Launching feedback-driven fuzzing on TrustZone TEE

Andrey Akimov, security researcher
LEADING INFORMATION SECURITY SERVICES PROVIDER

870+ clients
1802 projects
16 years in information security
1300 vulnerabilities found in 2018

19 industries

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

75 research papers
100+ experts
165 talks at international conferences

Acknowledgments

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

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

**GlobalPlatform**

**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
• 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](https://github.com
  • [github: trustonic-tee-driver](https://github.com
• Old Qualcomm leak with Trustonic Kinibi SDK qcom_leaked_sources.zip
  • secure world headers
  • secure world static libraries
  • documentation
  • etc.
Trustonic Kinibi
Normal World and Secure World

Architecture

Normal-World
- Applications
  - Service Provider Provisioning Agent
  - Trusted Application Connector (TLC)
  - Root Provisioning Agent
  - <t-base Client API
  - <t-base daemon
  - kernel
  - <t-base driver

Secure-World
- Content Mgt TA
- SIP/OEM System Containers
- TSM-Operated Containers
- System TA
- Trusted Application (TA)
- <t-base Internal API
- Runtime Management
- Secure Drivers
- <t-base micro-Kernel

Developer’s view

Normal-World
- Applications
- GlobalPlatform Trusted Applications
- GlobalPlatform TEE Client API
- mcClient API
- Runtime Management
- Secure Drivers
- <t-base driver
- <t-base micro-Kernel

Secure-World
- Trusted Application Connector
- GlobalPlatform TEE Internal API
- tAPI
- Runtime Management
- Secure Drivers
- <t-base daemon

ARM TrustZone® enabled SoC
Normal World
Exploring Android file system
- Keymaster
  - access to key information
- Fingerprintd
  - biometrics
- Samsung Pay
- ...
• 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
• `/system/vendor/lib64/libMcClient.so` – trustlet communication
  • `mcOpenSession`
  • `mcMallocWsm`
  • `mcNotify`
  • ...

• `/system/vendor/lib64/libMcRegistry.so` – registry management
  • `mcRegistryStoreAuthToken`
  • `mcRegistryStoreSp`
  • ...

Trustonic Kinibi
Client API
• /system/vendor/bin/mcDriverDaemon
• Communicates through @mcdaemon socket
• SELinux
  • u:object_r:mobicoredaemon_exec:s0
• 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
• 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
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-------------------+
<table>
<thead>
<tr>
<th>G950FXXU3CRGH_G950FOXMX3CRGH_SER.zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>+---Firmware content-------------------</td>
</tr>
<tr>
<td>+-- AP_G950FXXU3CRGH_CL14023573_QB19093103_REV00_user_low_ship_meta.tar.md5</td>
</tr>
<tr>
<td>+-- BL_G950FXXU3CRGH_CL14023573_QB19093103_REV00_user_low_ship.tar.md5</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>+---------------------------------------</td>
</tr>
<tr>
<td>+---BL_G950FXXU3CRGH...---+</td>
</tr>
<tr>
<td>+-- cm.bin.lz4</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>+-- sboot.bin.lz4</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>+----------------------</td>
</tr>
<tr>
<td>+----------------------</td>
</tr>
</tbody>
</table>
```

• 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

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

```
<table>
<thead>
<tr>
<th>Name</th>
<th>Start</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP_BL2</td>
<td>0000000000000000</td>
<td>0000000000200000</td>
</tr>
<tr>
<td>AP_BL31_IMG</td>
<td>0000000000200000</td>
<td>00000000002A0000</td>
</tr>
<tr>
<td>AP_BL31_unpacker</td>
<td>00000000002A0000</td>
<td>00000000005A0000</td>
</tr>
<tr>
<td>AP_BL32</td>
<td>00000000005A0000</td>
<td>0000000001430000</td>
</tr>
<tr>
<td>AP_BL33</td>
<td>0000000001430000</td>
<td>0000000001C3110</td>
</tr>
</tbody>
</table>
```
• 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    | 07F24000| 07F24000|
RTM      | 07F36000| 07F36000|
DRCRYPTO | 07F49000| 07F49000|
TLPROXY  | 07F54000| 07F54000|
MCTL     | 07F54000| 07F56000|```
• Kinibi kernel – MTK
• Runtime manager – RTM
• Some trusted drivers – drcrypto, ...
• Some trusted applications – STH2, ...
• Internal API library - mclib
• Trusted applications - TA, CM system TA, SP TAs
• Reside in Android file system
• Identified by GUID
Trustonic Kinibi
Inter-world communication flow
• MobiCore Load Format – MCLF
  • [github: mcLoadFormat.h](https://github.com/mclf/mcLoadFormat.h)
    • IDA Pro loader
    • Ghidra loader
  • Signed binaries
  • 32-bit executables
  • Uninitialized fields
    • tciBuffer_ptr
    • tciBuffer_len
    • mclibEntry
    • ...
  • Internal API via mclib
- All external calls are through mclib entry field in MCLF header
- Easy to emulate such an isolated code
- Easy to wrap in fuzzing environment
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
• 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
• 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
• 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

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

```c
_TLAPI_EXTERN_C void tlApiLogPrintf(
    const char *fmt,
    ...);
```
• Dispatch function
  • tlApiLibEntry

  // tlrunc

typedef void (*tlApiEntry_t)(int num);

void (*tlApiLibEntry)(int num) __attribute__((weak));
void tlApiEntry(int num) __attribute__((noplt));
__attribute__((constructor)) void init()
{
  tlApiLibEntry = tlApiEntry;
}

  // tllib.c

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

- 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

```bash
rem ida_auto.bat

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

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

root@artik:/targets/07010000000000000000000000000000000000000# ./tlrun < test
mem1 = 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 ---
Fuzzing
Poexali!
• QEMU and AFL QEMU patches issues
  • toolchain
• AFL instrumentation issues
  • Study AFL thoroughly
<table>
<thead>
<tr>
<th>process timing</th>
<th>overall results</th>
</tr>
</thead>
<tbody>
<tr>
<td>run time: 0 days, 22 hrs, 30 min, 39 sec</td>
<td>cycles done: 715</td>
</tr>
<tr>
<td>last new path: 0 days, 7 hrs, 35 min, 16 sec</td>
<td>total paths: 490</td>
</tr>
<tr>
<td>last uniq path: 0 days, 1 hrs, 44 min, 44 sec</td>
<td>uniq crashes: 62</td>
</tr>
<tr>
<td>last uniq hang: none seen yet</td>
<td>uniq hangs: 0</td>
</tr>
<tr>
<td>cycle progress</td>
<td>map coverage</td>
</tr>
<tr>
<td>now processing: 483 (98.57%)</td>
<td>map density: 1.86% / 3.29%</td>
</tr>
<tr>
<td>paths timed out: 0 (0.00%)</td>
<td>count coverage: 2.42 bits/tuple</td>
</tr>
<tr>
<td>stage progress</td>
<td>findings in depth</td>
</tr>
<tr>
<td>now trying: splice 3</td>
<td>favored paths: 69 (14.08%)</td>
</tr>
<tr>
<td>stage execs: 26/96 (27.08%)</td>
<td>new edges on: 99 (26.20%)</td>
</tr>
<tr>
<td>total execs: 47.8M</td>
<td>total crashes: 73.9K (62 unique)</td>
</tr>
<tr>
<td>exec speed: 663.1/sec</td>
<td>total tmouts: 127 (22 unique)</td>
</tr>
<tr>
<td>fuzzing strategy yields</td>
<td>path geometry</td>
</tr>
<tr>
<td>bit flips: 67/1.50M, 25/1.50M, 14/1.50M</td>
<td>levels: 22</td>
</tr>
<tr>
<td>byte flips: 4/187k, 1/35.8k, 2/35.5k</td>
<td>pending: 0</td>
</tr>
<tr>
<td>arithmetics: 39/2.01M, 4/1.12M, 2/537k</td>
<td>pend fav: 0</td>
</tr>
<tr>
<td>known ints: 21/171k, 9/764k, 34/1.32M</td>
<td>own finds: 489</td>
</tr>
<tr>
<td>dictionary: 0/0, 0/0, 0/55.8k</td>
<td>imported: n/a</td>
</tr>
<tr>
<td>havoc: 232/16.1M, 97/20.9M</td>
<td>stability: 100.00%</td>
</tr>
<tr>
<td>trim: 41.48%/57.5k, 80.27%</td>
<td>[cpu008: 24%]</td>
</tr>
</tbody>
</table>
23 trustlets – 477 crashes

afl-cmin – 225 unique cases
Crash analysis
Crash analysis

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

```bash
# 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
```

```python
# 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")
```
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

```plaintext

```
Results

- [https://security.samsungmobile.com/securityUpdate.smsb](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.
- Aleksandr Ruiz: SVE-2018-13326
- Gruskovnjak Jordan: SVE-2019-13921
- Julian Jackson: SVE-2019-14031
- Artyom Skrob 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.
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 CO
00000010: 01 0F 00 00 00 00 FF CO | 01 0F 00 00 03 03 10 10
00000020: 00 00 00 03 83 00 00 77 | 10 AC 0A 00 00 00 00 00
00000030: 00 00 00 6C 6C 6C 6C 6C | 6C 6C 6C 6C 6C 6C 6C 6C
00000040: 6C 6C 6C 6C 6C 6C 6C 6C | 6C 6C 6C 6C 6C 6C 6C 6C
00000050: 6C 6C 6C 6C 6C 6C 6C 6C | 6C 6C 6C 6C 6C 6C 6C 6C
00000060: 6C 6C 6C 6C 6C 6C 6C 6C | 6C 6C 6C 6C 6C 6C 6C 6C
00000070: 6C 6C 6C 6C 6C 6C 6C 6C | 6C 6C 6C 6C 6C 6C 6C 6C
00000080: 6C 6C 6C 6C 6C 6C 6C 6C | 6C 6C 6C 6C 6C 6C 6C 6C
00000090: 6C 6C 6C 6C 6C 6C 6C 6C | 6C 6C 6C 6C 6C 6C 6C 6C
```

Root cause
• 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)
• **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](#)
• 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
• 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)

<table>
<thead>
<tr>
<th>ROP gadget</th>
<th>ROP gadget</th>
<th>JOP gadget</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDR R2, [R1]</td>
<td>MOV R0, R4</td>
<td>ADDS R7, R7, #1</td>
</tr>
<tr>
<td>STRB.W R0, [R2],#1</td>
<td>POP {R3-R7,PC}</td>
<td>ORR.W R4, R4, #0x200</td>
</tr>
<tr>
<td>STR R2, [R1]</td>
<td>BLX LR</td>
<td>BLX R1</td>
</tr>
</tbody>
</table>
• ROPgadget --binary tlrun --thumb --range 0x1000-0xbeb44

• grep -E "; b.+ r[0-9]+$"
• 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

\[ \text{op } Rn!, \{\text{reglist}\} \]

where:

- \( \text{op} \) is either:
  - \( \text{LDMIA} \): Load multiple, increment after
  - \( \text{STMIA} \): Store multiple, increment after

- \( Rn \) is the register containing the base address. \( Rn \) must be in the range \( r0-r7 \).

- \( \text{reglist} \) is a comma-separated list of low registers or low-register ranges.

• ROPgadget --binary tlrun --thumb --range 0x1000-0xbeb44
• grep -E "; b.+ r[0-9]+$"
• grep -E "ldm.."
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
• 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
• 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
• Reverse-engineering Samsung Exynos 9820 bootloader and TZ
Thanks for your attention!

Andrey Akimov,
security researcher

tg: @e13fter