_Get_Memmap failed: %u
PTBLcounter: tzbsp_query_rpmb failed
counter: rpmb init fail {%x} {%x}
(%u)secboot_get_fuse_info_from_image falied %u
(%u)secboot_init_fuses falied %u
(%u)secboot_authenticate falied %u
Cipher Init failed
SetParam Mode failed
SetParam Key failed
SetParam IV failed
Cipher decrypt failed
Cipher encrypt failed
tzbsp_hmac failed
HMAC computation failed
HMAC comparison failed
nhcsRNG failed for cipher key
RNG failed for hmac key
Setup flow info failed
Setup cipher info failed
Encryption failed
HMAC creation failed
Header validation failed
HMAC validation failed
Decryption failed
rollback ver update failed (%u) %u, %u
```

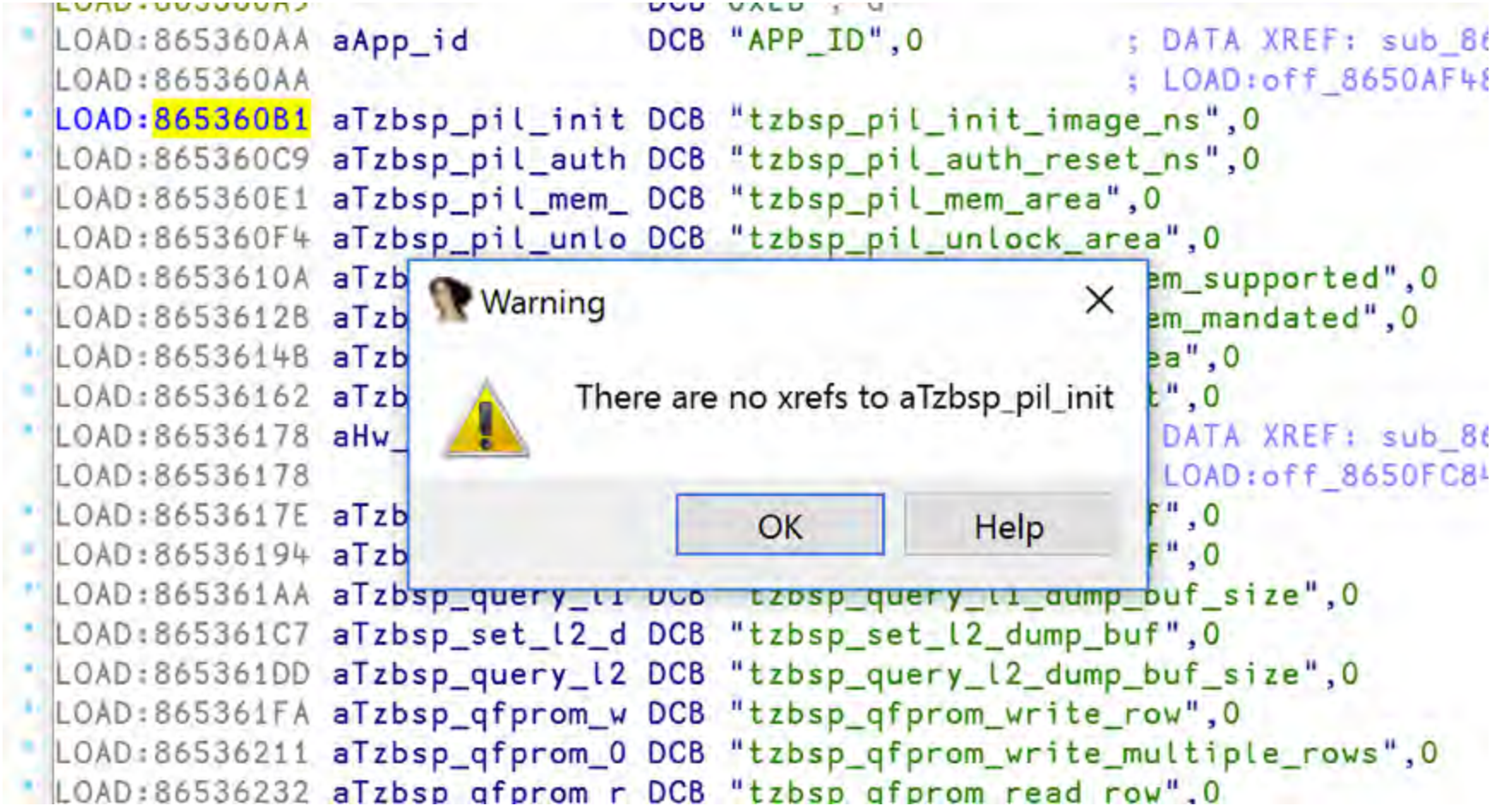
System calls

- TrustZone system calls are a good initial target for patching
- Now that we have access to the trustzone image let's start by locating the exported system calls.
- You can find the name of the system calls using `strings` and `grep`

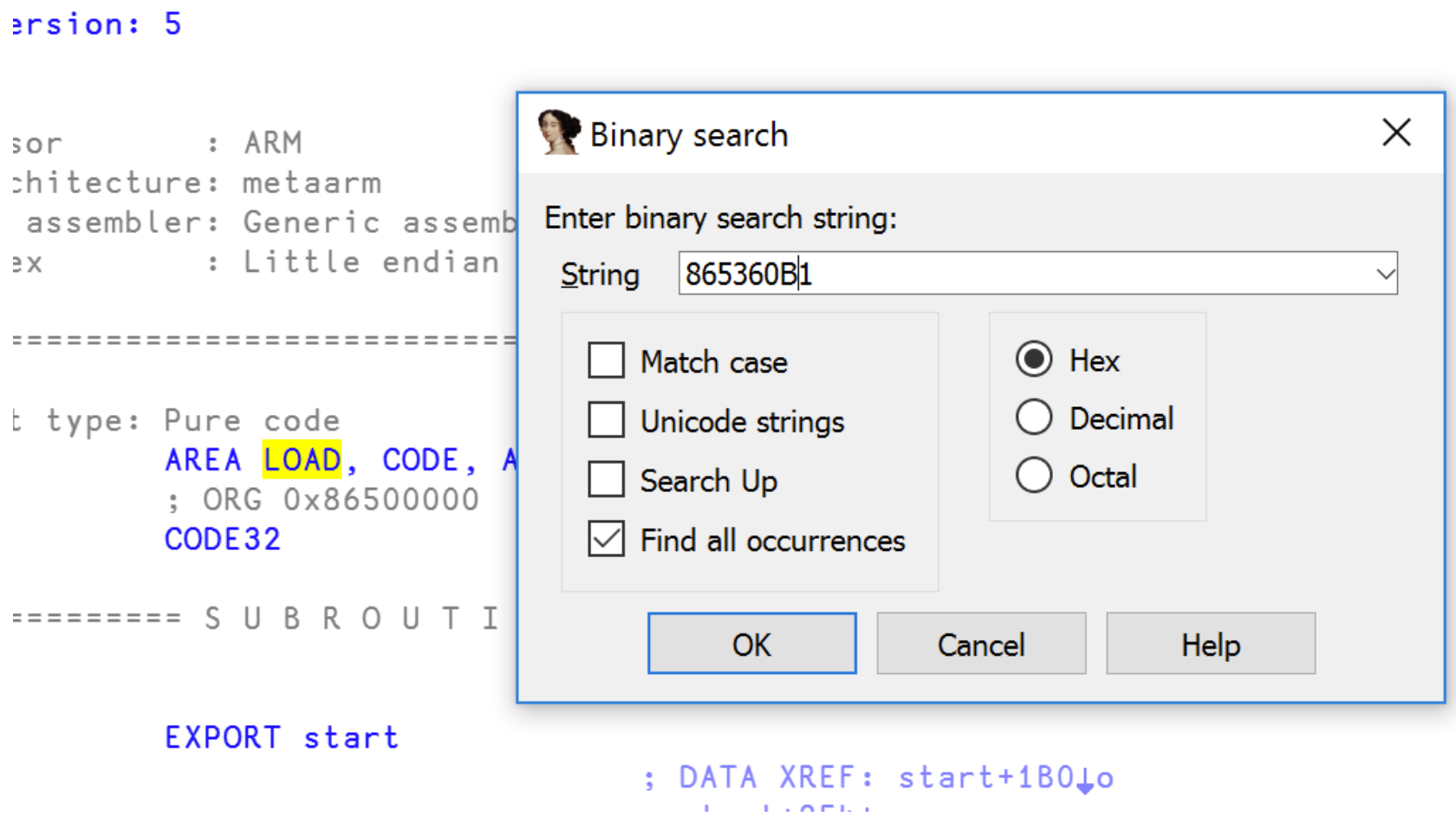
Syscalls

```
opcode@ubuntu:~/Desktop$ strings tz.img | grep ^tzbsp_ | grep -v -E '\(| '
tzbsp_pil_init_image_ns
tzbsp_pil_auth_reset_ns
tzbsp_pil_mem_area
tzbsp_pil_unlock_area
tzbsp_pil_is_subsystem_supported
tzbsp_pil_is_subsystem_mandated
tzbsp_pil_get_mem_area
tzbsp_pil_modem_reset
tzbsp_set_cpu_ctx_buf
tzbsp_set_l1_dump_buf
tzbsp_query_l1_dump_buf_size
tzbsp_set_l2_dump_buf
tzbsp_query_l2_dump_buf_size
tzbsp_qfprom_write_row
tzbsp_qfprom_write_multiple_rows
tzbsp_qfprom_read_row
tzbsp_qfprom_rollback_write_row
tzbsp_prng_getdata_syscall
tzbsp_resource_config
tzbsp_dcvs_create_group
tzbsp_dcvs_register_core
tzbsp_dcvs_set_alg_params
tzbsp_dcvs_init
```

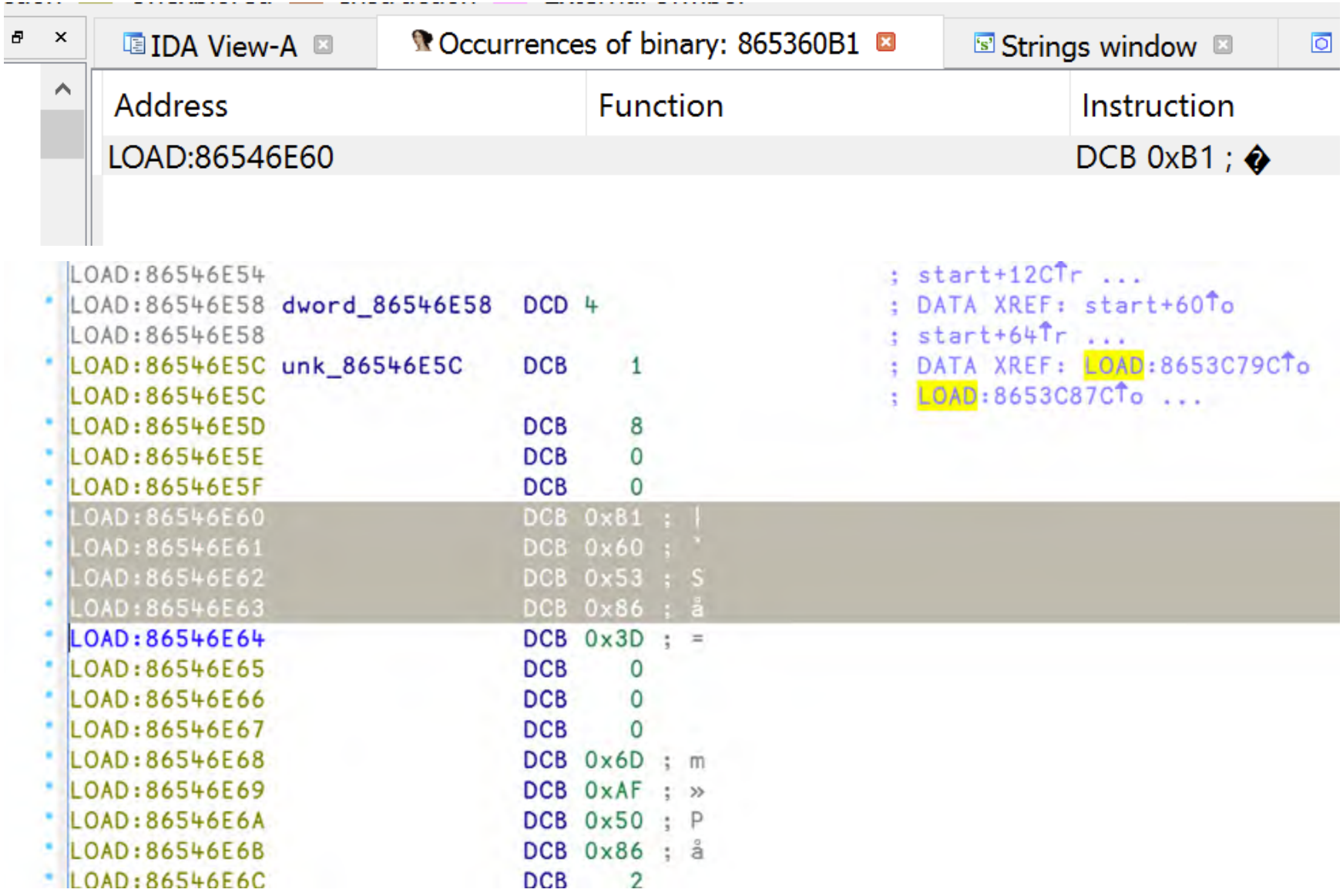
Syscalls - no xref



Searching xref



Search result



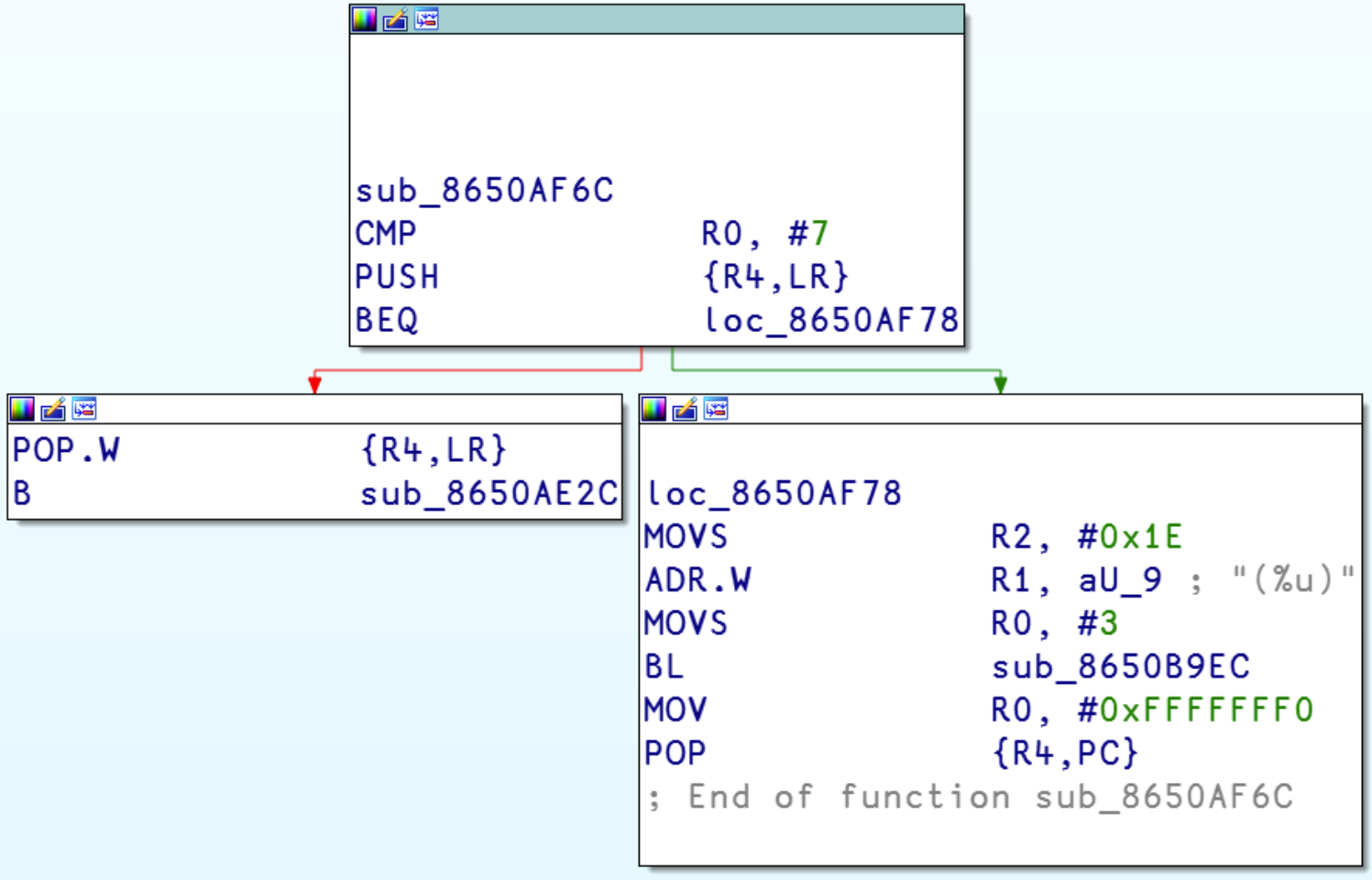
Pointer to syscall name

```
LOAD:86546E5C unk_86546E5C    DCB    1              ; DATA XREF: LOAD:8653C79C↑o
LOAD:86546E5C                                       ; LOAD:8653C87C↑o ...
LOAD:86546E5D                 DCB    8
LOAD:86546E5E                 DCB    0
LOAD:86546E5F                 DCB    0
LOAD:86546E60                 DCD aTzbsp_pil_init   ; "tzbsp_pil_init_image_ns"
LOAD:86546E64                 DCB 0x3D ; =
LOAD:86546E65                 DCB    0
LOAD:86546E66                 DCB    0
LOAD:86546E67                 DCB    0
LOAD:86546E68                 DCB 0x6D ; m
LOAD:86546E69                 DCB 0xAF ; »
LOAD:86546E6A                 DCB 0x50 ; P
LOAD:86546E6B                 DCB 0x86 ; å
LOAD:86546E6C                 DCB    2
LOAD:86546E6D                 DCB    0
LOAD:86546E6E                 DCB    0
```

Pointer to the syscall code

```
LOAD:86546E5E                   DCB    0
LOAD:86546E5F                   DCB    0
LOAD:86546E60                   DCD aTzbsp_pil_init     ; "tzbsp_pil_init_image_ns"
LOAD:86546E64                   DCB 0x3D ; =
LOAD:86546E65                   DCB    0
LOAD:86546E66                   DCB    0
LOAD:86546E67                   DCB    0
LOAD:86546E68                   DCD 0x8650AF6D
LOAD:86546E6C                   DCB    2
LOAD:86546E6D                   DCB    0
LOAD:86546E6E                   DCB    0
LOAD:86546E6F                   DCB    0
```

`tzbsp_pil_init_image_ns` syscall



There is a pattern!

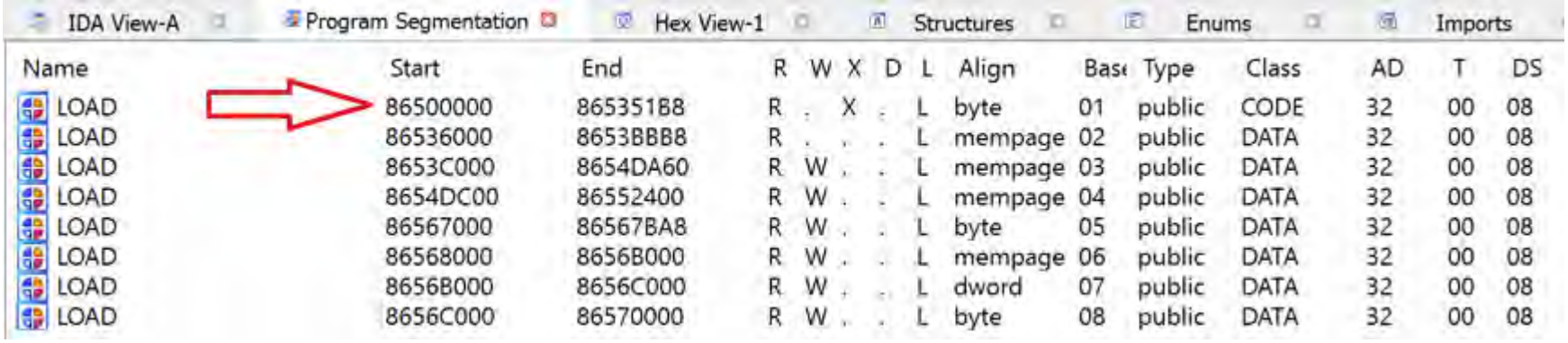
```
LOAD:86546E58                                         ; start+64↑r ...
LOAD:86546E5C dword_86546E5C  DCD 0x801               ; DATA XREF: LOAD:8653C79C↑o
LOAD:86546E5C                                         ; LOAD:8653C87C↑o ...
LOAD:86546E60                 DCD aTzbsp_pil_init     ; "tzbsp_pil_init_image_ns"
LOAD:86546E64                 DCD 0x3D
LOAD:86546E68                 DCD sub_8650AF6C+1
LOAD:86546E6C                 DCD 2
LOAD:86546E70                 DCD 4
LOAD:86546E74                 DCD 4
LOAD:86546E78                 DCD 0x805
LOAD:86546E7C                 DCD aTzbsp_pil_auth     ; "tzbsp_pil_auth_reset_ns"
LOAD:86546E80                 DCD 0x3D
LOAD:86546E84                 DCD sub_8650B188+1
LOAD:86546E88                 DCD 1
LOAD:86546E8C                 DCD 4
LOAD:86546E90                 DCD 0x802
LOAD:86546E94                 DCD aTzbsp_pil_mem_     ; "tzbsp_pil_mem_area"
LOAD:86546E98                 DCD 0xD
LOAD:86546E9C                 DCD sub_8650AB6E+1
LOAD:86546EA0                 DCD 3
LOAD:86546EA4                 DCD 4
LOAD:86546EA8                 DCD 4
LOAD:86546EAC                 DCD 4
LOAD:86546EB0                 DCD 0x806
LOAD:86546EB4                 DCD aTzbsp_pil_unlo     ; "tzbsp_pil_unlock_area"
```

SMC table format

```
LOAD:86546E78    DCD 0x805                ; (ServiceId<<10|CommandId)
LOAD:86546E7C    DCD aTzbsp_pil_auth      ; "tzbsp_pil_auth_reset_ns"
LOAD:86546E80    DCD 0x3D
LOAD:86546E84    DCD sub_86508188+1       ; Pointer to the syscall handler
LOAD:86546E88    DCD 1                    ; Number of args
LOAD:86546E8C    DCD 4                    ; Size of arg
```

- Detailed table format explanation:
  - http://bits-please.blogspot.sg/2015/08/exploring-qualcomms-trustzone.html
- Now we can patch a system call

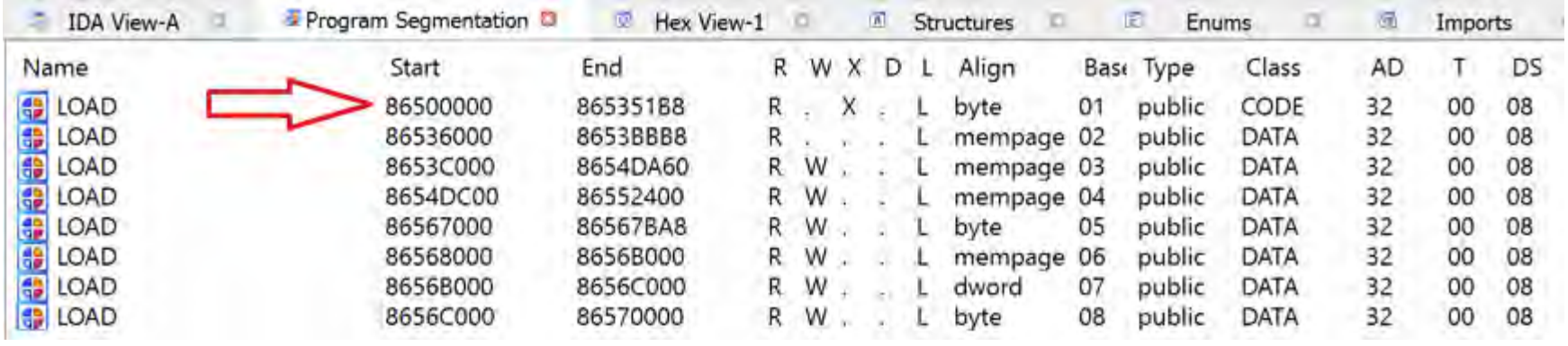
Patching ELF headers (segments)



Name	Start	End	R	W	X	D	L	Align	Base	Type	Class	AD	T	DS
LOAD	86500000	86535188	R	.	X	.	L	byte	01	public	CODE	32	00	08
LOAD	86536000	8653BBB8	R	.	.	.	L	mempage	02	public	DATA	32	00	08
LOAD	8653C000	8654DA60	R	W	.	.	L	mempage	03	public	DATA	32	00	08
LOAD	8654DC00	86552400	R	W	.	.	L	mempage	04	public	DATA	32	00	08
LOAD	86567000	86567BA8	R	W	.	.	L	byte	05	public	DATA	32	00	08
LOAD	86568000	8656B000	R	W	.	.	L	mempage	06	public	DATA	32	00	08
LOAD	8656B000	8656C000	R	W	.	.	L	dword	07	public	DATA	32	00	08
LOAD	8656C000	86570000	R	W	.	.	L	byte	08	public	DATA	32	00	08

- There is only one executable segment on the original TZ image
- The first experiment was to patch the `get_version` system call
- To give more space for the new code we expanded the segment to the maximum allowed value

Patching ELF headers (segments)

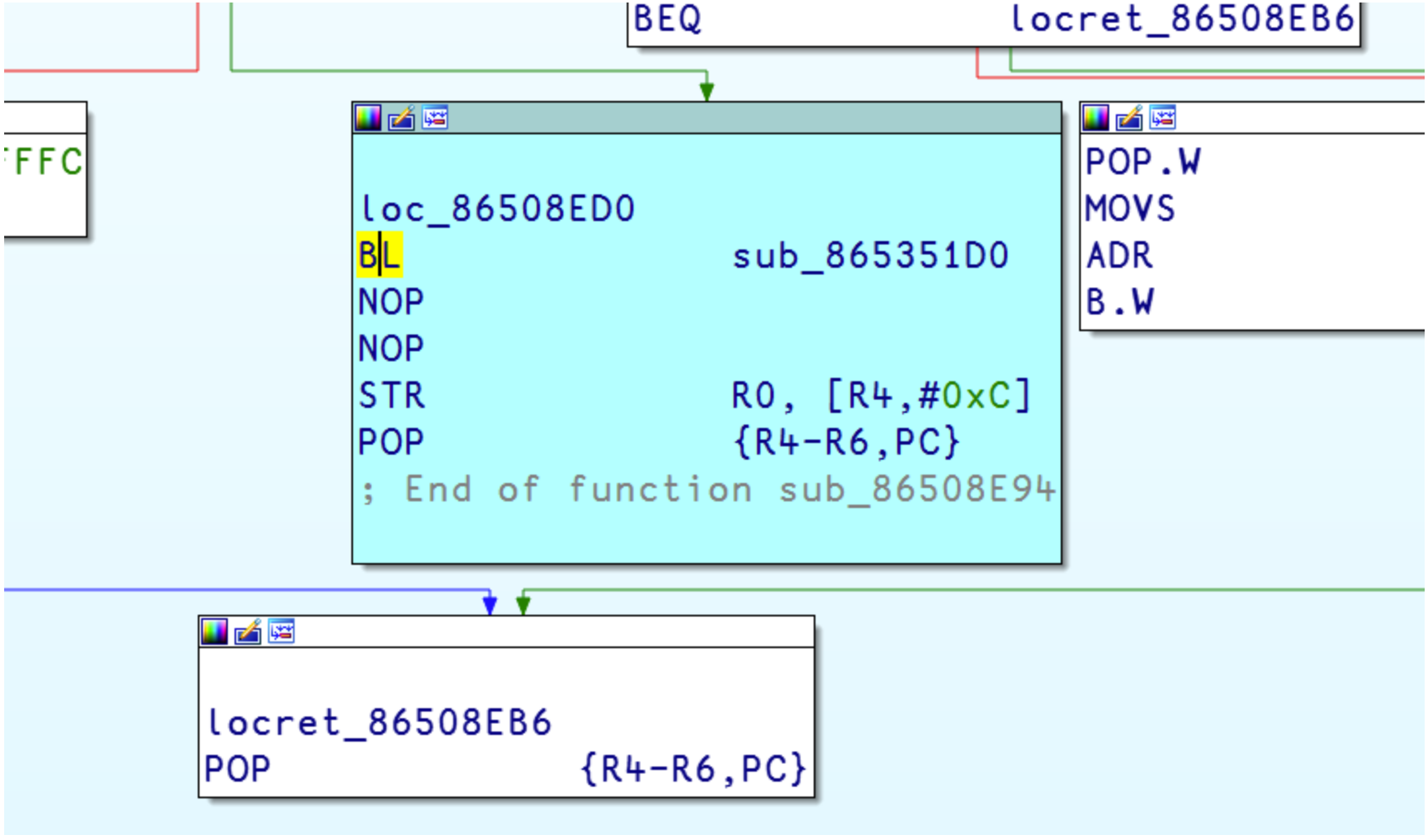


IDA View-A | Program Segmentation | Hex View-1 | Structures | Enums | Imports

Name	Start	End	R	W	X	D	L	Align	Base	Type	Class	AD	T	DS
LOAD	86500000	865351B8	R	.	X	.	L	byte	01	public	CODE	32	00	08
LOAD	86536000	8653BBB8	R	.	.	.	L	mempage	02	public	DATA	32	00	08
LOAD	8653C000	8654DA60	R	W	.	.	L	mempage	03	public	DATA	32	00	08
LOAD	8654DC00	86552400	R	W	.	.	L	mempage	04	public	DATA	32	00	08
LOAD	86567000	86567BA8	R	W	.	.	L	byte	05	public	DATA	32	00	08
LOAD	86568000	8656B000	R	W	.	.	L	mempage	06	public	DATA	32	00	08
LOAD	8656B000	8656C000	R	W	.	.	L	dword	07	public	DATA	32	00	08
LOAD	8656C000	86570000	R	W	.	.	L	byte	08	public	DATA	32	00	08

- Expand (maximize) eXecutable segment
  - range `0x86500000 - 0x865351b8`
  - range `0x86500000 - 0x86536000`

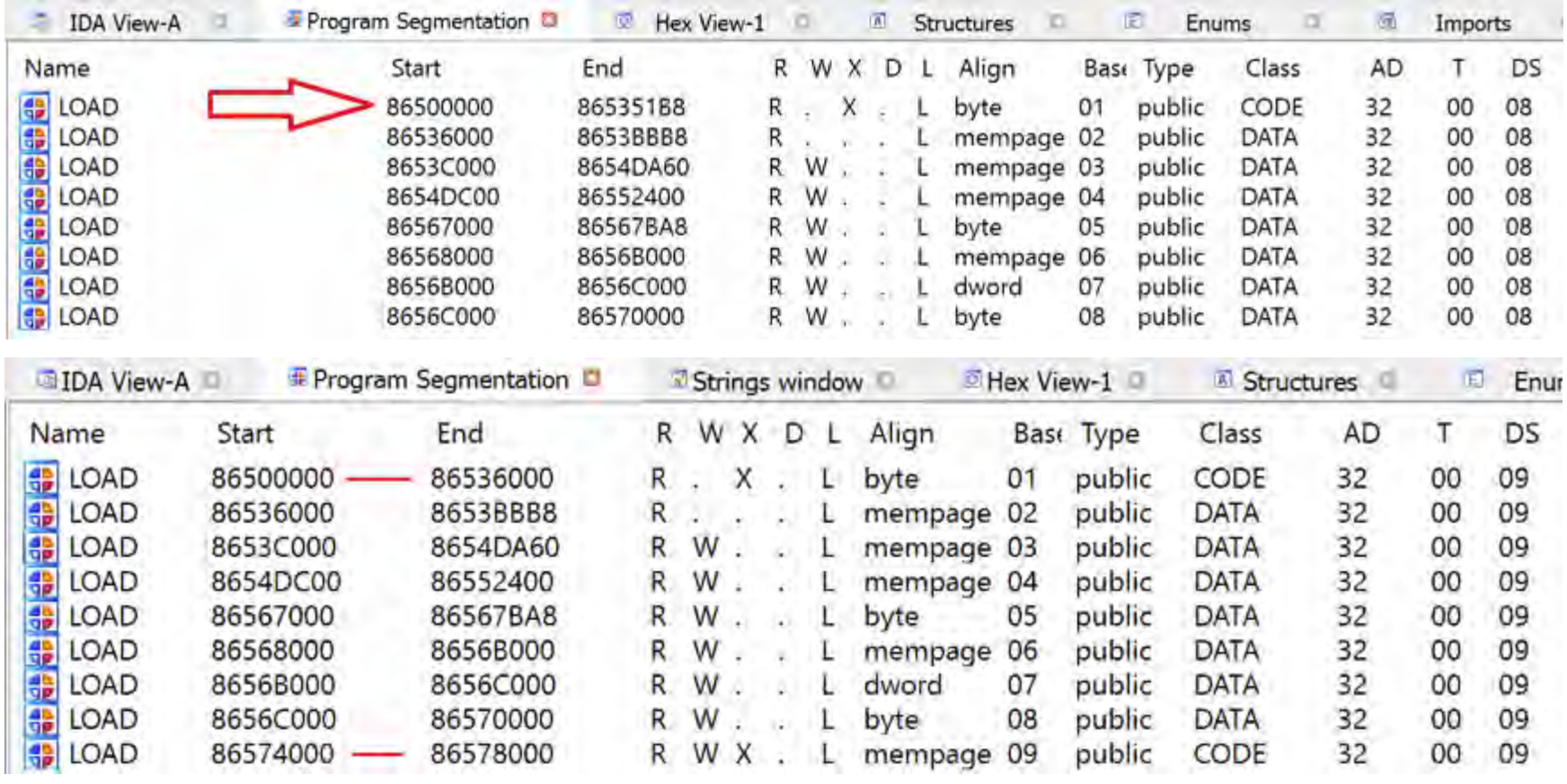
Patching `get_version`



Patching problems

- I created a new function at the end of the expanded segment and patched the `get_version` with a branch to the new function.
- It works! `get_version` was returning a new value.
- To have even more space to create new functions I decided to create a new segment in the TrustZone image

Patching ELF - new executable segment



IDA View-A | Program Segmentation | Hex View-1 | Structures | Enums | Imports

Name	Start	End	R	W	X	D	L	Align	Base	Type	Class	AD	T	DS
LOAD	86500000	86535188	R	.	X	.	L	byte	01	public	CODE	32	00	08
LOAD	86536000	8653BBB8	R	.	.	.	L	mempage	02	public	DATA	32	00	08
LOAD	8653C000	8654DA60	R	W	.	.	L	mempage	03	public	DATA	32	00	08
LOAD	8654DC00	86552400	R	W	.	.	L	mempage	04	public	DATA	32	00	08
LOAD	86567000	86567BA8	R	W	.	.	L	byte	05	public	DATA	32	00	08
LOAD	86568000	8656B000	R	W	.	.	L	mempage	06	public	DATA	32	00	08
LOAD	8656B000	8656C000	R	W	.	.	L	dword	07	public	DATA	32	00	08
LOAD	8656C000	86570000	R	W	.	.	L	byte	08	public	DATA	32	00	08

IDA View-A | Program Segmentation | Strings window | Hex View-1 | Structures | Enu

Name	Start	End	R	W	X	D	L	Align	Base	Type	Class	AD	T	DS
LOAD	86500000	86536000	R	.	X	.	L	byte	01	public	CODE	32	00	09
LOAD	86536000	8653BBB8	R	.	.	.	L	mempage	02	public	DATA	32	00	09
LOAD	8653C000	8654DA60	R	W	.	.	L	mempage	03	public	DATA	32	00	09
LOAD	8654DC00	86552400	R	W	.	.	L	mempage	04	public	DATA	32	00	09
LOAD	86567000	86567BA8	R	W	.	.	L	byte	05	public	DATA	32	00	09
LOAD	86568000	8656B000	R	W	.	.	L	mempage	06	public	DATA	32	00	09
LOAD	8656B000	8656C000	R	W	.	.	L	dword	07	public	DATA	32	00	09
LOAD	8656C000	86570000	R	W	.	.	L	byte	08	public	DATA	32	00	09
LOAD	86574000	86578000	R	W	X	.	L	mempage	09	public	CODE	32	00	09

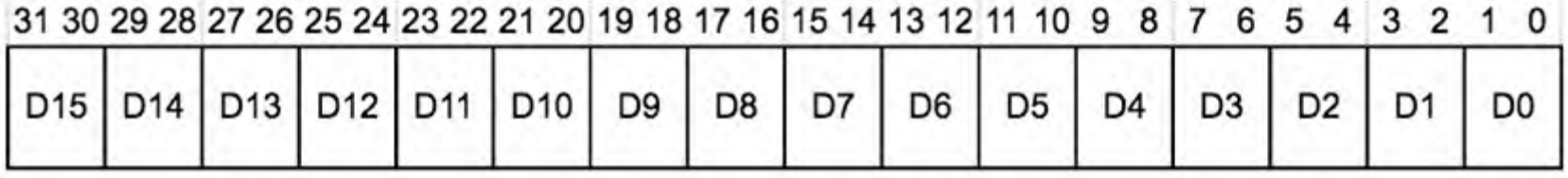
New segment

- Patched `get_version` again to branch to the new segment
- Phone freezes for a while and reboots!
- Suspected the reason is some memory protection after triple-checking the permissions of the new segment
- Solution: **disable** memory protection!

DACR register

- Domain Access Control Register
- **All regions of memory** have an associated **domain**. A domain is the primary access control mechanism for a region of memory.
- Holds the access permissions for a maximum of 16 domains.
- Protection of each domain encoded inside 2-bit fields

Figure 4.35. DACR bit assignments



DACR register

Table 4.57. DACR bit assignments

Bits	Name	Description
-	D<n>[a]	The fields D15-D0 in the register define the access permissions for each one of the 16 domains. b00 = No access. Any access generates a domain fault. b01 = Client. Accesses are checked against the access permission bits in the TLB entry. b10 = Reserved. Any access generates a domain fault. b11 = Manager. Accesses are not checked against the access permission bits in the TLB entry, so a permission fault cannot be generated. Attempting to execute code in a page that has the TLB *eXecute Never* (XN) attribute set does not generate an abort.

- 11 → Access **not checked**!

Patching

- After disabling DACR, executing code in the new segment works!!!
- Only one problem!
  - The phone **freezes** if you try to shutdown the phone!
- Somehow the disabled DACR protection interferes with the shutdown process.
- Solution?
  - Disable DACR before jumping to the new segment
  - Enable DACR again after return!

Patch - DACR disable/enable

```
sub_86535200                                    ; CODE XREF: sub_865351D0+4↑p

var_28          = -0x28

                STMFD           SP!, {R1-R8,LR}
                STR             R0, [SP,#0x24+var_28]!
                MOV             R0, #0xFFFFFFFF
                MCR             p15, 0, R0,c3,c0, 0 ; DISABLE DACR
                LDR             R0, [SP+0x28+var_28],#4
                STR             LR, [SP,#0x24+var_28]!
                BLX             loc_86574000 ; BRANCH TO NEW SEGMENT
                LDR             LR, [SP+0x28+var_28],#4
                STR             R0, [SP,#0x24+var_28]!
                MOV             R0, #0x55555555 ; ENABLE DACR
                MCR             p15, 0, R0,c3,c0, 0
                LDR             R0, [SP+0x28+var_28],#4
                LDMFD           SP!, {R1-R8,LR}
                BX              LR
; End of function sub_86535200
```

Generating ARM code

- At the start of the project I had only 1 option: to use `GNU as` assembler. It was a nightmare!
- Fortunately some months later the `Keystone Engine` assembler framework was released and I could use Python to generate the arm code! Easy!
- http://www.keystone-engine.org/

Executing your code

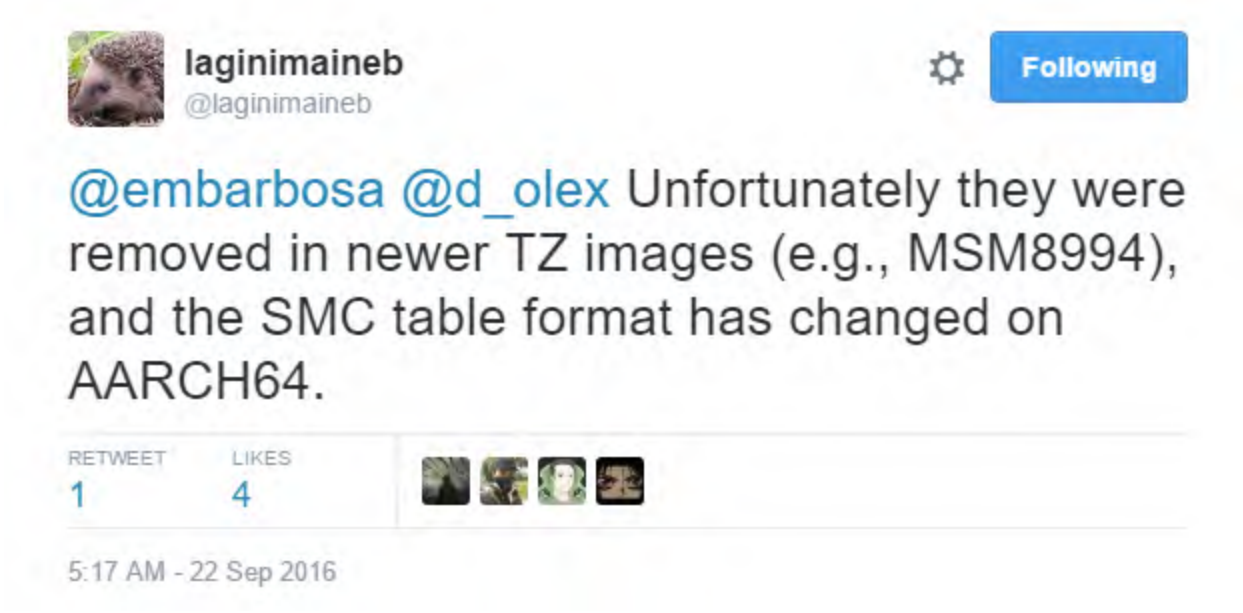
- Just create a device driver
- Linux provides the `scm` and `scm_call` functions!
- Tip:
  - Sometimes building the open source Linux kernel of an Android device is an impossible mission
    - Again, is Android really open source? :)
  - You can extract the symbols of the binary kernel using this little wonderful tool: https://github.com/glandium/extract-symvers and build your device driver
  - "Building a Linux kernel module without the exact kernel headers": https://glandium.org/blog/?p=2664

Undocumented

- That's all you need to create TZ code for your device
- We need more reverse engineering of TZ
- There are some functions that are really difficult to understand/reverse
- References to devices and memory mapped I/O regions where I couldn't find any documentation

Reversing TZ - Bad news ಠ_ಠ

- Things are changing...



- They removed the `tzbsp` strings and modified the syscall table format!
  - (ノ °□°) ノ ⌒ ┻━┻

Not all is lost yet

- Latest version of Xiaomi TZ (this week)
- There are still a few `tzbsp` strings available

LOAD:0000000...	0000000E	C	tz_mpu_rg_cfg
LOAD:0000000...	00000023	C	tzbps_es_set_ice_key: invalid buf\n
LOAD:0000000...	00000034	C	tzbps_es_set_ice_key: invalid request parameter(s)\n
LOAD:0000000...	0000002E	C	tzbsp application rpmb version rollback label
LOAD:0000000...	00000027	C	tzbsp version counter cipher key label
LOAD:0000000...	00000025	C	tzbsp version counter hmac key label
LOAD:0000000...	00000016	C	tzbsp_hmac256 failed!
LOAD:0000000...	0000001D	C	tzbsp_psci_cpu_boot_notifier
LOAD:0000000...	0000001C	C	tzbsp_register_isr() failed
LOAD:0000000...	00000005	C	t&#65533;W&#65533;

Same process applies

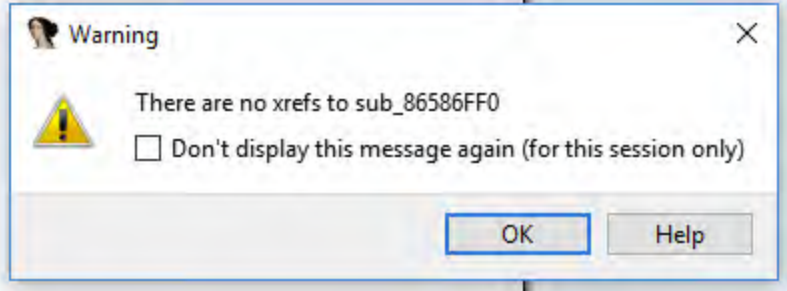
```
00865E8B2C aTzbps_es_set_0 DCB "tzbps_es_set_ice_key: invalid buf",0xA,0
00865E8B2C                                         ; DATA XREF: sub_86586FF0+114↑o
00865E8B2C                                         ; sub_86586FF0+11C↑o
```

```
loc_865870B4                ; "(%x)"
ADRP            X1, #aX@PAGE
ADRP            X2, #aTzbps_es_set_i@PAGE ; "tzbps_es_set_ice_key: invalid request p"...
ADD             X1, X1, #aX@PAGEOFF ; "(%x)"
ADD             X2, X2, #aTzbps_es_set_i@PAGEOFF ; "tzbps_es_set_ice_key: invalid request p"...
MOV             W0, #3
BL              sub_8654894C
MOV             W0, #0xFFFFFFF0
```

```
; Attributes: bp-based frame

sub_86586FF0

var_88= -0x88
var_68= -0x68
var_48= -0x48
var_40= -0x40
var_30= -0x30
var_20= -0x20
var_10= -0x10
var_s0=  0
```



Search again ...

```
LOAD:00000000865EED65                   DCB    0
LOAD:00000000865EED66                   DCB    0
LOAD:00000000865EED67                   DCB    0
LOAD:00000000865EED68                   DCB 0xF0
LOAD:00000000865EED69                   DCB 0x6F ; o
LOAD:00000000865EED6A                   DCB 0x58 ; X
LOAD:00000000865EED6B                   DCB 0x86 ; å
LOAD:00000000865EED6C                   DCB    0
LOAD:00000000865EED6D                   DCB    0
LOAD:00000000865EED6E                   DCB    0
```

Table found!

```
LOAD:00000000865EED66                 DCB     0
LOAD:00000000865EED67                 DCB     0
LOAD:00000000865EED68                 DCQ sub_86586FF0
LOAD:00000000865EED70                 DCB     0
LOAD:00000000865EED71                 DCB     0
LOAD:00000000865EED72                 DCB     0
LOAD:00000000865EED73                 DCB     0
LOAD:00000000865EED74                 DCB     3
LOAD:00000000865EED75                 DCB  0x10
LOAD:00000000865EED76                 DCB     0
LOAD:00000000865EED77                 DCB     2
LOAD:00000000865EED78                 DCB     1
LOAD:00000000865EED79                 DCB     0
LOAD:00000000865EED7A                 DCB     0
LOAD:00000000865EED7B                 DCB     0
LOAD:00000000865EED7C                 DCB     0
LOAD:00000000865EED7D                 DCB     0
LOAD:00000000865EED7E                 DCB     0
LOAD:00000000865EED7F                 DCB     0
LOAD:00000000865EED80                 DCQ sub_86587120
LOAD:00000000865EED88                 DCB     0
```

- There are no more pointer to strings
- Detection of table can be easily automated with IDAPython

Next (1/2)

- I have now full access to TZ and a framework that allow me to patch the TZ image to execute any experiment
- No need for NDA, dev boards, emulation. Freedom to learn!
- No TrustZone debugger! We are blind now.
  - Idea: implement a debugging interface by patching TZ

Next (2/2)

- We need to find other devices that allow us to write on the TZ partition or find more methods to access TZ
  - Don't blame me if you brick your phone!
  - I'm trying to unlock other devices. Will post any new information on my Twitter account.
- Have fun with TZ!
  - but no rootkits, please!
  - Rootkits are lame :)

Thank you!

Greetz!

- Sheng Di @sheng0x64
- TrustZone Jedi Hacker Master Gal Beniamini @laginimaineb
- Jonathan Levin @Morpheus_____

References 1/2

Best references about TrustZone hacking/internals:

1. http://bits-please.blogspot.sg/
2. http://technologeeks.com/files/TZ.pdf
3. http://technologeeks.com/files/TrustZone.pdf

Reference 2/2

1. http://blog.csdn.net/u011279649/article/details/45250979
2. http://huaqianlee.github.io/2015/08/23/Android/高通Android设备启动流程分析-从power-on上电到Home-Lanucher启动/
3. http://forum.xda-developers.com/showthread.php?t=1769411&page=24
4. http://www8.hp.com/h20195/v2/getpdf.aspx/4AA5-6428ENW.pdf?ver=1.0
5. https://www.arm.com/files/pdf/Tech_seminar_TrustZone_v7_PUBLIC.pdf
6. https://android.googlesource.com/platform/prebuilts/gcc/linux-x86/arm/arm-eabi-4.6/
7. https://www.isc2cares.org/uploadedFiles/wwwisc2caresorg/Content/Android-Security-Report-FrostSullivan.pdf