BREAKING SAMSUNG'S ARM TRUSTZONE

Quarkslab

Black Hat USA 2019

OUR TEAM

Maxime Peterlin - *R&D Engineer at Quarkslab*

@pandasec_

Alexandre Adamski - *R&D Engineer at Quarkslab*

@NeatMonster_

Joffrey Guilbon - *R&D Engineer at Quarkslab*

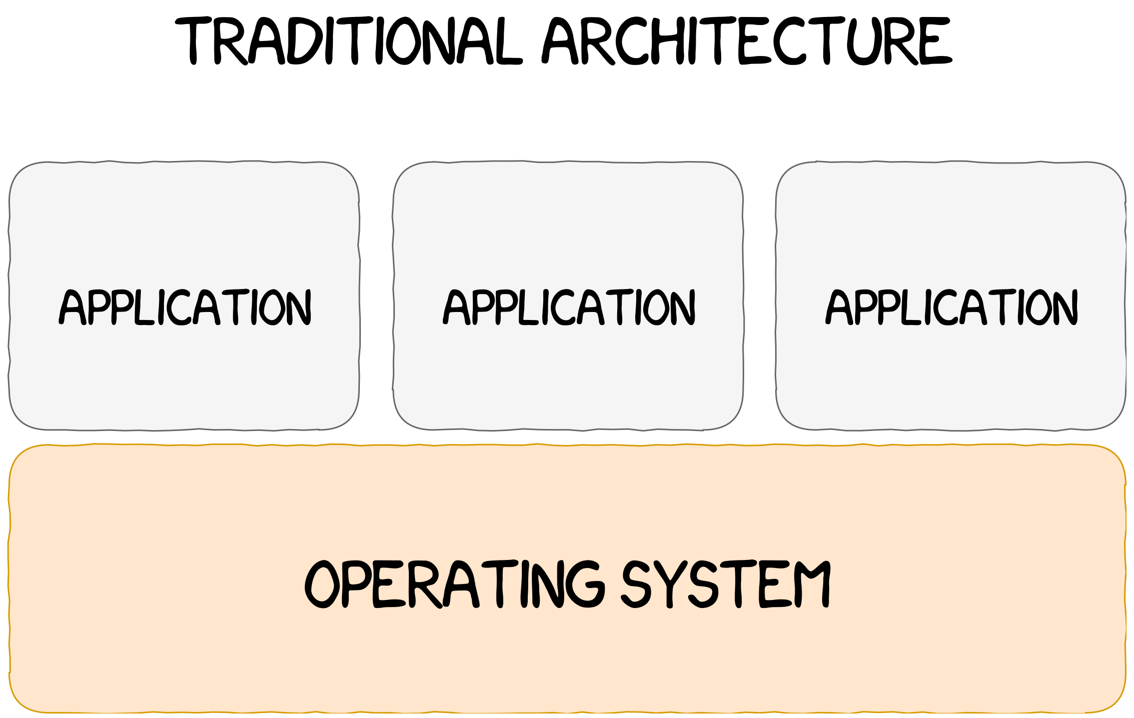
@patateQBool

PRESENTATION OUTLINE

- Current state of embedded security
- Introduction to the ARM TrustZone technology
- Samsung's TrustZone Overview
- Trusted Components
- Vulnerability Research Tools
- Vulnerability Analysis
- Exploitation
- Post-Exploitation Demonstrations

CURRENT STATE OF EMBEDDED SECURITY

A LONG TIME AGO...



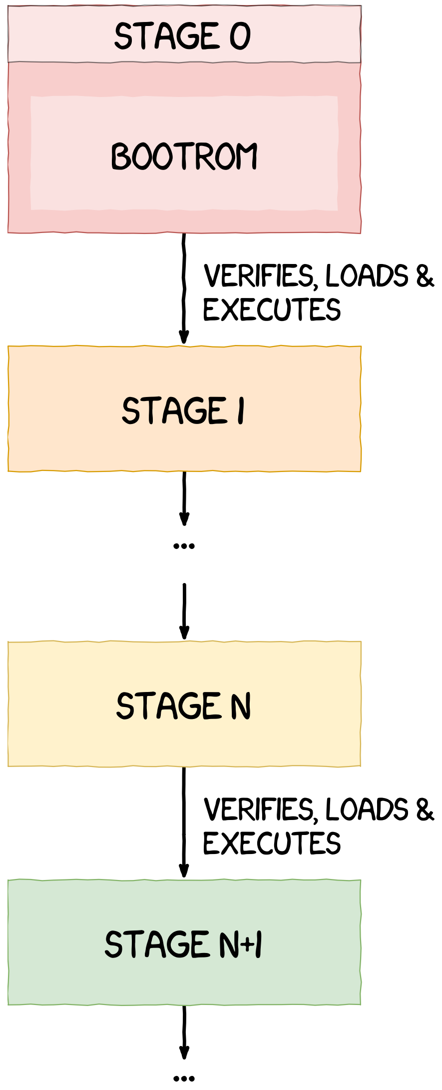
- Kernel unbreakable...?

HOW DO WE PROTECT OURSELVES...

- ... if the kernel is corrupted during the boot process?
- ... if the kernel is corrupted when the system is already running?

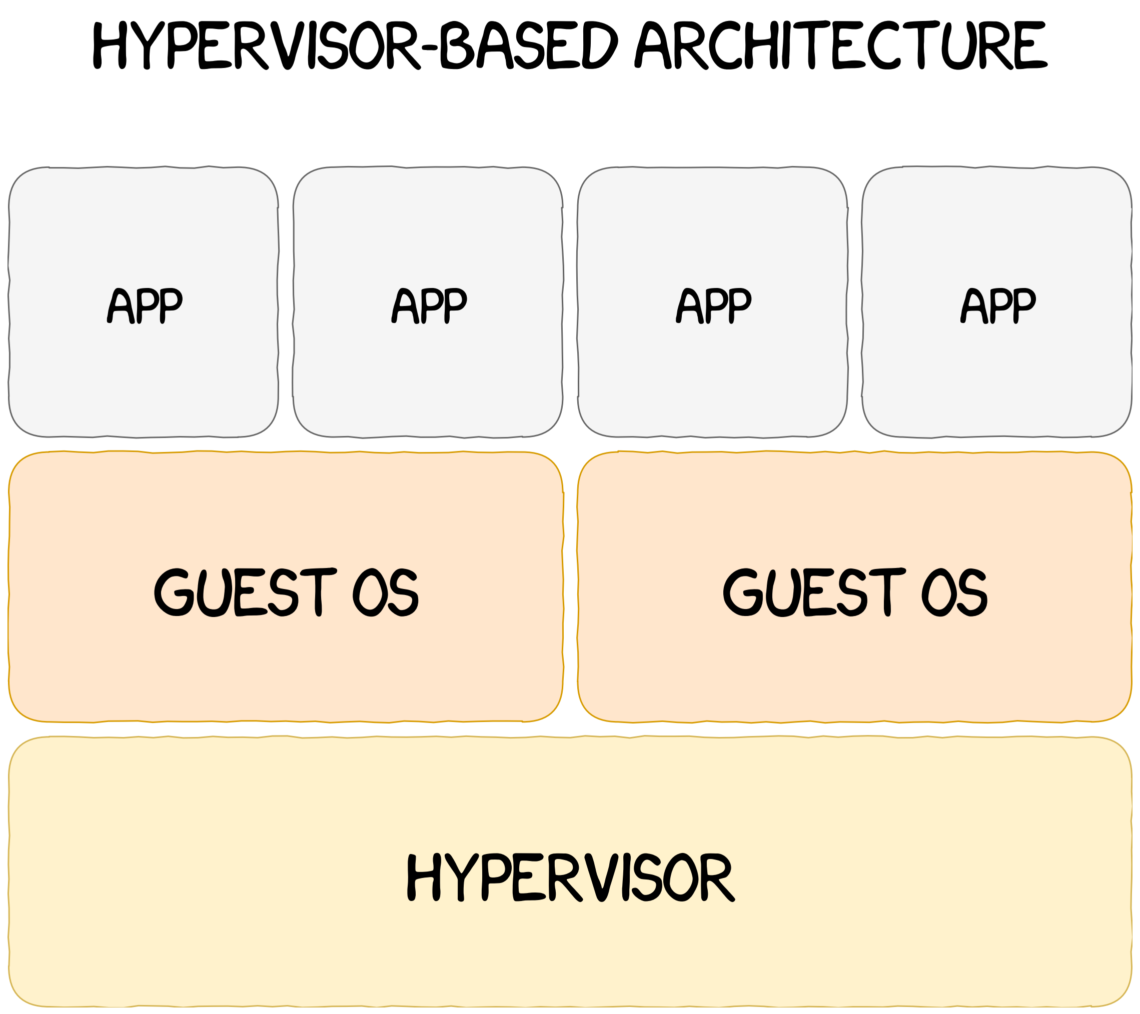
PROTECTION DURING THE BOOT PROCESS

Secure Boot



- Prevent the execution of untrusted or unauthorized code on end users devices

RUNTIME PROTECTION USING AN HYPERVISOR

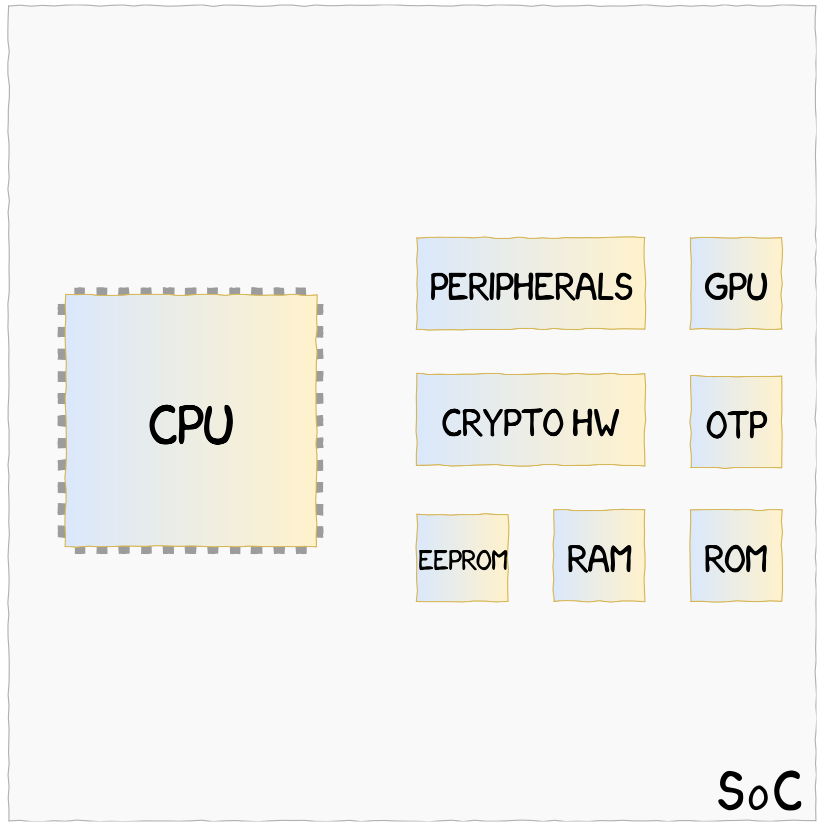


- Hypervisor based guest kernel protection
- **Problem:** VM escapes and hypervisor compromissions

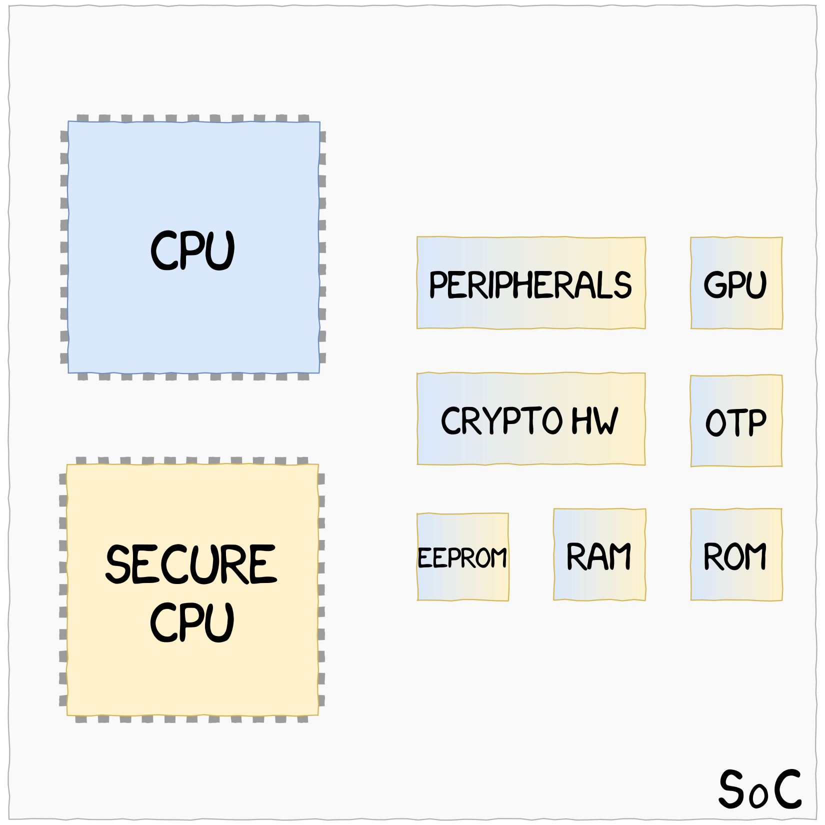
TRUSTED EXECUTION ENVIRONMENTS

Taken from *Le TEE, nouvelle ligne de défense dans les mobiles*, SSTIC 2013

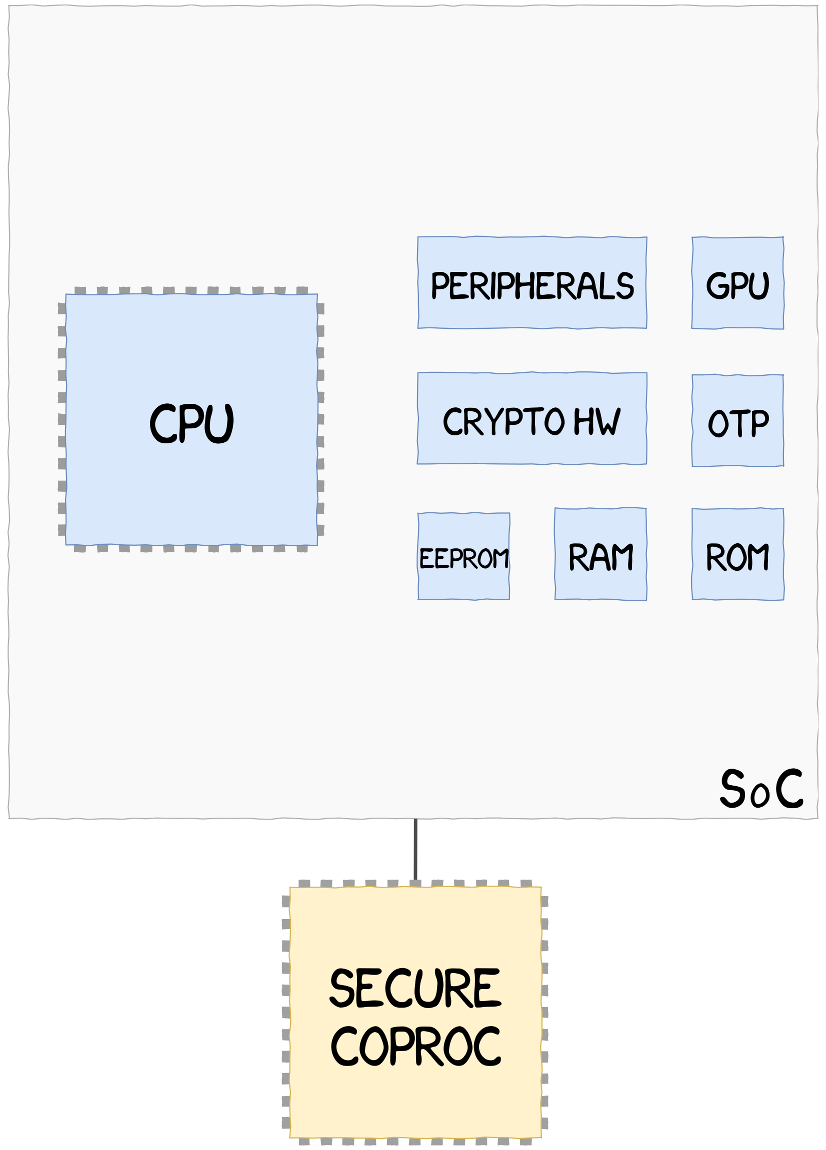
Virtual Processor
(e.g. ARM TrustZone)



On-SoC Processor
(e.g. Apple SEP)



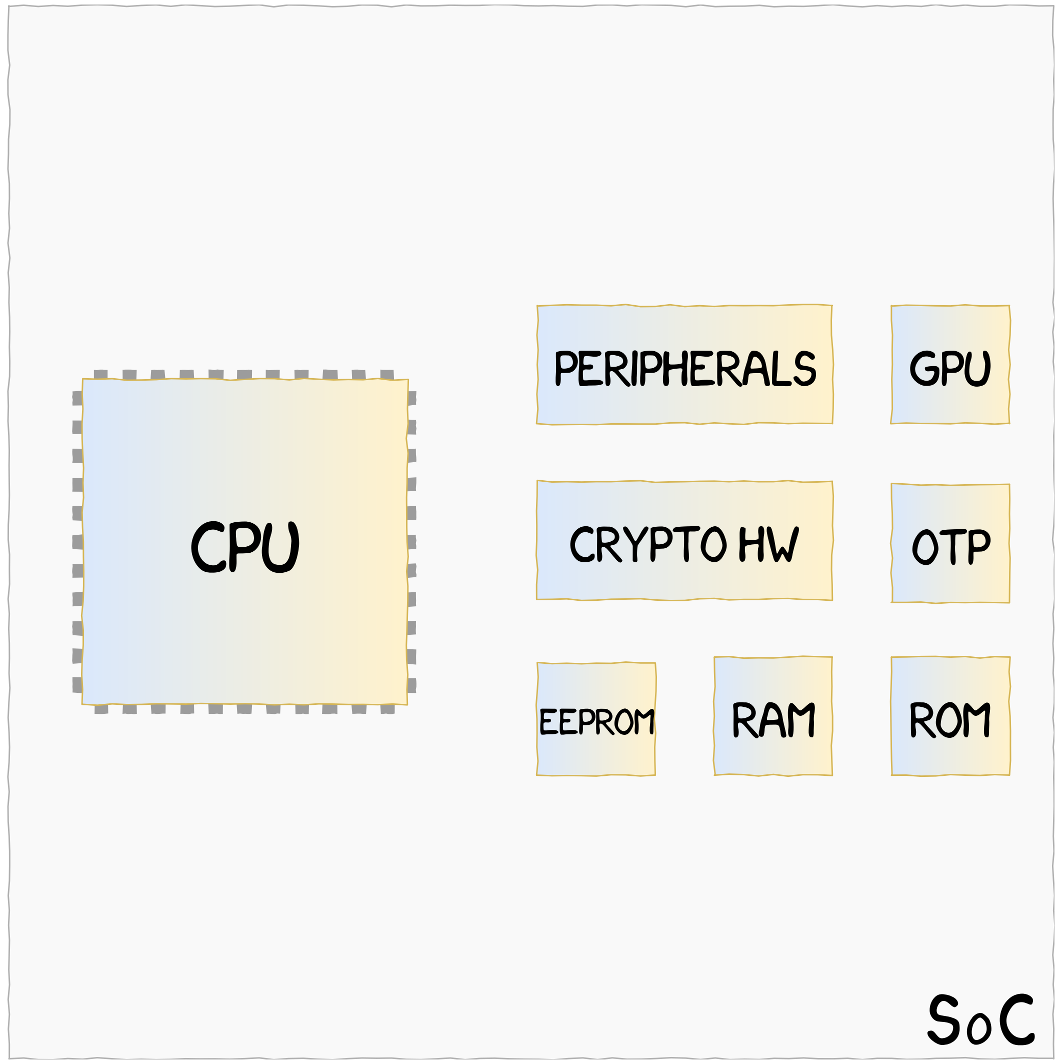
External Coprocessor
(e.g. Google Titan M)



ARM TRUSTZONE TECHNOLOGY

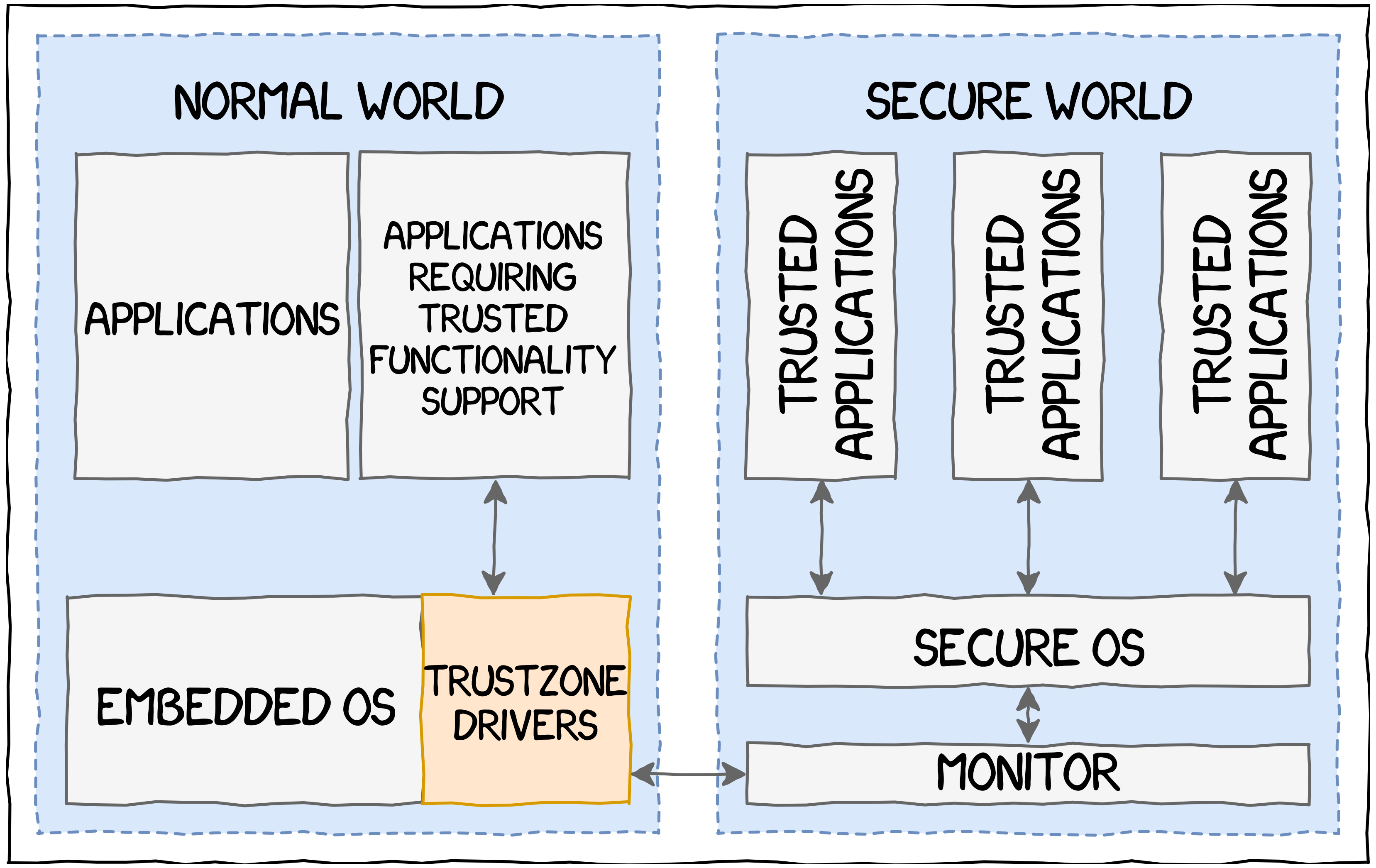
OVERVIEW

ARM TrustZone is a system-wide hardware isolation mechanism



- **Hardware architecture**
  - Partitioning of all the SoC's hardware and software resources
  - TZPC, TZASC, TZMA, etc.
- **Software architecture**
  - Software implementation used in secure state
  - Communications between secure and non-secure components

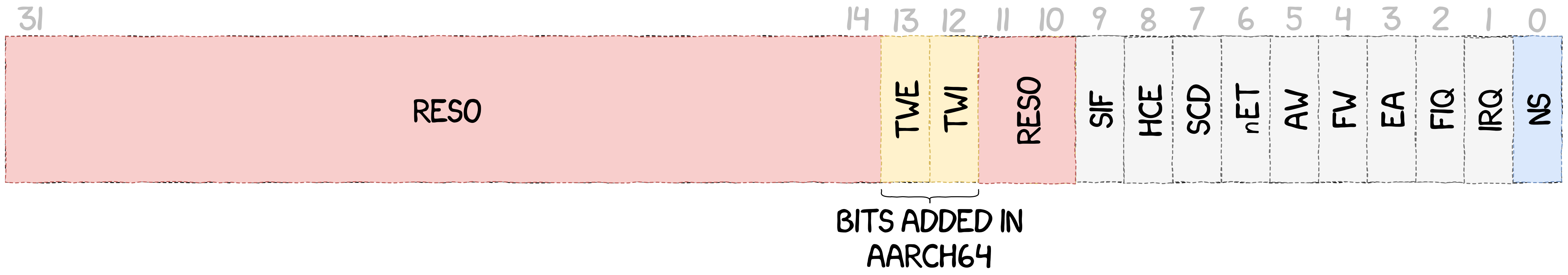
SECURE AND NON-SECURE WORLDS



- **Secure World**
  - Runs trusted code
  - Performs sensitive operations
- **Normal World**
  - Considered as compromised by design
  - Performs non-sensitive operations

SECURE CONFIGURATION REGISTER

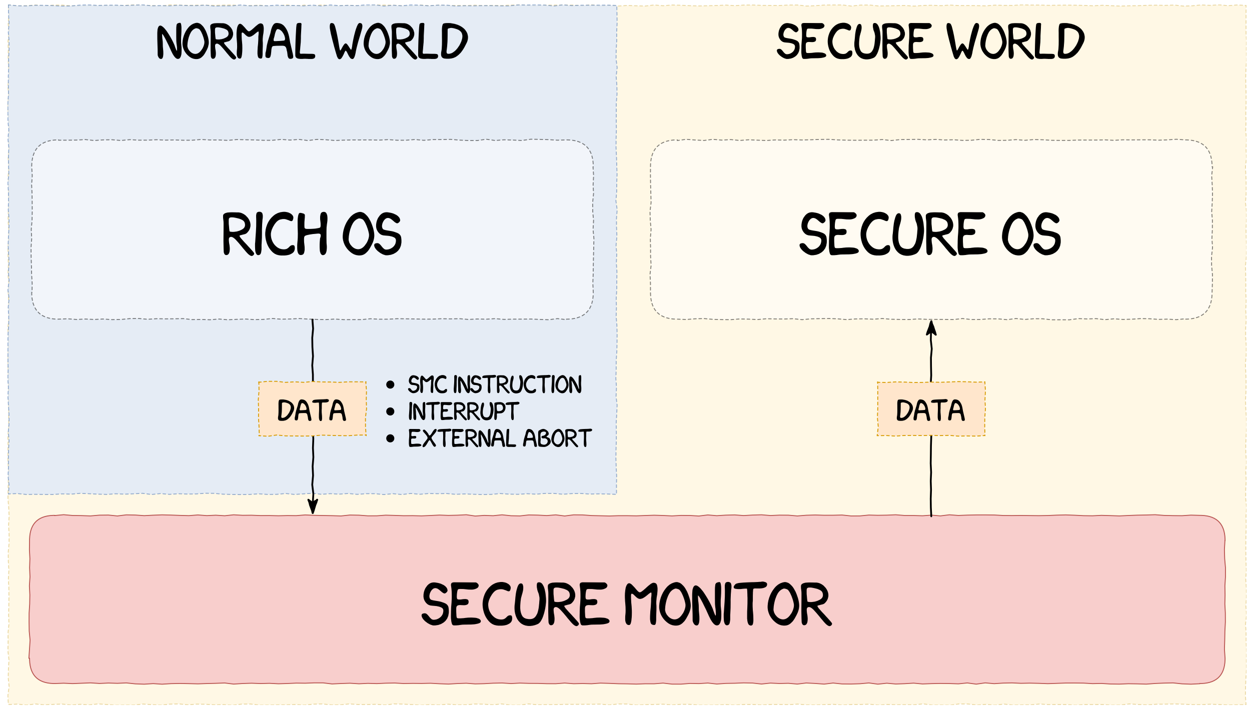
The **Secure** (or **Non-Secure**) state of the CPU is determined by the least significant bit of the **Secure Configuration Register** (SCR)

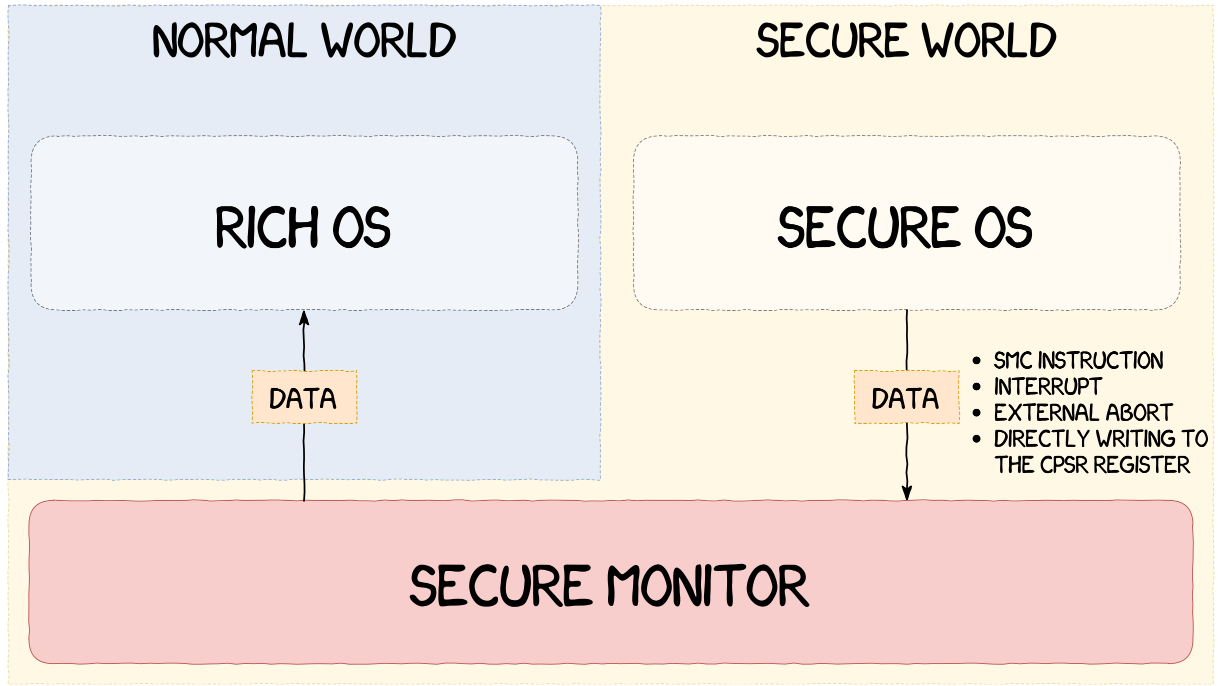


NS BIT [0]:

- 0 SECURE STATE
- 1 NON-SECURE STATE

COMMUNICATING BETWEEN WORLDS

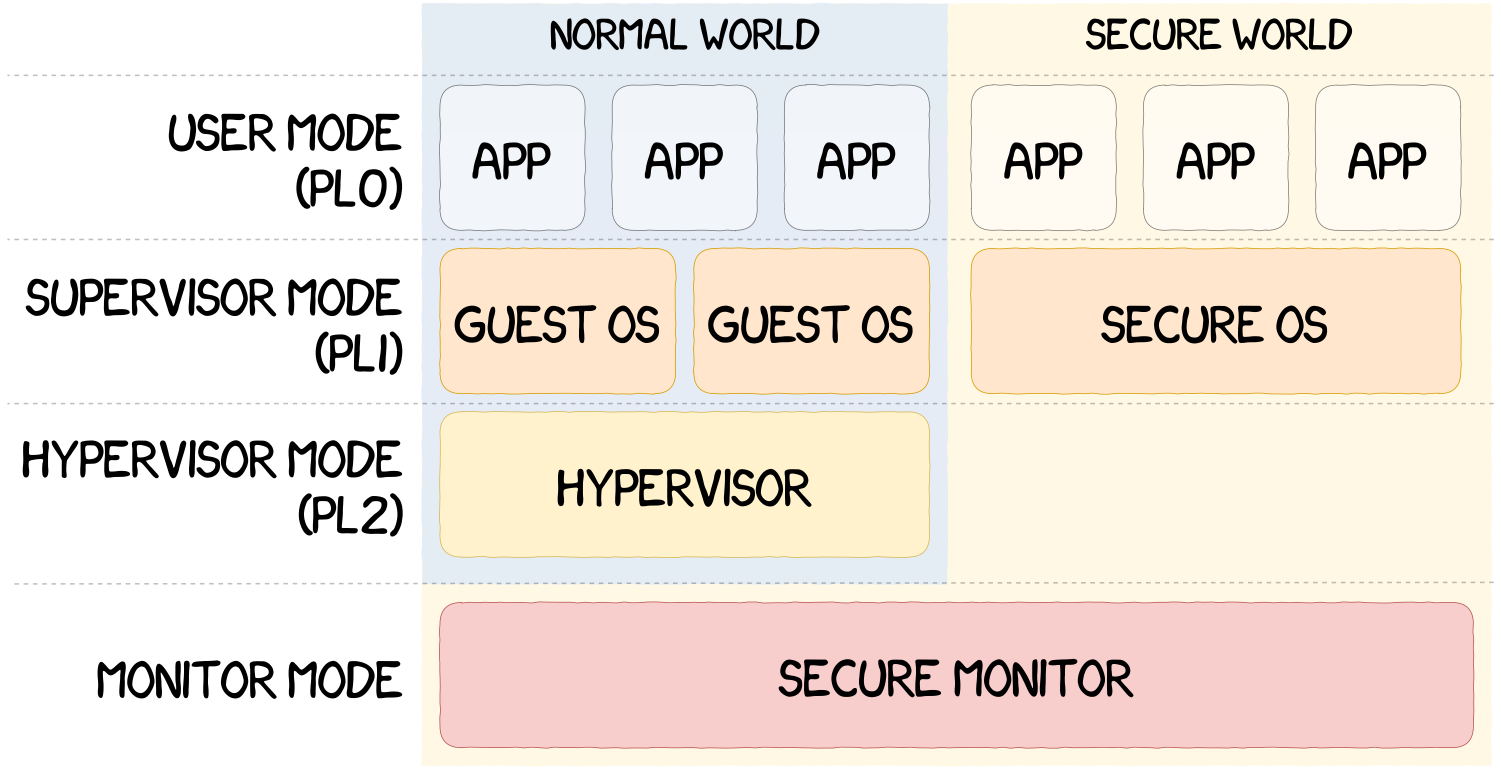




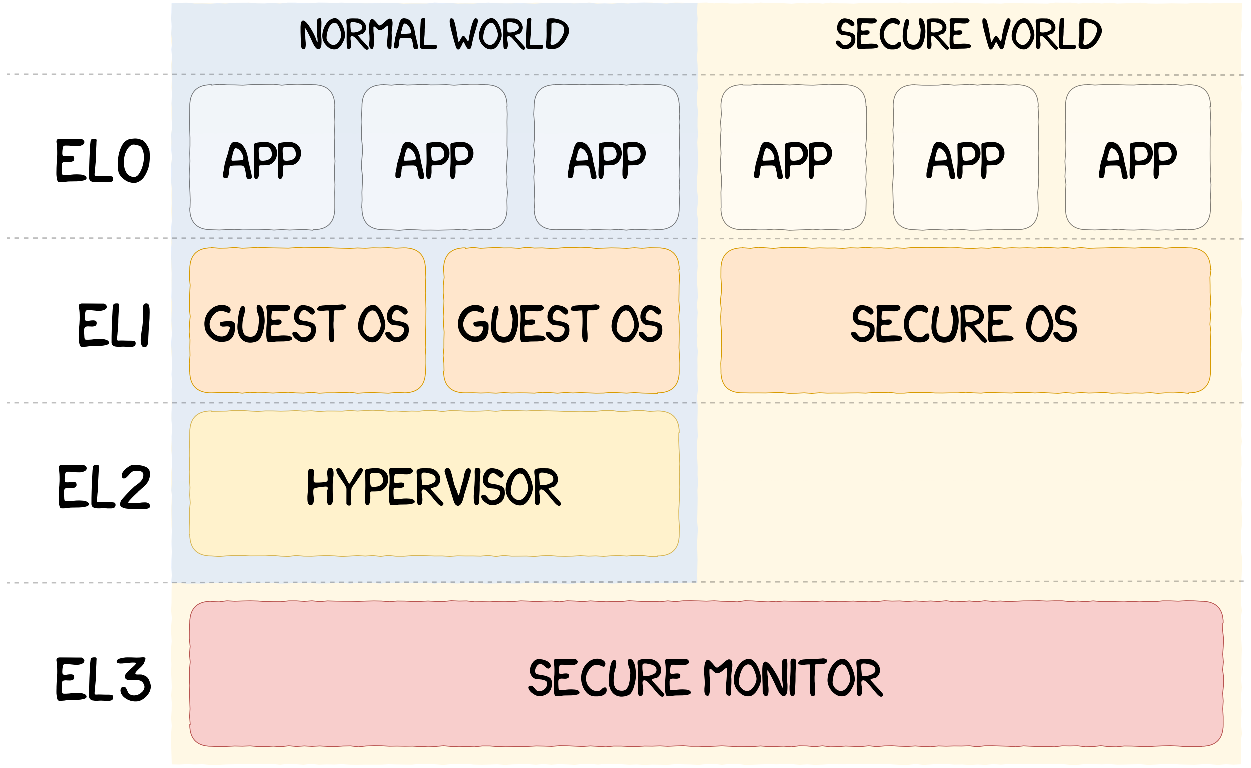
- Switches between worlds are performed by the **Secure Monitor**
  - Runs at the highest privilege level (**EL3** in ARMv8/**Monitor Mode** in ARMv7)
- Data exchanged through
  - Exceptions
  - Interruptions
  - Writing to the PSTATE/CPSR registers (privileged operation)

PRIVILEGES SEPARATION

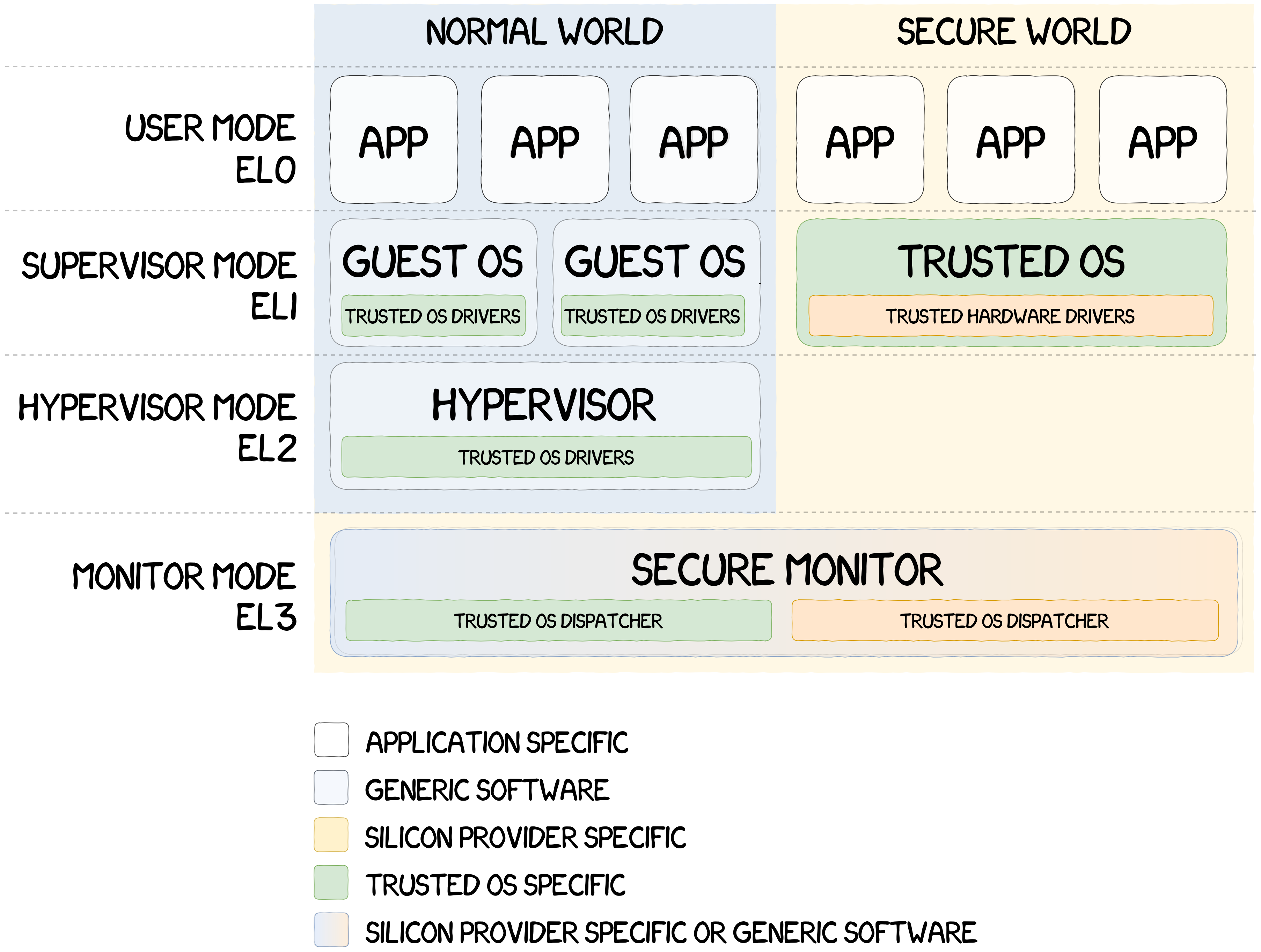
ARMv7 Privilege Levels



ARMv8 Exception Levels



THE TRUSTED COMPONENTS FRAGMENTATION ISSUE

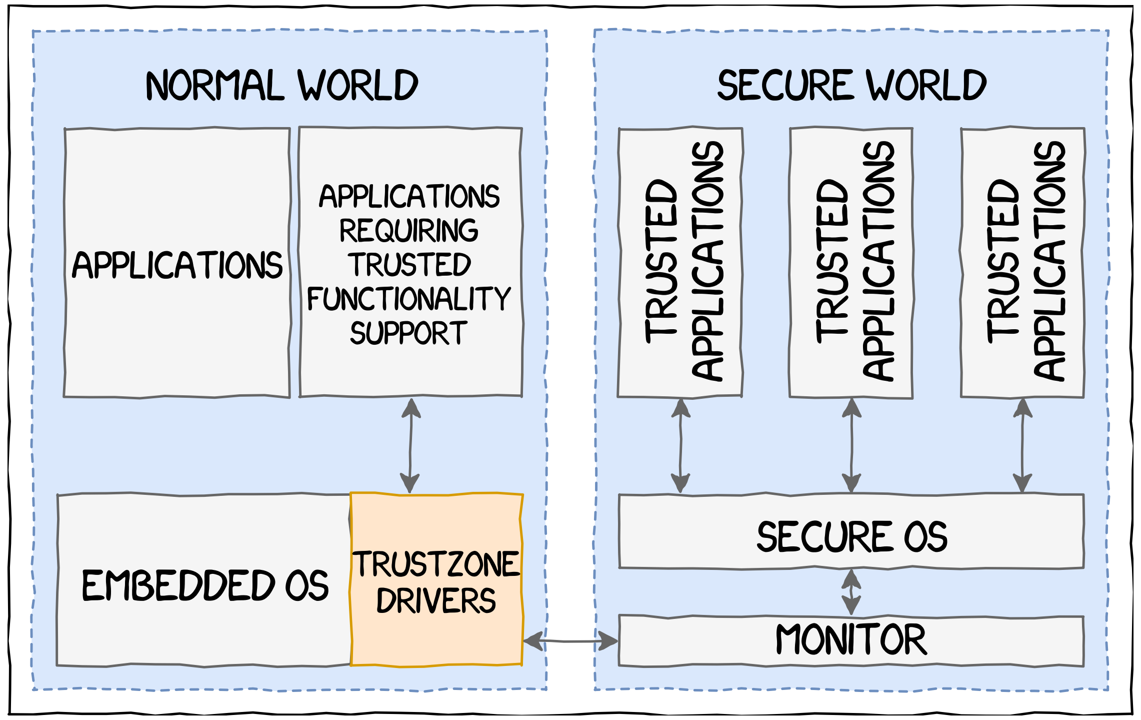


- **Privilege escalation by design**
  - No hardware isolation between S-EL1 and EL3
  - Access to all the physical memory
  - Will be fixed in ARMv8.4 with *Secure Partitions*
- **Fragmentation**
  - System developed by different vendors
  - Cooperation and mutual trust required

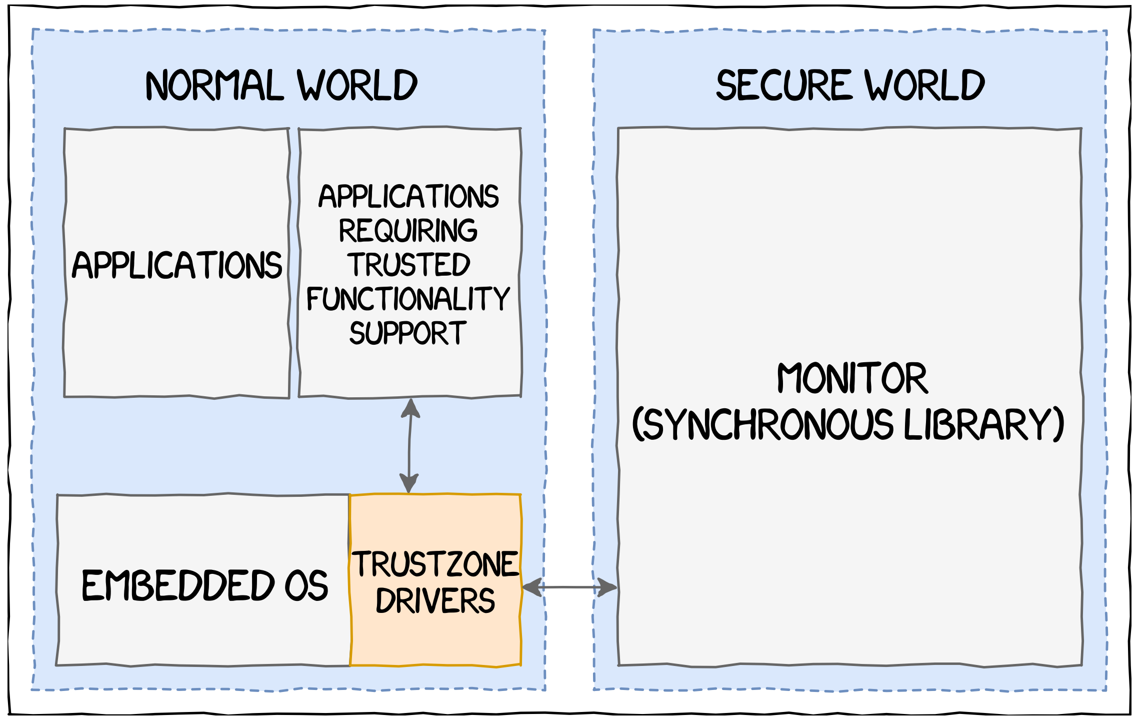
TRUSTZONE SOFTWARE ARCHITECTURE

Several implementations of the software stack running in TrustZone are possible

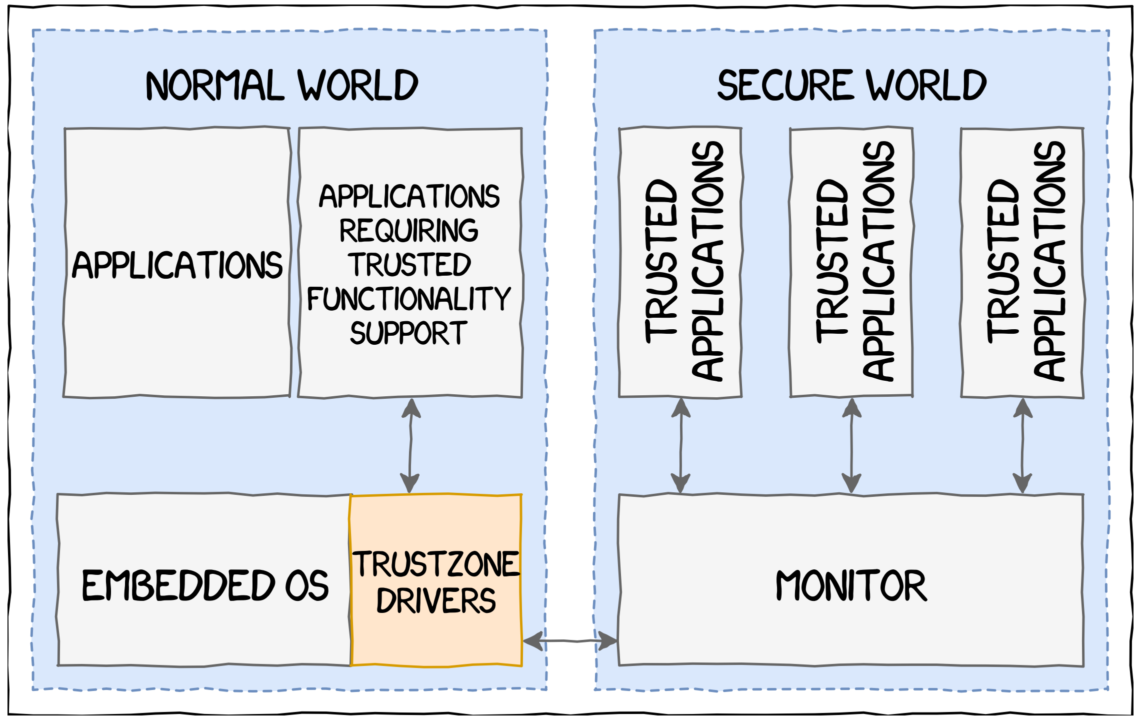
Operating System



Synchronous Library



Intermediate Option



TRUSTZONE'S USE CASES

- Accessing hardware-backed features:
  - Cryptographic engine
  - Credentials storage (Hardware-backed Keystore)
  - True random number generator
  - ...
- Digital Rights Management (by leveraging the cryptographic engine)
- Protecting and monitoring of the Normal World by the Secure World
  - **Example:** Samsung's Real-Time Kernel Protection (RKP) and Periodic Kernel Measurement (PKM)

SAMSUNG'S ARM TRUSTZONE

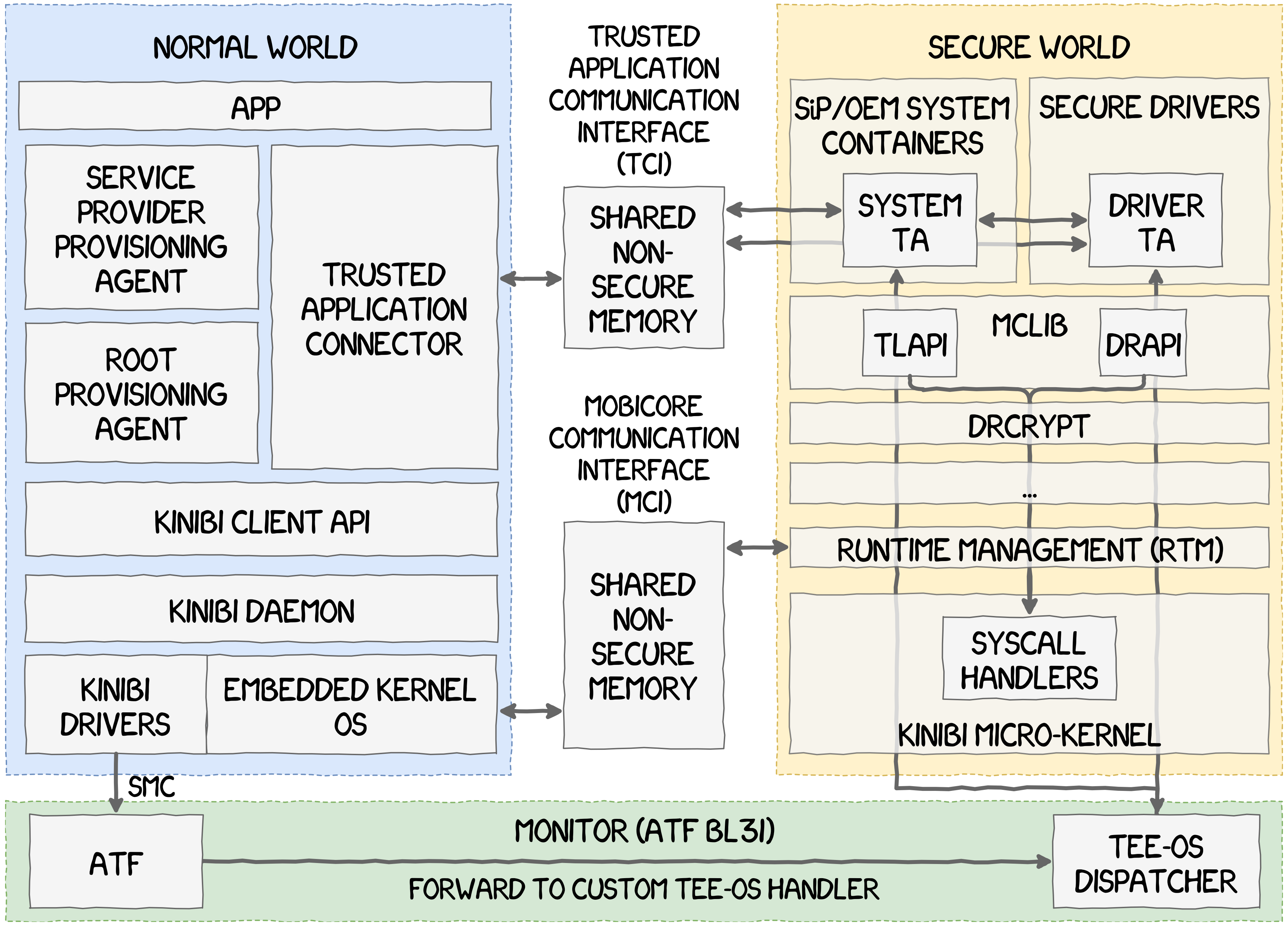
OVERVIEW

- **Samsung Devices**
  - Use both Samsung's Exynos and Qualcomm's Snapdragon SoCs
    - The same phone models can have different SoCs depending on the country
- **Samsung's TrustZone**
  - Found only on Exynos SoCs
  - First used in the Samsung Galaxy S3
  - Trusted OS used:
    - **Kinibi** developed by Trustonic (Galaxy S3 to Galaxy S9)
    - **TEEGRIS** developed by Samsung (Galaxy S10)
    - Both are used in other models too
    - This talk will focus on **Kinibi**

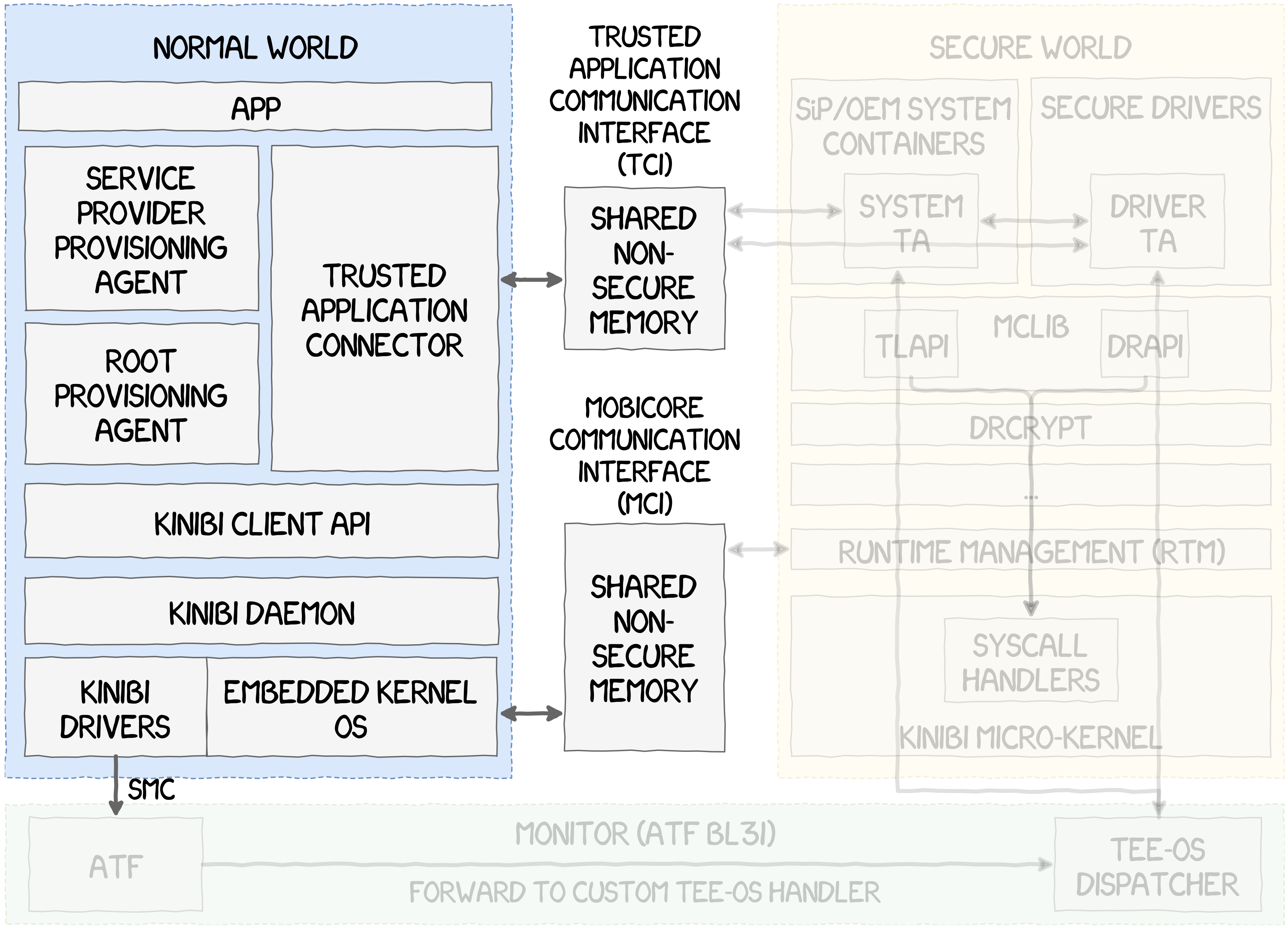
PREVIOUS WORKS

- Reverse Engineering Samsung S6 SBOOT (2-part article series) by **Fernand Lone Sang**
  - ARM Trusted Firmware usage on Samsung devices and extraction process from an OTA of the TEE-OS
- Unbox Your Phone (3-part article series) by **Daniel Komaromy**
  - Reverse-engineering of the Trusted OS and exploitation of vulnerabilities in trustlets
- Trust Issues: Exploiting TrustZone TEEs by **Gal Beniamini**
  - Security analysis of different Trusted Execution Environments

SAMSUNG'S TRUSTZONE ARCHITECTURE

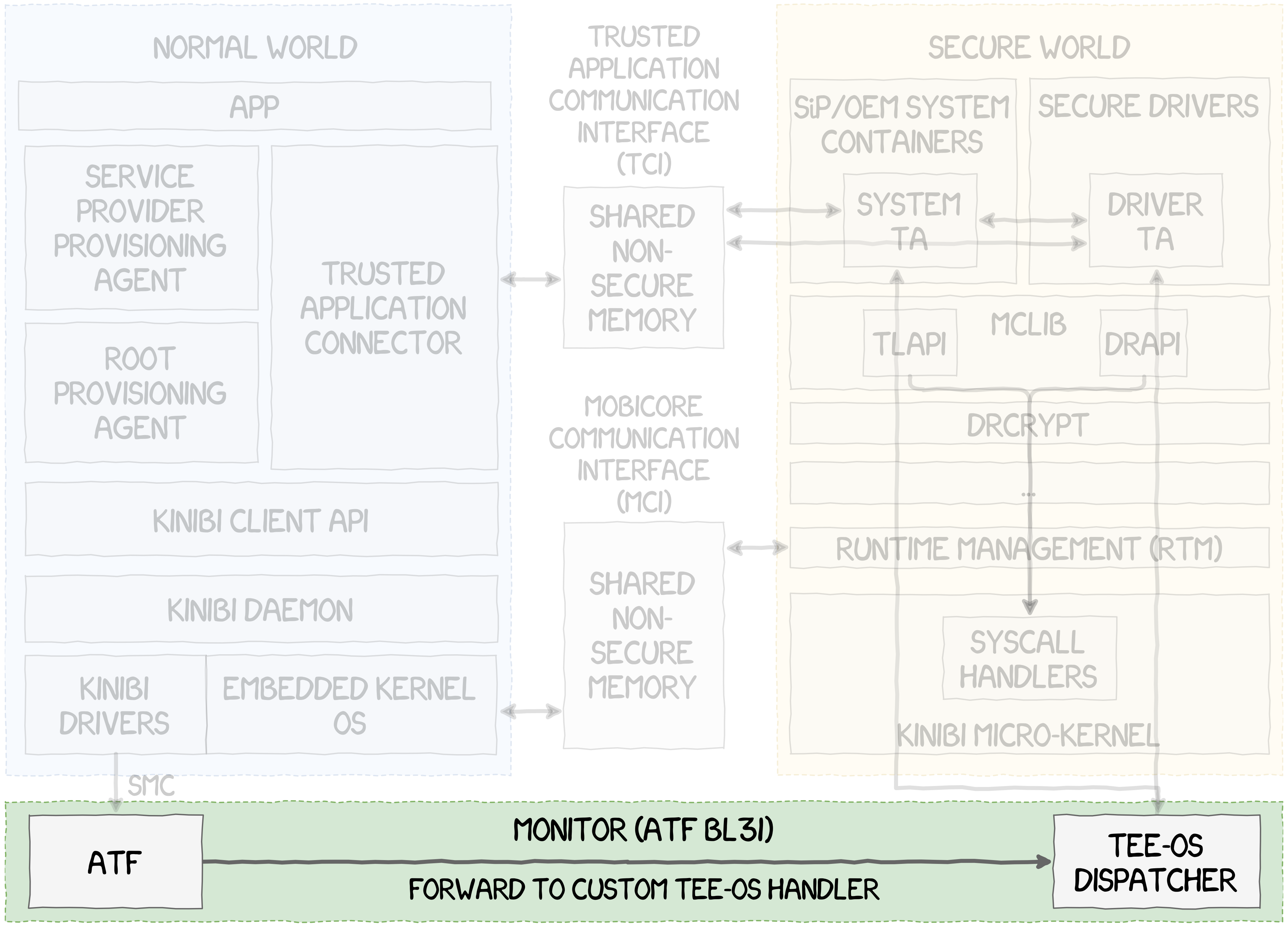


NORMAL WORLD COMPONENTS



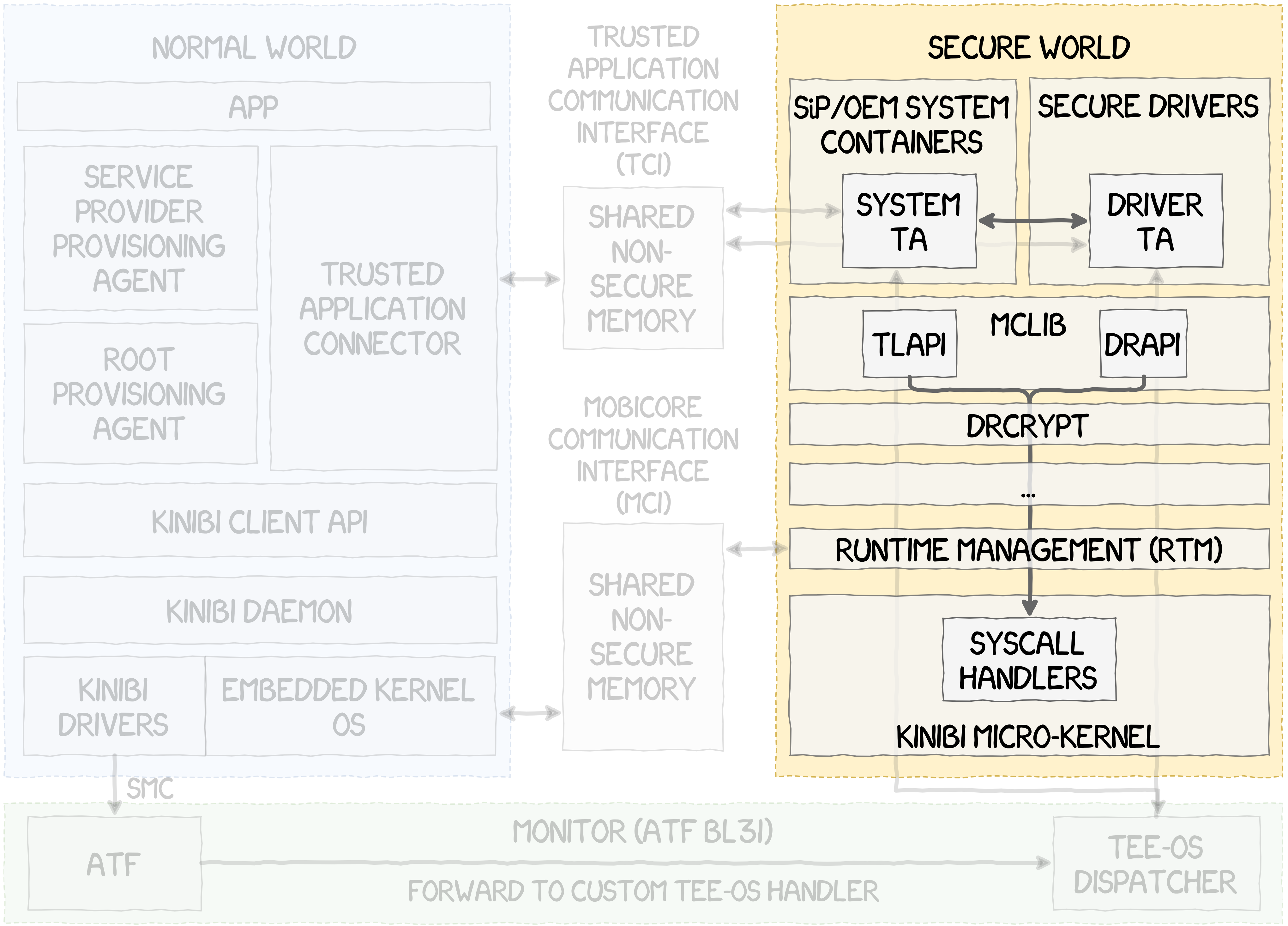
- Drivers, daemons, libraries and interfaces used for communicating with the Secure World
- Communications pass through SMCs and shared memory buffers

SECURE MONITOR



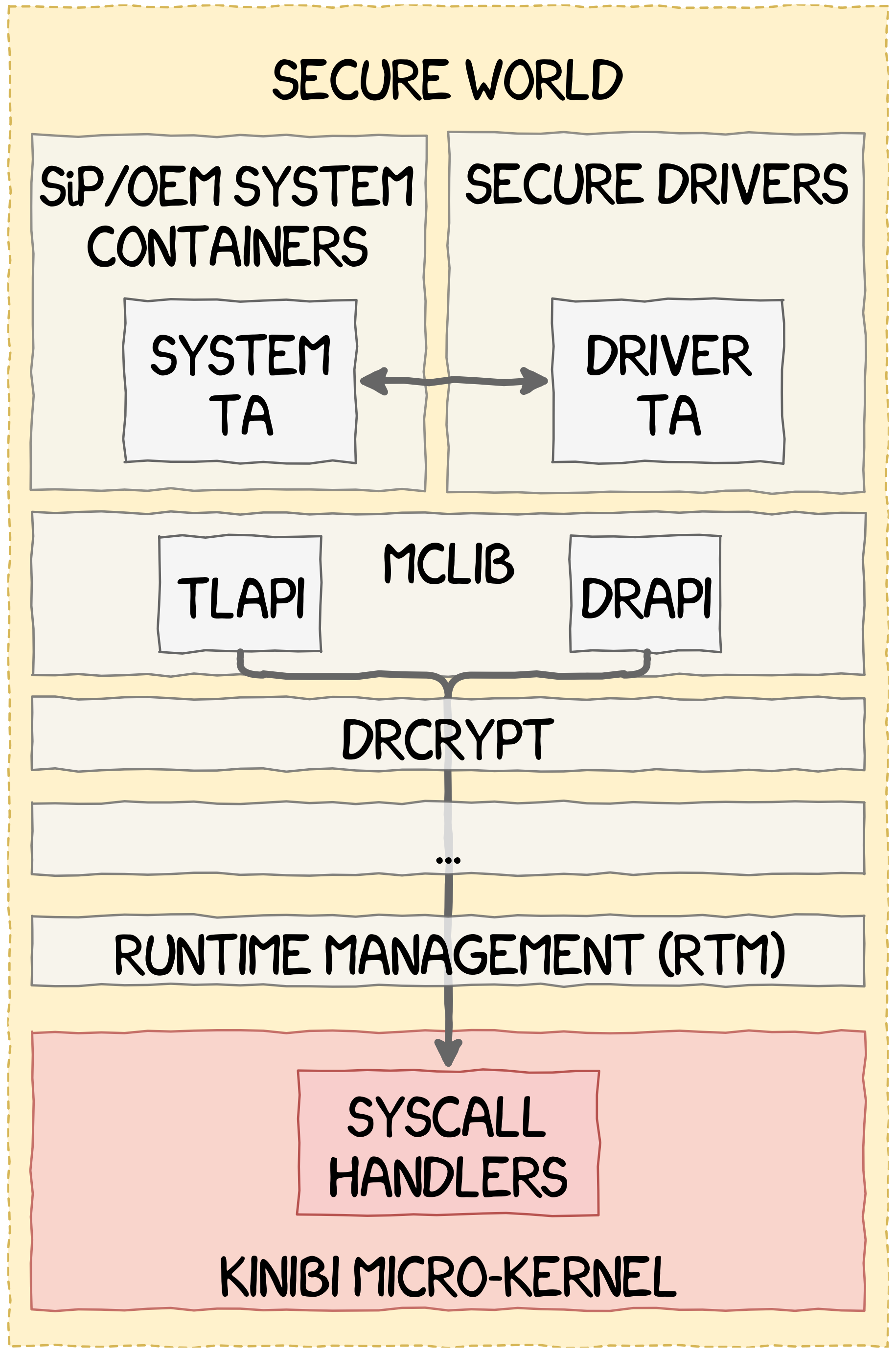
- **ARM Trusted Firmware**
  - Open-source reference implementation of Secure World software provided by ARM
  - Contains a modular secure monitor implementation
  - Custom SMC handlers, called runtime services, can be added to fit the vendors requirements
  - **Example:** runtime services are used by Samsung to forward SMCs handled by Kinibi

SECURE WORLD COMPONENTS



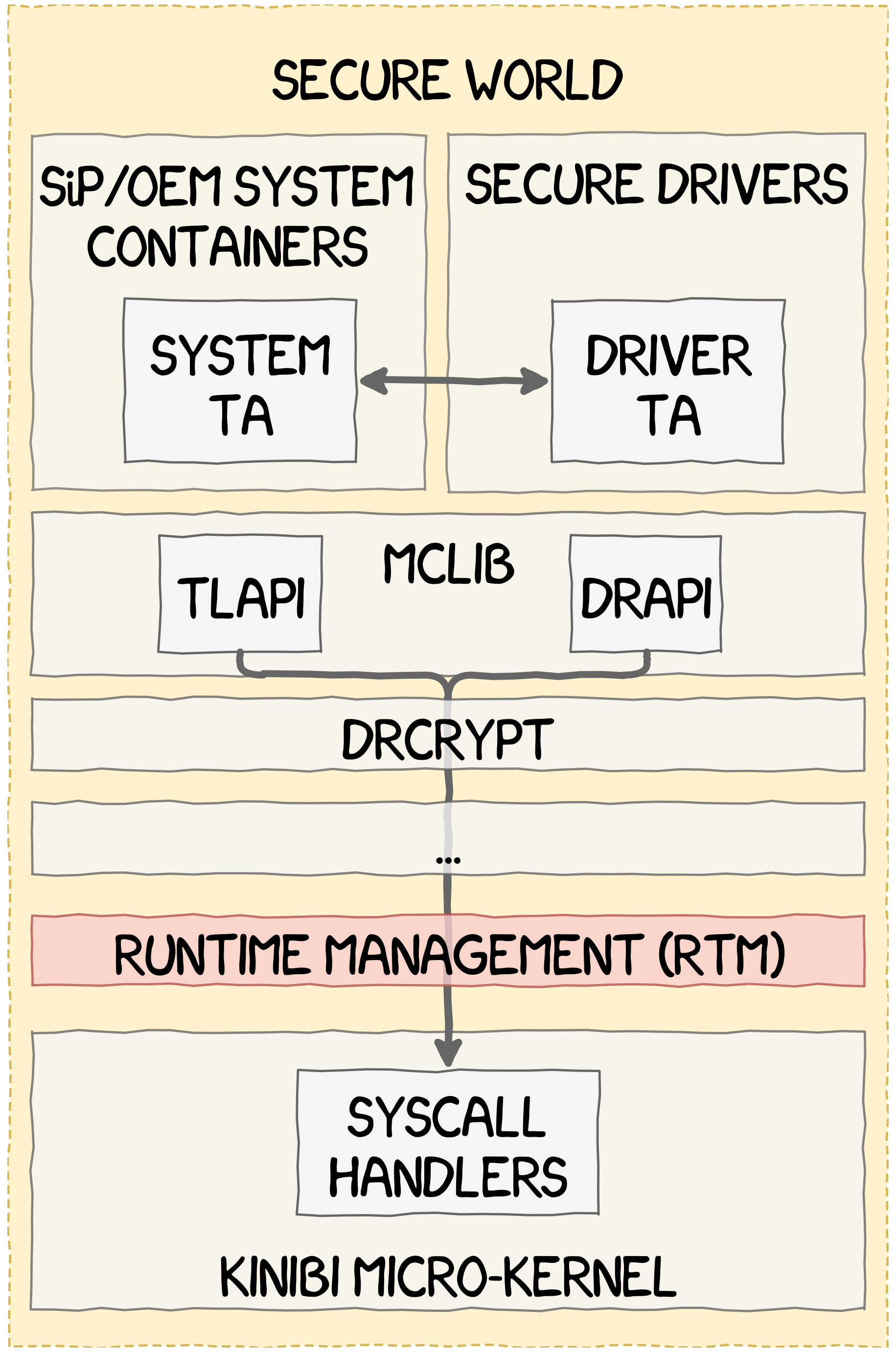
- Secure world based on a micro-kernel architecture

MTK: KINIBI'S MICRO KERNEL



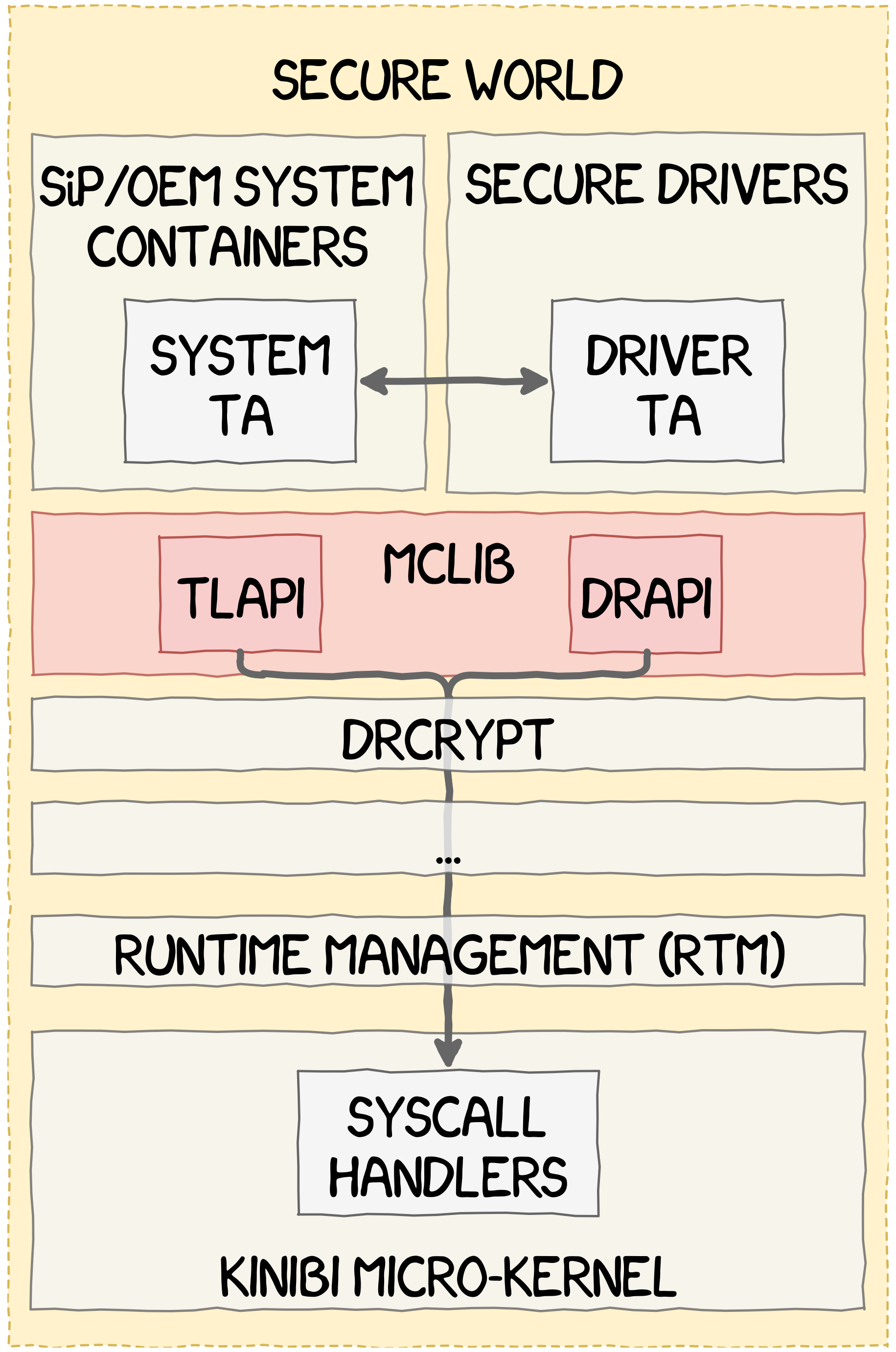
- Kinibi is a 32-bit OS developed by **Trustonic**
  - Used to be called *Mobicore* and *t-base*
- **MTK**: micro-kernel and only component running in S-EL1
- Provides syscalls (SVCs)
  - Memory mapping, process creation, SMCs, etc.
  - SVCs available depend on the privileges of the calling process
- Loads other components (embedded drivers, etc.) and especially **RTM**

RUN-TIME MANAGER



- Special Secure World trusted application equivalent to the init process on Linux
- **Main tasks**
  - starting and managing processes
  - notifying trustlets of incoming data from the NWd
- **Implements communication channels**
  - Inter-Process Communications
  - Mobicore Communication Interface (MCI)
    - A communication channel with the Normal World based on the Mobicore Control Protocol (MCP)

MCLIB: KINIBI'S STANDARD LIBRARY

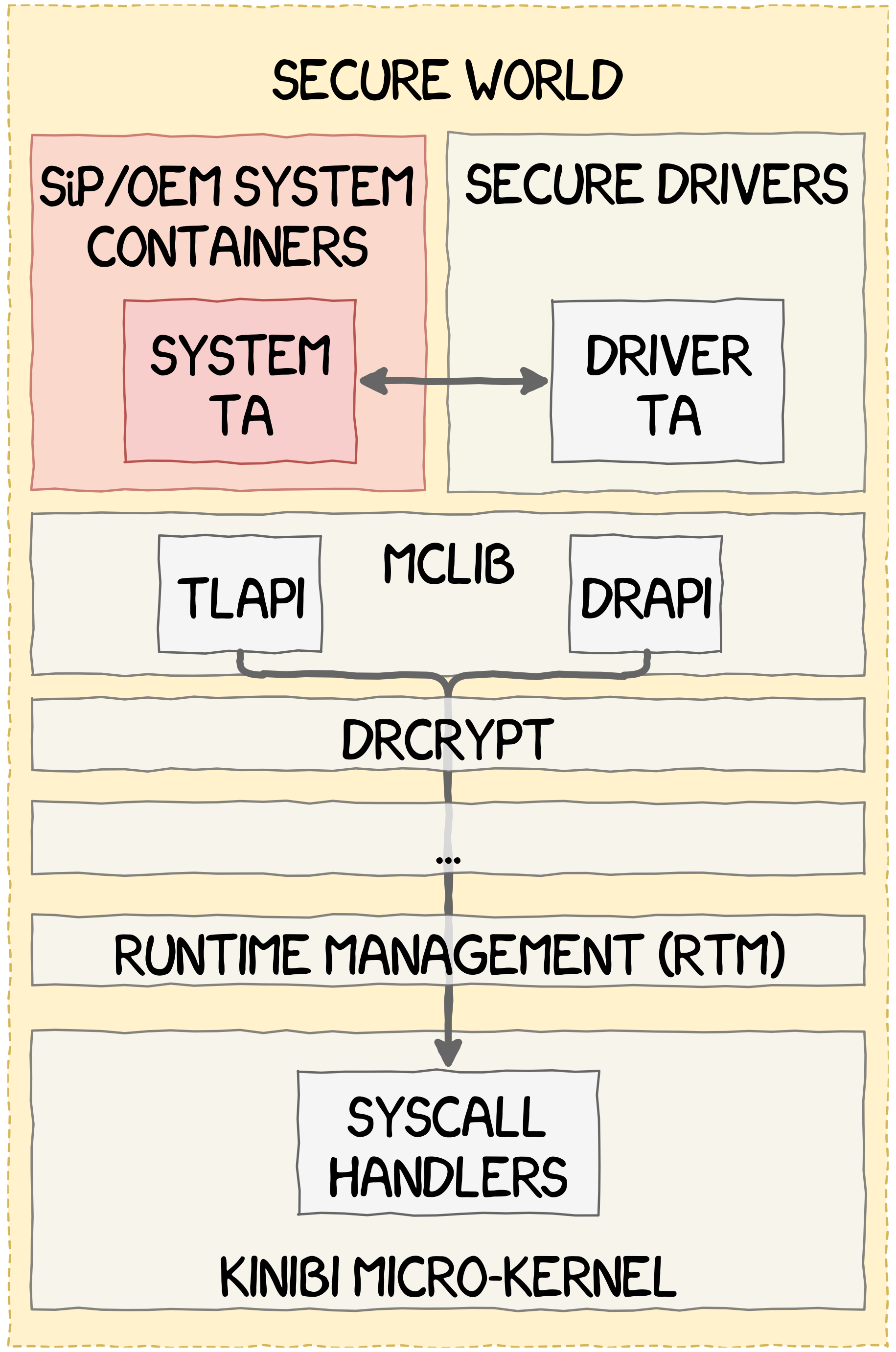


- Provides standard functions to Trusted Applications, Secure Drivers and RTM
- Separated into two APIs:
  - **TlApi:** set of functions used by trusted applications
  - **DrApi:** set of functions used by secure drivers
- Useful during exploitation to find gadgets

- **TlApi call example**

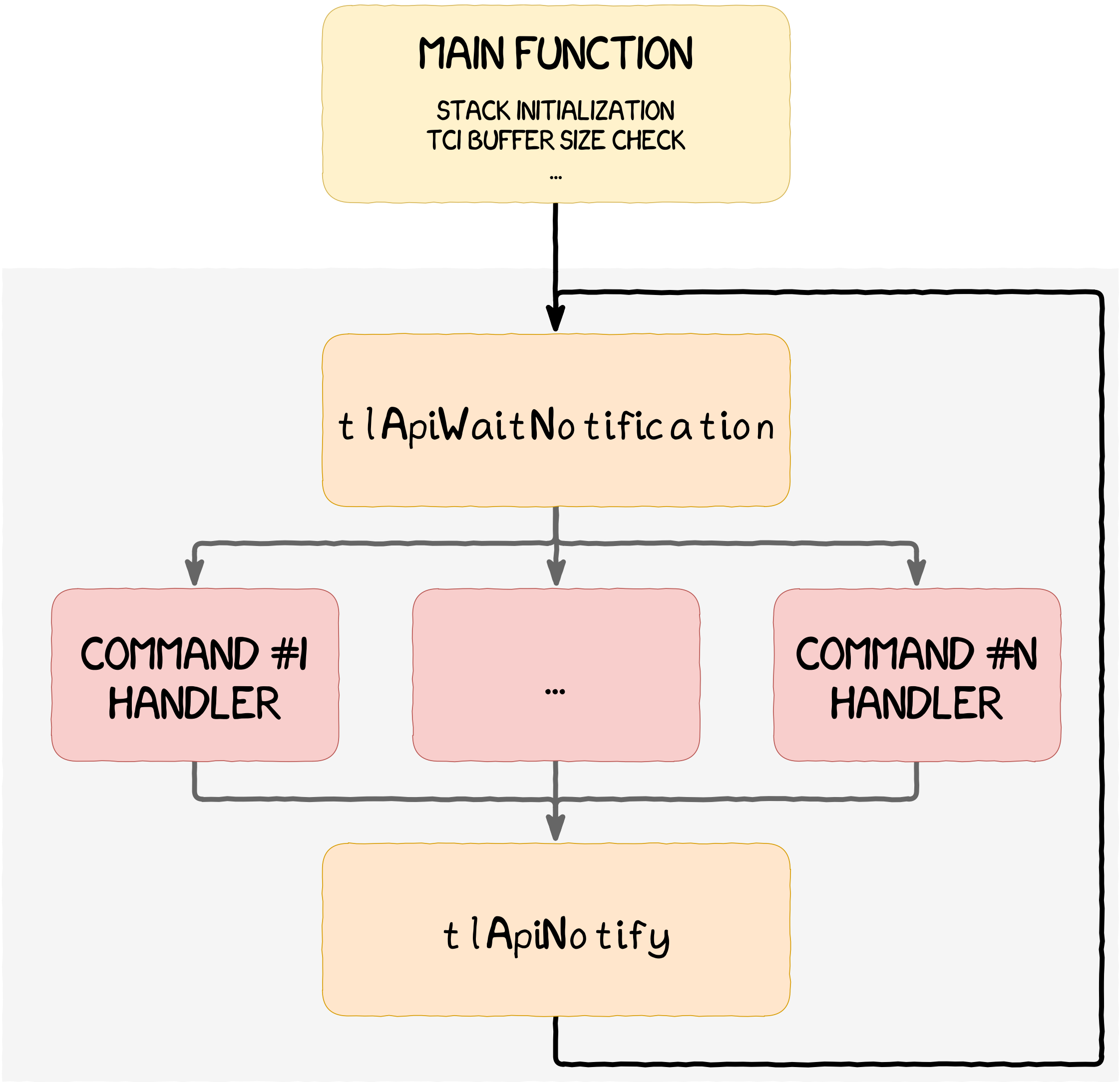
```
; _DWORD tlApiWaitNotification(_DWORD timeout)
MOV.W           R1, #0x1000
LDR.W           R2, [R1,#(tlApiLibEntry - 0x1000)]
MOV             R1, R0
MOVS            R0, #6
BX              R2
```

TRUSTED APPLICATIONS



- Secure World equivalent of regular applications in the Normal World (run at S-EL0)
- Allow trusted third-parties to extend the functionalities of the TEE-OS
  - Trusted UI, DRM, storage of secrets, etc.
- Signed binaries loaded directly from the Normal World (so are SDs)

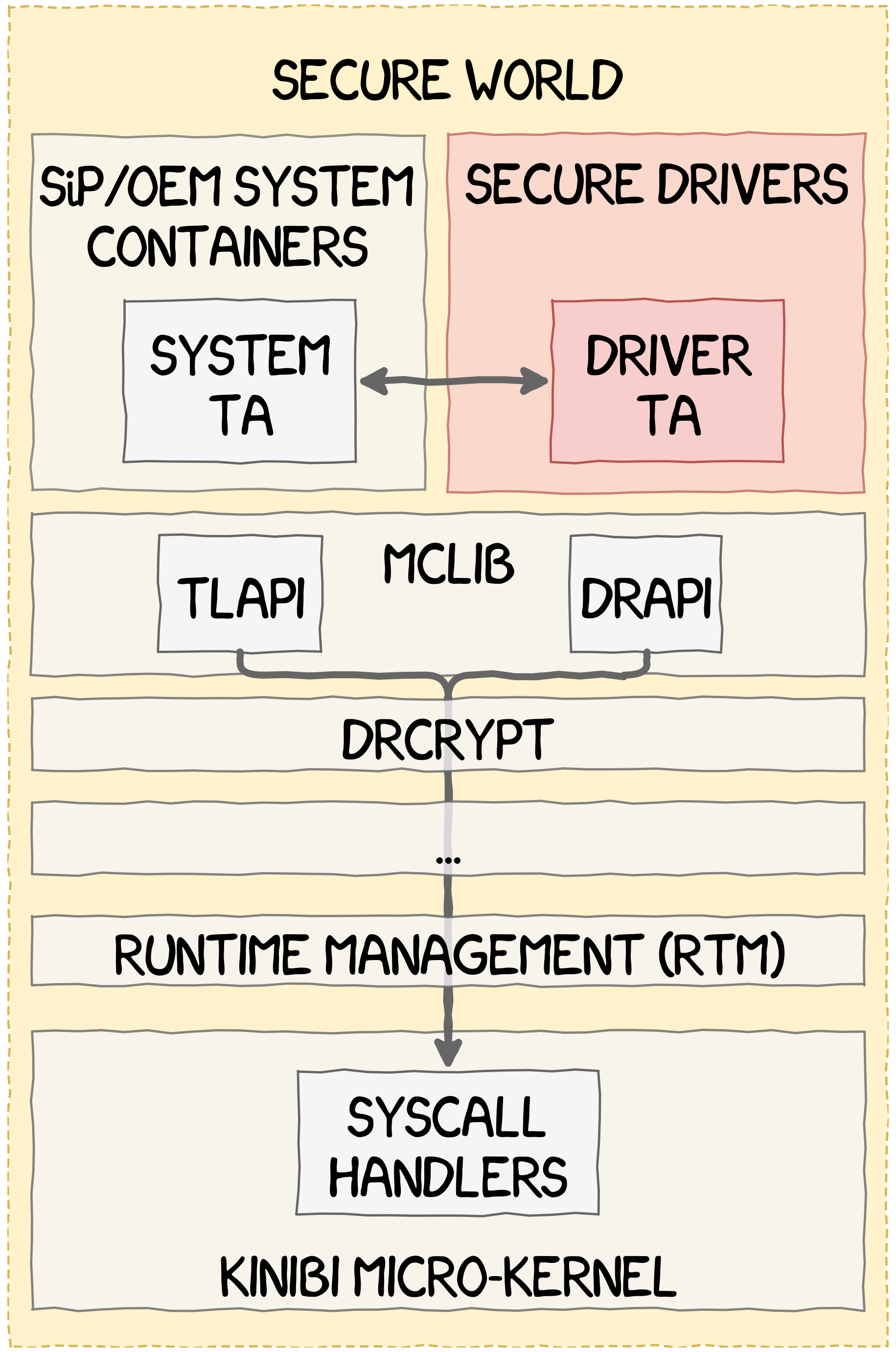
TRUSTED APPLICATIONS LIFE-CYCLE



- Communications with the Normal World made through world-shared memory (named TCI buffer by Trustonic)
- The TCI buffer contains commands to be handled by the trustlet
- TCI buffer contains commands to be handled by the trustlet

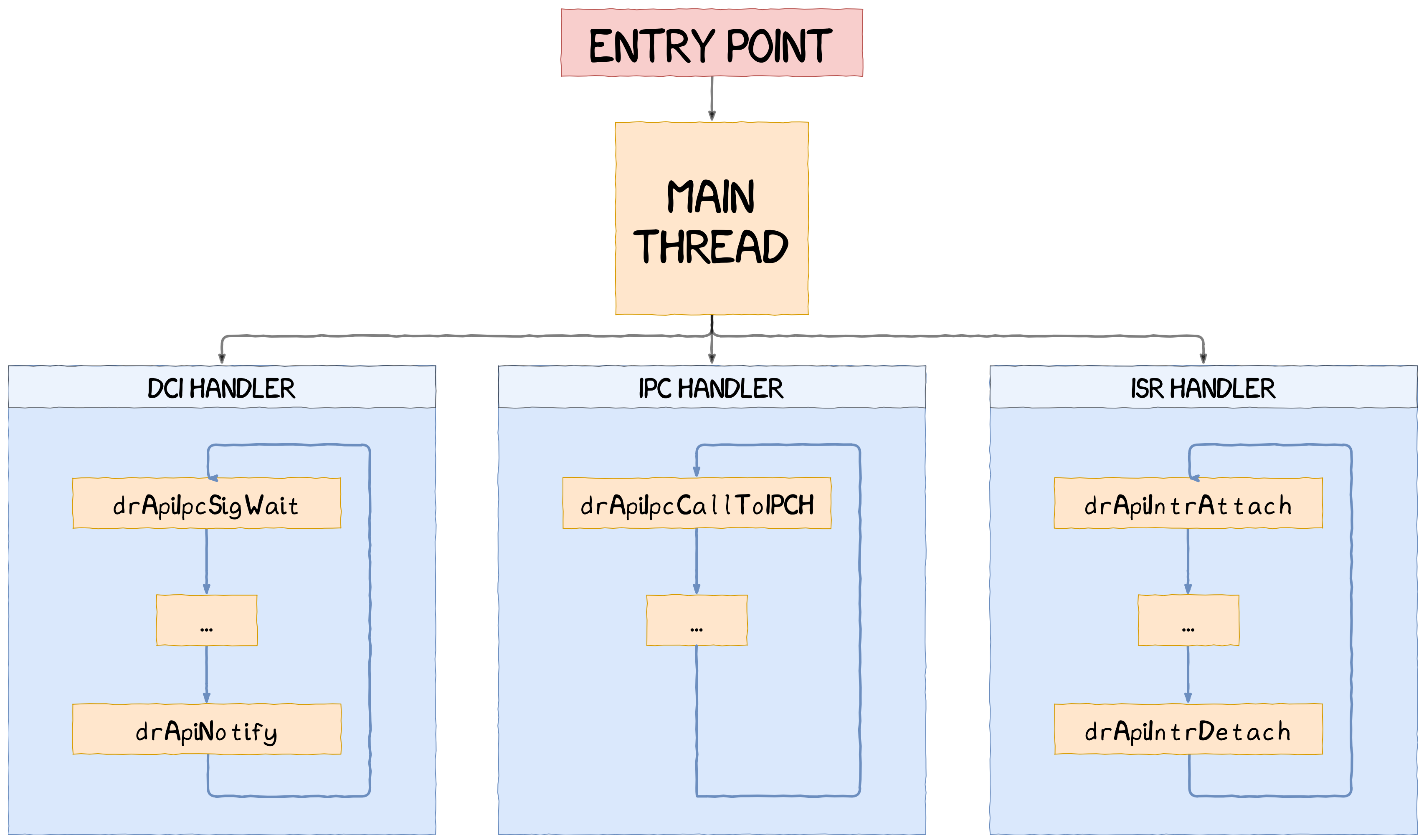
- **Notifications**
  - `tlApiWaitNotification`
  - `tlApiNotify`

SECURE DRIVERS



- Special type of Trusted Applications
- Run at S-EL0 but have higher software-define privileges
- Have access to a richer set of API and syscalls
- Are used by trustlets as an interface to access physical memory and reach secure peripherals in a controlled manner
- Communications with TAs made through IPCs and shared memory

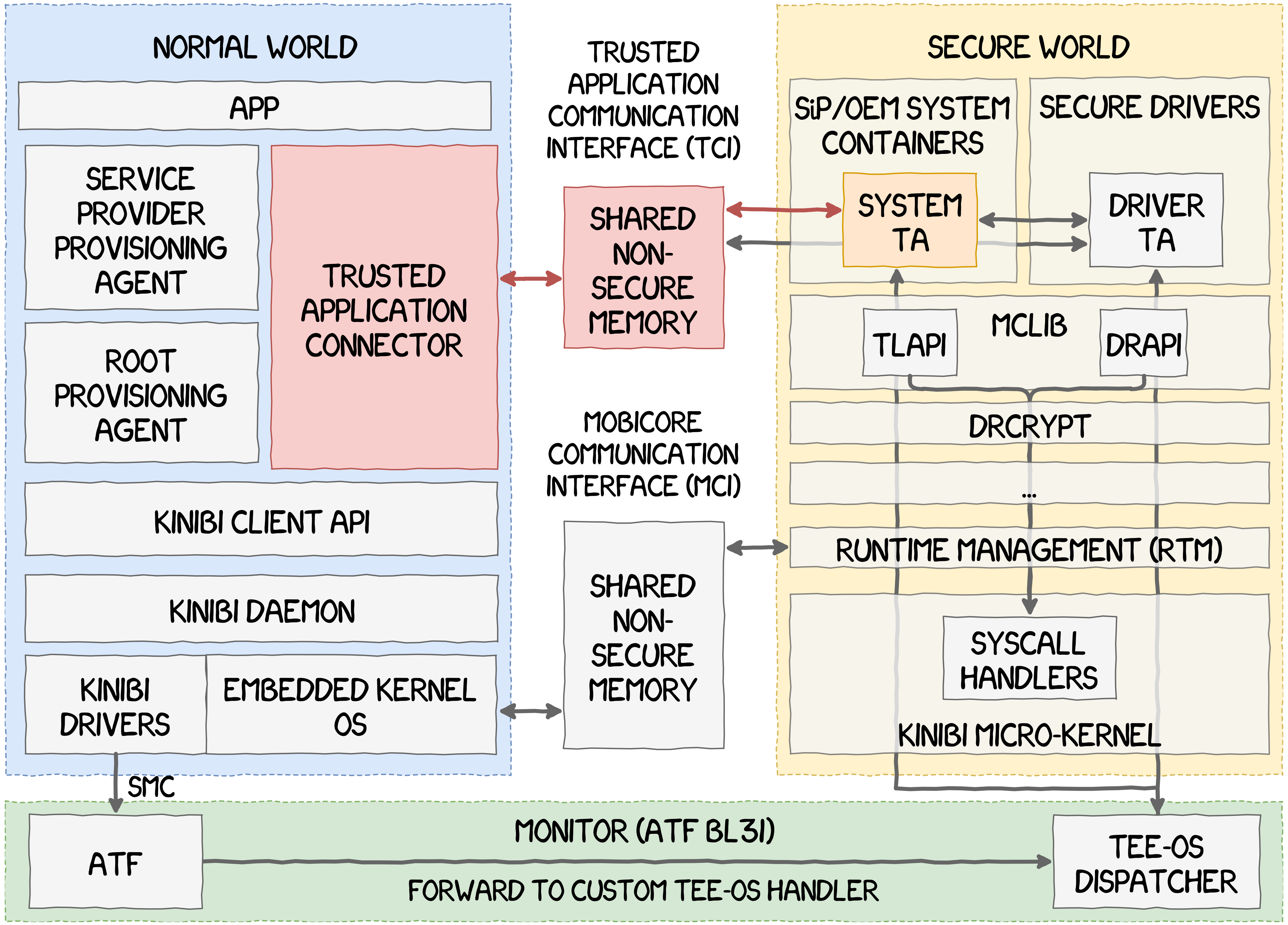
SECURE DRIVERS LIFE-CYCLE



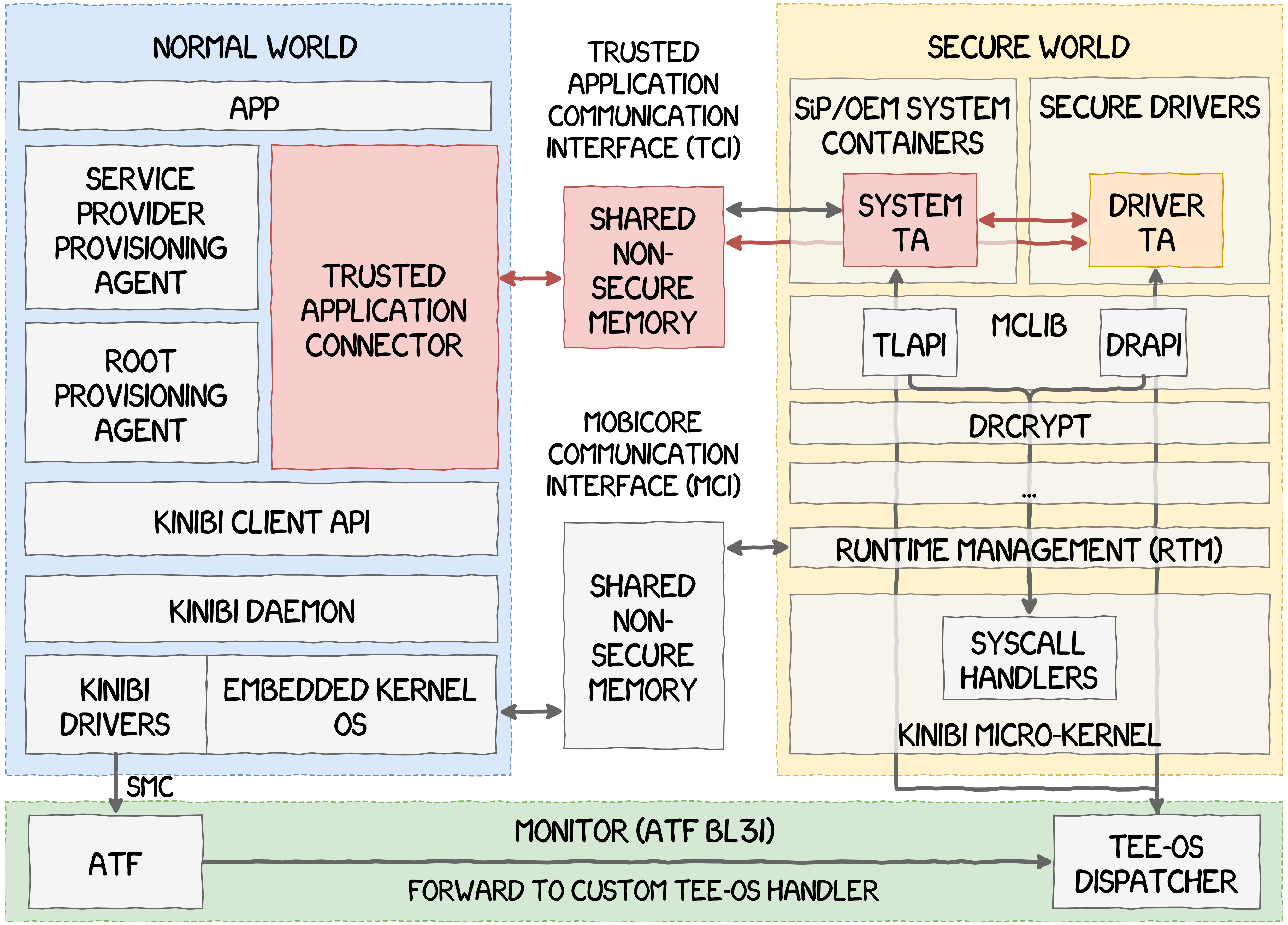
- **Multi-threaded application**
  - **DCI:** Normal World communications
  - **IPC:** trustlet communications
- **Trustlet interactions**
  - Retrieves IPC data by mapping the entire trustlet
  - Notifications using `drApiIpcCallToIPCH`

VULNERABILITY RESEARCH TOOLS

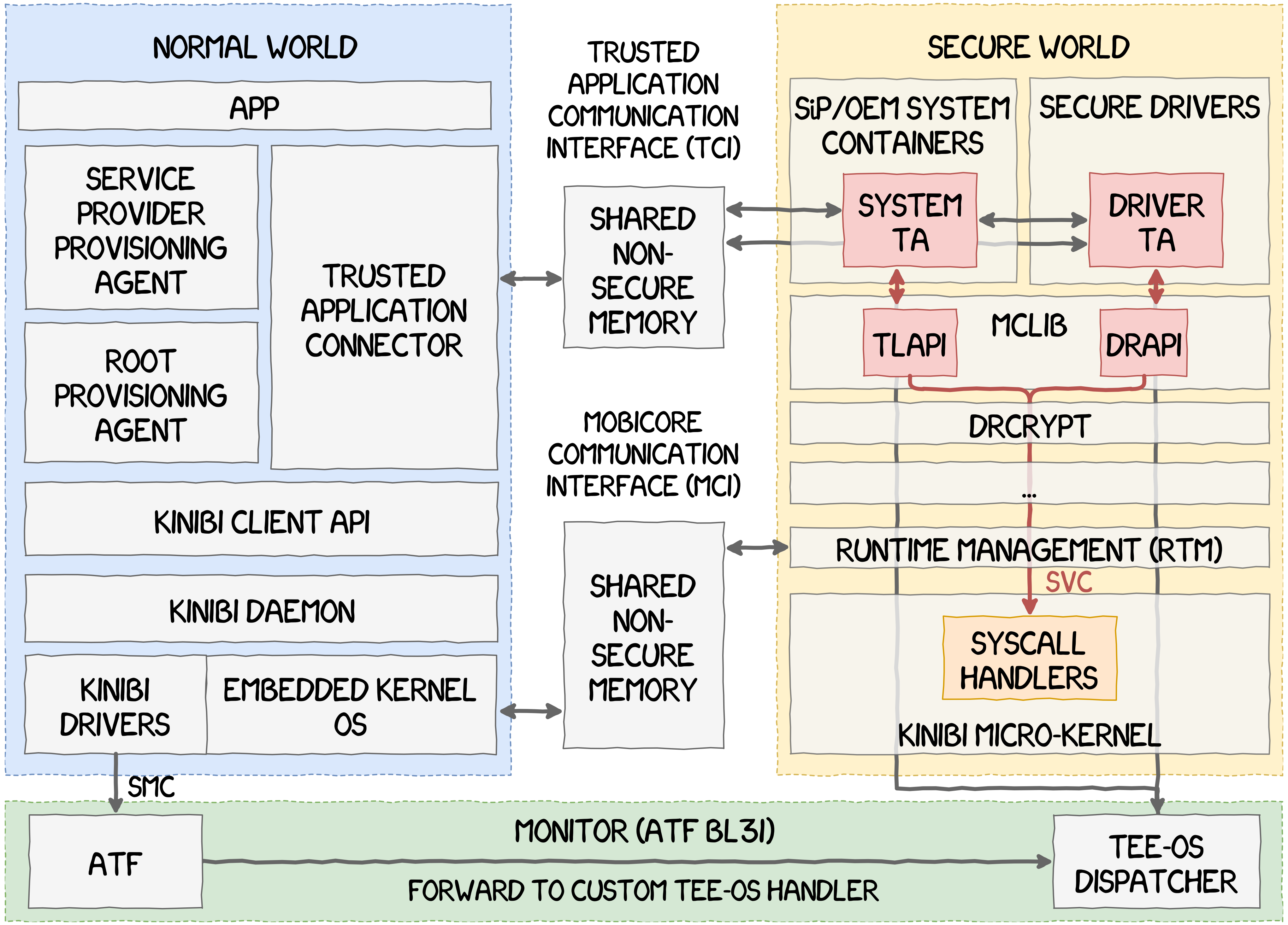
ATTACK SURFACE



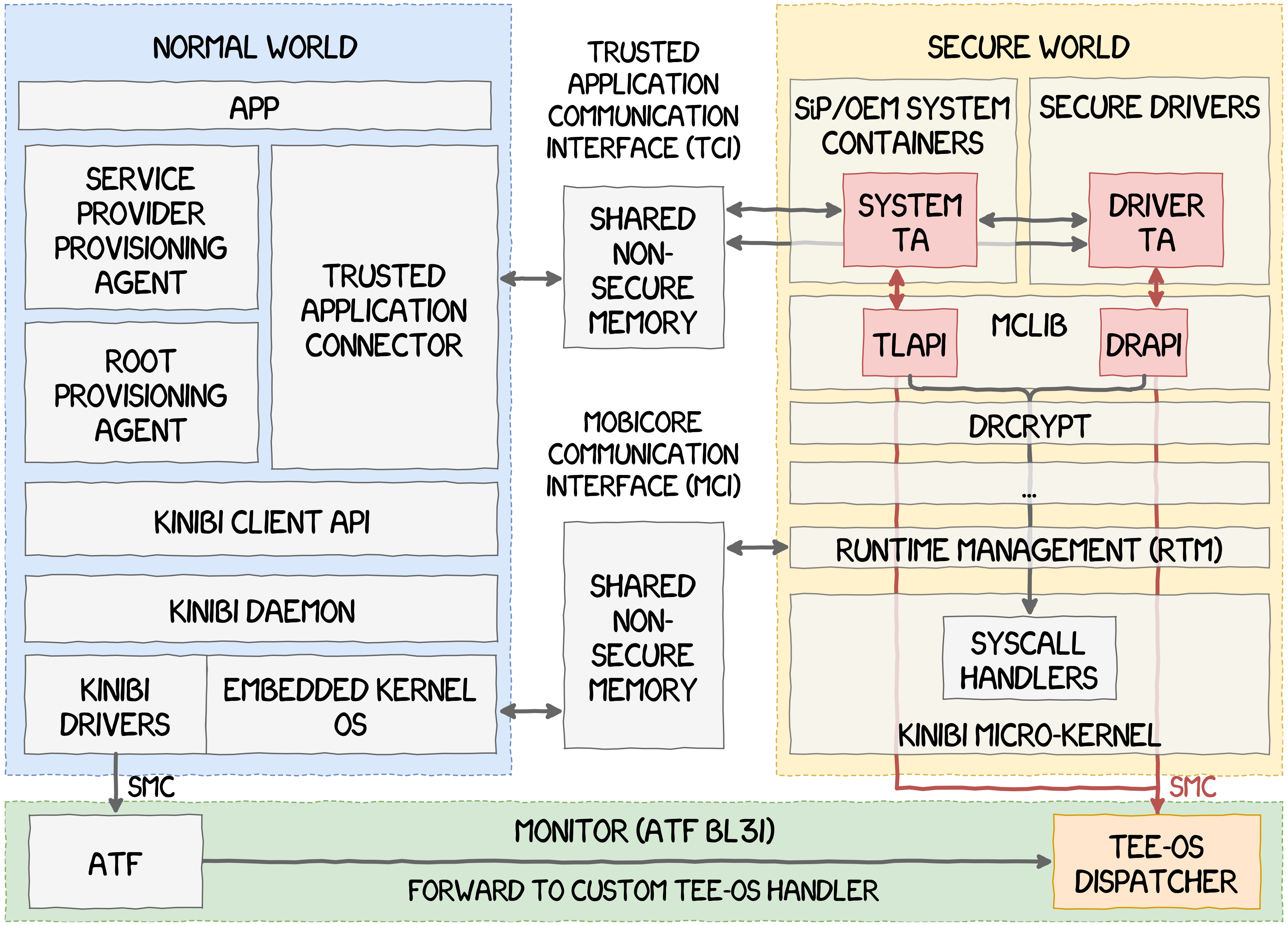
ATTACK SURFACE



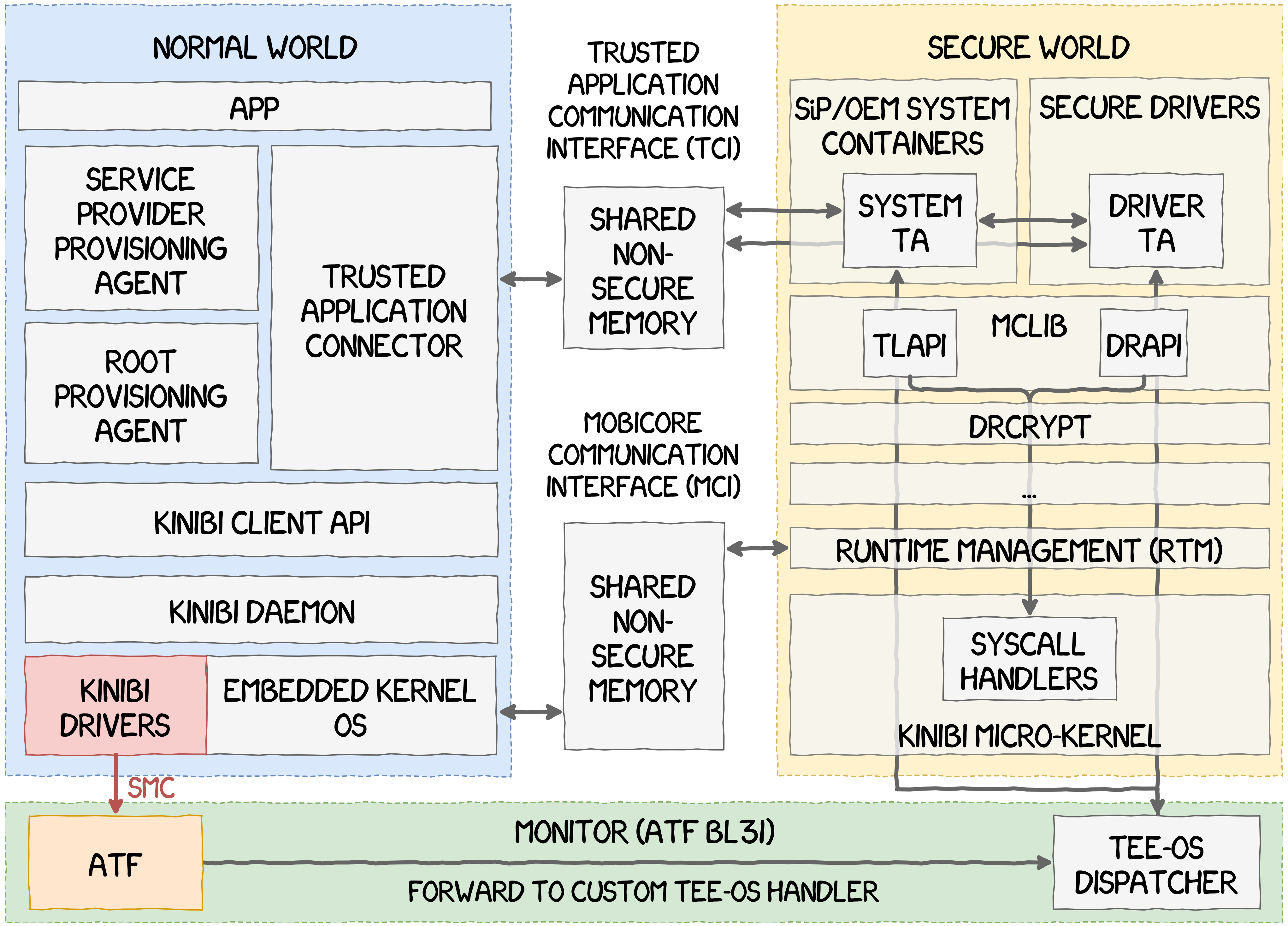
ATTACK SURFACE



ATTACK SURFACE



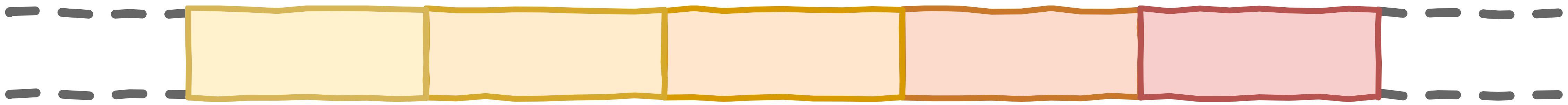
ATTACK SURFACE



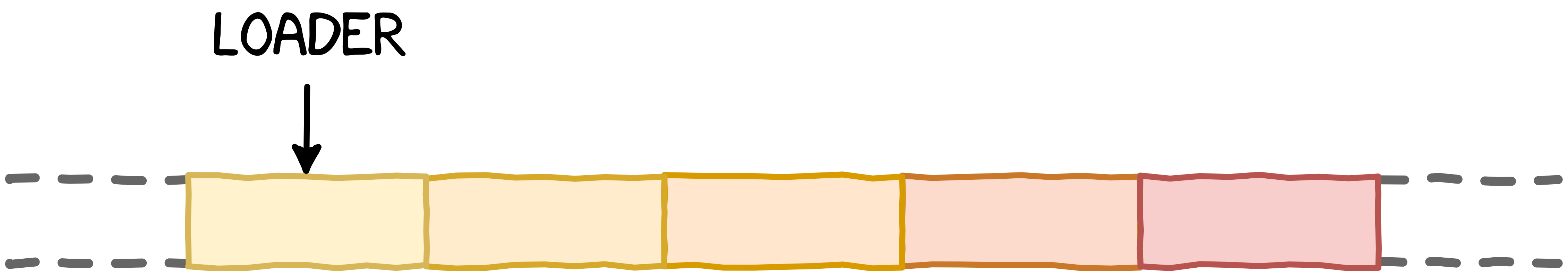
ATTACK SURFACE

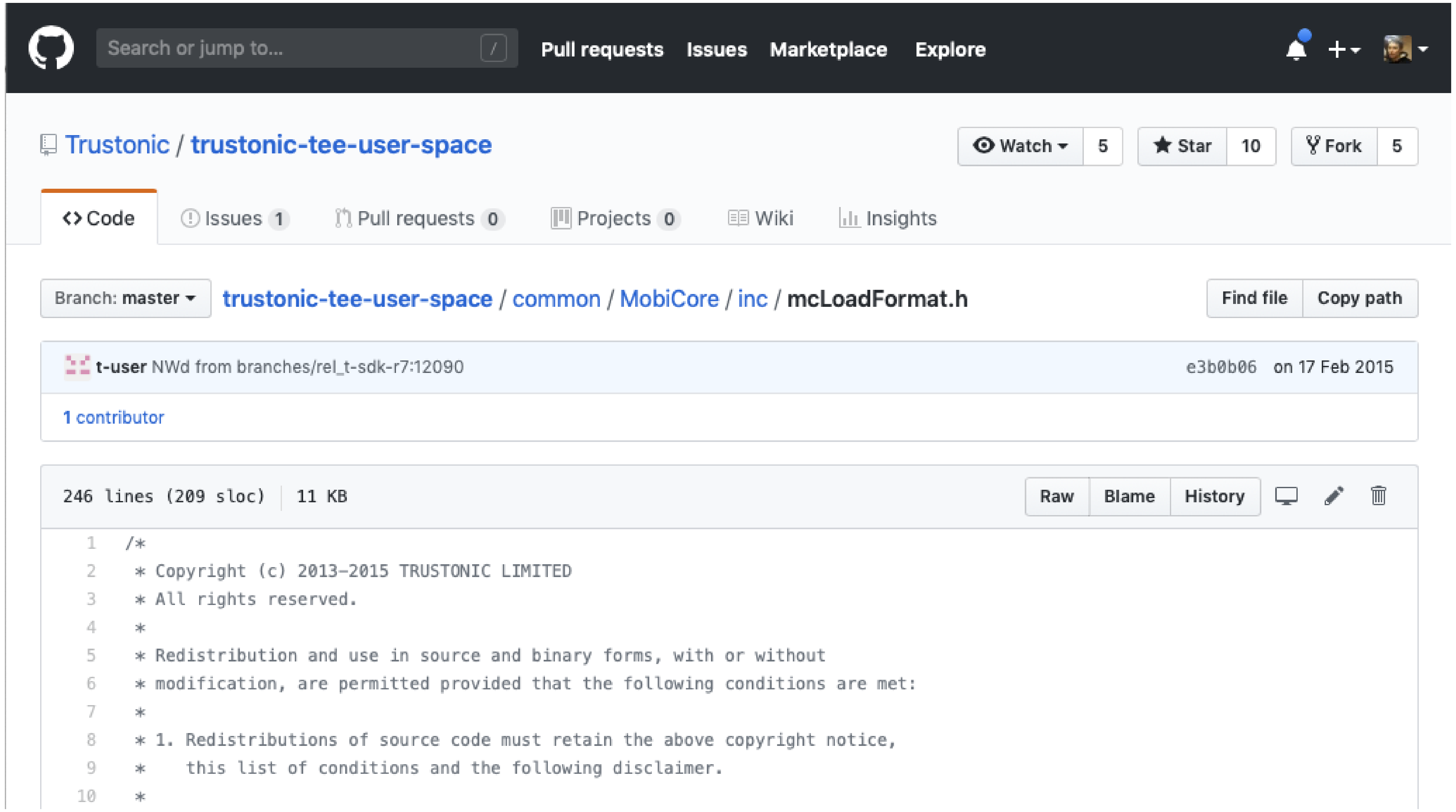
- Must be reacheable from the Normal World
- ATF is open-source, probably heavily reviewed
- Trusted Applications are low-hanging fruits

OUR JOURNEY IN 5 STEPS

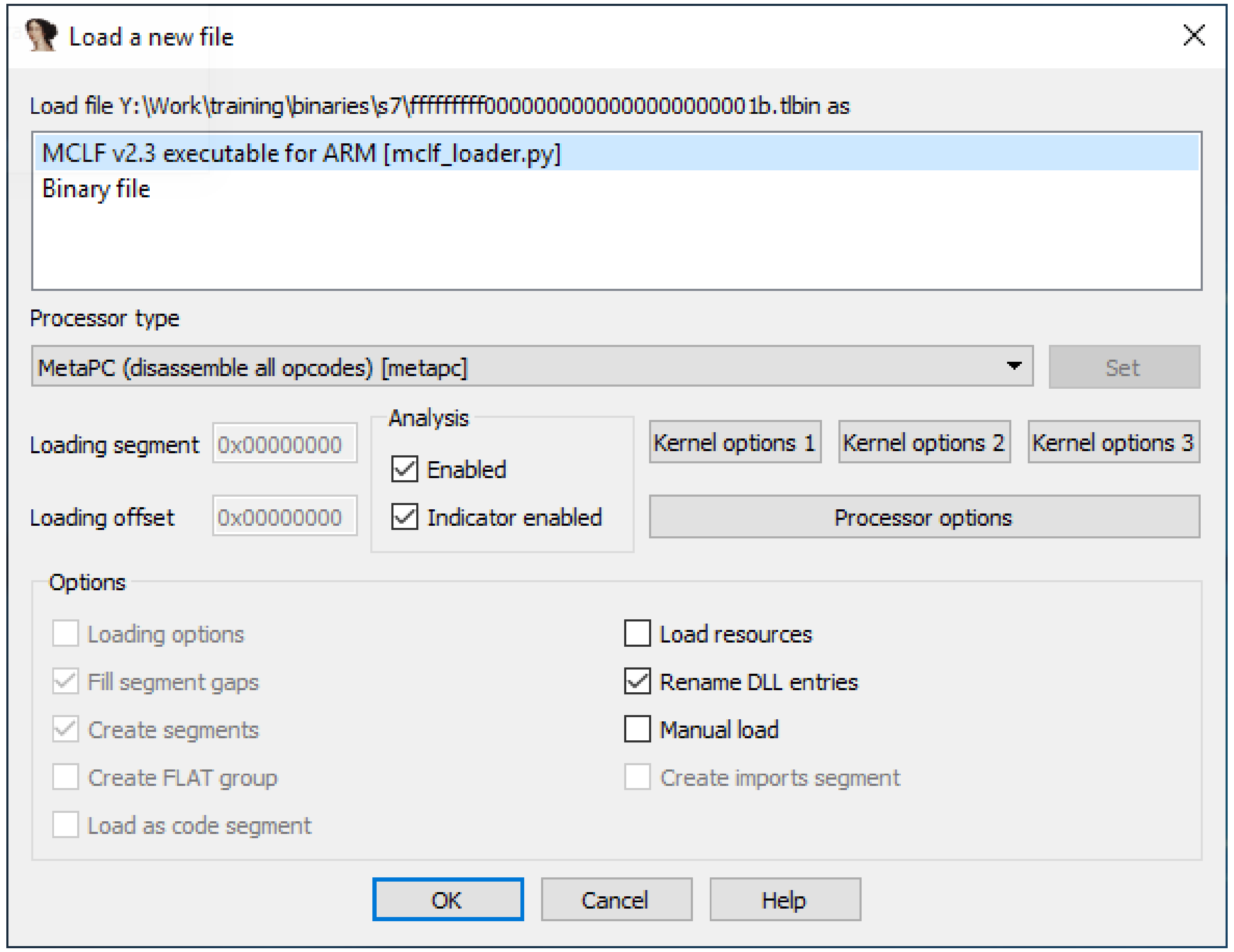


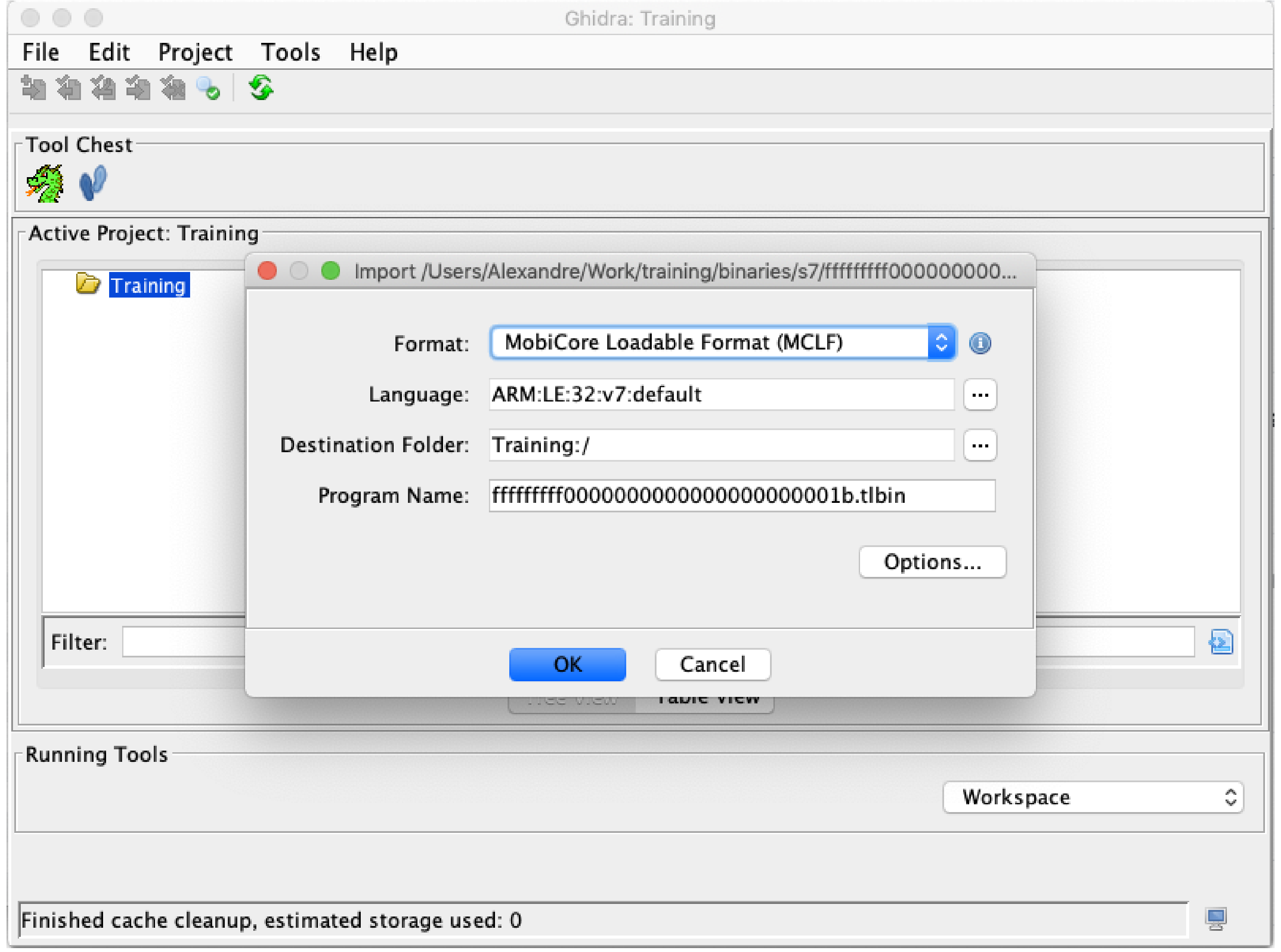
STEP #1 - LOADING INTO IDA/GHIDRA





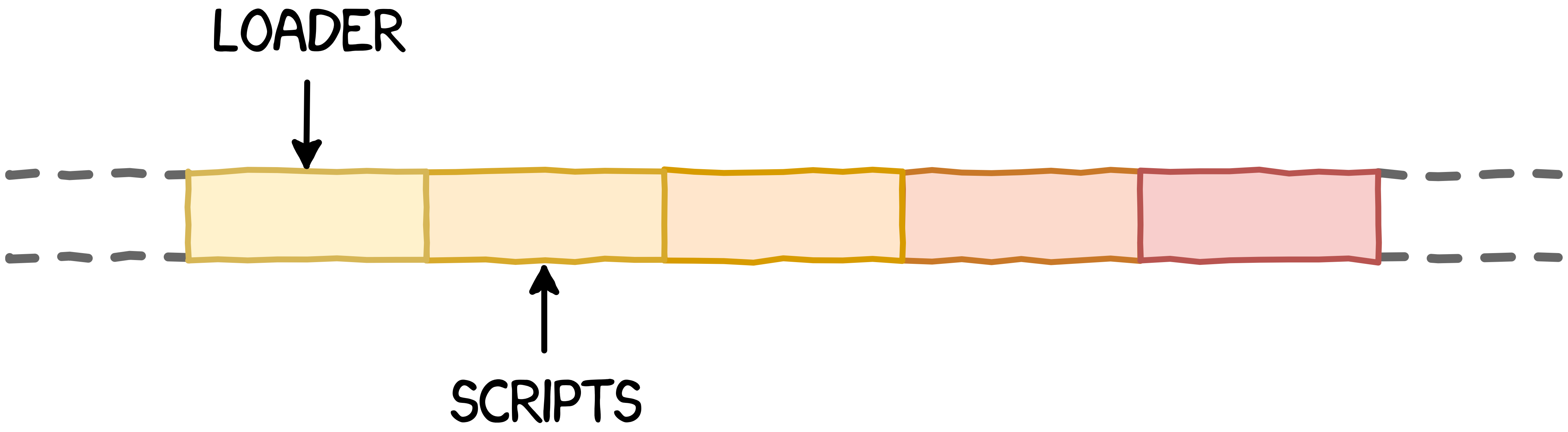
- Proprietary File Format - MobiCore Loadable Format (MCLF)



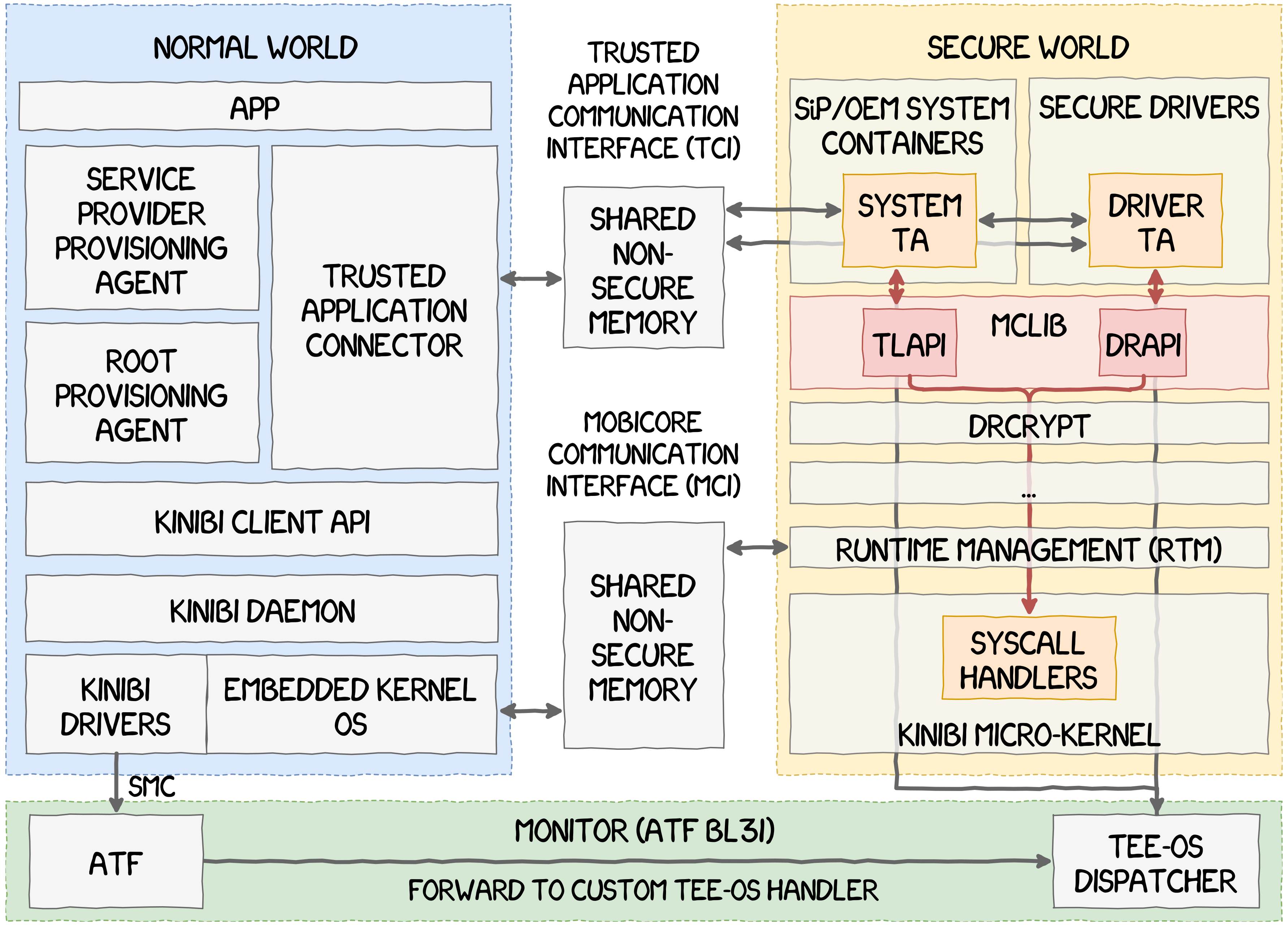


- mclf-ida-loader
- mclf-ghidra-loader

STEP #2 - IDENTIFYING FUNCTIONS



MCLIB - STANDARD LIBRARY



- Renames tlApi/drAPI functions
- Sets the functions prototypes

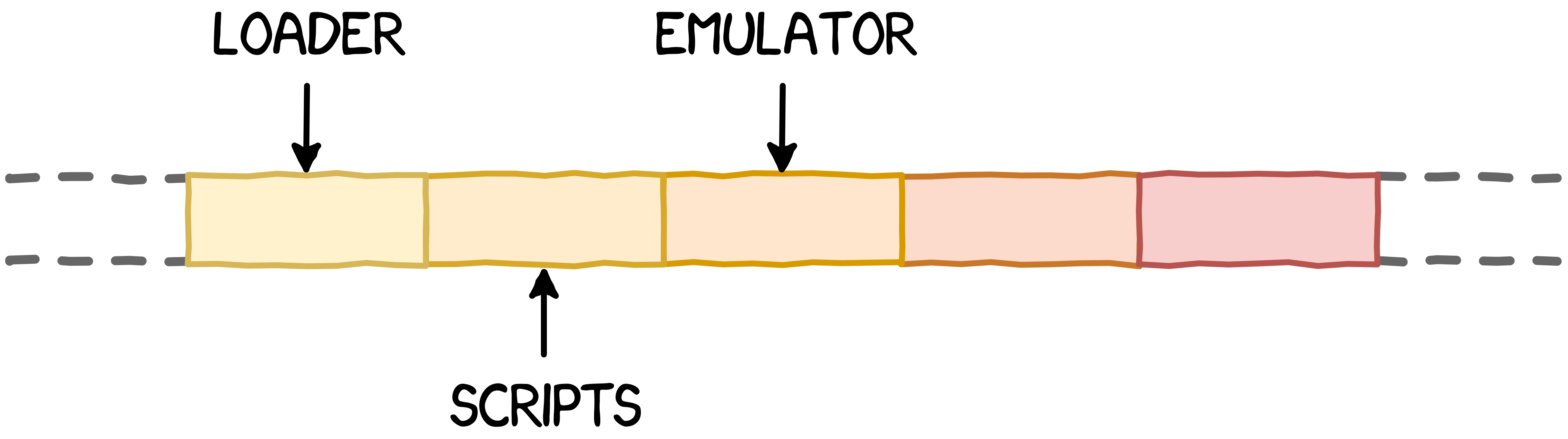
```
.text:00023EA8 ; =============== S U B R O U T I N E =======================================
.text:00023EA8
.text:00023EA8
.text:00023EA8 sub_23EA8                               ; CODE XREF: _entry-4A80↑p
.text:00023EA8                 LDR             R1, =dword_1000
.text:00023EAA                 LDR.W           R2, [R1,#(tlApiLibEntry - 0x1000)]
.text:00023EAE                 MOV             R1, R0
.text:00023EB0                 MOVS            R0, #6
.text:00023EB2                 BX              R2
.text:00023EB2 ; End of function sub_23EA8
.text:00023EB2
.text:00023EB2 ; ---------------------------------------------------------------------------
.text:00023EB4 off_23EB4       DCD dword_1000          ; DATA XREF: sub_23EA8↑r
.text:00023EB8
.text:00023EB8 ; =============== S U B R O U T I N E =======================================
.text:00023EB8
.text:00023EB8
.text:00023EB8 sub_23EB8                               ; CODE XREF: sub_26CC+1A↑p
.text:00023EB8                                         ; sub_347C+C↑p
.text:00023EB8                 LDR             R3, =dword_1000
.text:00023EBA                 PUSH            {R4,R5}
.text:00023EBC                 LDR.W           R4, [R3,#(tlApiLibEntry - 0x1000)]
.text:00023EC0                 MOV             R3, R2
.text:00023EC2                 MOV             R2, R1
.text:00023EC4                 MOV             R1, R0
.text:00023EC6                 MOV             R12, R4
.text:00023EC8                 POP             {R4,R5}
.text:00023ECA                 MOVS            R0, #0xE
.text:00023ECC                 BX              R12
.text:00023ECC ; End of function sub_23EB8
```

Before

```
.text:00023EA8 ; =============== S U B R O U T I N E =======================================
.text:00023EA8
.text:00023EA8
.text:00023EA8 ; _DWORD __cdecl tlApiWaitNotification(_DWORD timeout)
.text:00023EA8 tlApiWaitNotification                   ; CODE XREF: _entry-4A80↑p
.text:00023EA8                 LDR             R1, =dword_1000
.text:00023EAA                 LDR.W           R2, [R1,#(tlApiLibEntry - 0x1000)]
.text:00023EAE                 MOV             R1, R0
.text:00023EB0                 MOVS            R0, #6
.text:00023EB2                 BX              R2
.text:00023EB2 ; End of function tlApiWaitNotification
.text:00023EB2
.text:00023EB2 ; ---------------------------------------------------------------------------
.text:00023EB4 off_23EB4       DCD dword_1000          ; DATA XREF: tlApiWaitNotification↑r
.text:00023EB8
.text:00023EB8 ; =============== S U B R O U T I N E =======================================
.text:00023EB8
.text:00023EB8
.text:00023EB8 ; _DWORD __cdecl tlApiRandomGenerateData(_DWORD alg, _DWORD randomBuffer, _DWORD randomLen)
.text:00023EB8 tlApiRandomGenerateData                 ; CODE XREF: sub_26CC+1A↑p
.text:00023EB8                                         ; sub_347C+C↑p
.text:00023EB8                 LDR             R3, =dword_1000
.text:00023EBA                 PUSH            {R4,R5}
.text:00023EBC                 LDR.W           R4, [R3,#(tlApiLibEntry - 0x1000)]
.text:00023EC0                 MOV             R3, R2
.text:00023EC2                 MOV             R2, R1
.text:00023EC4                 MOV             R1, R0
.text:00023EC6                 MOV             R12, R4
.text:00023EC8                 POP             {R4,R5}
.text:00023ECA                 MOVS            R0, #0xE
.text:00023ECC                 BX              R12
.text:00023ECC ; End of function tlApiRandomGenerateData
```

After

STEP #3 - MANUALLY FINDING VULNERABILITIES



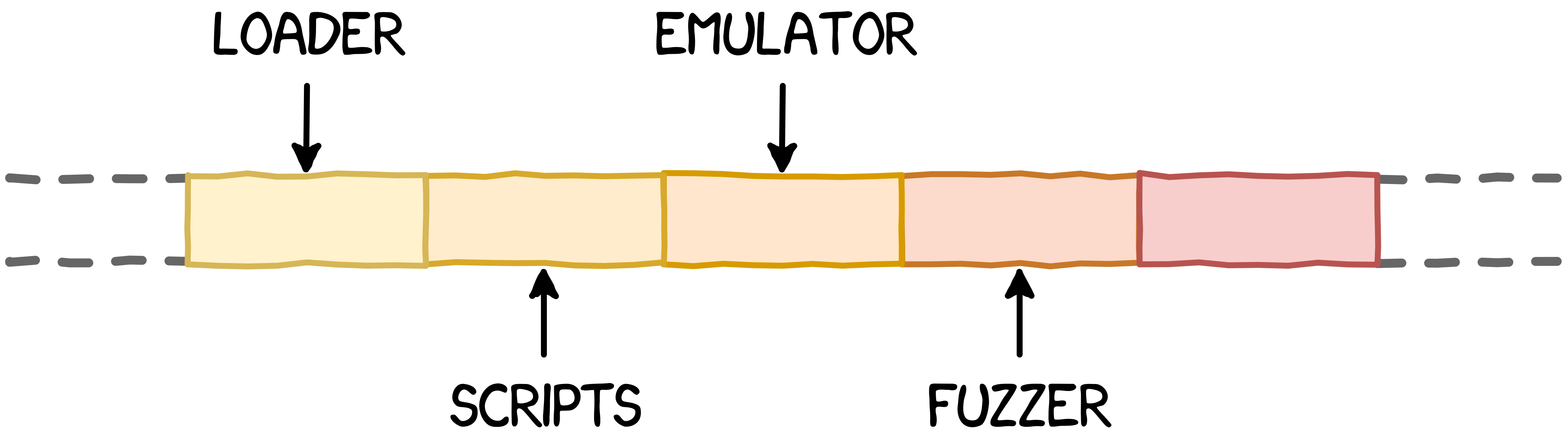
TRUSTLETS EMULATOR

- Based on Unicorn (external project)
- Split into simple tasks:
  - Loading the MCLF binary
  - Mapping the shared memory buffer
  - Hooking the McLib functions

TRUSTLETS EMULATOR

```
python emulator.py *41.tlbin cmd1.bin --tci 0x40100 -v
[+] Binary is a trustlet
[+] Trustlet size = 0x1ba4c
[+] Mapping text section at 0x00001000 with a size of 0x4874
[+] Mapping data section at 0x00007000 with a size of 0x168
[+] Mapping BSS section at 0x00007168 with a size of 0x17070
[+] Mapping region at 0x07d00000 (0x1 bytes)
[+] Mapping TCI buffer at 0x00100000 with a size of 0x40100
[i] drApiLogvPrintf(u'ICCC:Trustlet ICCC::Starting\n')
[+] Loading input data
[i] drApiLogvPrintf(u'TL ICCC: we got a command: 1\n')
[i] drApiLogvPrintf(u'ICCC: Initialize failed - tamper fuse set\n')
[i] drApiLogvPrintf(u'ICCC: Measurements result ret = 65548, ret hex = 1000c\n')
[i] drApiLogvPrintf(u'iccc: ICCC save data@#\n')
[i] drApiLogvPrintf(u'Iccc_phys_read failed\n')
[i] drApiLogvPrintf(u'ICCC: check magic failed\n')
[i] drApiLogvPrintf(u'End of ICCC_Init, ret=1000c\n')
[i] drApiLogvPrintf(u'ICCC: Error writing Trustboot flag\n')
[+] tlApiNotify: Quitting!
```

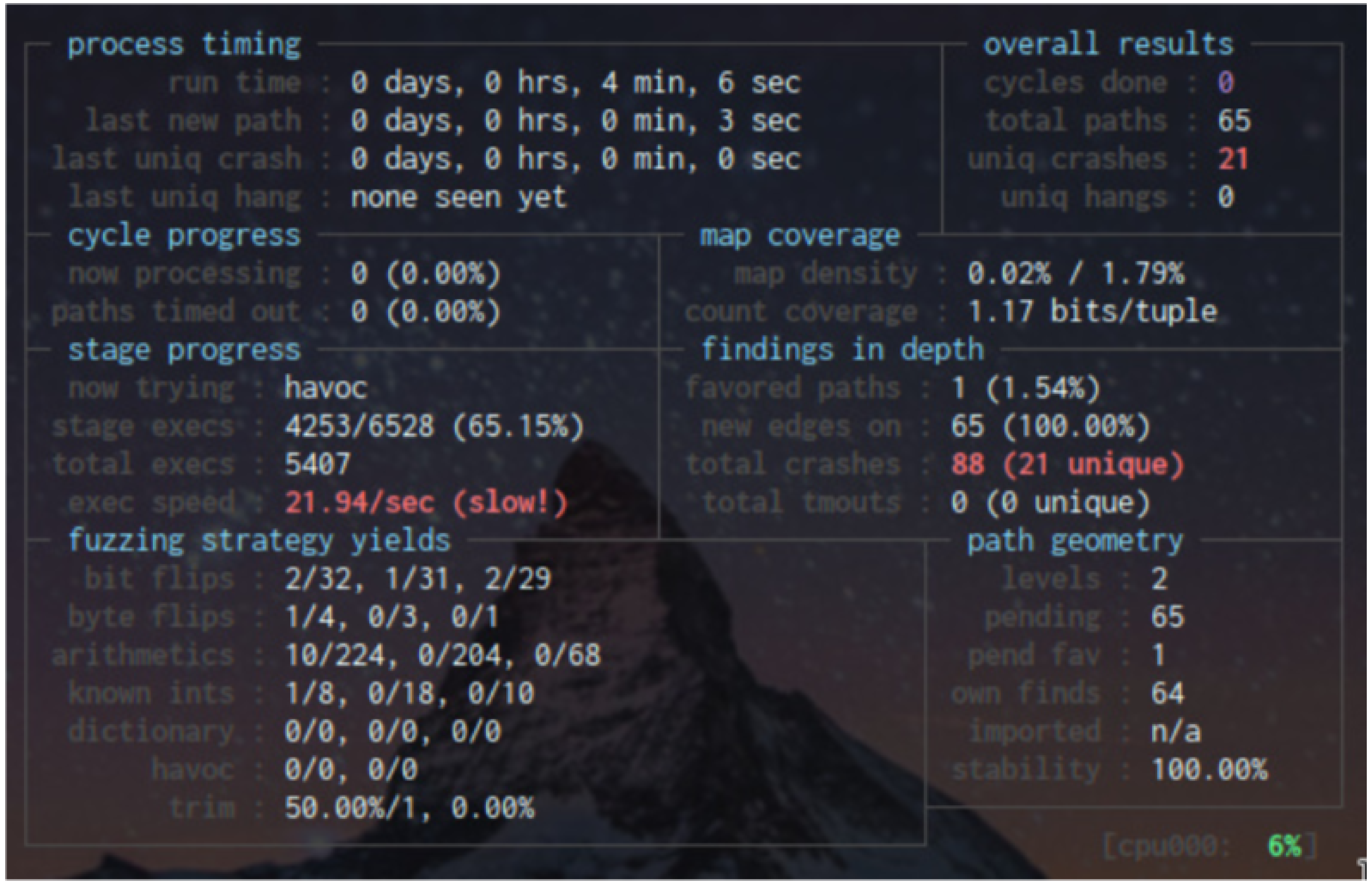
STEP #4 - FINDING VULNERABILITIES AUTOMATICALLY



TRUSTLETS FUZZER

- Based on AFL_Unicorn (internal project)
  - Interfaces the fuzzer AFL with Unicorn
  - Usability and performance improvements
  - 100% of the code is written in Python!

TRUSTLETS FUZZER



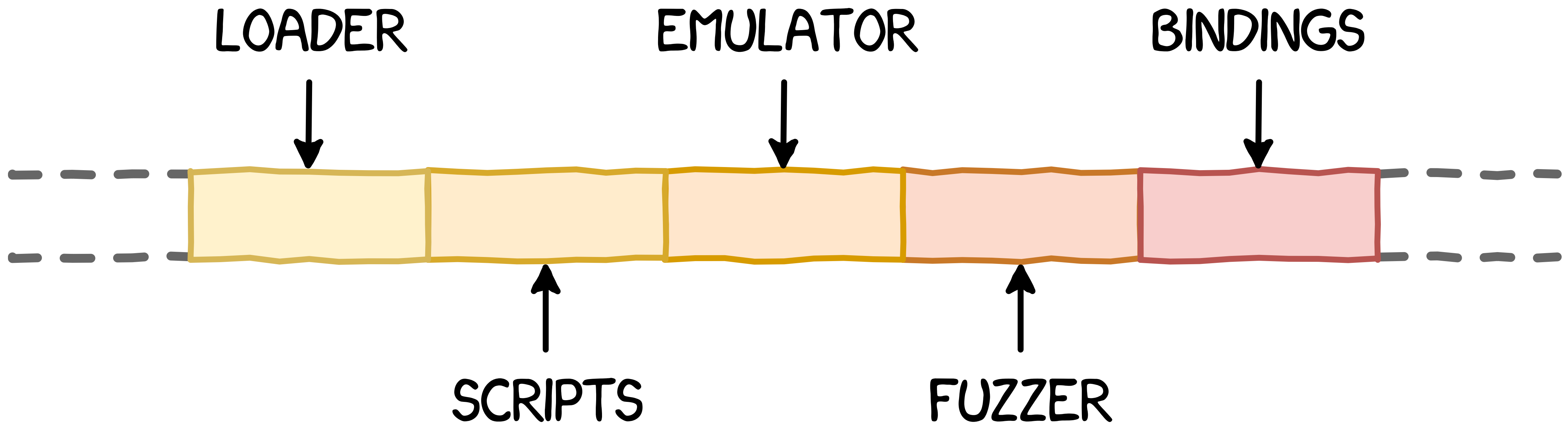
TRUSTLETS SYMBOLIC EXECUTOR

- Based on Manticore by Trail of Bits
- Uses very simple strategies:
  - Mark the shared memory buffer symbolic
  - Explore all the paths of the trustlet
  - Check reads or writes to memory
  - Ask the solver for an invalid address

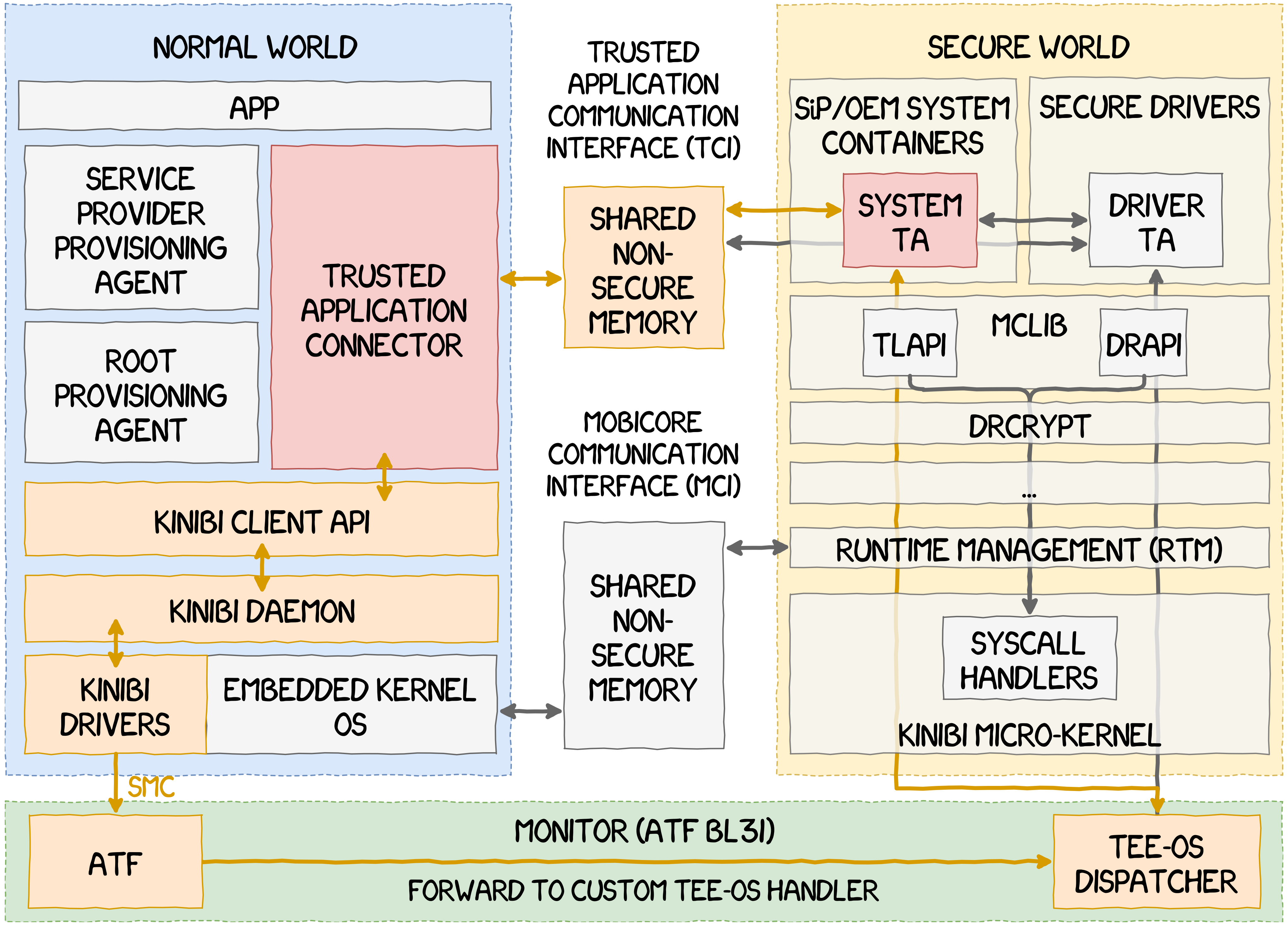
CRASH EXAMPLE

```
Command line:
  '../tainter.py -s 1036 ffffffff000000000000000000000005.tlbin.elf -t -c coverage.txt'
Status:
  Invalid symbolic memory access (mode:r)

================ PROC: 00 ================
Memory:
0000000000001000-0000000000008000  r x 00000094 ffffffff000000000000000000000005.tlbin.elf
  0000000000009000-0000000000024000  rw  00006dc8 ffffffff000000000000000000000005.tlbin.elf
  0000000000100000-0000000000101000   rw 00000000
  0000000007d00000-0000000007d01000   rx 00000000 CPU:
INSTRUCTION: 0x00000000000059ec:        pld     [r1, #0x80]
APSR: 0x0000000060000000
R0 : 0x0000000000009aac
R1 : <BitVecExtract at 7f2571dbdeb8-T>
R10: 0x0000000000000000
R11: 0x0000000000000000
R12: 0x0000000000000000
R13: 0x0000000000023a28
```

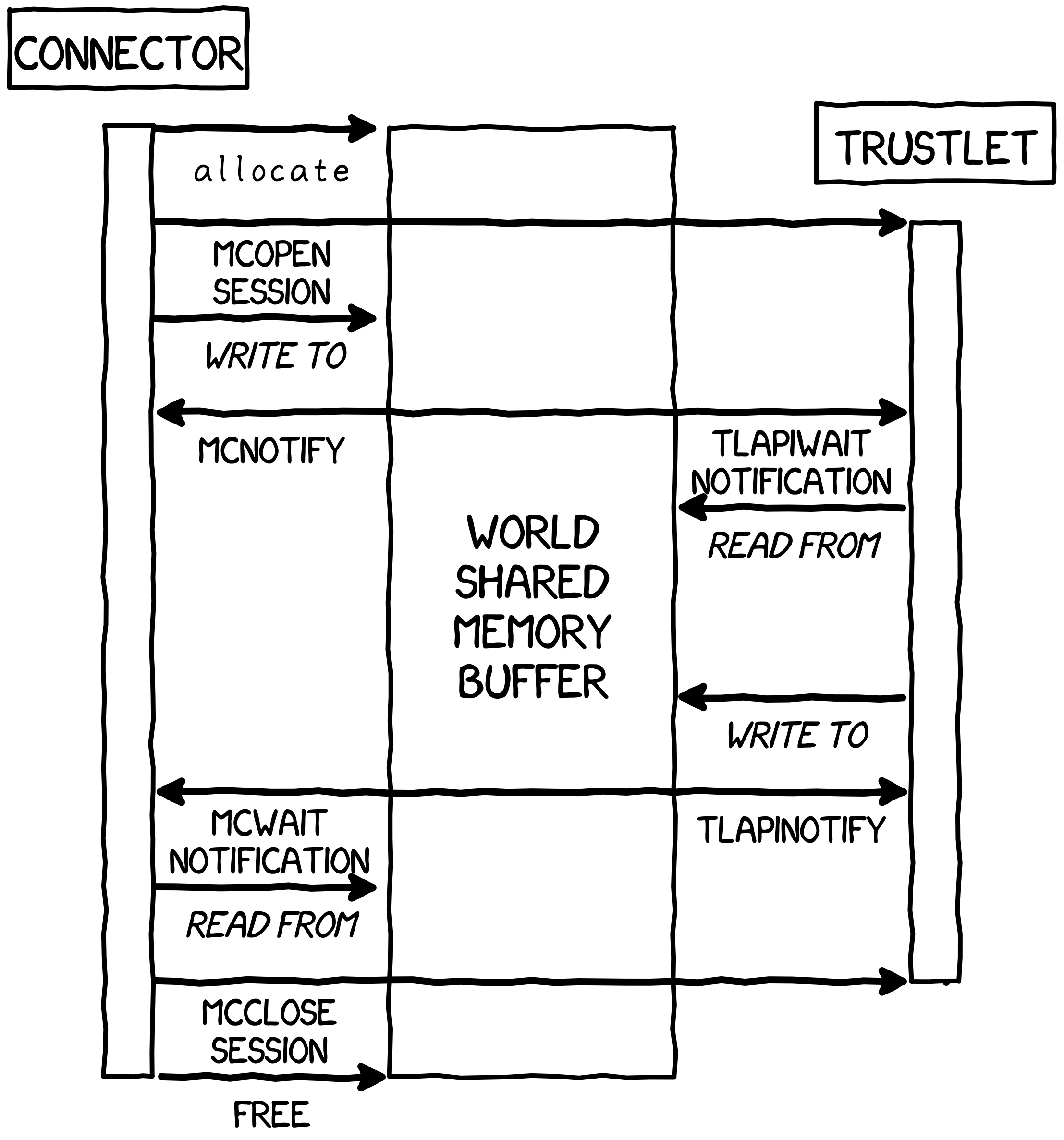
STEP #5 - EXPLOITING THE VULNERABILITIES



SOFTWARE STACK



CLIENT API



CONNECTOR
TRUSTLET
allocate
MCOPEN SESSION
WRITE TO
MCNOTIFY
TLAPIWAIT NOTIFICATION
READ FROM
WORLD SHARED MEMORY BUFFER
WRITE TO
MCWAIT NOTIFICATION
TLAPINOTIFY
READ FROM
MCCLOSE SESSION
FREE

PYTHON BINDINGS

- Writing C is tedious, writing Python is a lot easier
- Bindings of the mcClient API called `pymcclient`
- Provides various utilities: hexdump, (dis)assemble, etc.
- Provides a command interpreter which is based on IPython

SCRIPT EXAMPLE

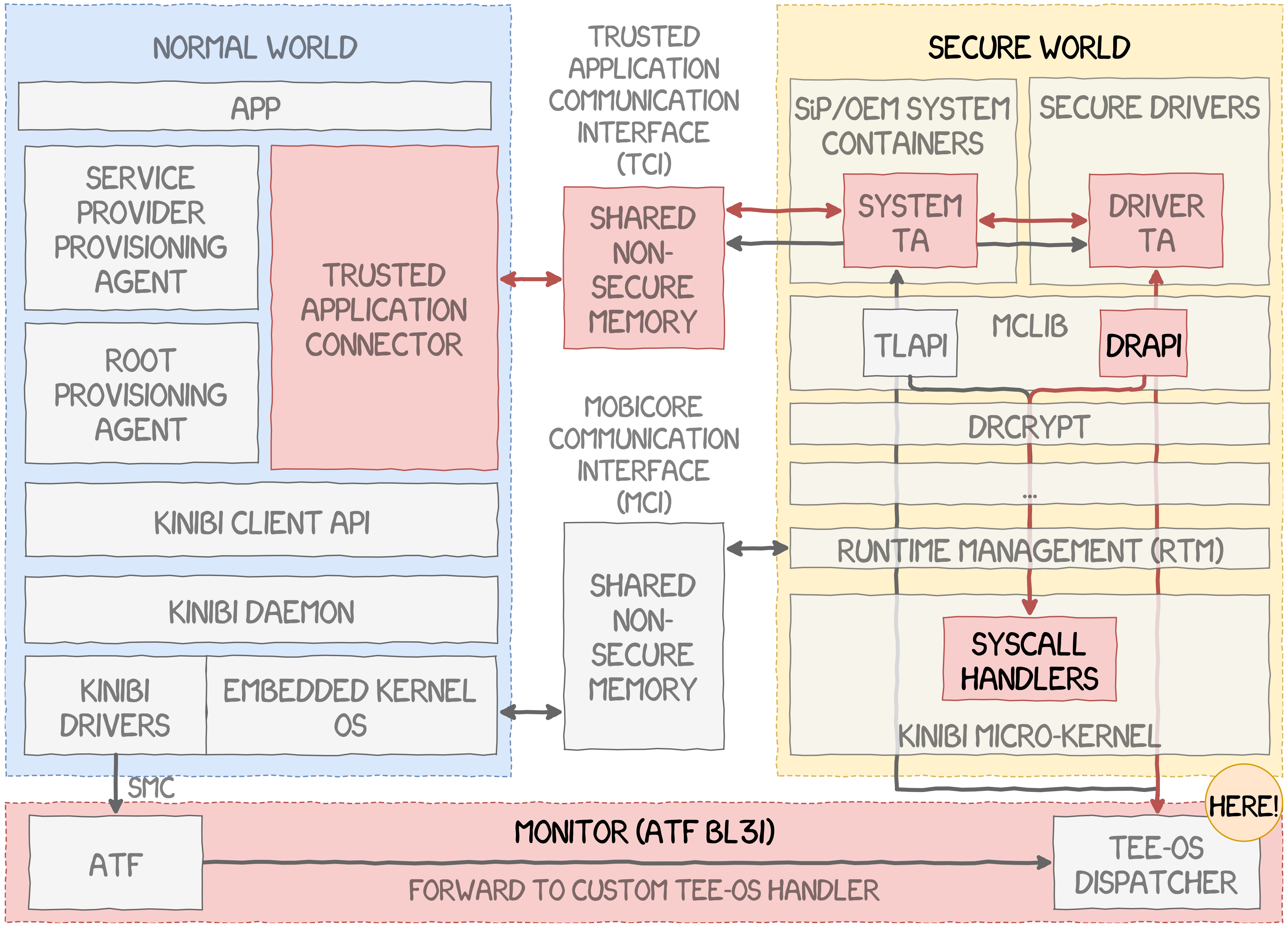
```
with Device(DEVICE_ID) as dev:
    with dev.buffer(TCI_BUFFER_SIZE) as tci:
        with open(TRUSTLET_FILE, "rb") as fd:
            buf = fd.read()

        with Trustlet(dev, tci, buf) as app:
            tci.seek(0)
            tci.write_dword(1)

            app.notify()
            app.wait_notification()

            tci.seek(0)
            print(tci.read_dword())
```

VULNERABILITY ANALYSIS & EXPLOITATION

OVERVIEW

- **Target:**
  - Samsung Galaxy S7 running Android 7.0
- **Main goal:**
  - Obtaining code execution in EL3
- **Prerequisites:**
  - Being part of the `radio` group
  - Being able to write files somewhere on the device

ATTACK PLAN

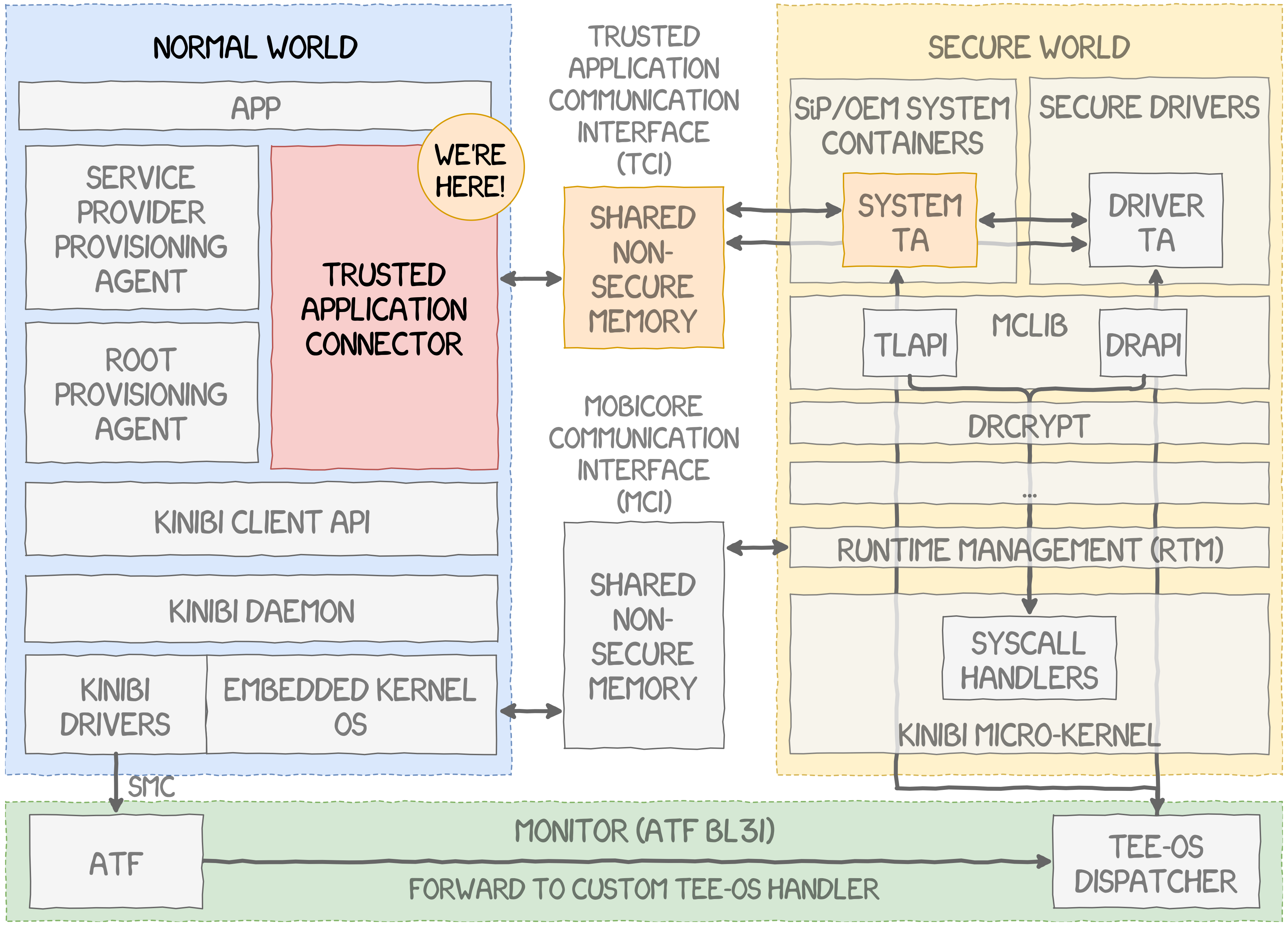

NORMAL WORLD
APP
SERVICE PROVIDER PROVISIONING AGENT
TRUSTED APPLICATION CONNECTOR
ROOT PROVISIONING AGENT
KINIBI CLIENT API
KINIBI DAEMON
KINIBI DRIVERS
EMBEDDED KERNEL OS
SMC
TRUSTED APPLICATION COMMUNICATION INTERFACE (TCI)
SHARED NON-SECURE MEMORY
MOBICORE COMMUNICATION INTERFACE (MCI)
SHARED NON-SECURE MEMORY
SECURE WORLD
SiP/OEM SYSTEM CONTAINERS
SECURE DRIVERS
SYSTEM TA
DRIVER TA
TLAPI
MCLIB
DRAPI
DRCRYPT
...
RUNTIME MANAGEMENT (RTM)
SYSCALL HANDLERS
KINIBI MICRO-KERNEL
HERE!
MONITOR (ATF BL31)
ATF
FORWARD TO CUSTOM TEE-OS HANDLER
TEE-OS DISPATCHER

SOFTWARE MITIGATIONS

Model	XN bit	Canary	ASLR	PIE
S6	✓	✗	✗	✗
S7	✓	✗	✗	✗
S8	✓	✗	✗	✗
S9	✓	✓	✗	✗

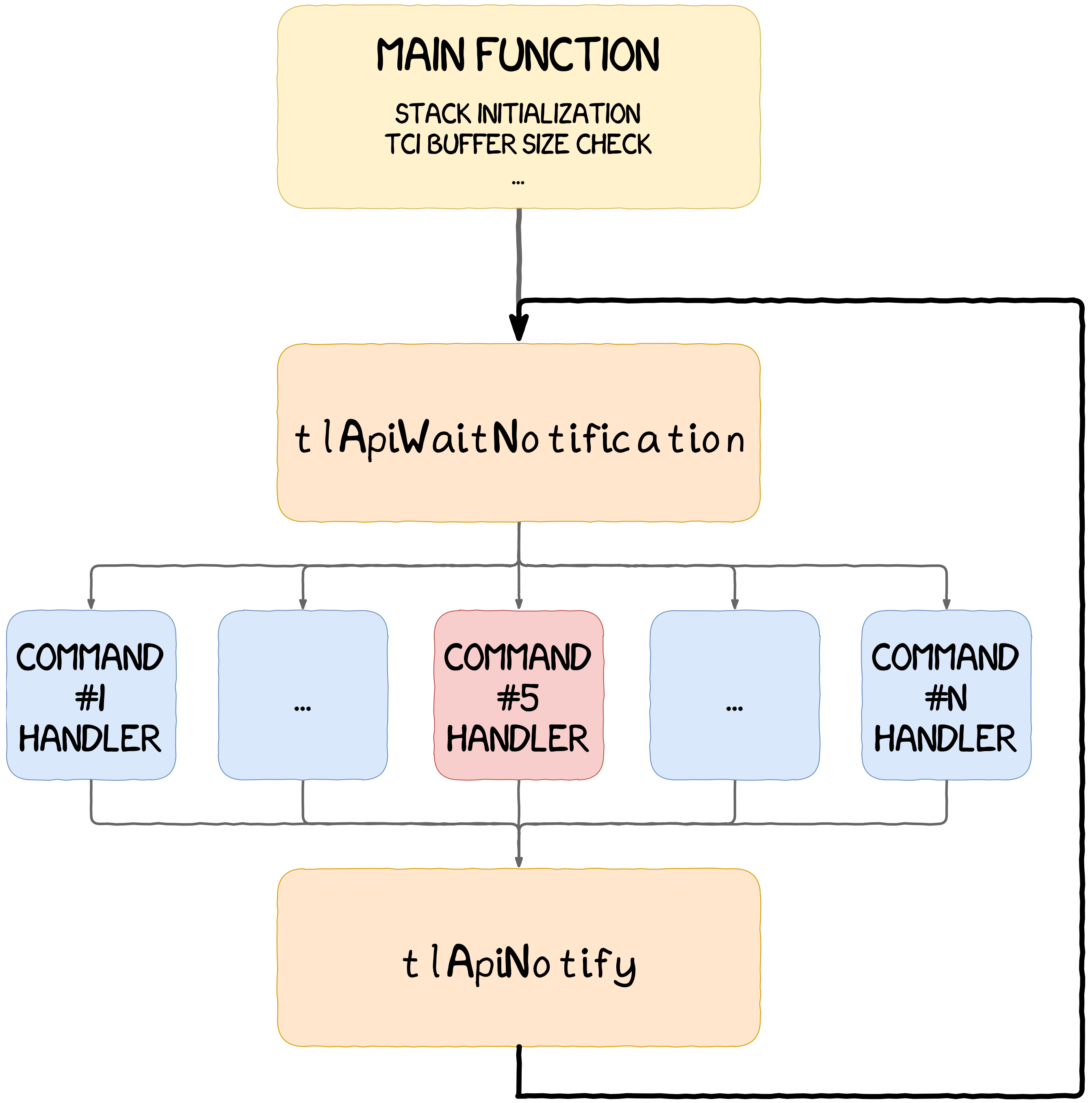
ATTACKING A TRUSTED APPLICATION

Overview



ATTACKING A TRUSTED APPLICATION

SEM Trustlet Vulnerability



Stack-based buffer overflow in the handler of the command ID #5

Call to `memcpy` at the beginning of the 5th command handler

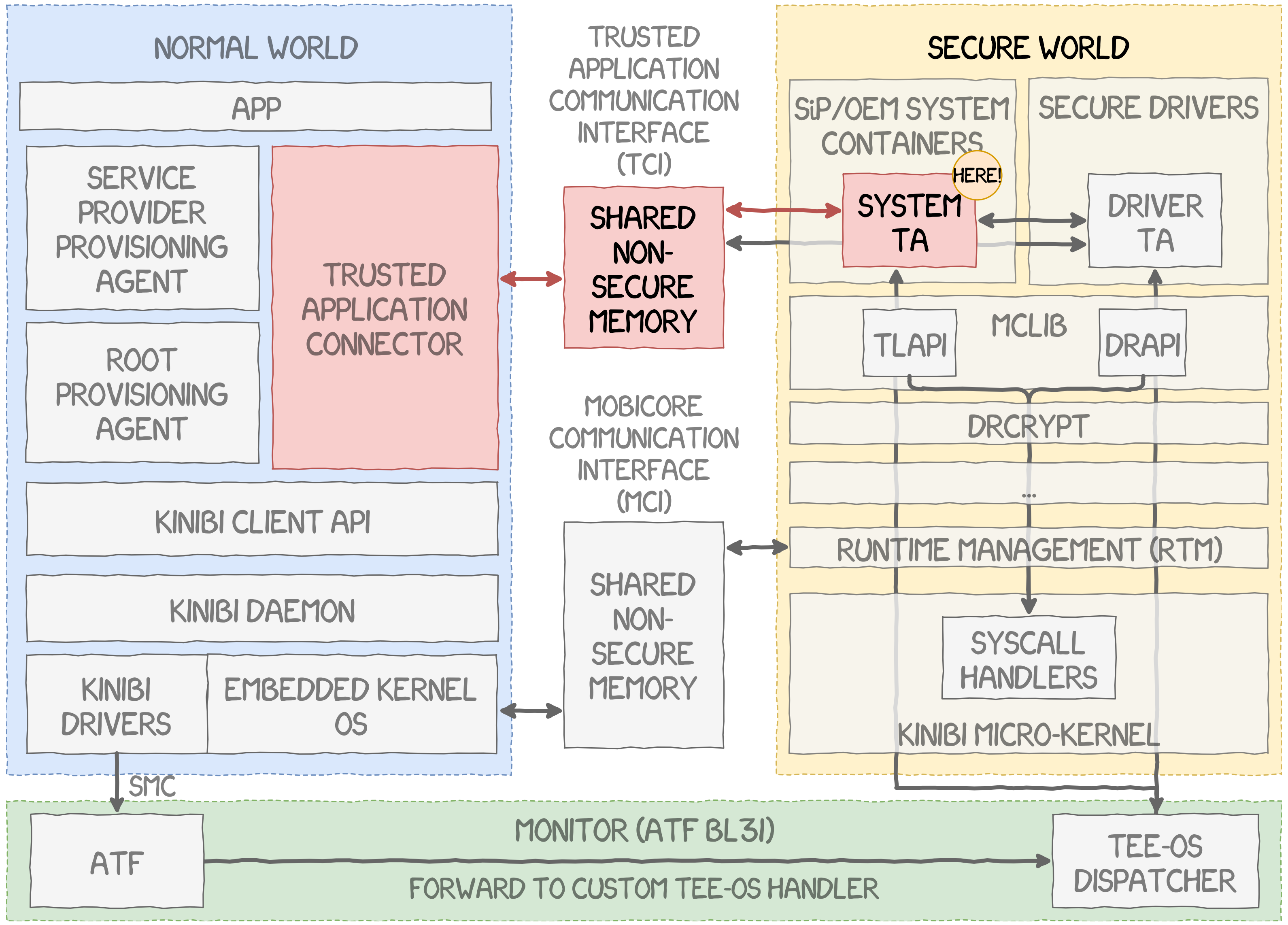
Before this call, the registers are set as follow:

- R0 = `SP+0x4F8-0xF0`, the destination buffer
- R1 = `tci_buffer + 0x8`, the source buffer
- R2 = `*(tci_buffer + 0x16808)`, the length of the buffer

```
.text:00020FB2    ADD.W    R1, R0, #0x16000
.text:00020FB6    MOV      R4, R1
.text:00020FB8    LDR.W    R2, [R1,#0x808]
.text:00020FBC    ADD.W    R1, R0, #8
.text:00020FC0    ADD.W    R0, SP, #0x4F8+var_F0
.text:00020FC4    BLX      memcpy_aligned
```

ATTACKING A TRUSTED APPLICATION

Exploitation Results



- Code execution in **S-EL0**
- **It is now possible to:**
  - Communicate with **Secure Drivers**
  - Make some syscalls (e.g. print characters, get system information, etc.)
- **Next target:** Secure Driver

ATTACKING A SECURE DRIVER

VALIDATOR Secure Driver Vulnerability

A vulnerability was found in the **VALIDATOR secure driver**

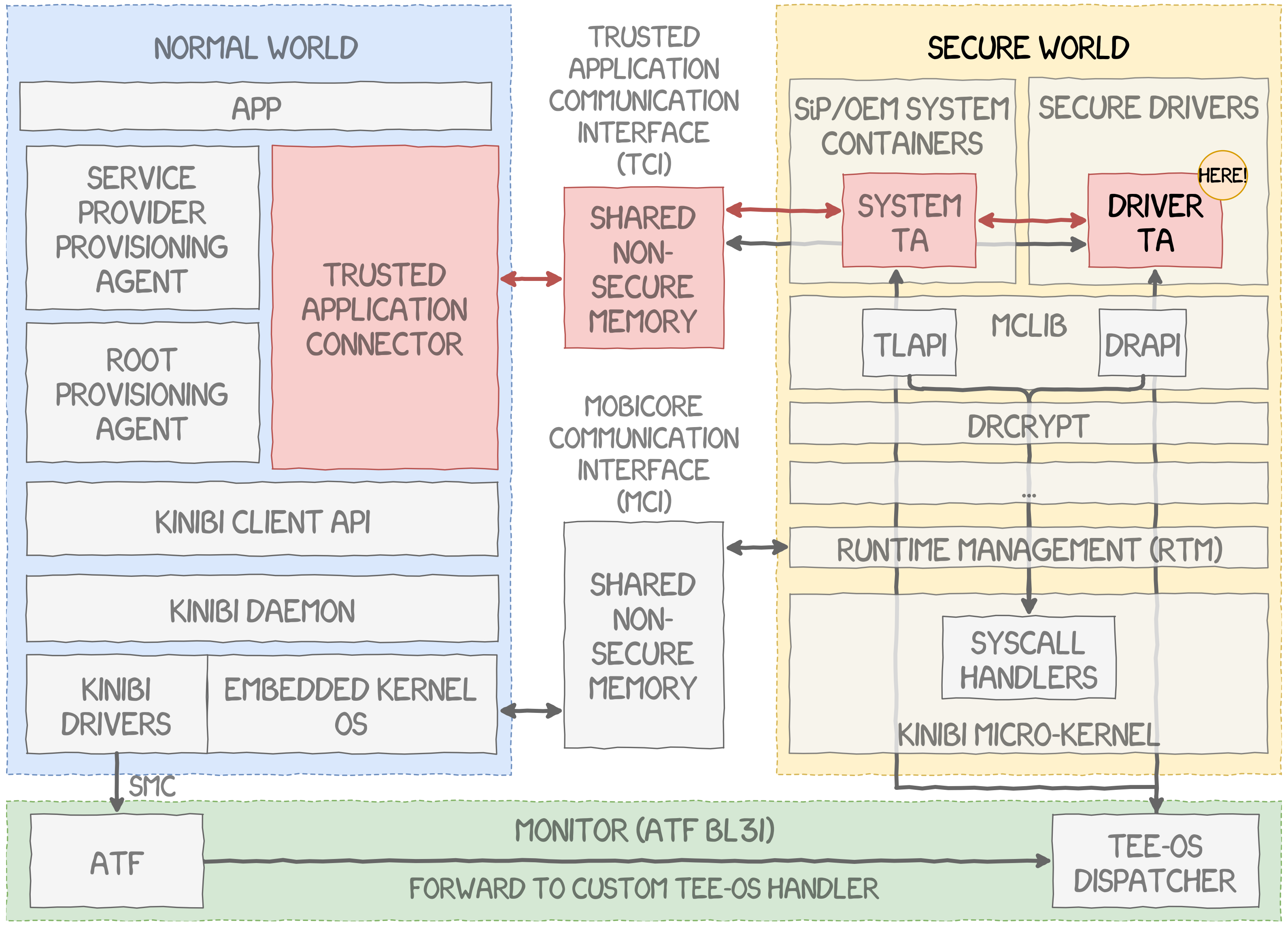
Stack-based buffer overflow in the handler of the command ID #15

Equivalent to the one found in the trustlet (i.e. `memcpy` in the stack and a user-controlled size)

```
.text:00001362                 MOVS            R2, #0x37 ; '7'
.text:00001364                 MOV             R1, R4
.text:00001366                 ADDS            R0, R6, #1
.text:00001368                 BLX             memcpy
```

ATTACKING A SECURE DRIVER

Exploitation Results



- Code execution in **S-EL0** but with higher privileges
- **It is now possible to:**
  - Communicate with the **RunTime Manager** (or RTM, an init-like process)
  - Access more syscalls (e.g. map physical memory, create threads, make SMCs, etc.)
- **Next target:** Trusted OS & Monitor

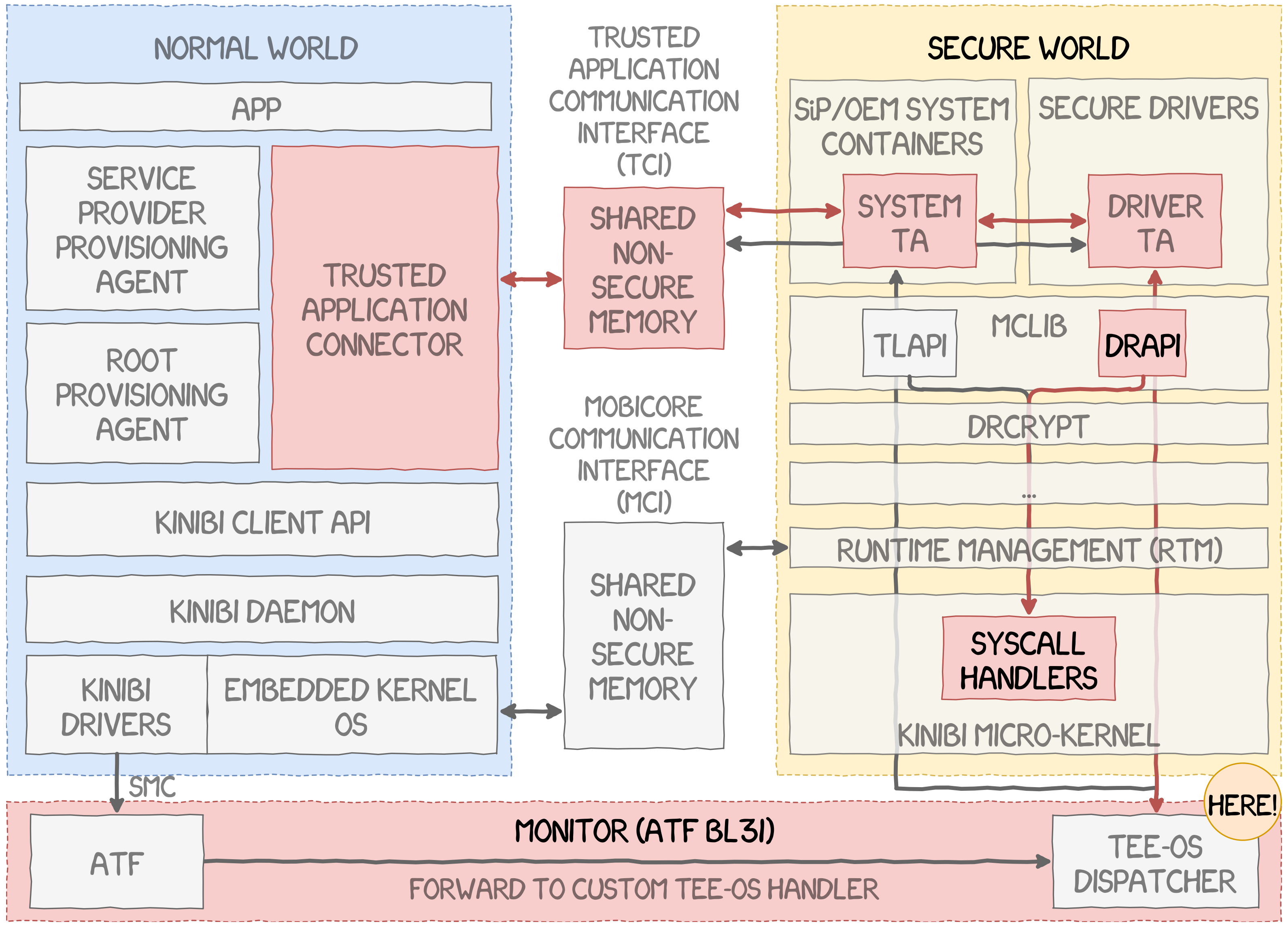
ATTACKING KINIBI AND THE MONITOR

Vulnerability Analysis

- **mmap:** secure and non-secure physical memory mapping syscall
- **Vulnerability**
  - Monitor mapped at `0x2022000`
  - Can be mapped using `mmap` to modify an SMC
  - Calling the hijacked SMC allows code execution in EL3
- **Patch**
  - Fixed in the newest versions by using a blacklist

ATTACKING KINIBI AND THE MONITOR

Exploitation Results



- Code execution in **EL3**
- Now possible to do anything we want!

POST-EXPLOITATION

TRUSTPWN FRAMEWORK

- Based on the previous vulnerabilities
- **Internals**
  - Uses the EL3 vulnerability to have arbitrary access to Kinibi
  - Adds a SVC and a drApi function to execute code in S-EL1 "natively"
    - SVCs and drApi functions are referenced in pointer arrays
- **Usage**
  - Read or write memory arbitrarily
  - Execute code in S-EL1 and EL3

DEMO

Finding the Master Key in the Monitor

FINDING THE MASTER KEY IN THE MONITOR

DrApi Reversing

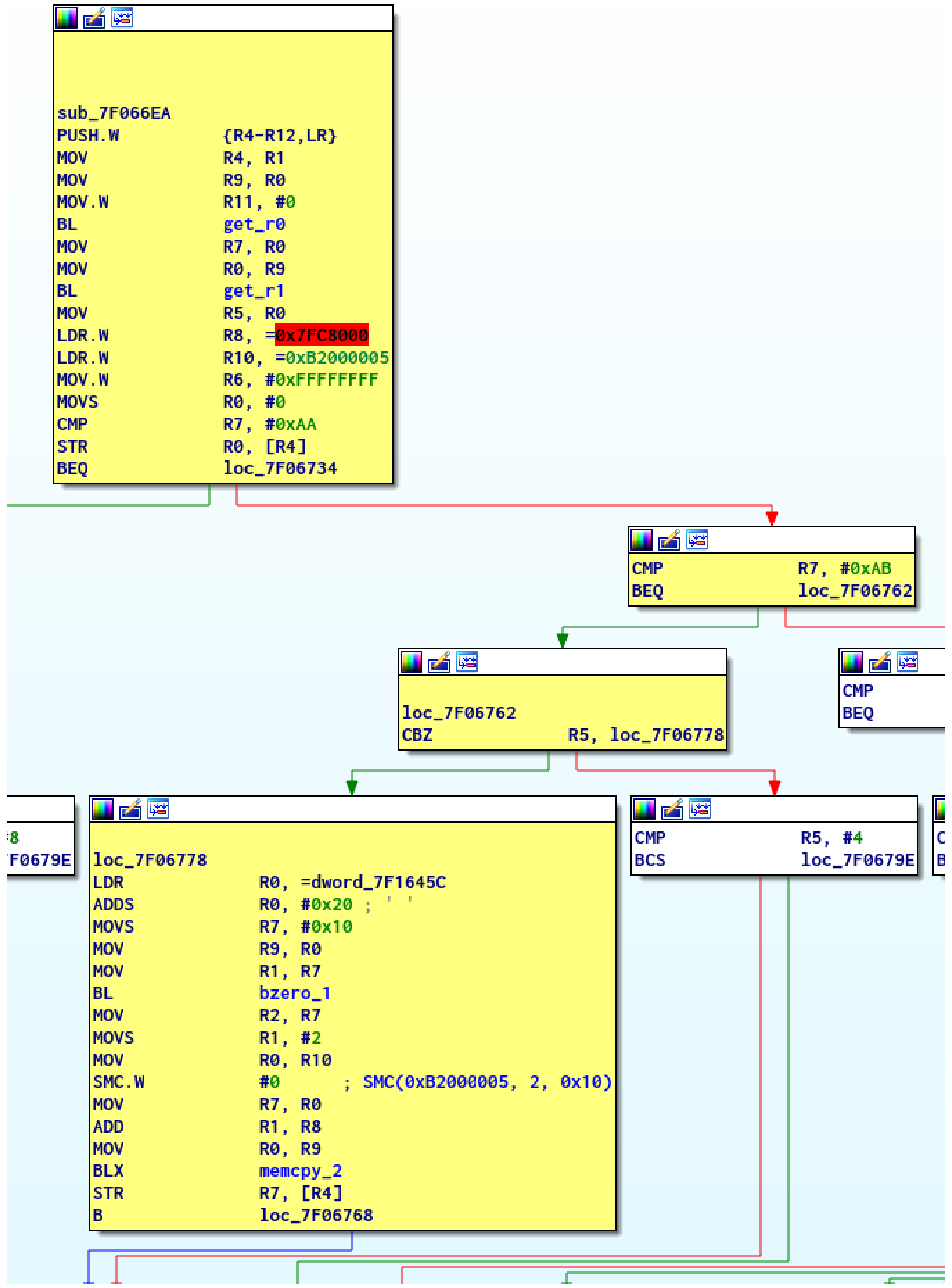
- Reversing the crypto-driver **drcrypto** (found embedded in Kinibi)

- **DrApi 0x1030**
  - Takes four possible command IDs (`0xAA, 0xAB, 0xAC, 0xAD`)
    - The interesting one is `0xAB`
  - Wrapper around SMC `0xB2000005`

- **SMC 0xB2000005**
  - SMC arguments:
    - **R0:** SMC ID
    - **R0:** command number (four possible values `[0-3]`)
    - **R1:** number of bytes to read
  - Reads 0x10 bytes of the master key at `0x101E4000`

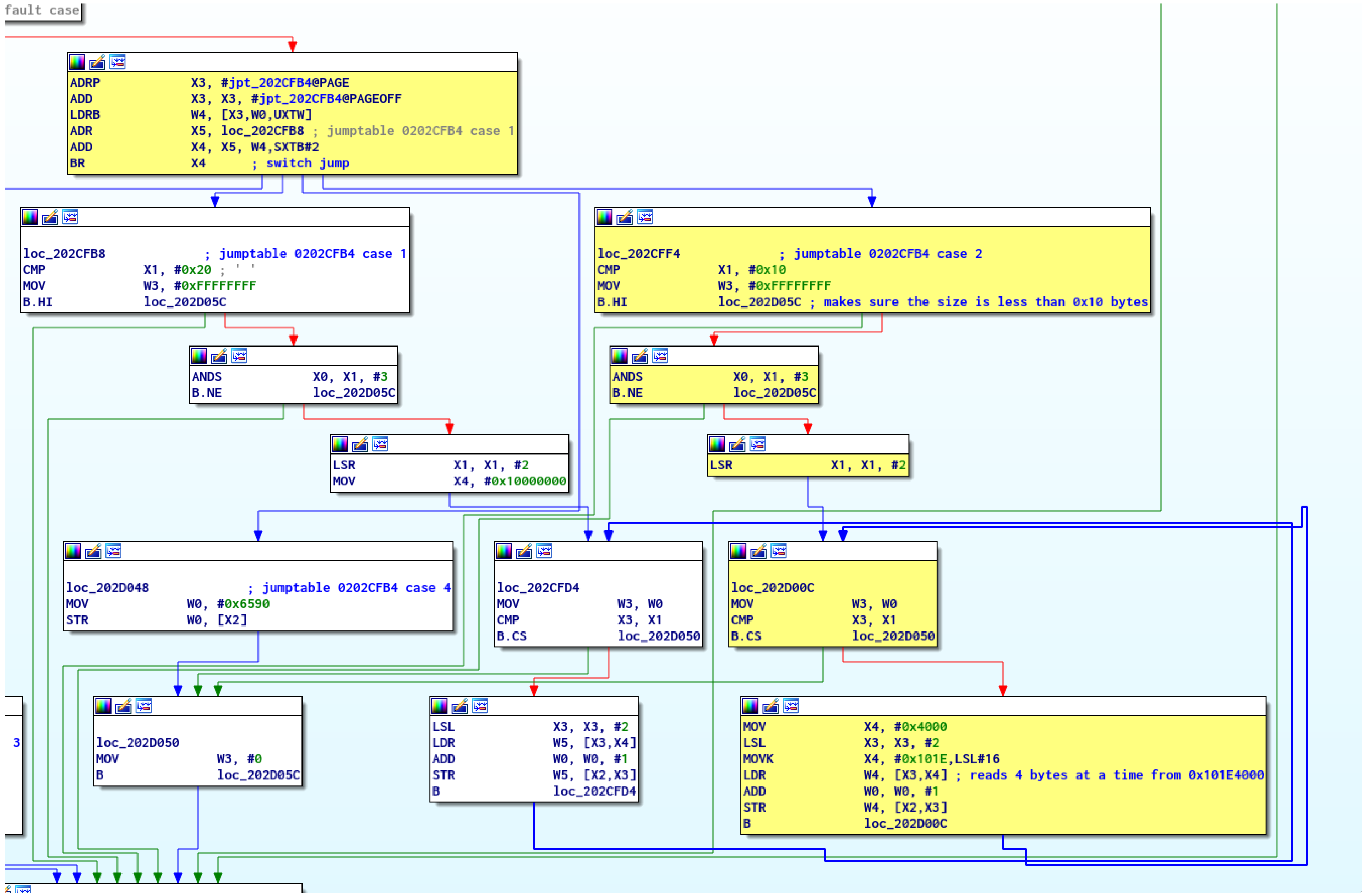
FINDING THE MASTER KEY IN THE MONITOR

DrApi Function



FINDING THE MASTER KEY IN THE MONITOR

SMC Function



DEMO

Bypassing Signature Checks

BYPASSING SIGNATURE CHECKS

Methodology

- Reversing RTM
- Finding the SHA-256 of the public key corresponding to the private key used to sign TAs and SDs
- Signature is verified using `tlApiSignatureVerify`
- Patch the checks and load your own TA or SD

FINDING THE MASTER KEY IN THE MONITOR

RTM Verifications

- **First check**

```
ROM:00006E62                 BL              tlApiSignatureVerify
ROM:00006E66                 LDR             R4, =0x40B00009
ROM:00006E68                 ADDS            R4, R4, #6
ROM:00006E6A                 CBNZ            R0, loc_6E7A
ROM:00006E6C                 LDRB.W          R0, [SP,#0xC0+var_44]
```

- **Second check**

```
ROM:000073E0                 BL              tlApiSignatureVerify
ROM:000073E4                 CBNZ            R0, loc_73FA
ROM:000073E6                 LDRB.W          R0, [SP,#0x1C0+var_48]
```

DEMO

Trusted-OS Instrumentation

TRUSTED-OS INSTRUMENTATION

Methodology

- Handles ARMv7 and Thumb
- Based on the **Undefined Instruction** exception
- **Undefined Instruction** handler is replaced by our own code
- Patch an instruction with the ARM undefined instruction `UDF 0xNNNN`
- When a breakpoint triggers the current context of the CPU is saved
  - Current context is saved
  - Overwritten instruction is executed

THANK YOU!