riscure

NULLCON
INTERNATIONAL SECURITY CONFERENCE

Fuzzing embedded (trusted) operating systems

Martijn Bogaard
Senior Security Analyst

martijn@riscure.com
@jmartijnb

Today's agenda

- Trusted Execution Environments?!?
- Why is fuzzing TEEs so difficult? (Is it...?)
- Fuzzing OP-TEE syscalls

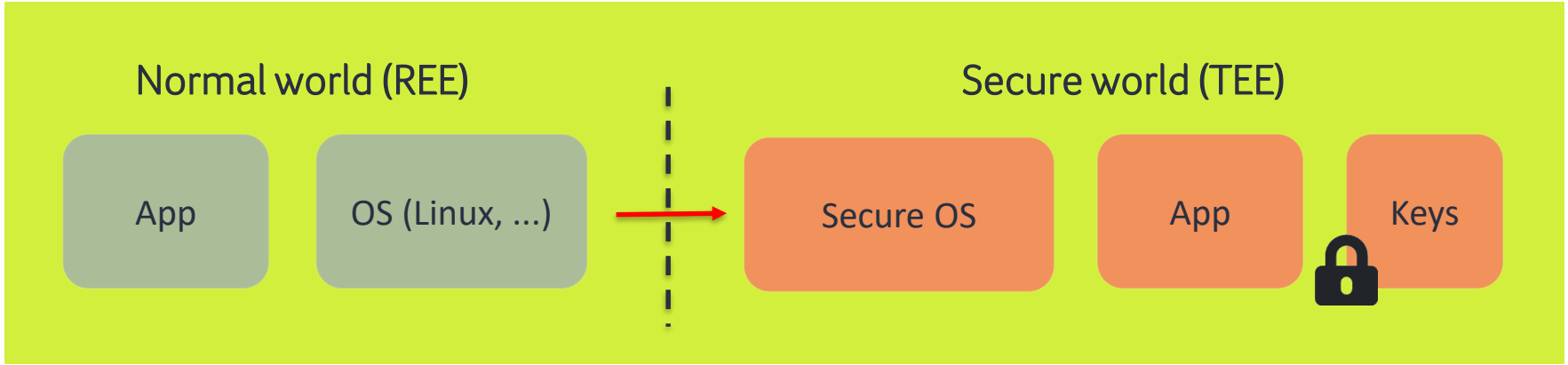
Research motivation

- Trusted Execution Environments become widespread
  - Mandatory in Android 6+ (conditionally)
  - Starting to pick up in other markets e.g. automotive

- Highly privileged component in modern chips
  - Way more than Android/Linux
  - Strictly isolated from the rest

- Often controls access to device's most secret crypto keys
  - KeyMaster, GateKeeper, DRM, Mobile Banking

Concept



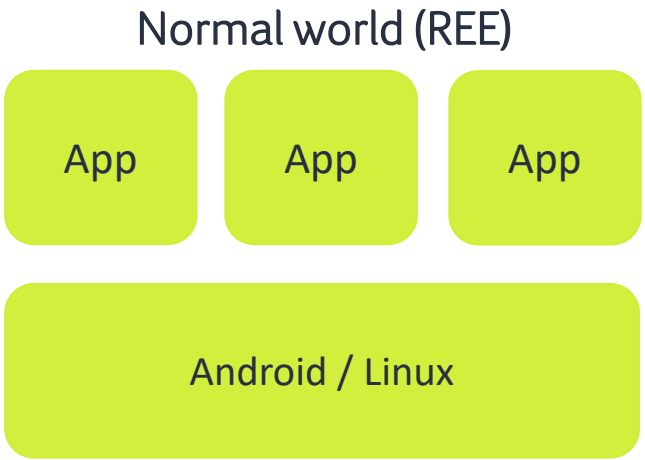
TEE Technology

- Arm TrustZone
- Intel SGX
- MIPS Virtualization
- RISC-V Keystone

TEE Technology

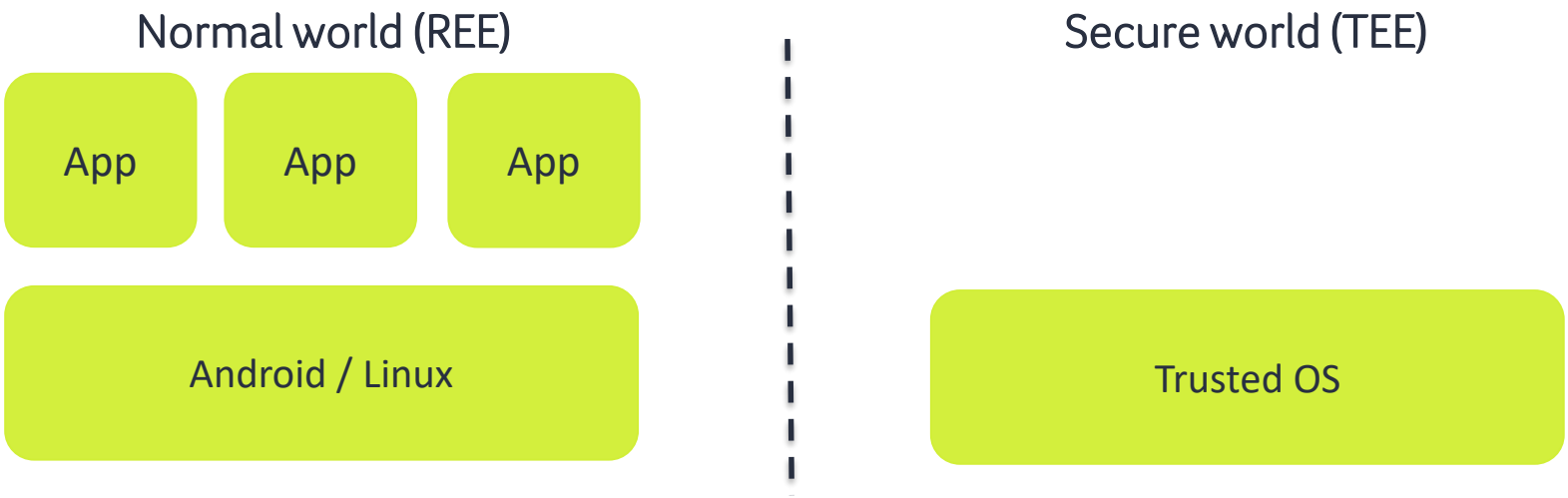
- **Arm TrustZone**
- Intel SGX
- MIPS Virtualization
- RISC-V Keystone

Trusted Execution Environment

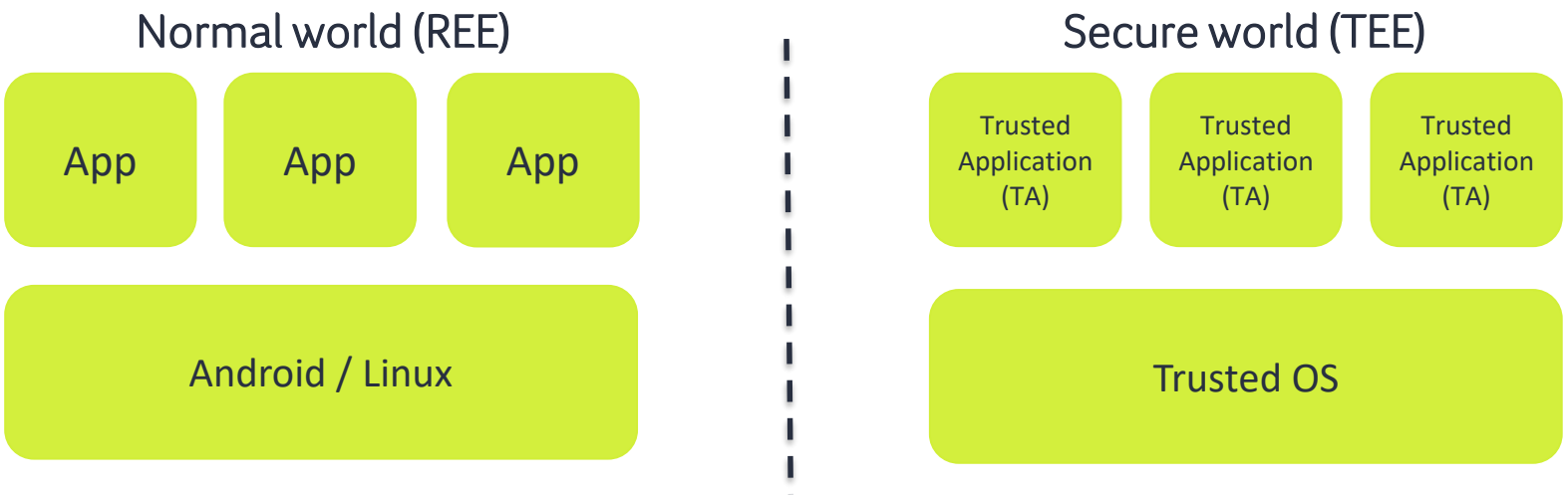


Trusted Execution Environment

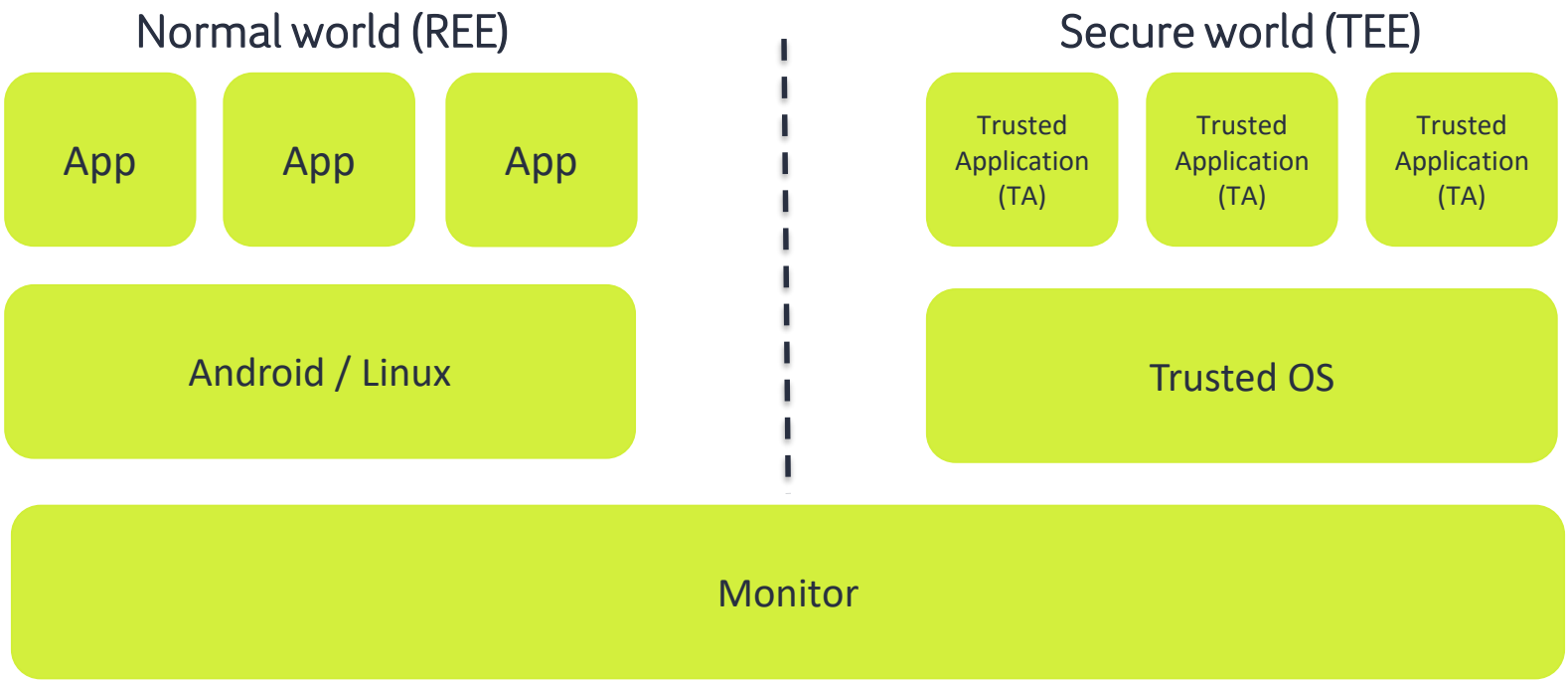
Trusted Execution Environment



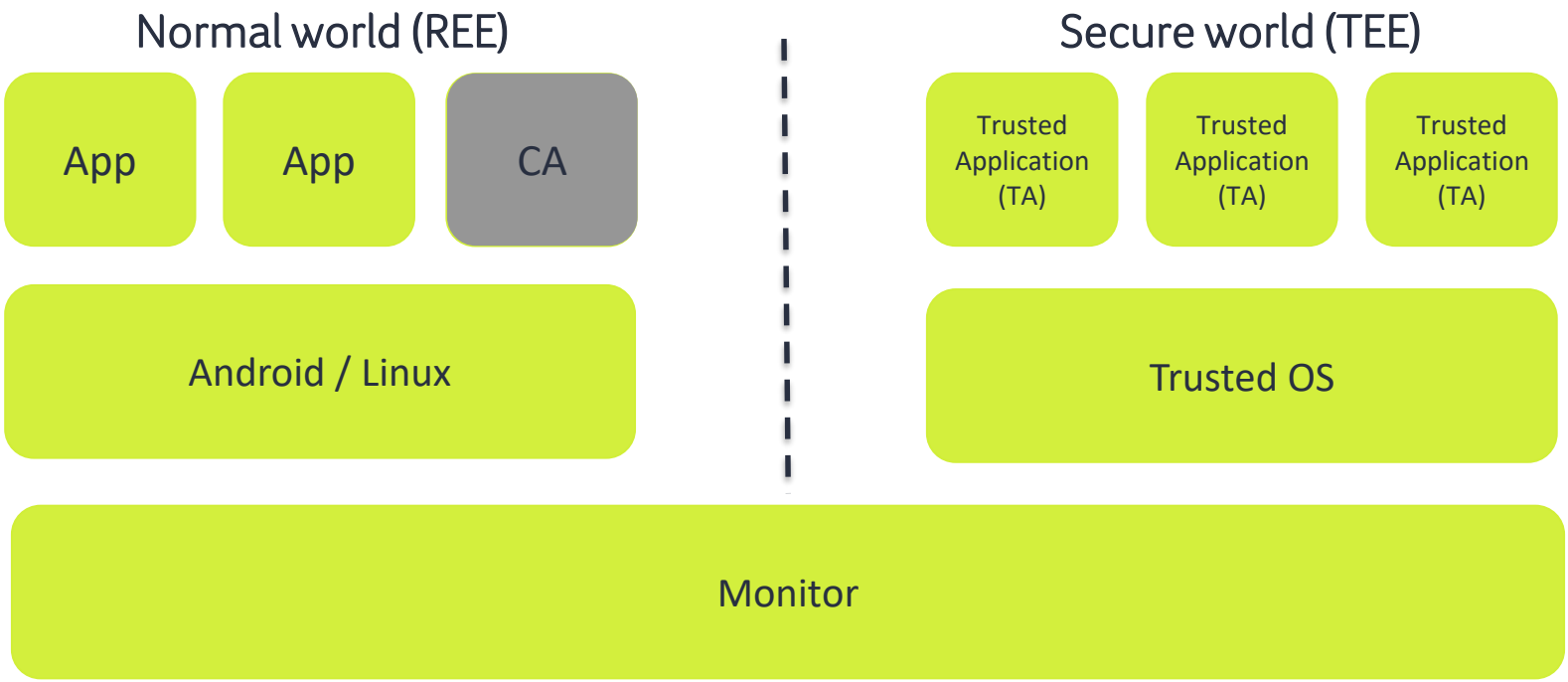
Trusted Execution Environment



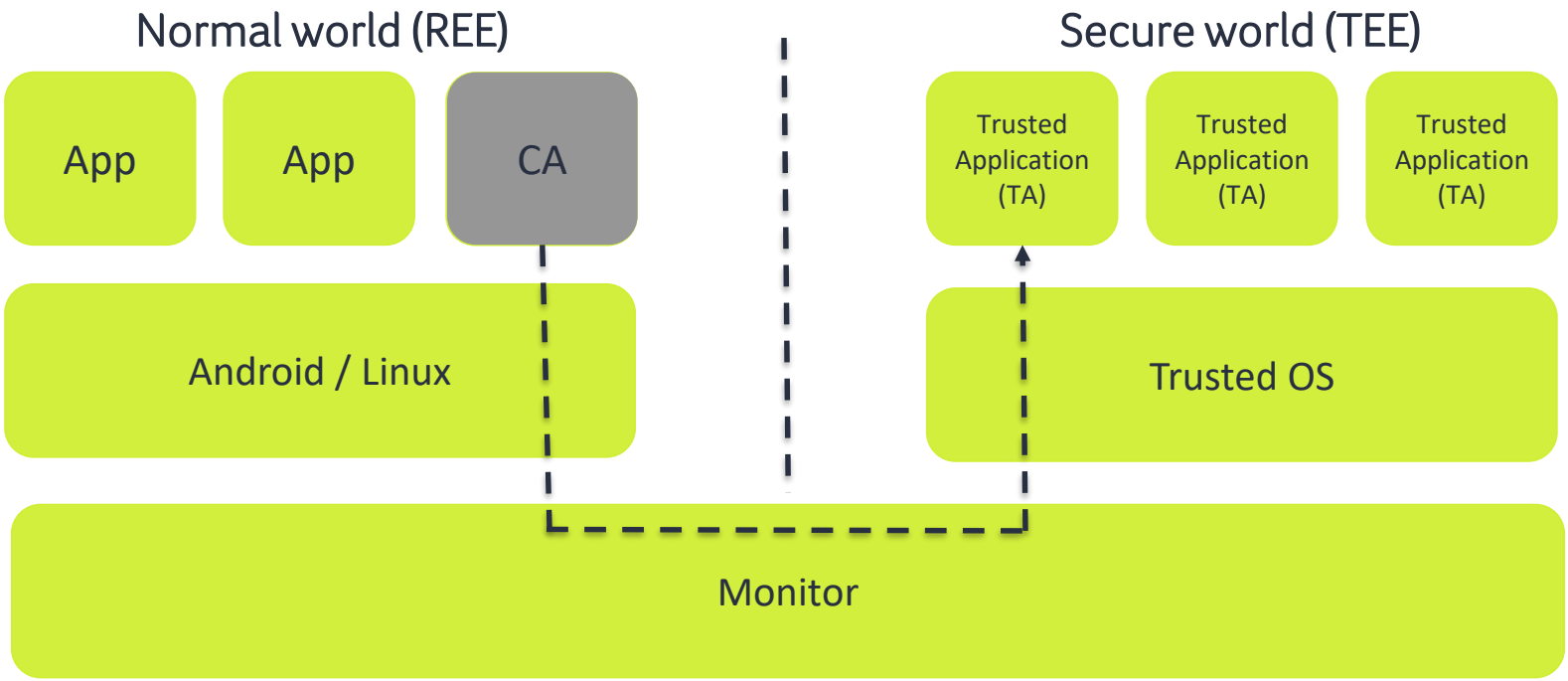
Trusted Execution Environment



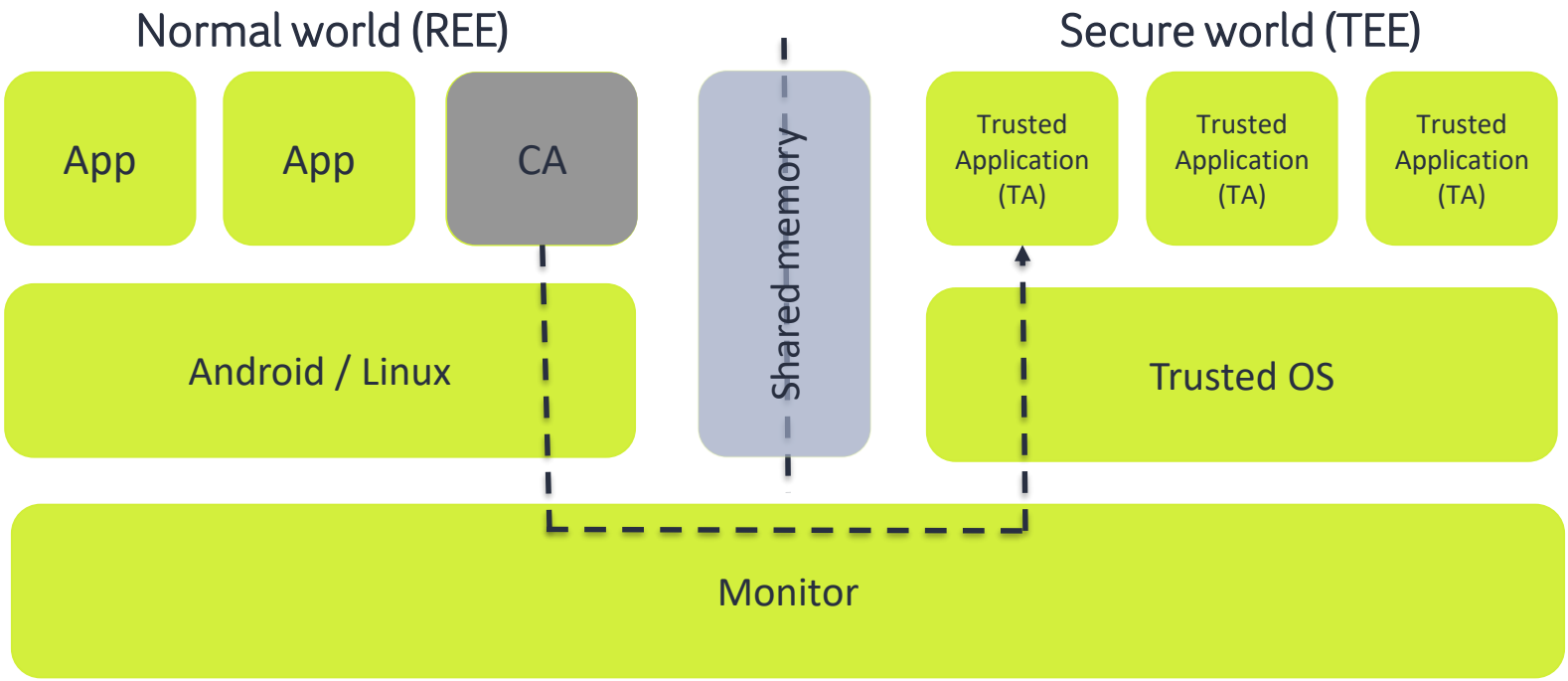
Trusted Execution Environment



Trusted Execution Environment



Trusted Execution Environment

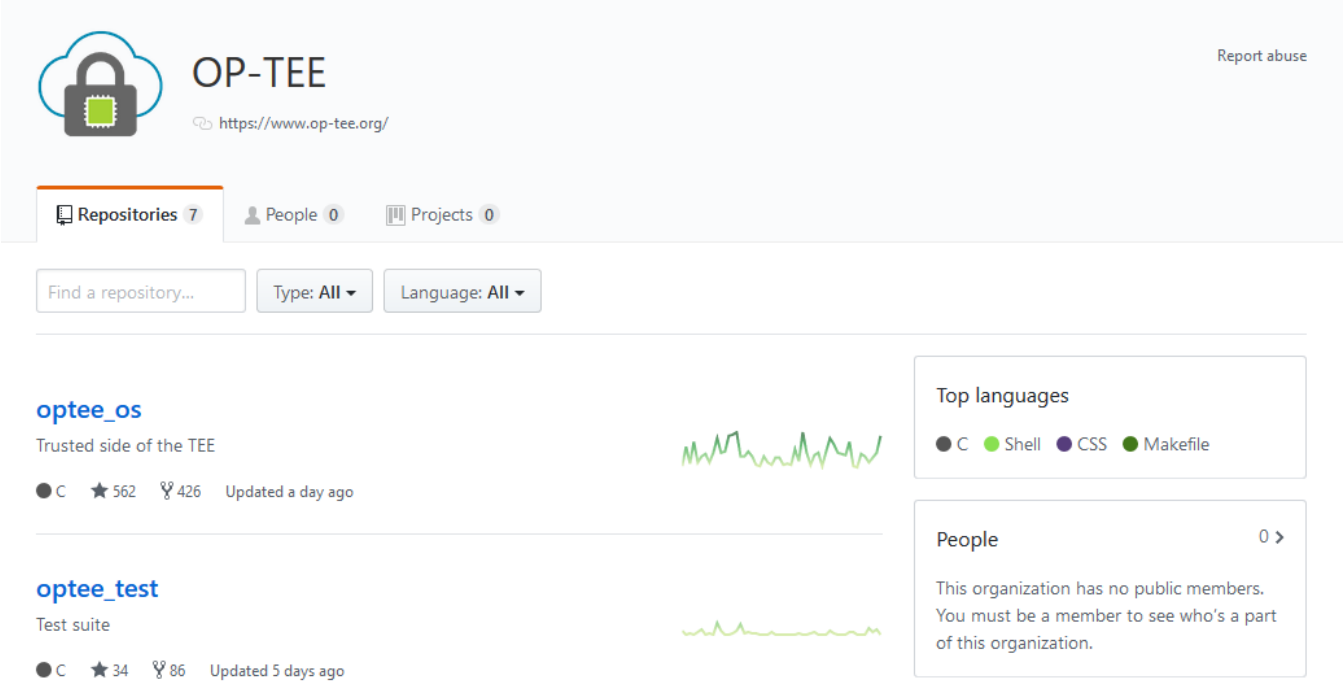


TrustZone based TEEs

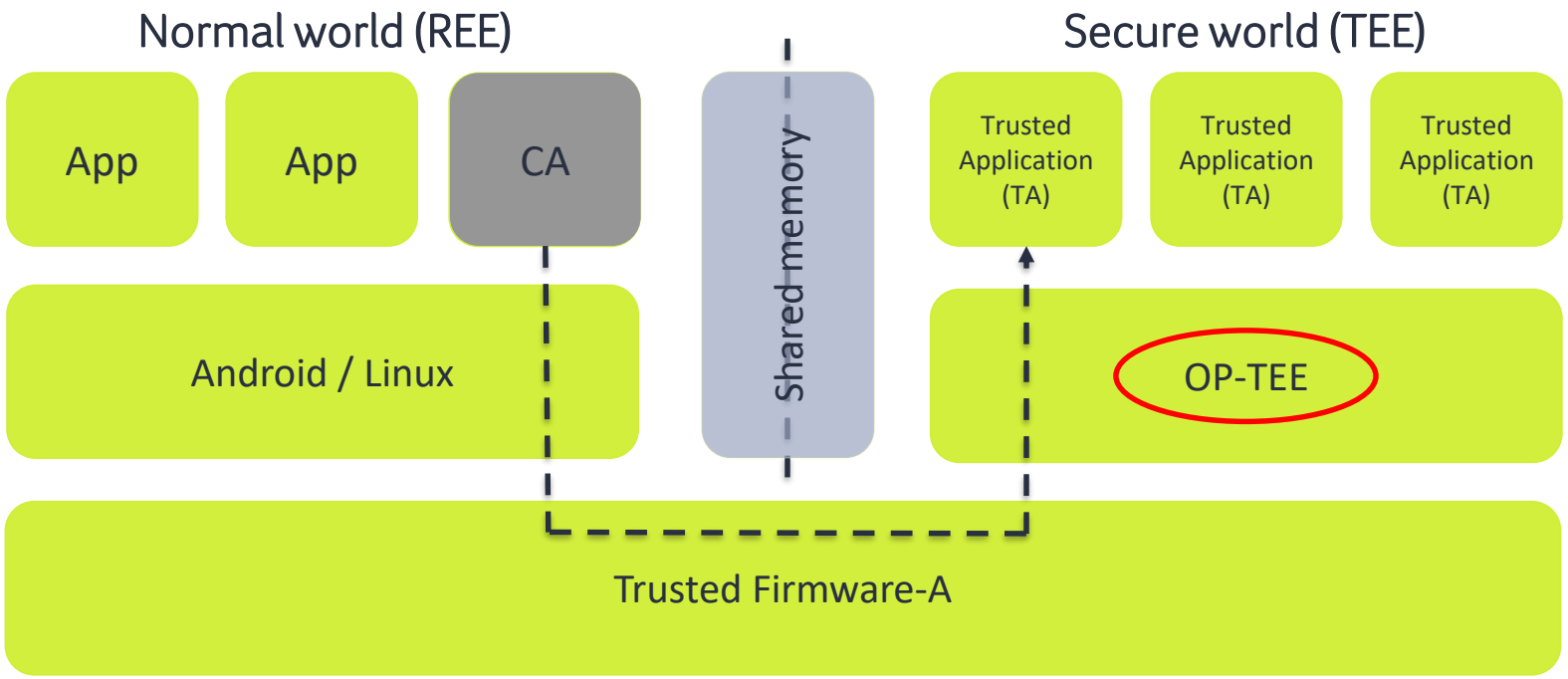
- Proprietary solutions
- Open-source
  - OP-TEE
  - Trusty

OP-TEE

- Developed by Linaro
- (Only?) widely deployed open-source TEE OS



Trusted Execution Environment



Trusted Applications

- Often written by chip vendor or OEM in C
- Global Platform API for compatibility between TEEs
  - Specifies exported symbols + TA API

→ Not like any normal world application!

Example: AES in TA

```
TEE_Result TA_InvokeCommandEntryPoint(void* sess_ctx, uint32_t cmd_id,
                                      uint32_t param_types, TEE_Param params[4])
{
    if (cmd_id == CMD_AES_ENCRYPT)  {
        [...]

        TEE_AllocateOperation(&op_handle, TEE_ALG_AES_CBC_NOPAD, TEE_MODE_ENCRYPT, AES128_KEY_SIZE);
        TEE_AllocateTransientObject(TEE_TYPE_AES, AES128_KEY_SIZE, &key_handle);
        TEE_InitRefAttribute(&attr, TEE_ATTR_SECRET_VALUE, key, AES128_KEY_BYTE_SIZE);
        TEE_PopulateTransientObject(key_handle, &attr, 1);
        TEE_SetOperationKey(op_handle, key_handle);
        TEE_CipherInit(op_handle, iv, iv_sz);
        TEE_CipherUpdate(op_handle,  buf_in, buf_in_len, buf_out, &buf_out_len);
    }
    else {
        return TEE_ERROR_BAD_PARAMETERS;
    }
}
```

Let’s go one layer deeper….

Example: AES in OP-TEE

Example: AES in OP-TEE

```
uint32_t obj_handle1;
uint32_t obj_handle2;

uint32_t state_handle;

syscall_cryp_obj_alloc(0xa0000010, 0x80, &obj_handle1);
```

Example: AES in OP-TEE

```
uint32_t obj_handle1;
uint32_t obj_handle2;

uint32_t state_handle;

syscall_cryp_obj_alloc(0xa0000010, 0x80, &obj_handle1);
syscall_cryp_state_alloc(0x10000110, 0x0, obj_handle1, 0x0, &state_handle);
```

Example: AES in OP-TEE

```
uint32_t obj_handle1;
uint32_t obj_handle2;

uint32_t state_handle;

syscall_cryp_obj_alloc(0xa0000010, 0x80, &obj_handle1);
syscall_cryp_state_alloc(0x10000110, 0x0, obj_handle1, 0x0, &state_handle);
syscall_cryp_obj_alloc(0xa0000010, 0x80, &obj_handle2);
```

Example: AES in OP-TEE

```
uint32_t obj_handle1;
uint32_t obj_handle2;

uint32_t state_handle;

syscall_cryp_obj_alloc(0xa0000010, 0x80, &obj_handle1);
syscall_cryp_state_alloc(0x10000110, 0x0, obj_handle1, 0x0, &state_handle);
syscall_cryp_obj_alloc(0xa0000010, 0x80, &obj_handle2);
syscall_cryp_obj_populate(obj_handle2, {c0000000, buf_key, 0x10}, 0x1);
```

Example: AES in OP-TEE

```
uint32_t obj_handle1;
uint32_t obj_handle2;

uint32_t state_handle;

syscall_cryp_obj_alloc(0xa0000010, 0x80, &obj_handle1);
syscall_cryp_state_alloc(0x10000110, 0x0, obj_handle1, 0x0, &state_handle);
syscall_cryp_obj_alloc(0xa0000010, 0x80, &obj_handle2);
syscall_cryp_obj_populate(obj_handle2, {c0000000, buf_key, 0x10}, 0x1);
syscall_cryp_obj_reset(obj_handle1);
```

Example: AES in OP-TEE

```
uint32_t obj_handle1;
uint32_t obj_handle2;

uint32_t state_handle;

syscall_cryp_obj_alloc(0xa0000010, 0x80, &obj_handle1);
syscall_cryp_state_alloc(0x10000110, 0x0, obj_handle1, 0x0, &state_handle);
syscall_cryp_obj_alloc(0xa0000010, 0x80, &obj_handle2);
syscall_cryp_obj_populate(obj_handle2, {c0000000, buf_key, 0x10}, 0x1);
syscall_cryp_obj_reset(obj_handle1);
syscall_cryp_obj_copy(obj_handle1, obj_handle2));
```

Example: AES in OP-TEE

```
uint32_t obj_handle1;
uint32_t obj_handle2;

uint32_t state_handle;

syscall_cryp_obj_alloc(0xa0000010, 0x80, &obj_handle1);
syscall_cryp_state_alloc(0x10000110, 0x0, obj_handle1, 0x0, &state_handle);
syscall_cryp_obj_alloc(0xa0000010, 0x80, &obj_handle2);
syscall_cryp_obj_populate(obj_handle2, {c0000000, buf_key, 0x10}, 0x1);
syscall_cryp_obj_reset(obj_handle1);
syscall_cryp_obj_copy(obj_handle1, obj_handle2));
syscall_cipher_init(state_handle, iv, 0x10);
```

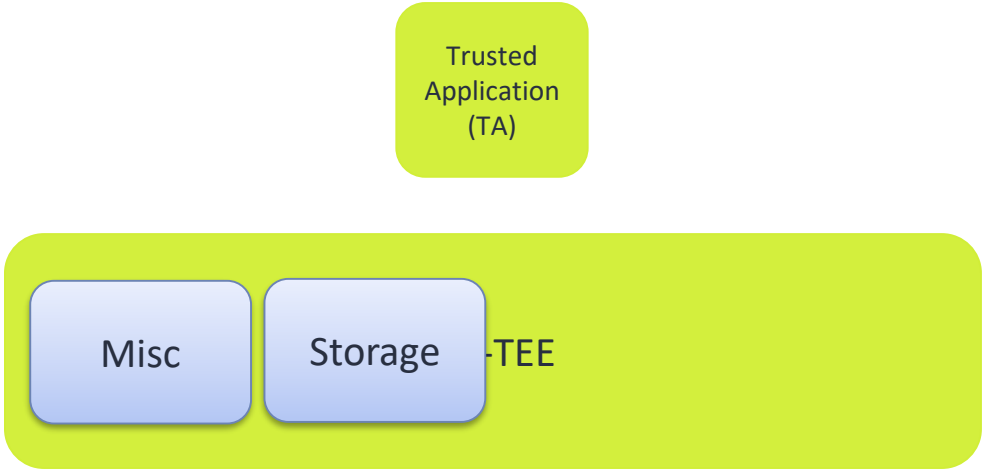
Example: AES in OP-TEE

```
uint32_t obj_handle1;
uint32_t obj_handle2;

uint32_t state_handle;

syscall_cryp_obj_alloc(0xa0000010, 0x80, &obj_handle1);
syscall_cryp_state_alloc(0x10000110, 0x0, obj_handle1, 0x0, &state_handle);
syscall_cryp_obj_alloc(0xa0000010, 0x80, &obj_handle2);
syscall_cryp_obj_populate(obj_handle2, {c0000000, buf_key, 0x10}, 0x1);
syscall_cryp_obj_reset(obj_handle1);
syscall_cryp_obj_copy(obj_handle1, obj_handle2));
syscall_cipher_init(state_handle, iv, 0x10);
syscall_cipher_update(state_handle, "Hello Nullcon!!!", 0x10, buf_out ,&buf_out_len);
```

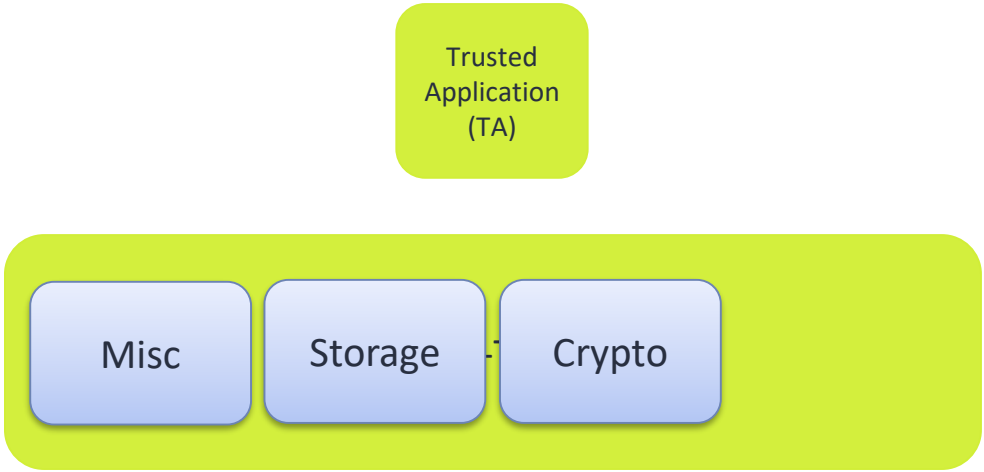
OP-TEE syscalls

OP-TEE syscalls

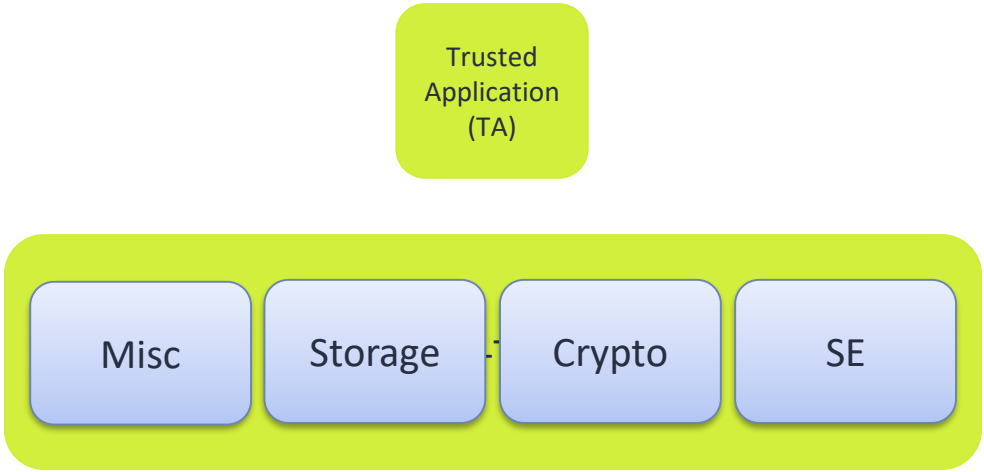
OP-TEE syscalls



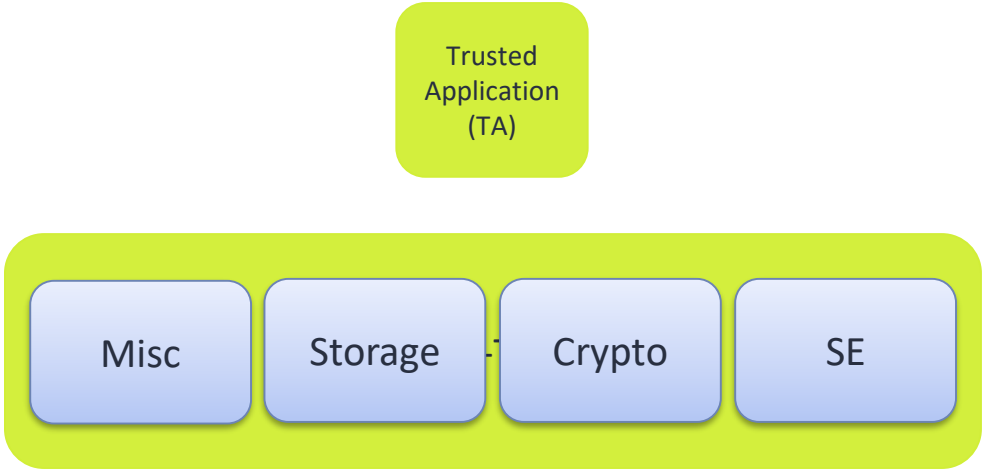
OP-TEE syscalls



OP-TEE syscalls



OP-TEE syscalls



Total: 70 syscalls

However, does *trusted* also mean secure?

Who guards the guardian...

Fuzzing

Fuzzing

- Random data
  - Prototype by colleagues in 2014
  - cat /dev/urandom > /dev/tee_smc

Fuzzing

- Random data
  - Prototype by colleagues in 2014
  - cat /dev/urandom > /dev/tee_smc
- Model-based

Fuzzing

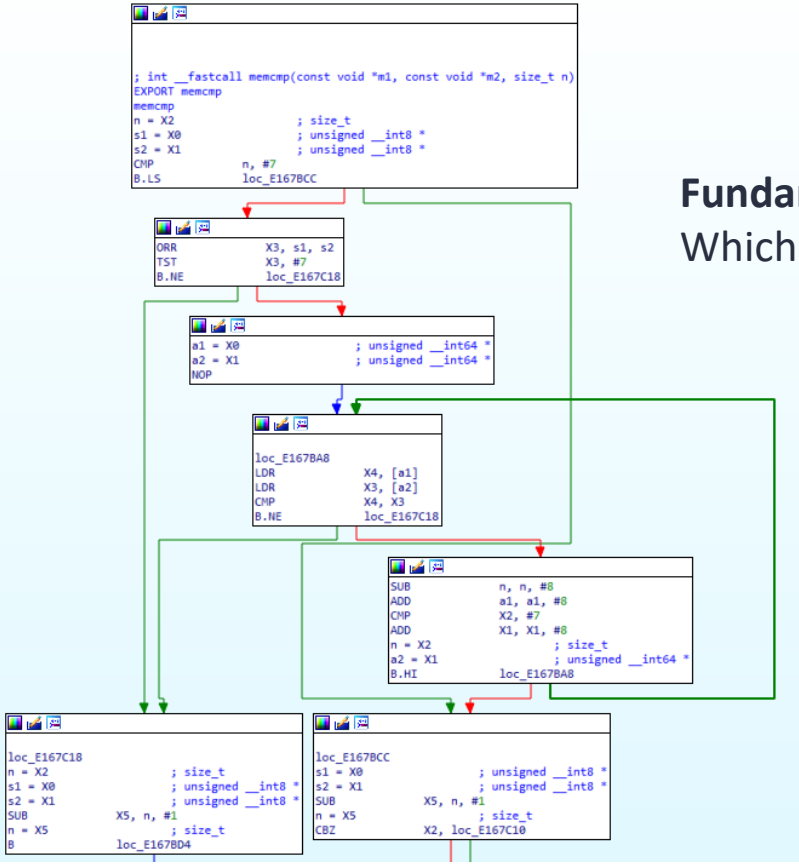
- Random data
  - Prototype by colleagues in 2014
  - cat /dev/urandom > /dev/tee_smc
- Model-based
- Coverage guided evolutionary fuzzing

Coverage guided evolutionary fuzzing

Simple algorithm:

1. Generate new input from collection of corpora
   - By applying 1 or more mutations (e.g. bit flips)
2. Run target with input
3. Collect code coverage information
4. If coverage information shows a previously unseen code path is taken, add to corpus queue

Coverage tracking



Fundamental question:
Which branches are taken?

Coverage tracking

```
; int __fastcall memcmp(const void *m1, const void *m2, size_t n)
EXPORT memcmp
memcmp

var_50= -0x50
var_40= -0x40
var_30= -0x30
var_20= -0x20
var_10= -0x10

n = X2                  ; size_t
m1 = X0                 ; const void *
m2 = X1                 ; const void *
; __unwind {
STP             X29, X30, [SP,#var_50]!
MOV             X29, SP
STP             X19, X20, [SP,#0x50+var_40]
MOV             X20, m1
s1 = X0                 ; unsigned __int8 *
s2 = X1                 ; unsigned __int8 *
MOV             X19, s2
STP             X21, X22, [SP,#0x50+var_30]
MOV             X21, n
STP             X23, X24, [SP,#0x50+var_20]
STR             X25, [SP,#0x50+var_10]
BL              __sanitizer_cov_trace_pc
CMP             n, #7
B.LS            loc 9C
```

```
BL              __sanitizer_cov_trace_pc
ORR             X0, s1, s2
TST             X0, #7
B.NE            loc_138
```

```
ADRP            X1, #off_1C0@PAGE ; "lib/libutils/isoc/newlib/memcmp.c"
ADRP            X0, #off_1D0@PAGE ; "lib/libutils/isoc/newlib/memcmp.c"
ADD             X25, X1, #off_1C0@PAGEOFF ; "lib/libutils/isoc/newlib/memcmp.c"
ADD             X24, X0, #off_1D0@PAGEOFF ; "lib/libutils/isoc/newlib/memcmp.c"
```

```
loc_58
a1 = X20                ; unsigned __int64 *
BL              __sanitizer_cov_trace_pc
```

Coverage tracking

```
; int __fastcall memcmp(const void *m1, const void *m2, size_t n)
EXPORT memcmp
memcmp

var_50= -0x50
var_40= -0x40
var_30= -0x30
var_20= -0x20
var_10= -0x10

n = X2                  ; size_t
m1 = X0                 ; const void *
m2 = X1                 ; const void *
; __unwind {
STP             X29, X30, [SP,#var_50]!
MOV             X29, SP
STP             X19, X20, [SP,#0x50+var_40]
MOV             X20, m1
s1 = X0                 ; unsigned __int8 *
s2 = X1                 ; unsigned __int8 *
MOV             X19, s2
STP             X21, X22, [SP,#0x50+var_30]
MOV             X21, n
STP             X23, X24, [SP,#0x50+var_20]
STR             X25, [SP,#0x50+var_10]
BL              __sanitizer_cov_trace_pc
CMP             n, #7
B.LS            loc 9C
```

```
BL              __sanitizer_cov_trace_pc
ORR             X0, s1, s2
TST             X0, #7
B.NE            loc_138
```

```
ADRP            X1, #off_1C0@PAGE ; "lib/libutils/isoc/newlib/memcmp.c"
ADRP            X0, #off_1D0@PAGE ; "lib/libutils/isoc/newlib/memcmp.c"
ADD             X25, X1, #off_1C0@PAGEOFF ; "lib/libutils/isoc/newlib/memcmp.c"
ADD             X24, X0, #off_1D0@PAGEOFF ; "lib/libutils/isoc/newlib/memcmp.c"
```

```
loc_58
a1 = X20                ; unsigned __int64 *
BL              __sanitizer_cov_trace_pc
```

Coverage tracking

```
; int __fastcall memcmp(const void *m1, const void *m2, size_t n)
EXPORT memcmp
memcmp

var_50= -0x50
var_40= -0x40
var_30= -0x30
var_20= -0x20
var_10= -0x10

n = X2                  ; size_t
m1 = X0                 ; const void *
m2 = X1                 ; const void *
; __unwind {
STP             X29, X30, [SP,#var_50]!
MOV             X29, SP
STP             X19, X20, [SP,#0x50+var_40]
MOV             X20, m1
s1 = X0                 ; unsigned __int8 *
s2 = X1                 ; unsigned __int8 *
MOV             X19, s2
STP             X21, X22, [SP,#0x50+var_30]
MOV             X21, n
STP             X23, X24, [SP,#0x50+var_20]
STR             X25, [SP,#0x50+var_10]
BL              __sanitizer_cov_trace_pc
CMP             n, #7
B.LS            loc 9C
```

```
BL              __sanitizer_cov_trace_pc
ORR             X0, s1, s2
TST             X0, #7
B.NE            loc_138
```

```
ADRP            X1, #off_1C0@PAGE ; "lib/libutils/isoc/newlib/memcmp.c"
ADRP            X0, #off_1D0@PAGE ; "lib/libutils/isoc/newlib/memcmp.c"
ADD             X25, X1, #off_1C0@PAGEOFF ; "lib/libutils/isoc/newlib/memcmp.c"
ADD             X24, X0, #off_1D0@PAGEOFF ; "lib/libutils/isoc/newlib/memcmp.c"
```

```
loc_58
a1 = X20                ; unsigned __int64 *
BL              __sanitizer_cov_trace_pc
```

Coverage tracking

```
; int __fastcall memcmp(const void *m1, const void *m2, size_t n)
EXPORT memcmp
memcmp

var_50= -0x50
var_40= -0x40
var_30= -0x30
var_20= -0x20
var_10= -0x10

n = X2                  ; size_t
m1 = X0                 ; const void *
m2 = X1                 ; const void *
; __unwind {
STP             X29, X30, [SP,#var_50]!
MOV             X29, SP
STP             X19, X20, [SP,#0x50+var_40]
MOV             X20, m1
s1 = X0                 ; unsigned __int8 *
s2 = X1                 ; unsigned __int8 *
MOV             X19, s2
STP             X21, X22, [SP,#0x50+var_30]
MOV             X21, n
STP             X23, X24, [SP,#0x50+var_20]
STR             X25, [SP,#0x50+var_10]
BL              __sanitizer_cov_trace_pc
CMP             n, #7
B.LS            loc 9C
```

```
BL              __sanitizer_cov_trace_pc
ORR             X0, s1, s2
TST             X0, #7
B.NE            loc_138
```

```
ADRP            X1, #off_1C0@PAGE ; "lib/libutils/isoc/newlib/memcmp.c"
ADRP            X0, #off_1D0@PAGE ; "lib/libutils/isoc/newlib/memcmp.c"
ADD             X25, X1, #off_1C0@PAGEOFF ; "lib/libutils/isoc/newlib/memcmp.c"
ADD             X24, X0, #off_1D0@PAGEOFF ; "lib/libutils/isoc/newlib/memcmp.c"
```

```
loc_58
a1 = X20                ; unsigned __int64 *
BL              __sanitizer_cov_trace_pc
```

Why is fuzzing operating systems difficult?

- Crashes
- Global state
- Coverage tracking
- Seeding
- Trace stability → threading, SMP, interrupts

Why is fuzzing operating systems difficult?

- Crashes
- Global state
- Coverage tracking
- Seeding
- Trace stability → threading, SMP, interrupts

A lot of progress for Linux and other mainstream Oss
e.g. AFL, Syzkaller, ...

Why is fuzzing operating systems difficult?

- Crashes
- Global state
- Coverage tracking
- Seeding
- Trace stability → threading, SMP, interrupts

A lot of progress for Linux and other mainstream Oss
e.g. AFL, Syzkaller, ...

Let's make use of that!

Goals

- Reuse an existing fuzzer (AFL)
  → Focus on the TEE challenge, not building a fuzzer
- Good, not perfect results (limited time)

Why is fuzzing *TEEs* difficult?

All before

+

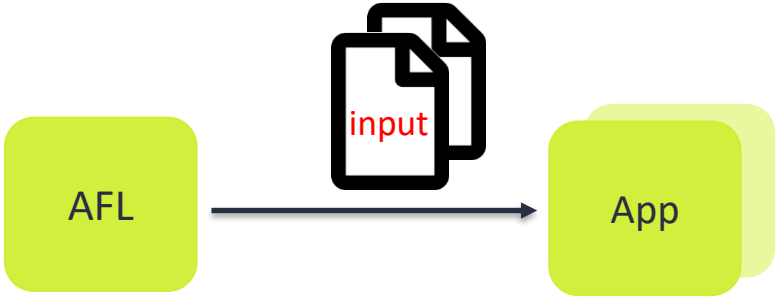
- Isolated environment
- Separate operating system
- Limited API
- **Seeding**

AFL

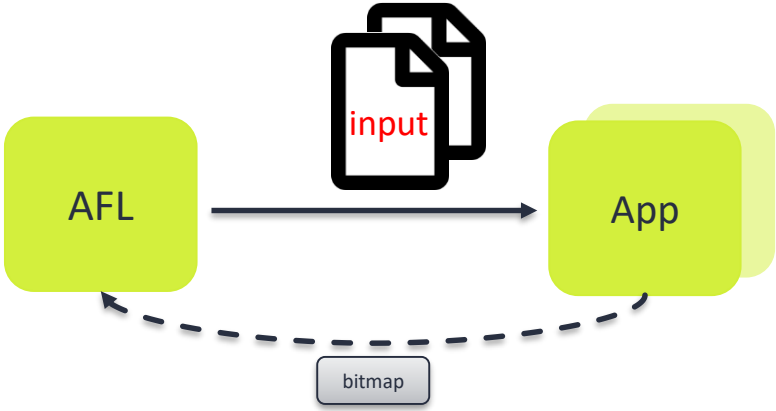
AFL

AFL

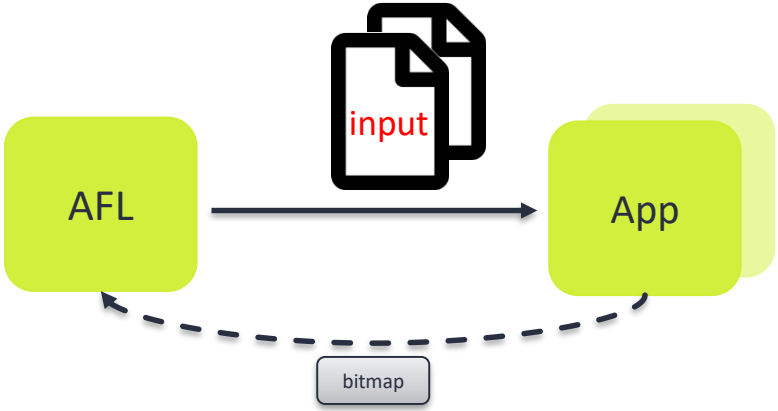
AFL



AFL

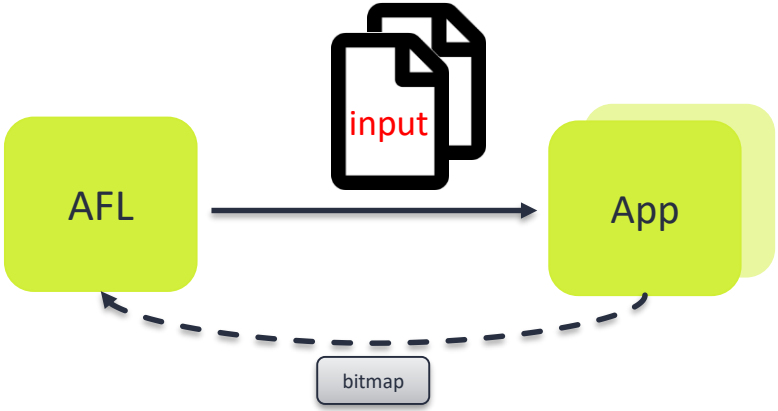


AFL




```
7ae0  00 00 00 00 00 00 00 00 00 00 01 00 00 00 00 00  ................
7b00  01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
7b70  00 00 00 01 00 00 00 00 00 00 00 00 00 00 00 00  ................
7b90  01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
7fa0  00 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00  ................
81e0  00 00 00 00 00 00 00 00 00 01 00 00 00 00 00 00  ................
8240  00 23 00 00 00 00 00 00 00 00 00 00 00 00 23 00  .#............#.
8250  00 00 00 00 23 00 00 00 23 00 00 00 00 00 00 00  ....#...#.......
8270  00 00 23 00 00 00 39 00 00 00 00 00 00 00 00 00  ..#...9.........
85f0  01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

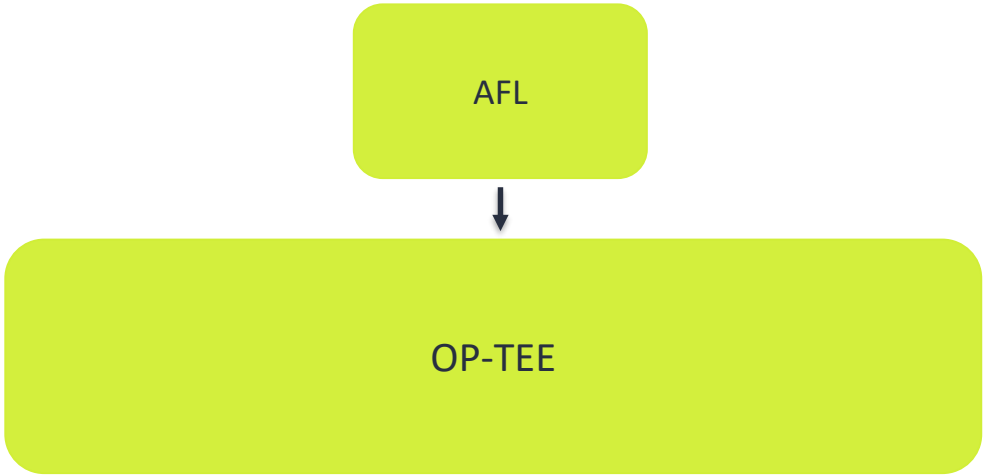
AFL



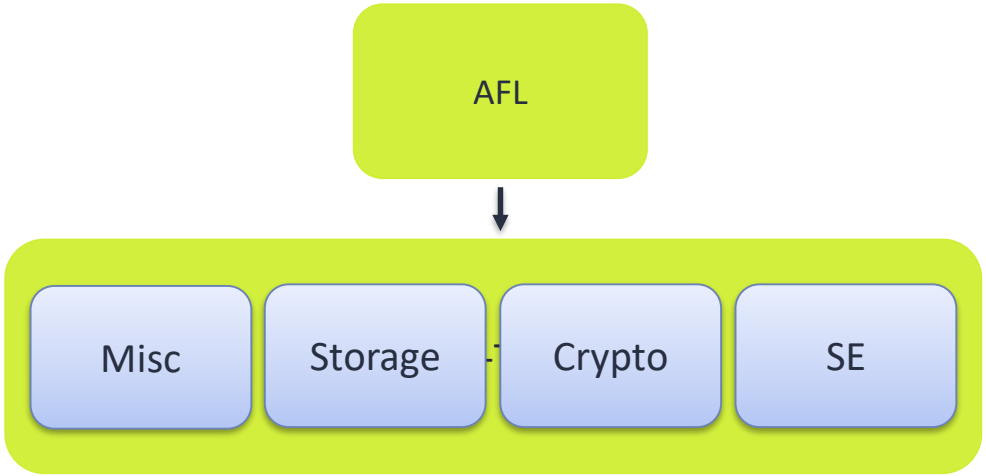

```
7ae0  00 00 00 00 00 00 00 00 00 00 01 00 00 00 00 00  ................
7b00  01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
7b70  00 00 00 01 00 00 00 00 00 00 00 00 00 00 00 00  ................
7b90  01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
7fa0  00 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00  ................
81e0  00 00 00 00 00 00 00 00 00 01 00 00 00 00 00 00  ................
8240  00 23 00 00 00 00 00 00 00 00 00 00 00 00 23 00  .#............#.
8250  00 00 00 00 23 00 00 00 23 00 00 00 00 00 00 00  ....#...#.......
8270  00 00 23 00 00 00 39 00 00 00 00 00 00 00 00 00  ..#...9.........
85f0  01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

Corresponds with $addr somewhere in binary

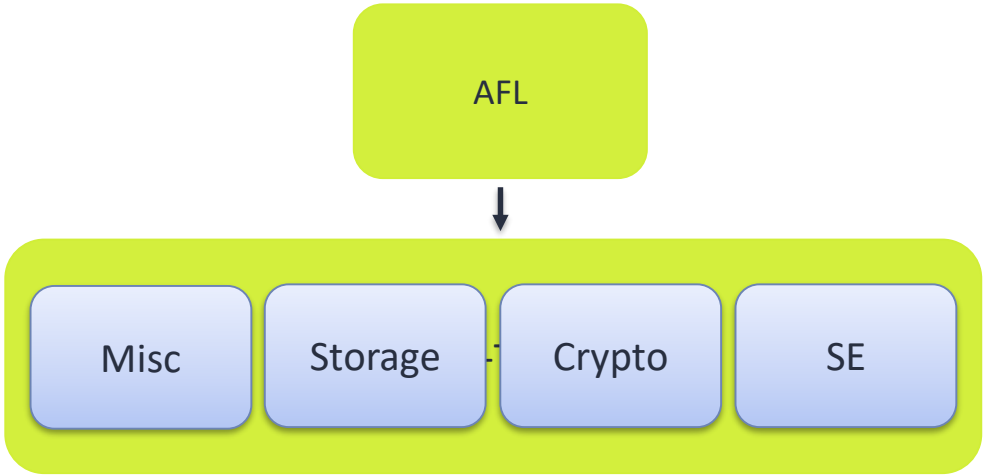
AFL as Trusted Application?



AFL as Trusted Application?

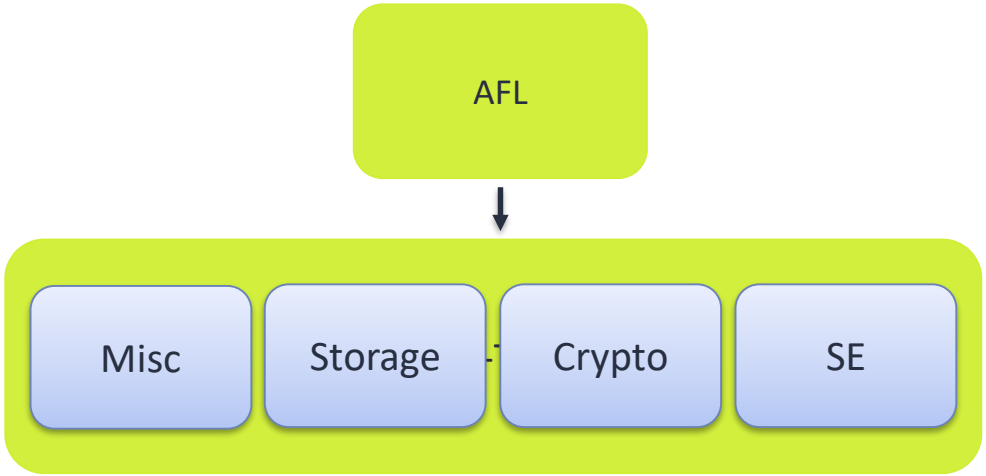


AFL as Trusted Application?



`fork(…)?` → `TEE_ERROR_NOT_IMPLEMENTED`

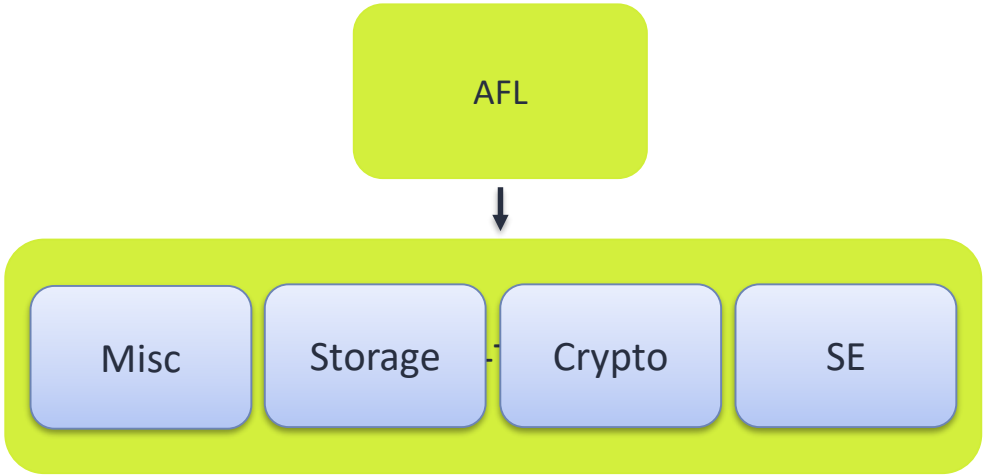
AFL as Trusted Application?



`fork(…)?` → `TEE_ERROR_NOT_IMPLEMENTED`

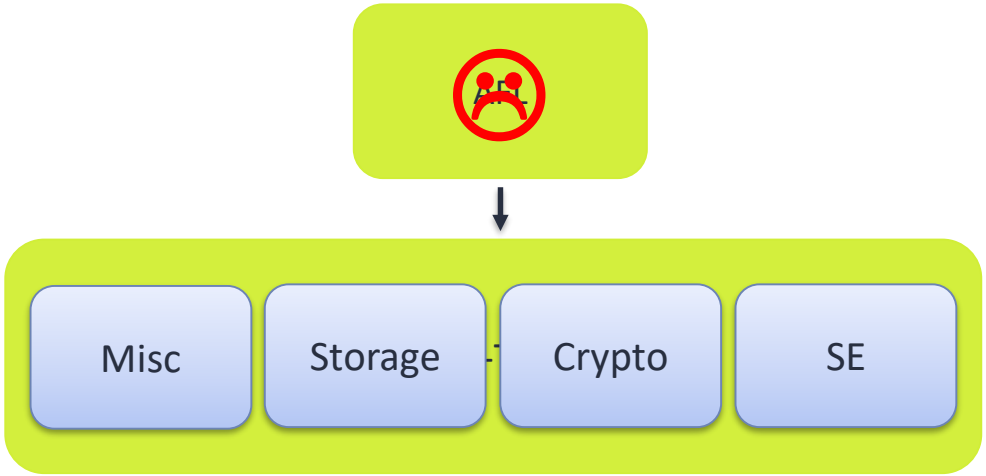
`execve(…)?` → `TEE_ERROR_NOT_IMPLEMENTED`

AFL as Trusted Application?



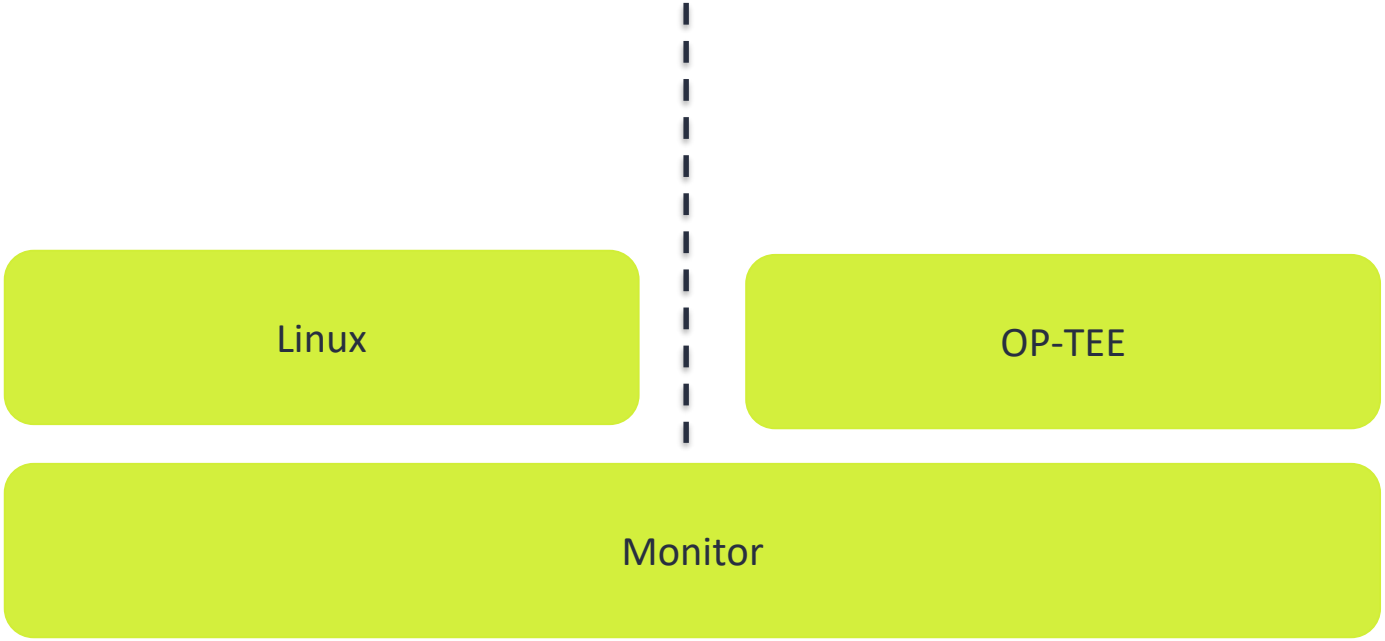

```
fork(…)?     → TEE_ERROR_NOT_IMPLEMENTED
execve(…)?   → TEE_ERROR_NOT_IMPLEMENTED
open(…)?     → TEE_ERROR_NOT_IMPLEMENTED
```

AFL as Trusted Application?

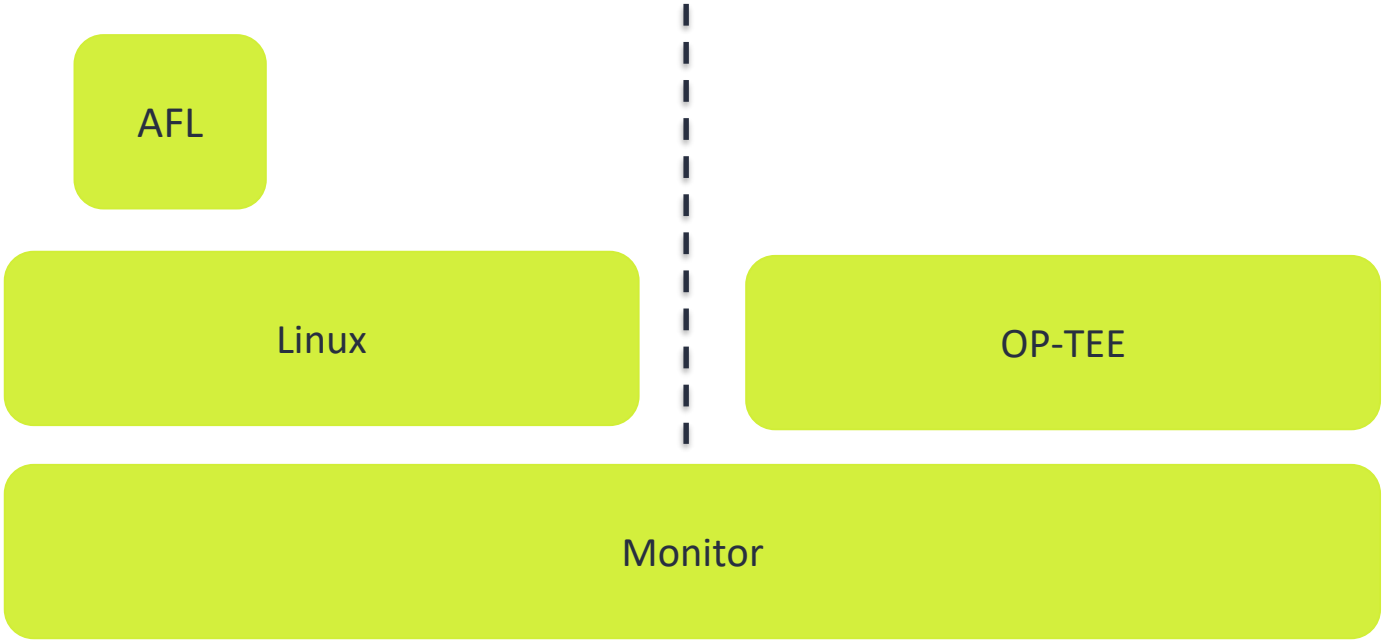



```
fork(…)?     → TEE_ERROR_NOT_IMPLEMENTED
execve(…)?   → TEE_ERROR_NOT_IMPLEMENTED
open(…)?     → TEE_ERROR_NOT_IMPLEMENTED
```

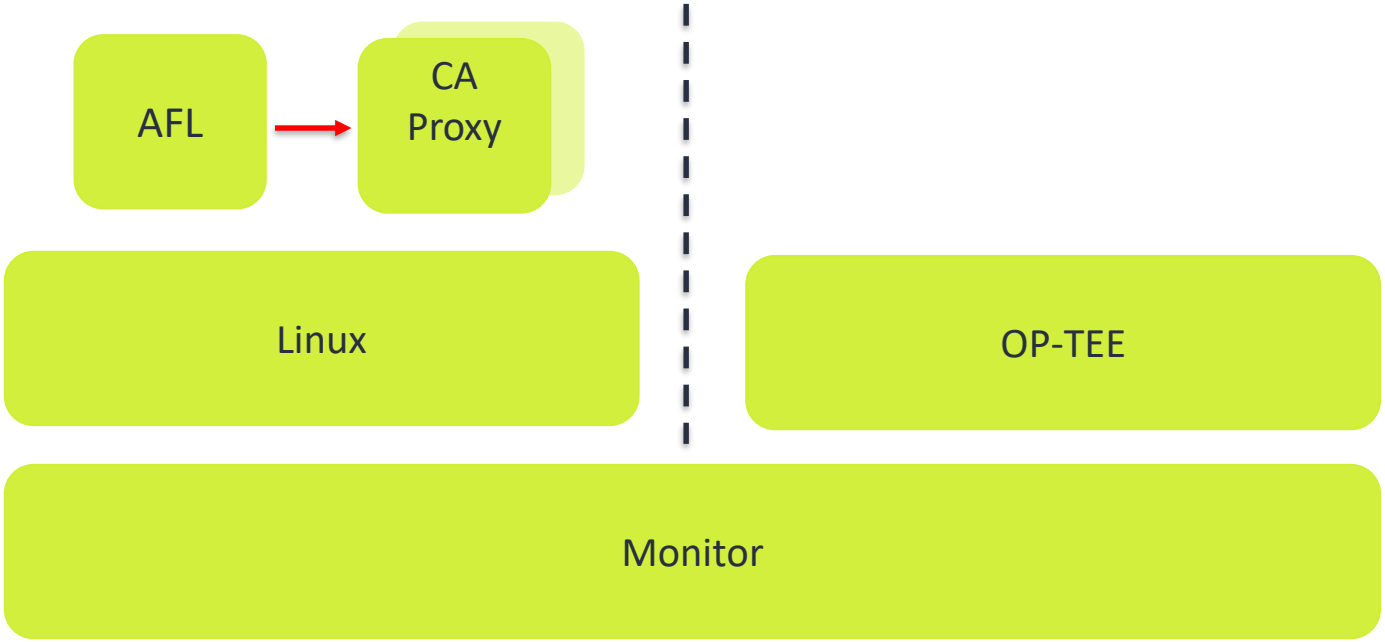
AFL as ~~Trusted~~ Linux Application



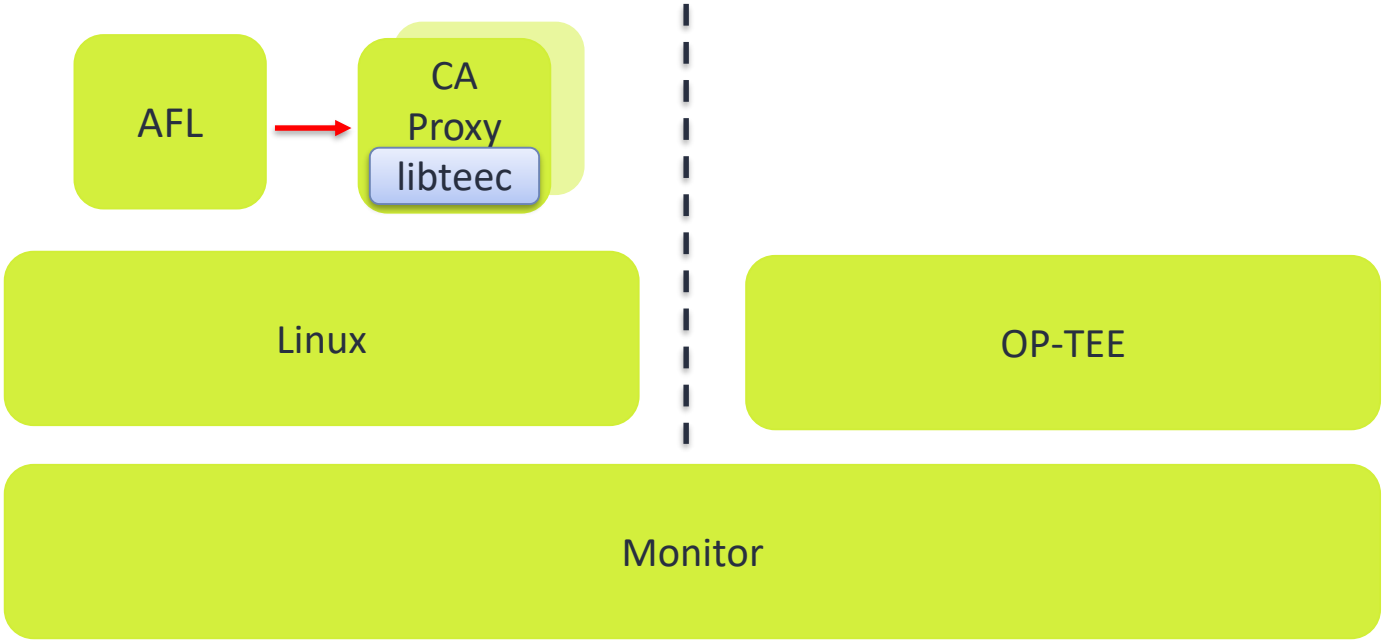
AFL as ~~Trusted~~ Linux Application



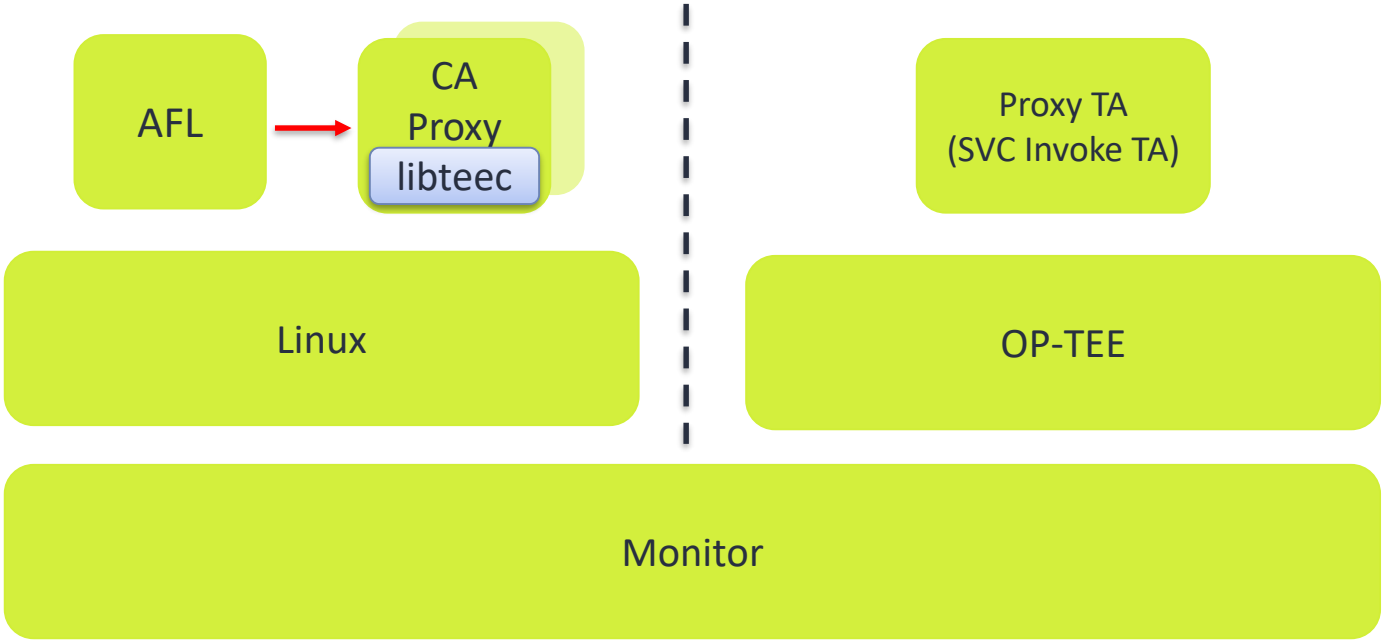
AFL as ~~Trusted~~ Linux Application



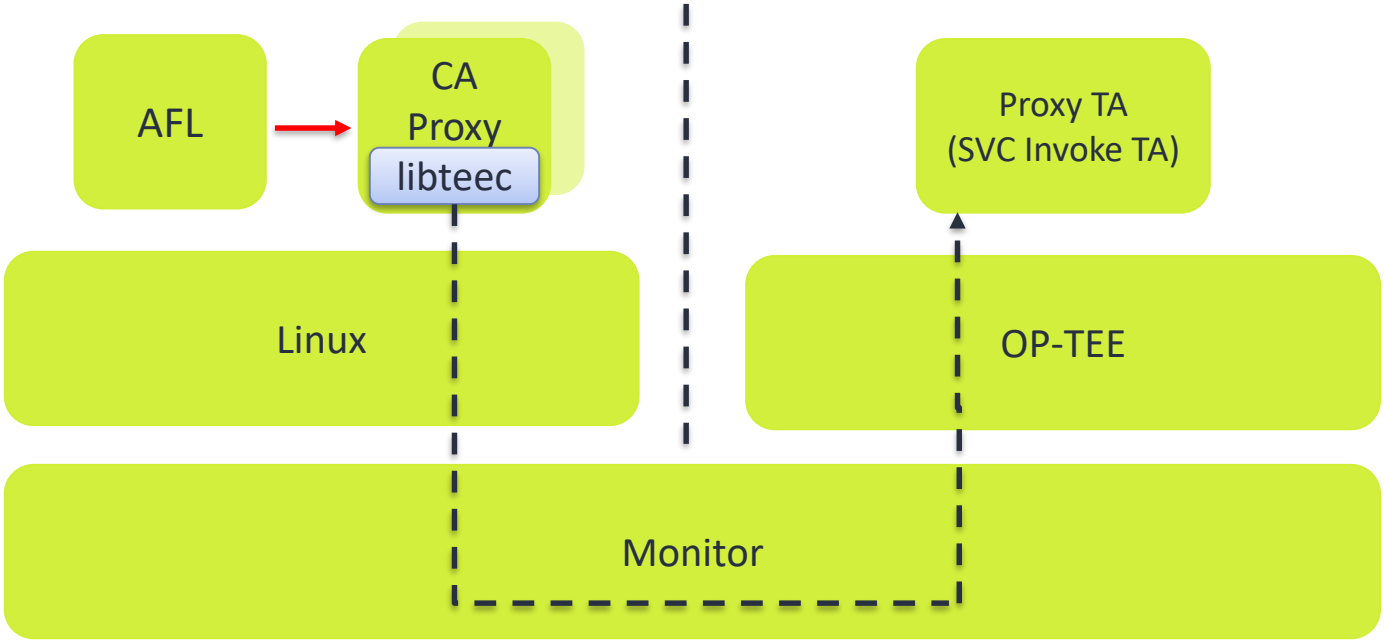
AFL as ~~Trusted~~ Linux Application



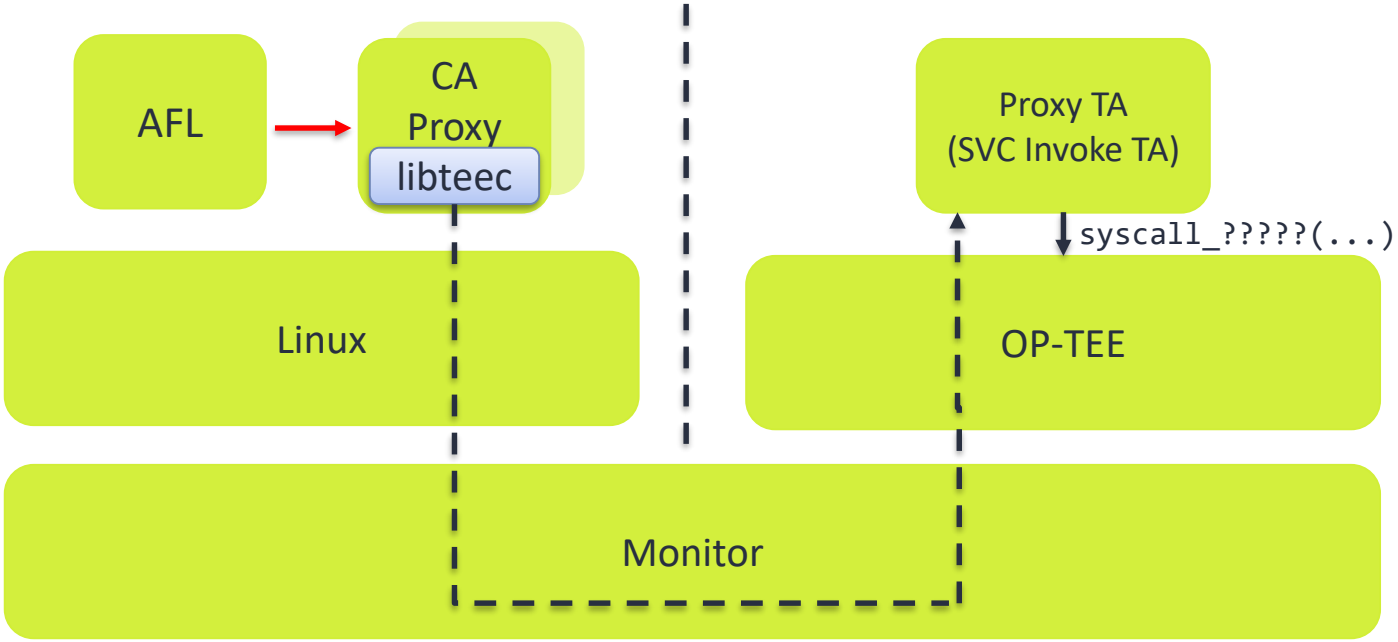
AFL as ~~Trusted~~ Linux Application



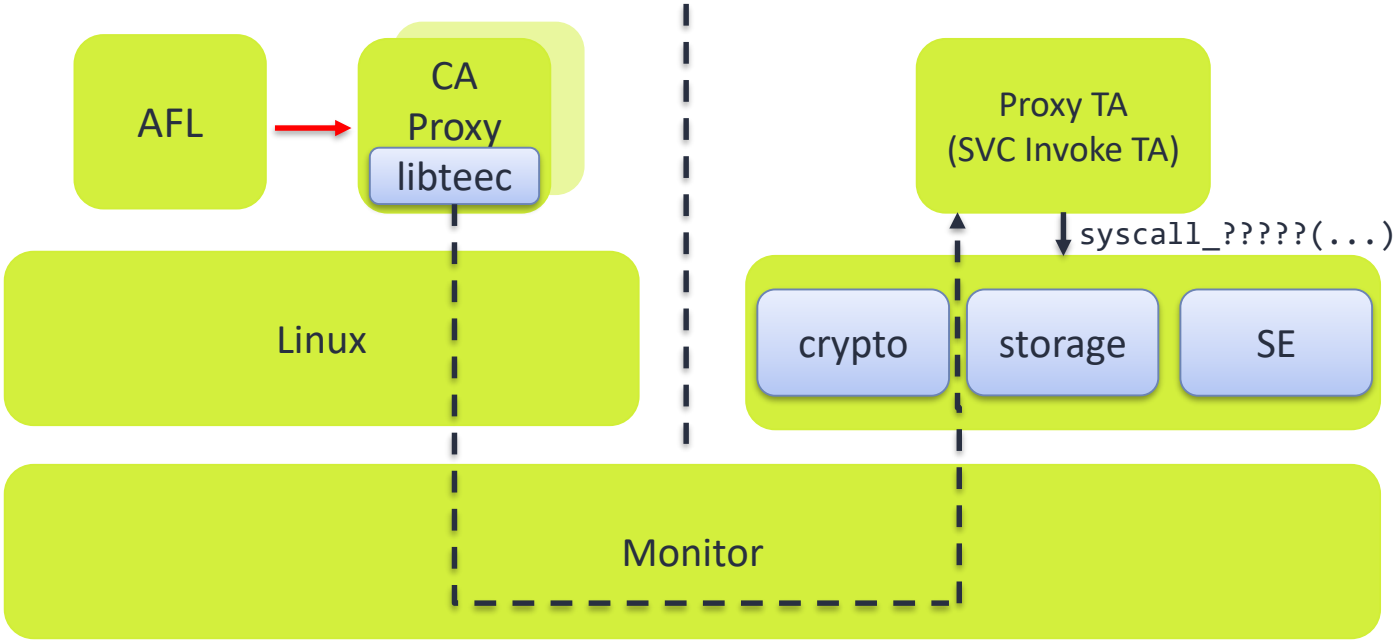
AFL as ~~Trusted~~ Linux Application



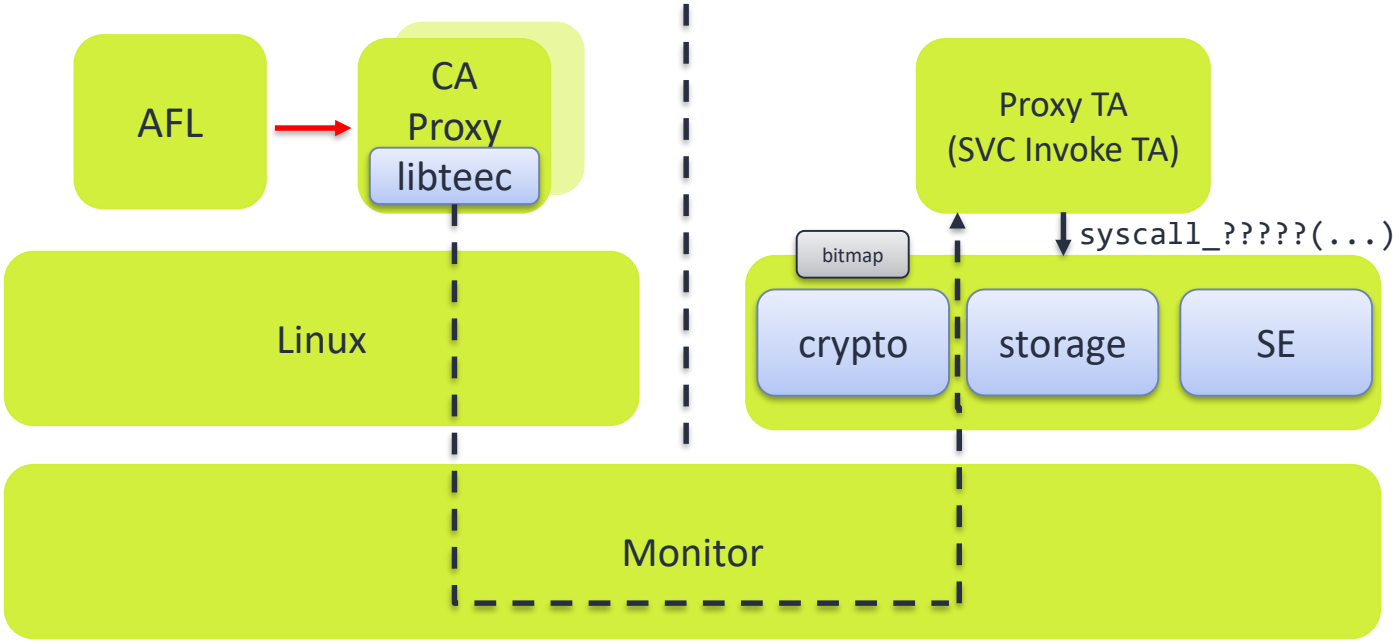
AFL as ~~Trusted~~ Linux Application



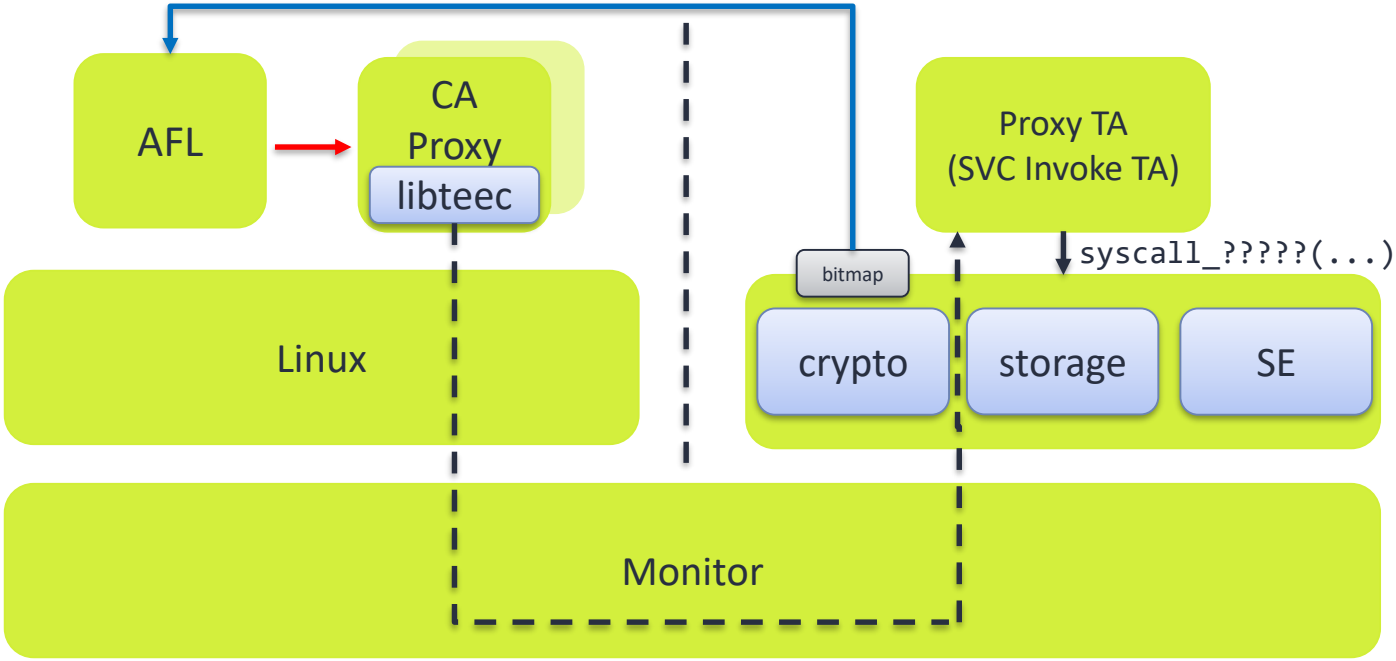
AFL as ~~Trusted~~ Linux Application



AFL as ~~Trusted~~ Linux Application



AFL as ~~Trusted~~ Linux Application



How to (randomly) invoke system calls using AFL?

AFL can only mutate a blob of (random) data by flipping bits or bytes...

We need to find a way to let AFL generate random system calls

And preferably not by rewriting the mutation engine

Hello Nullcon!

```
00000000: 0100 0000 4a00 0000 1800 0001 1000 0000  ....J...........
00000010: ff00 0000 0000 0000 0a48 656c 6c6f 204e  .........Hello N
00000020: 756c 6c63 6f6e 2100                      ullcon!.
```

Hello Nullcon!

```
00000000: 0100 0000 4a00 0000 1800 0001 1000 0000  ....J...........
00000010: ff00 0000 0000 0000 0a48 656c 6c6f 204e  .........Hello N
00000020: 756c 6c63 6f6e 2100                      ullcon!.
```

```
t[0] = malloc(16);
memcpy(t[0], "\x0aHello Nullcon!\x00", 16);
r[0] = utee_log(t[0], 0x10);
free(t[0]);
```

Hello Nullcon!

```
00000000: 0100 0000 4a00 0000 1800 0001 1000 0000  ....J...........
00000010: ff00 0000 0000 0000 0a48 656c 6c6f 204e  .........Hello N
00000020: 756c 6c63 6f6e 2100                      ullcon!.
```

```
t[0] = malloc(16);
memcpy(t[0], "\x0aHello Nullcon!\x00", 16);
r[0] = utee_log(t[0], 0x10);
free(t[0]);
```

Syscall wrapper function

System calls as data

- Syscalls consists of id + up to 8 arguments
  - Values
  - Pointers to data, structures, etc
  - Pointers to structures with pointers, etc.
- Syscall arguments often depend on prev. syscall
  - E.g. returned handles

→ **Argument encoding is the hardest part!**

System calls as data

- Simple binary format encoding 1 or more syscall invokes
  - Contains arguments inline except buffer content
  - Goal: every bit flip results in a slightly different invoke

- After invoke data follows section with raw data
  - Strings, buffer content, etc.
  - Can be flexible referenced from argument info

System calls as data

```
typedef enum {
    ARG_NONE,

    ARG_VALUE_NULL,
    ARG_VALUE_8,
    ARG_VALUE_16,
    ARG_VALUE_32,
    ARG_VALUE_64,

    ARG_BUFFER_ALLOC,
    ARG_BUFFER_REF,
    ARG_BUFFER_DEREF32,
    ARG_BUFFER_DEREF64,

    ARG_DATA_SHARED,
    ARG_DATA_PRIVATE,

    ARG_RETURN_VALUE,

    ARG_TYPE_MAX
} svc_arg_type_t;
```

System calls as data

```
00000000: 0100 0000 4a00 0000 1800 0001 1000 0000  ....J...........
00000010: ff00 0000 0000 0000 0a48 656c 6c6f 204e  .........Hello N
00000020: 756c 6c63 6f6e 2100                      ullcon!.
```

System calls as data

```
00000000: 0100 0000 4a00 0000 1800 0001 1000 0000  ....J...........
00000010: ff00 0000 0000 0000 0a48 656c 6c6f 204e  .........Hello N
00000020: 756c 6c63 6f6e 2100                      ullcon!.
```

Syscall id

System calls as data

```
00000000: 0100 0000 4a00 0000 1800 0001 1000 0000  ....J...........
00000010: ff00 0000 0000 0000 0a48 656c 6c6f 204e  .........Hello N
00000020: 756c 6c63 6f6e 2100                      ullcon!.
```

Argument types

→ 1 nibble per argument

System calls as data

```
00000000: 0100 0000 4a00 0000 1800 0001 1000 0000  ....J...........
00000010: ff00 0000 0000 0000 0a48 656c 6c6f 204e  .........Hello N
00000020: 756c 6c63 6f6e 2100                      ullcon!.
```

Argument types

0xa: argument 0 is a buffer with in-line data

System calls as data

```
00000000: 0100 0000 4a00 0000 1800 0001 1000 0000  ....J...........
00000010: ff00 0000 0000 0000 0a48 656c 6c6f 204e  .........Hello N
00000020: 756c 6c63 6f6e 2100                      ullcon!.
```

Argument types

0xa: argument 0 is a buffer with in-line data

0x4: argument 1 is a 32-bit integer value

System calls as data

```
00000000: 0100 0000 4a00 0000 1800 0001 1000 0000  ....J...........
00000010: ff00 0000 0000 0000 0a48 656c 6c6f 204e  .........Hello N
00000020: 756c 6c63 6f6e 2100                      ullcon!.
```

Argument 0: buffer

Argument 1: value

System calls as data

```
00000000: 0100 0000 4a00 0000 1800 0001 1000 0000  ....J...........
00000010: ff00 0000 0000 0000 0a48 656c 6c6f 204e  .........Hello N
00000020: 756c 6c63 6f6e 2100                      ullcon!.
```

Argument 0:

Buffer offset (12-bit) → 0x18
Buffer length (20-bit) → 0x10

System calls as data

```
00000000: 0100 0000 4a00 0000 1800 0001 1000 0000  ....J...........
00000010: ff00 0000 0000 0000 0a48 656c 6c6f 204e  .........Hello N
00000020: 756c 6c63 6f6e 2100                      ullcon!.
```

Argument 0:

Buffer offset (12-bit) → 0x18
Buffer length (20-bit) → 0x10
Data

System calls as data

```
00000000: 1b00 0000 4406 0000 1000 00a0 8000 0000
00000010: 0080 0000 0f00 0000 4447 0600 1001 0010
00000020: 0000 0000 0040 0000 0000 0000 0180 0000
00000030: 1e00 0000 c704 0000 0040 0000 0000 00c0
00000040: 8000 0001 0100 0000 1500 0000 a704 0000
00000050: 0140 0000 9000 0001 1000 0000 1600 0000
00000060: a764 0a00 0140 0000 a000 0001 1000 0000
00000070: 0300 0100 b000 8000 ff00 0000 0000 0000
00000080: 0001 0203 0405 0607 0809 0a0b 0c0d 0e0f
00000090: 0000 0000 0000 0000 0000 0000 0000 0000
000000a0: 4865 6c6c 6f20 4e75 6c6c 636f 6e21 2100
000000b0: 1000 0000 0000 0000
```

```
cryp_obj_alloc(0xa0000010, 0x80, &obj_handle);
cryp_state_alloc(0x10000110, 0, obj_handle, 0, &cryp_handle);
cryp_obj_populate (cryp_handle,
                   {c0000000, "\x00\x01[...]\x0e\x0f"}, 1);
cipher_init(cryp_handle, "\x00\x00[...]\x00\x00", 0x10);
cipher_update(cryp_handle, "Hello Nullcon!!\x00", 0x10,
              buf_out, &buf_out_len);
```

System calls as data

```
SYSCALL_INFO syscalls[] = {
  DEF_CALL(log,              SCN_LOG,              2, { ARG_BUF_IN_ADDR | ARG_BUF_LEN_ARG(1), ARG_VALUE })
  DEF_CALL(panic,            SCN_PANIC,            2, { ARG_VALUE })
  DEF_CALL(get_property,     SCN_GET_PROPERTY,     7, { ARG_VALUE, ARG_VALUE, ARG_BUF_IN_ADDR | ARG_BUF_LEN_ARG(3),
                                                        ARG_VALUE_INOUT_PTR, ARG_VALUE_INOUT_PTR, ARG_VALUE,
                                                        ARG_VALUE_OUT_PTR  })
  DEF_CALL(get_time,         SCN_GET_TIME,         2, { ARG_VALUE, ARG_BUF_OUT_ADDR | ARG_BUF_SIZE(sizeof(TEE_Time)) })
  DEF_CALL(set_ta_time,      SCN_SET_TA_TIME,      1, { ARG_BUF_IN_ADDR | ARG_BUF_SIZE(sizeof(TEE_Time)) })
  DEF_CALL(cryp_state_alloc, SCN_CRYP_STATE_ALLOC, 5, { ARG_VALUE, ARG_VALUE, ARG_HANDLE, ARG_HANDLE,
                                                        ARG_HANDLE_OUT_PTR })

  [...]
};
```

System calls as data

```
SYSCALL_INFO syscalls[] = {
  DEF_CALL(log,              SCN_LOG,              2, { ARG_BUF_IN_ADDR | ARG_BUF_LEN_ARG(1), ARG_VALUE })
  DEF_CALL(panic,            SCN_PANIC,            2, { ARG_VALUE })
  DEF_CALL(get_property,     SCN_GET_PROPERTY,     7, { ARG_VALUE, ARG_VALUE, ARG_BUF_IN_ADDR | ARG_BUF_LEN_ARG(3),
                                                        ARG_VALUE_INOUT_PTR, ARG_VALUE_INOUT_PTR, ARG_VALUE,
                                                        ARG_VALUE_OUT_PTR  })
  DEF_CALL(get_time,         SCN_GET_TIME,         2, { ARG_VALUE, ARG_BUF_OUT_ADDR | ARG_BUF_SIZE(sizeof(TEE_Time)) })
  DEF_CALL(set_ta_time,      SCN_SET_TA_TIME,      1, { ARG_BUF_IN_ADDR | ARG_BUF_SIZE(sizeof(TEE_Time)) })
  DEF_CALL(cryp_state_alloc, SCN_CRYP_STATE_ALLOC, 5, { ARG_VALUE, ARG_VALUE, ARG_HANDLE, ARG_HANDLE,
                                                        ARG_HANDLE_OUT_PTR })

  [...]
};
```

Mandatory

System calls as data

```
SYSCALL_INFO syscalls[] = {
  DEF_CALL(log,              SCN_LOG,              2, { ARG_BUF_IN_ADDR | ARG_BUF_LEN_ARG(1), ARG_VALUE })
  DEF_CALL(panic,            SCN_PANIC,            2, { ARG_VALUE })
  DEF_CALL(get_property,     SCN_GET_PROPERTY,     7, { ARG_VALUE, ARG_VALUE, ARG_BUF_IN_ADDR | ARG_BUF_LEN_ARG(3),
                                                        ARG_VALUE_INOUT_PTR, ARG_VALUE_INOUT_PTR, ARG_VALUE,
                                                        ARG_VALUE_OUT_PTR  })
  DEF_CALL(get_time,         SCN_GET_TIME,         2, { ARG_VALUE, ARG_BUF_OUT_ADDR | ARG_BUF_SIZE(sizeof(TEE_Time)) })
  DEF_CALL(set_ta_time,      SCN_SET_TA_TIME,      1, { ARG_BUF_IN_ADDR | ARG_BUF_SIZE(sizeof(TEE_Time)) })
  DEF_CALL(cryp_state_alloc, SCN_CRYP_STATE_ALLOC, 5, { ARG_VALUE, ARG_VALUE, ARG_HANDLE, ARG_HANDLE,
                                                        ARG_HANDLE_OUT_PTR })

  [...]
};
```

Mandatory

Optional

Effectively this technique allows calling any function, not just syscalls!

How do we give AFL a good set of inputs to start from?

Creating them by hand is very tedious...

Seeding

- Difficult for the fuzzer to explore paths without good set of inputs (corpora)
- Ideally the start set covers the full interface

Test suite

324 commits | 1 branch | 27 releases | 25 contributors | View license

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

André Draszik and jforissier benchmark_1000: fix compilation against musl (uint) ... Latest commit 3733864 9 days ago

cert	cert: add some test certificates	9 months ago
host	benchmark_1000: fix compilation against musl (uint)	4 days ago
package/testsuite/global_platform/api_1....	Move GlobalPlatform tests to their own test suite	2 years ago
scripts	Add scripts/file_to_c.py	9 months ago
ta	fix compilation against musl (inttypes)	4 days ago
.gitignore	Move GlobalPlatform tests to their own test suite	2 years ago
Android.mk	xtest: add --stats applet	15 days ago
CMakeLists.txt	cmake: locate files WRT to project home directory	24 days ago
CMakeToolchain.txt	Cmake support for xtest only (not TAs)	a year ago
LICENSE.md	Create an explicit LICENSE file	3 years ago
Makefile	xtest: use imported OpenSSL	9 months ago
Notice.md	Changing from old STM CLA to the new DCO	3 years ago
README.md	xtest: fix 32bit/64bit build directive when GP suite is enabled	2 years ago

README.md

OP-TEE sanity testsuite

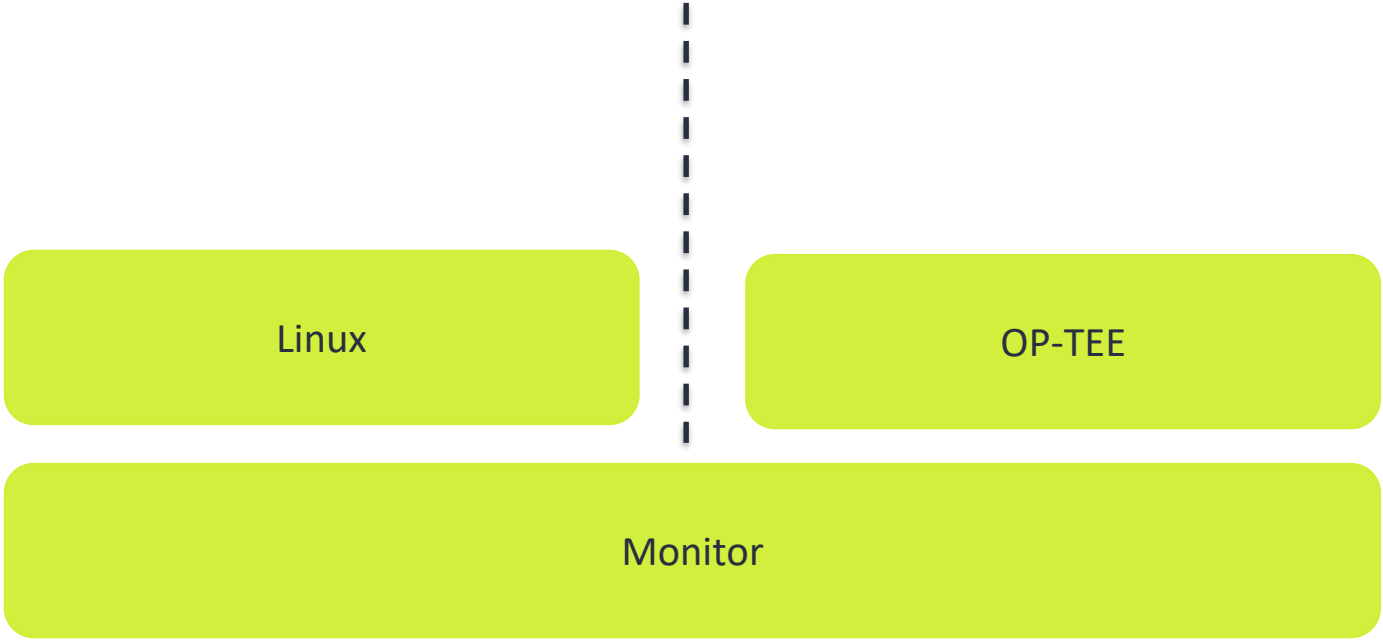
The optee_test git contains the source code for the TEE sanity testsuite in Linux using the ARM(R) TrustZone(R) technology. It is distributed under the GPLv2 and BSD 2-clause open-source licenses. For a general overview of OP-TEE, please see the Notice.md file.

Can we use the test suite to seed AFL?

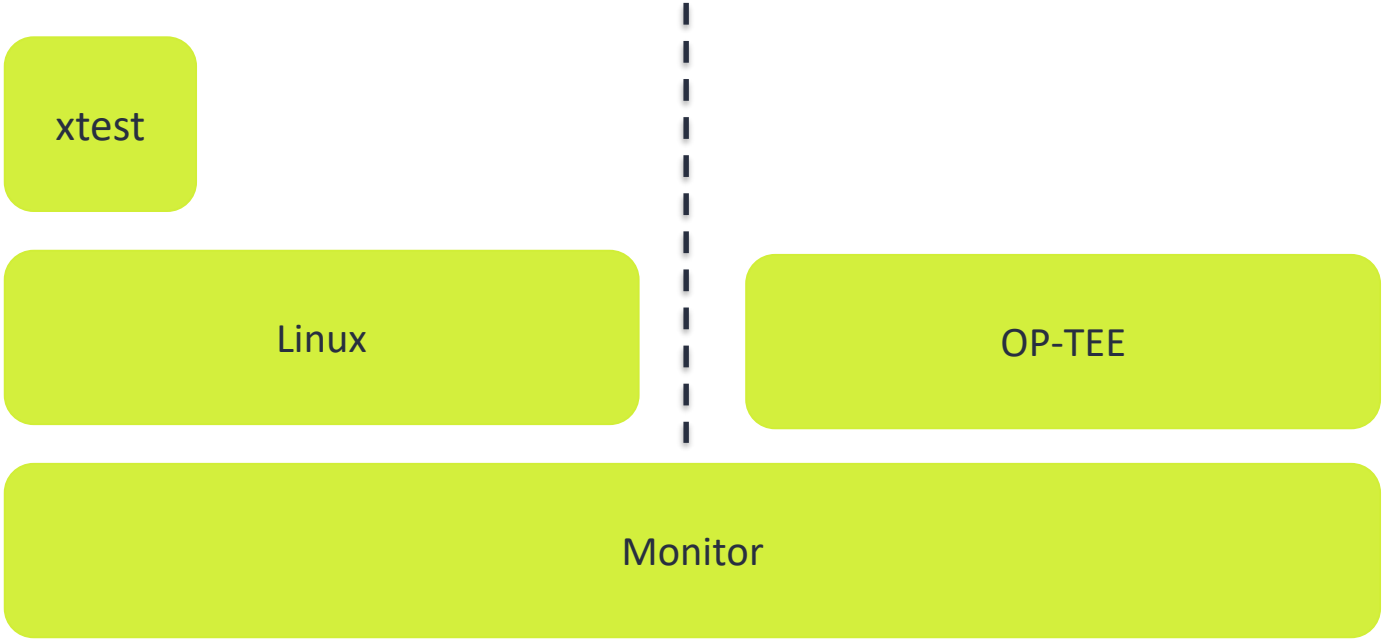
Test suite

- Contains thousands of (regression) tests
- **Covers pretty much all syscalls!**

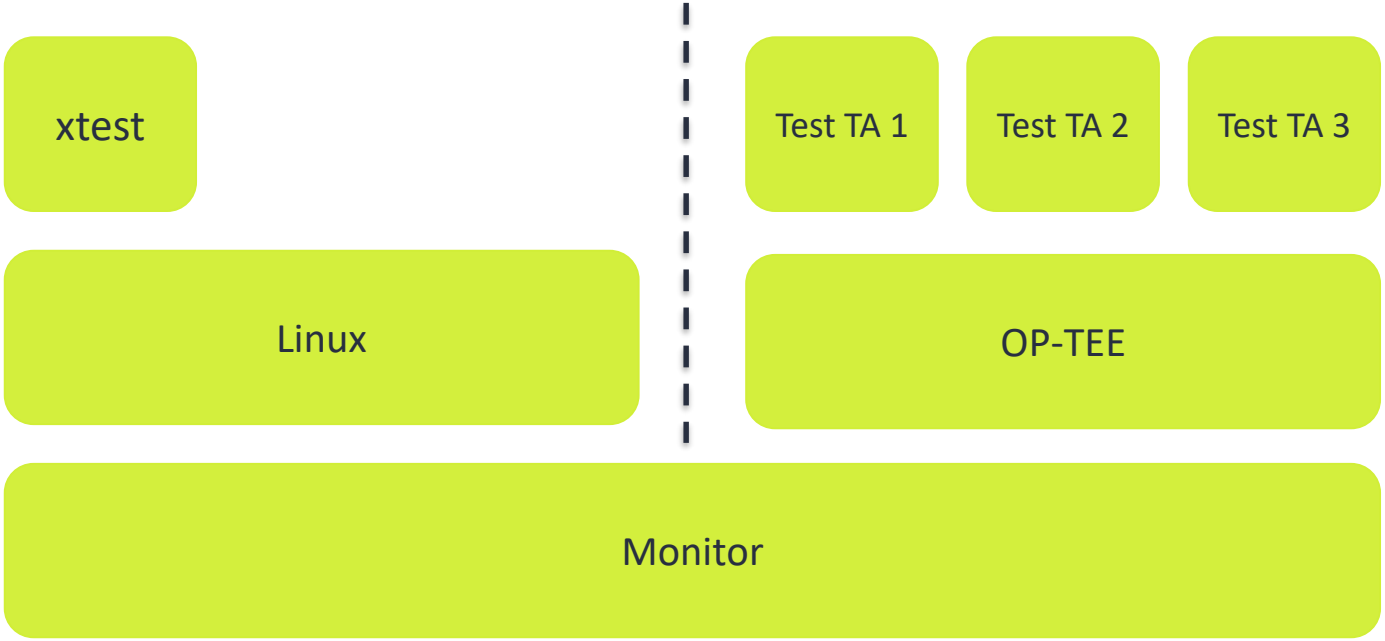
Test case → corpus



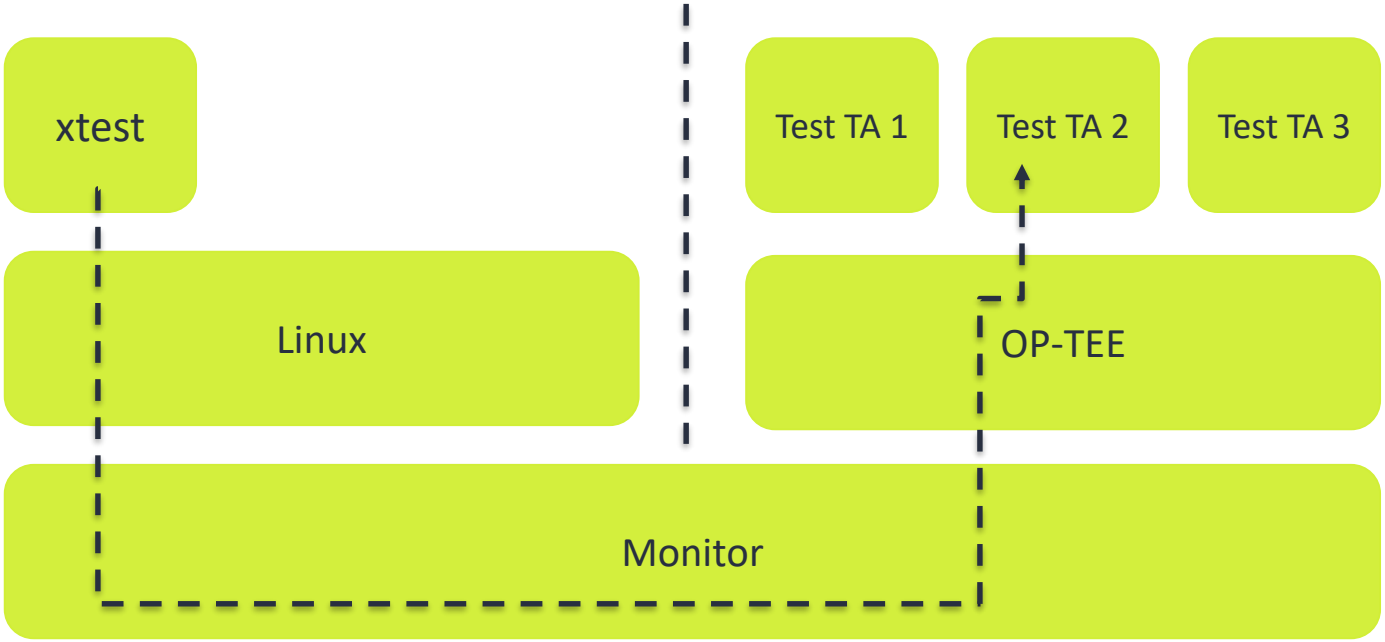
Test case → corpus



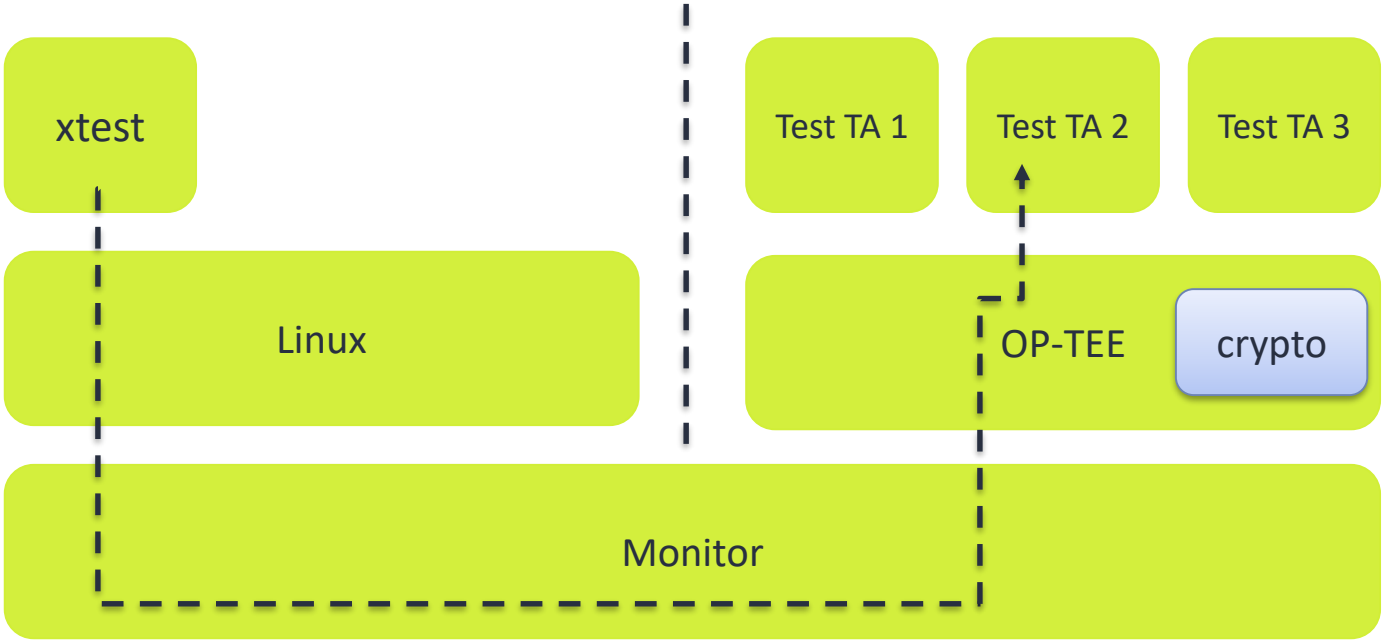
Test case → corpus



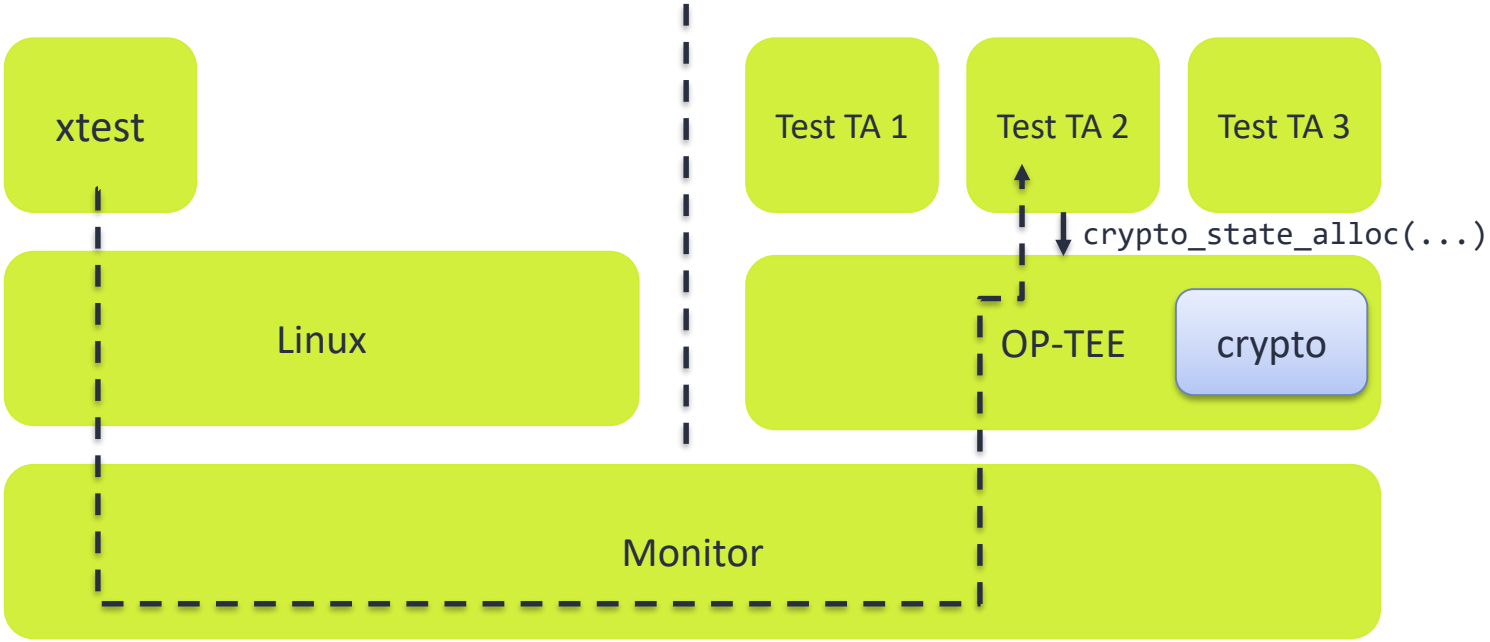
Test case → corpus



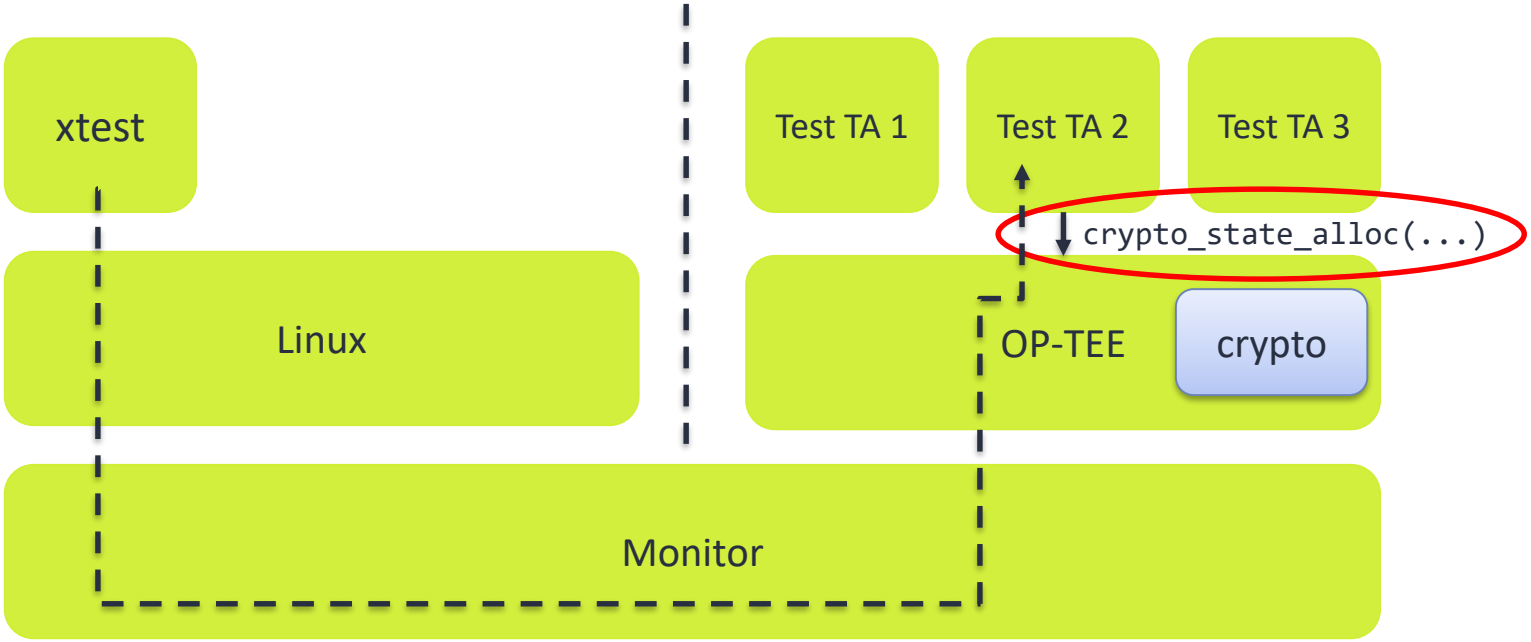
Test case → corpus



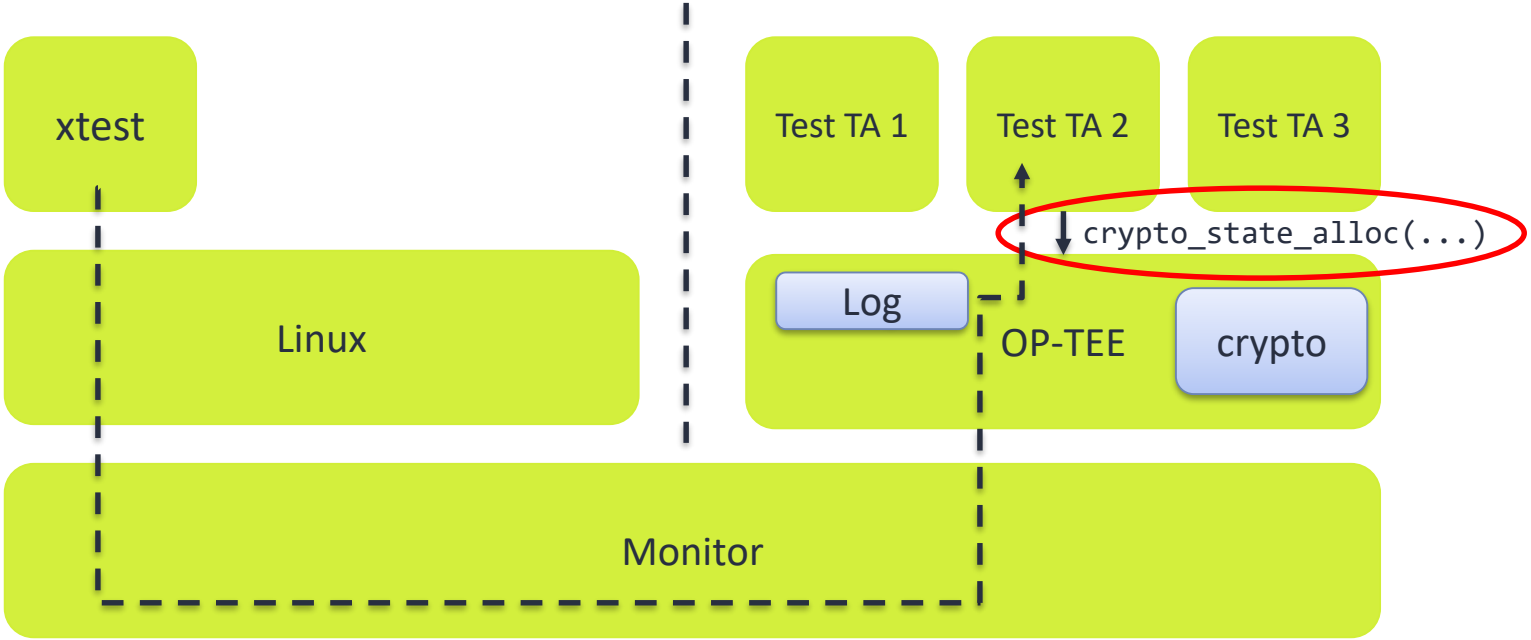
Test case → corpus



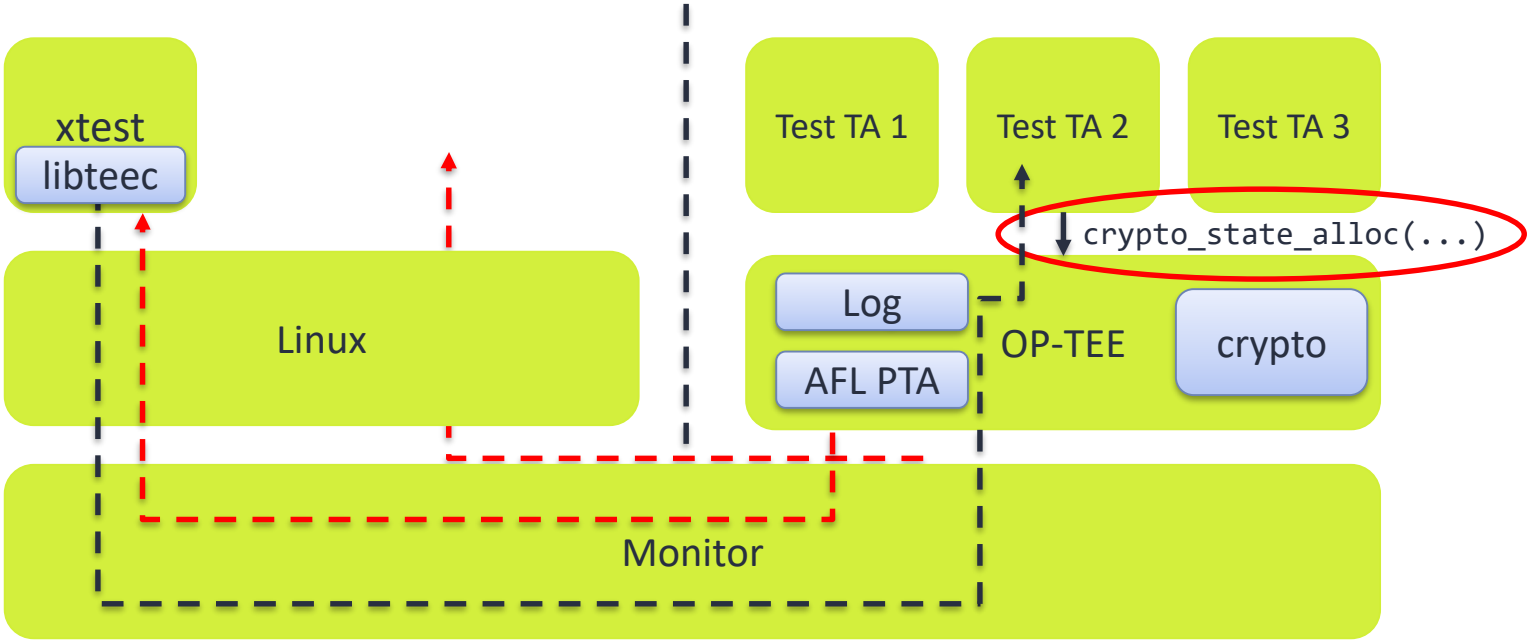
Test case → corpus



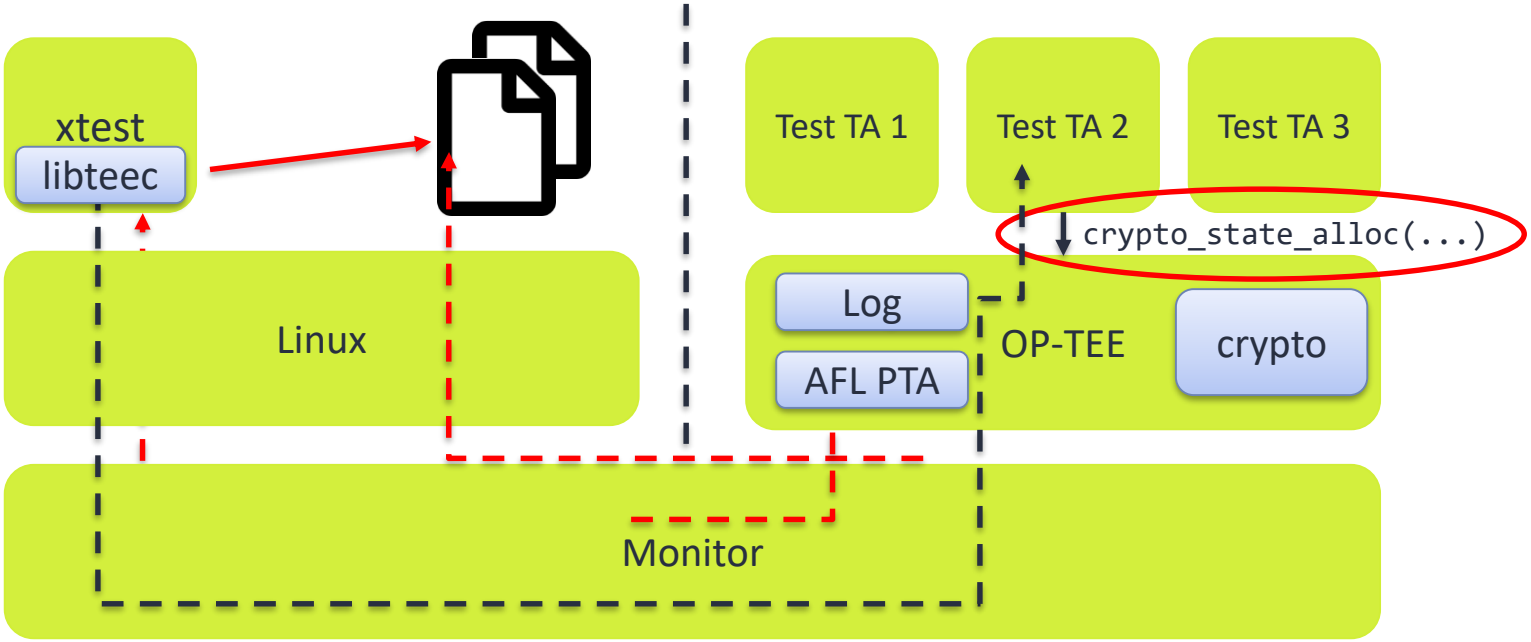
Test case → corpus



Test case → corpus



Test case → corpus



Test case → corpus

```
SYSCALL_INFO syscalls[] = {
  DEF_CALL(log,              SCN_LOG,              2, { ARG_BUF_IN_ADDR | ARG_BUF_LEN_ARG(1), ARG_VALUE })
  DEF_CALL(panic,            SCN_PANIC,            2, { ARG_VALUE })
  DEF_CALL(get_property,     SCN_GET_PROPERTY,     7, { ARG_VALUE, ARG_VALUE, ARG_BUF_IN_ADDR | ARG_BUF_LEN_ARG(3),
                                                        ARG_VALUE_INOUT_PTR, ARG_VALUE_INOUT_PTR, ARG_VALUE,
                                                        ARG_VALUE_OUT_PTR  })
  DEF_CALL(get_time,         SCN_GET_TIME,         2, { ARG_VALUE, ARG_BUF_OUT_ADDR | ARG_BUF_SIZE(sizeof(TEE_Time)) })
  DEF_CALL(set_ta_time,      SCN_SET_TA_TIME,      1, { ARG_BUF_IN_ADDR | ARG_BUF_SIZE(sizeof(TEE_Time)) })
  DEF_CALL(cryp_state_alloc, SCN_CRYP_STATE_ALLOC, 5, { ARG_VALUE, ARG_VALUE, ARG_HANDLE, ARG_HANDLE,
                                                        ARG_HANDLE_OUT_PTR })

  [...]
};
```

Test case → corpus

```
SYSCALL_INFO syscalls[] = {
  DEF_CALL(log,              SCN_LOG,             2, { ARG_BUF_IN_ADDR | ARG_BUF_LEN_ARG(1), ARG_VALUE })
  DEF_CALL(panic,            SCN_PANIC,           2, { ARG_VALUE })
  DEF_CALL(get_property,     SCN_GET_PROPERTY,    7, { ARG_VALUE, ARG_VALUE, ARG_BUF_IN_ADDR | ARG_BUF_LEN_ARG(3),
                                                       ARG_VALUE_INOUT_PTR, ARG_VALUE_INOUT_PTR, ARG_VALUE,
                                                       ARG_VALUE_OUT_PTR  })
  DEF_CALL(get_time,         SCN_GET_TIME,        2, { ARG_VALUE, ARG_BUF_OUT_ADDR | ARG_BUF_SIZE(sizeof(TEE_Time)) })
  DEF_CALL(set_ta_time,      SCN_SET_TA_TIME,     1, { ARG_BUF_IN_ADDR | ARG_BUF_SIZE(sizeof(TEE_Time)) })
  DEF_CALL(cryp_state_alloc, SCN_CRYP_STATE_ALLOC, 5, { ARG_VALUE, ARG_VALUE, ARG_HANDLE, ARG_HANDLE,
                                                       ARG_HANDLE_OUT_PTR })

  [...]
};
```

```
cryp_obj_alloc[27](a0000010, 80, 40000dfc)
 [*0x40000dfc => 1e4660]
cryp_state_alloc[15](10000110, 0, 1e4660, 0, 40020a88)
 [*0x40020a88 => 1e44e0]
cryp_obj_alloc[27](a0000010, 80, 40000e6c)
 [*0x40000e6c => 1e3fa0]
cryp_obj_populate[30](1e3fa0, *40000df0:18, 1)
 attr 0 { id: c0000000, a: 40023290, b: 10 }
cryp_obj_reset[29](1e4660)
cryp_obj_copy[31](1e4660, 1e3fa0)
cipher_init[21](1e44e0, *40024270:10, 10)
cipher_update[22](1e44e0, *400222b0:10, 10, *400222b0:10,
                  40000e38=10)
```

Test case → corpus

```
SYSCALL_INFO syscalls[] = {
  DEF_CALL(log,              SCN_LOG,              2, { ARG_BUF_IN_ADDR | ARG_BUF_LEN_ARG(1), ARG_VALUE })
  DEF_CALL(panic,            SCN_PANIC,            2, { ARG_VALUE })
  DEF_CALL(get_property,     SCN_GET_PROPERTY,     7, { ARG_VALUE, ARG_VALUE, ARG_BUF_IN_ADDR | ARG_BUF_LEN_ARG(3),
                                                        ARG_VALUE_INOUT_PTR, ARG_VALUE_INOUT_PTR, ARG_VALUE,
                                                        ARG_VALUE_OUT_PTR  })
  DEF_CALL(get_time,         SCN_GET_TIME,         2, { ARG_VALUE, ARG_BUF_OUT_ADDR | ARG_BUF_SIZE(sizeof(TEE_Time)) })
  DEF_CALL(set_ta_time,      SCN_SET_TA_TIME,      1, { ARG_BUF_IN_ADDR | ARG_BUF_SIZE(sizeof(TEE_Time)) })
  DEF_CALL(cryp_state_alloc, SCN_CRYP_STATE_ALLOC, 5, { ARG_VALUE, ARG_VALUE, ARG_HANDLE, ARG_HANDLE,
                                                        ARG_HANDLE_OUT_PTR })

  [...]
};
```

```
cryp_obj_alloc[27](a0000010, 80, 40000dfc)
 [*0x40000dfc => 1e4660]
cryp_state_alloc[15](10000110, 0, 1e4660, 0, 40020a88)
 [*0x40020a88 => 1e44e0]
cryp_obj_alloc[27](a0000010, 80, 40000e6c)
 [*0x40000e6c => 1e3fa0]
cryp_obj_populate[30](1e3fa0, *40000df0:18, 1)
 attr 0 { id: c0000000, a: 40023290, b: 10 }
cryp_obj_reset[29](1e4660)
cryp_obj_copy[31](1e4660, 1e3fa0)
cipher_init[21](1e44e0, *40024270:10, 10)
cipher_update[22](1e44e0, *400222b0:10, 10, *400222b0:10,
                  40000e38=10)
```

```
00000000: 1b00 0000 4406 0000 1000 00a0 8000 0000
00000010: 0080 0000 0f00 0000 4447 0600 1001 0010
00000020: 0000 0000 0040 0000 0000 0000 0180 0000
00000030: 1b00 0000 4406 0000 1000 00a0 8000 0000
00000040: 0280 0000 1e00 0000 c704 0000 0240 0000
00000050: 0000 00c0 b000 0001 0100 0000 1d00 0000
00000060: 0700 0000 0040 0000 1f00 0000 7700 0000
00000070: 0040 0000 0240 0000 1500 0000 a704 0000
00000080: 0140 0000 c000 0001 1000 0000 1600 0000
00000090: a764 0a00 0140 0000 d000 0001 1000 0000
000000a0: 0300 0100 e000 8000 ff00 0000 0000 0000
000000b0: 0001 0203 0405 0607 0809 0a0b 0c0d 0e0f
000000c0: 0000 0000 0000 0000 0000 0000 0000 0000
000000d0: 4865 6c6c 6f20 4e75 6c6c 636f 6e21 2100
000000e0: 1000 0000 0000 0000
```

Test case → corpus

```
SYSCALL_INFO syscalls[] = {
  DEF_CALL(log,              SCN_LOG,              2, { ARG_BUF_IN_ADDR | ARG_BUF_LEN_ARG(1), ARG_VALUE })
  DEF_CALL(panic,            SCN_PANIC,            2, { ARG_VALUE })
  DEF_CALL(get_property,     SCN_GET_PROPERTY,     7, { ARG_VALUE, ARG_VALUE, ARG_BUF_IN_ADDR | ARG_BUF_LEN_ARG(3),
                                                        ARG_VALUE_INOUT_PTR, ARG_VALUE_INOUT_PTR, ARG_VALUE,
                                                        ARG_VALUE_OUT_PTR  })
  DEF_CALL(get_time,         SCN_GET_TIME,         2, { ARG_VALUE, ARG_BUF_OUT_ADDR | ARG_BUF_SIZE(sizeof(TEE_Time)) })
  DEF_CALL(set_ta_time,      SCN_SET_TA_TIME,      1, { ARG_BUF_IN_ADDR | ARG_BUF_SIZE(sizeof(TEE_Time)) })
  DEF_CALL(cryp_state_alloc, SCN_CRYP_STATE_ALLOC, 5, { ARG_VALUE, ARG_VALUE, ARG_HANDLE, ARG_HANDLE,
                                                        ARG_HANDLE_OUT_PTR })

  [...]
};
```

```
cryp_obj_alloc[27](a0000010, 80, 40000dfc)
 [*0x40000dfc => 1e4660]
cryp_state_alloc[15](10000110, 0, 1e4660, 0, 40020a88)
 [*0x40020a88 => 1e44e0]
cryp_obj_alloc[27](a0000010, 80, 40000e6c)
 [*0x40000e6c => 1e3fa0]
cryp_obj_populate[30](1e3fa0, *40000df0:18, 1)
 attr 0 { id: c0000000, a: 40023290, b: 10 }
cryp_obj_reset[29](1e4660)
cryp_obj_copy[31](1e4660, 1e3fa0)
cipher_init[21](1e44e0, *40024270:10, 10)
cipher_update[22](1e44e0, *400222b0:10, 10, *400222b0:10,
                  40000e38=10)
```

```
00000000: 1b00 0000 4406 0000 1000 00a0 8000 0000
00000010: 0080 0000 0f00 0000 4447 0600 1001 0010
00000020: 0000 0000 0040 0000 0000 0000 0180 0000
00000030: 1b00 0000 4406 0000 1000 00a0 8000 0000
00000040: 0280 0000 1e00 0000 c704 0000 0240 0000
00000050: 0000 00c0 b000 0001 0100 0000 1d00 0000
00000060: 0700 0000 0040 0000 1f00 0000 7700 0000
00000070: 0040 0000 0240 0000 1500 0000 a704 0000
00000080: 0140 0000 c000 0001 1000 0000 1600 0000
00000090: a764 0a00 0140 0000 d000 0001 1000 0000
000000a0: 0300 0100 e000 8000 ff00 0000 0000 0000
000000b0: 0001 0203 0405 0607 0809 0a0b 0c0d 0e0f
000000c0: 0000 0000 0000 0000 0000 0000 0000 0000
000000d0: 4865 6c6c 6f20 4e75 6c6c 636f 6e21 2100
000000e0: 1000 0000 0000 0000
```

Test case → corpus

```
00000000: 1b00 0000 4406 0000 1000 00a0 8000 0000
00000010: 0080 0000 0f00 0000 4447 0600 1001 0010
00000020: 0000 0000 0040 0000 0000 0000 0180 0000
00000030: 1b00 0000 4406 0000 1000 00a0 8000 0000
00000040: 0280 0000 1e00 0000 c704 0000 0240 0000
00000050: 0000 00c0 b000 0001 0100 0000 1d00 0000
00000060: 0700 0000 0040 0000 1f00 0000 7700 0000
00000070: 0040 0000 0240 0000 1500 0000 a704 0000
00000080: 0140 0000 c000 0001 1000 0000 1600 0000
00000090: a764 0a00 0140 0000 d000 0001 1000 0000
000000a0: 0300 0100 e000 8000 ff00 0000 0000 0000
000000b0: 0001 0203 0405 0607 0809 0a0b 0c0d 0e0f
000000c0: 0000 0000 0000 0000 0000 0000 0000 0000
000000d0: 4865 6c6c 6f20 4e75 6c6c 636f 6e21 2100
000000e0: 1000 0000 0000 0000
```

```
b[0] = malloc(8);
cryp_obj_alloc(0xa0000010, 0x80, b[0]);
b[1] = malloc(8);
cryp_state_alloc(0x10000110, 0x0, *((uint32_t*)b[0]), 0x0, b[1]);
b[2] = malloc(8);
cryp_obj_alloc(0xa0000010, 0x80, b[2]);
cryp_obj_populate(*((uint32_t*)b[2]),
                  {c0000000, "\x00\x01\x02[...]\x0d\x0e\x0f"},
                  0x1);
cryp_obj_reset(*((uint32_t*)b[0]));
cryp_obj_copy(*((uint32_t*)b[0]), *((uint32_t*)b[2]));
t[1] = malloc(16);
memcpy(t[1], "\x00[...]\x00", 16);
cipher_init(*((uint32_t*)b[1]), t[1], 0x10);
free(t[1]);
t[1] = malloc(16);
memcpy(t[1], "Hello Nullcon!!\x00", 16);
b[3] = malloc(16);
t[4] = malloc(8);
memcpy(t[4], "\x10\x00\x00\x00\x00\x00\x00\x00", 8);
cipher_update(*((uint32_t*)b[1]), t[1], 0x10, b[3], t[4]);
```

Test case → corpus

```
00000000: 1b00 0000 4406 0000 1000 00a0 8000 0000
00000010: 0080 0000 0f00 0000 4447 0600 1001 0010
00000020: 0000 0000 0040 0000 0000 0000 0180 0000
00000030: 1b00 0000 4406 0000 1000 00a0 8000 0000
00000040: 0280 0000 1e00 0000 c704 0000 0240 0000
00000050: 0000 00c0 b000 0001 0100 0000 1d00 0000
00000060: 0700 0000 0040 0000 1f00 0000 7700 0000
00000070: 0040 0000 0240 0000 1500 0000 a704 0000
00000080: 0140 0000 c000 0001 1000 0000 1600 0000
00000090: a764 0a00 0140 0000 d000 0001 1000 0000
000000a0: 0300 0100 e000 8000 ff00 0000 0000 0000
000000b0: 0001 0203 0405 0607 0809 0a0b 0c0d 0e0f
000000c0: 0000 0000 0000 0000 0000 0000 0000 0000
000000d0: 4865 6c6c 6f20 4e75 6c6c 636f 6e21 2100
000000e0: 1000 0000 0000 0000
```

```
b[0] = malloc(8);
cryp_obj_alloc(0xa0000010, 0x80, b[0]);
b[1] = malloc(8);
cryp_state_alloc(0x10000110, 0x0, *((uint32_t*)b[0]), 0x0, b[1]);
b[2] = malloc(8);
cryp_obj_alloc(0xa0000010, 0x80, b[2]);
cryp_obj_populate(*((uint32_t*)b[2]),
                  {c0000000, "\x00\x01\x02[...]\x0d\x0e\x0f"},
                  0x1);
cryp_obj_reset(*((uint32_t*)b[0]));
cryp_obj_copy(*((uint32_t*)b[0]), *((uint32_t*)b[2]));
t[1] = malloc(16);
memcpy(t[1], "\x00[...]\x00", 16);
cipher_init(*((uint32_t*)b[1]), t[1], 0x10);
free(t[1]);
t[1] = malloc(16);
memcpy(t[1], "Hello Nullcon!!\x00", 16);
b[3] = malloc(16);
t[4] = malloc(8);
memcpy(t[4], "\x10\x00\x00\x00\x00\x00\x00\x00", 8);
cipher_update(*((uint32_t*)b[1]), t[1], 0x10, b[3], t[4]);
```

Test case → corpus

```
* regression_6001 Test TEE_CreatePersistentObject
o regression_6001.1 Storage id: 00000001
Write trace to /tmp/trace/filevMd213
  regression_6001.1 OK
o regression_6001.2 Storage id: 80000000
Write trace to /tmp/trace/filecVP4s4
  regression_6001.2 OK
  regression_6001 OK

* regression_6002 Test TEE_OpenPersistentObject
o regression_6002.1 Storage id: 00000001
Write trace to /tmp/trace/filej3Ssal
  regression_6002.1 OK
o regression_6002.2 Storage id: 80000000
Write trace to /tmp/trace/fileS5ZNYT
  regression_6002.2 OK
  regression_6002 OK

* regression_6003 Test TEE_ReadObjectData
o regression_6003.1 Storage id: 00000001
Write trace to /tmp/trace/fileTOTS1E
  regression_6003.1 OK
o regression_6003.2 Storage id: 80000000
Write trace to /tmp/trace/fileABYL1H
  regression_6003.2 OK
  regression_6003 OK

* regression_6004 Test TEE_WriteObjectData
o regression_6004.1 Storage id: 00000001
Write trace to /tmp/trace/fileeS1ZaV
  regression_6004.1 OK
o regression_6004.2 Storage id: 80000000
```

DEMO

Trace (in)stability

- Same input should always result in same bitmap output
- However:
  - Threading
  - Interrupts
  - RPC calls
  - Global state

→ AFL thinks input results in new code path while it doesn't!

Trace (in)stability

```
struct tee_ta_session {
    TAILQ_ENTRY(tee_ta_session) link;
    TAILQ_ENTRY(tee_ta_session) link_tsd;
    struct tee_ta_ctx *ctx;
    TEE_Identity clnt_id;
    bool cancel;
    bool cancel_mask;
    TEE_Time cancel_time;
    void *user_ctx;
    uint32_t ref_count;
    struct condvar refc_cv;
    struct condvar lock_cv;
    int lock_thread;
    bool unlink;
#if defined(CFG_AFL_ENABLE)
    struct afl_ctx* afl_ctx;
    struct afl_svc_trace_ctx* svc_trace_ctx;
#endif
};
```

Trace (in)stability

```
struct tee_ta_session {
    TAILQ_ENTRY(tee_ta_session) link;
    TAILQ_ENTRY(tee_ta_session) link_tsd;
    struct tee_ta_ctx *ctx;
    TEE_Identity clnt_id;
    bool cancel;
    bool cancel_mask;
    TEE_Time cancel_time;
    void *user_ctx;
    uint32_t ref_count;
    struct condvar refc_cv;
    struct condvar lock_cv;
    int lock_thread;
    bool unlink;
#if defined(CFG_AFL_ENABLE)
    struct afl_ctx* afl_ctx;
    struct afl_svc_trace_ctx* svc_trace_ctx;
#endif
};
```

Trace (in)stability

```
struct tee_ta_session {
    TAILQ_ENTRY(tee_ta_session) link;
    TAILQ_ENTRY(tee_ta_session) link_tsd;
    struct tee_ta_ctx *ctx;
    TEE_Identity clnt_id;
    bool cancel;
    bool cancel_mask;
    TEE_Time cancel_time;
    void *user_ctx;
    uint32_t ref_count;
    struct condvar refc_cv;
    struct condvar lock_cv;
    int lock_thread;
    bool unlink;
#if defined(CFG_AFL_ENABLE)
    struct afl_ctx* afl_ctx;
    struct afl_svc_trace_ctx* svc_trace_ctx;
#endif
};
```

```
typedef struct afl_ctx {
    bool enabled;
    char bitmap[MAP_SIZE];
    uint64_t prev_loc;
};
```

Trace (in)stability

```
struct tee_ta_session {
    TAILQ_ENTRY(tee_ta_session) link;
    TAILQ_ENTRY(tee_ta_session) link_tsd;
    struct tee_ta_ctx *ctx;
    TEE_Identity clnt_id;
    bool cancel;
    bool cancel_mask;
    TEE_Time cancel_time;
    void *user_ctx;
    uint32_t ref_count;
    struct condvar refc_cv;
    struct condvar lock_cv;
    int lock_thread;
    bool unlink;
#if defined(CFG_AFL_ENABLE)
    struct afl_ctx* afl_ctx;
    struct afl_svc_trace_ctx* svc_trace_ctx;
#endif
};
```

tpidrro_el0 (MSR)

```
typedef struct afl_ctx {
    bool enabled;
    char bitmap[MAP_SIZE];
    uint64_t prev_loc;
};
```

OP-TEE-2018-0005

OP-TEE-2018-0005

```
utee_log(“Hello Nullcon!”, 0xd); -> svc -> syscall_log()
```

OP-TEE-2018-0005

utee_log(“Hello Nullcon!”, 0xd); -> svc -> syscall_log()

```
27  void syscall_log(const void *buf __maybe_unused, size_t len __maybe_unused)
28  {
29  #ifdef CFG_TEE_CORE_TA_TRACE
30          char *kbuf;
31
32          if (len == 0)
33                  return;
34
35          kbuf = malloc(len + 1);
36          if (kbuf == NULL)
37                  return;
38
39          if (tee_svc_copy_from_user(kbuf, buf, len) == TEE_SUCCESS) {
40                  kbuf[len] = '\0';
41                  trace_ext_puts(kbuf);
42          }
43
44          free(kbuf);
45  #endif
46  }
```

OP-TEE-2018-0005

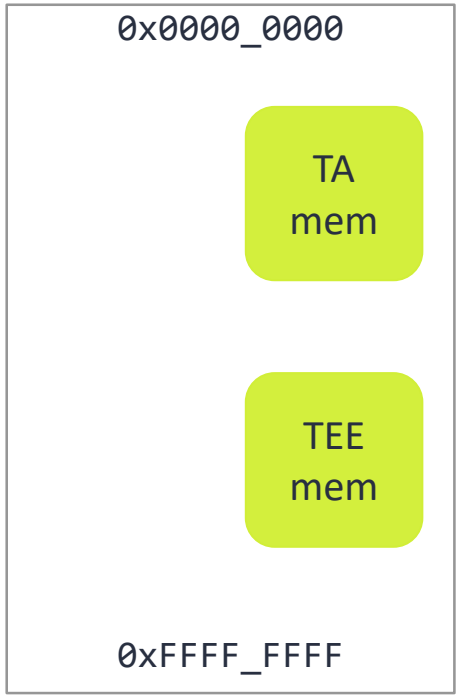
utee_log("Hello Nullcon!", 0xd); -> svc -> syscall_log()

```
27  void syscall_log(const void *buf __maybe_unused, size_t len __maybe_unused)
28  {
29  #ifdef CFG_TEE_CORE_TA_TRACE
30          char *kbuf;
31
32          if (len == 0)
33                  return;
34
35          kbuf = malloc(len + 1);
36          if (kbuf == NULL)
37                  return;
38
39          if (tee_svc_copy_from_user(kbuf, buf, len) == TEE_SUCCESS) {
40                  kbuf[len] = '\0';
41                  trace_ext_puts(kbuf);
42          }
43
44          free(kbuf);
45  #endif
46  }
```

OP-TEE-2018-0005

utee_log("Hello Nullcon!", 0xd); -> svc -> syscall_log()

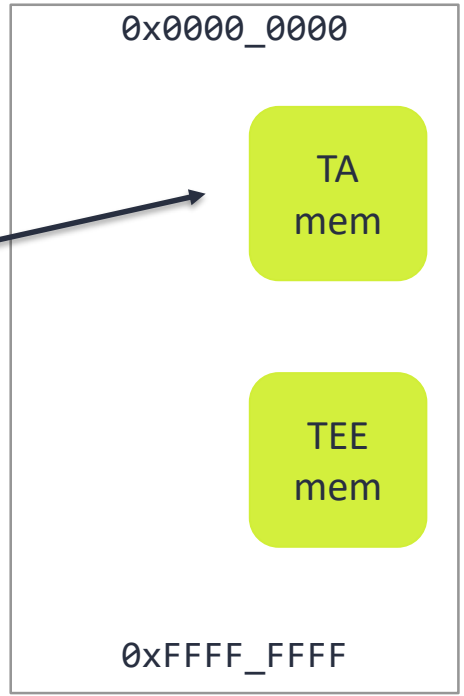
```
27  void syscall_log(const void *buf __maybe_unused, size_t len __maybe_unused)
28  {
29  #ifdef CFG_TEE_CORE_TA_TRACE
30          char *kbuf;
31
32          if (len == 0)
33                  return;
34
35          kbuf = malloc(len + 1);
36          if (kbuf == NULL)
37                  return;
38
39          if (tee_svc_copy_from_user(kbuf, buf, len) == TEE_SUCCESS) {
40                  kbuf[len] = '\0';
41                  trace_ext_puts(kbuf);
42          }
43
44          free(kbuf);
45  #endif
46  }
```



OP-TEE-2018-0005

utee_log("Hello Nullcon!", 0xd); -> svc -> syscall_log()

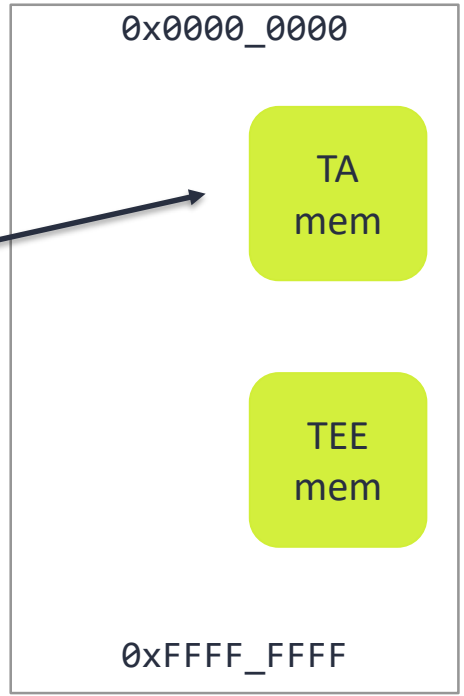
```
27  void syscall_log(const void *buf __maybe_unused, size_t len __maybe_unused)
28  {
29  #ifdef CFG_TEE_CORE_TA_TRACE
30          char *kbuf;
31
32          if (len == 0)
33                  return;
34
35          kbuf = malloc(len + 1);
36          if (kbuf == NULL)
37                  return;
38
39          if (tee_svc_copy_from_user(kbuf, buf, len) == TEE_SUCCESS) {
40                  kbuf[len] = '\0';
41                  trace_ext_puts(kbuf);
42          }
43
44          free(kbuf);
45  #endif
46  }
```



OP-TEE-2018-0005

utee_log("Hello Nullcon!", 0xd); -> svc -> syscall_log()

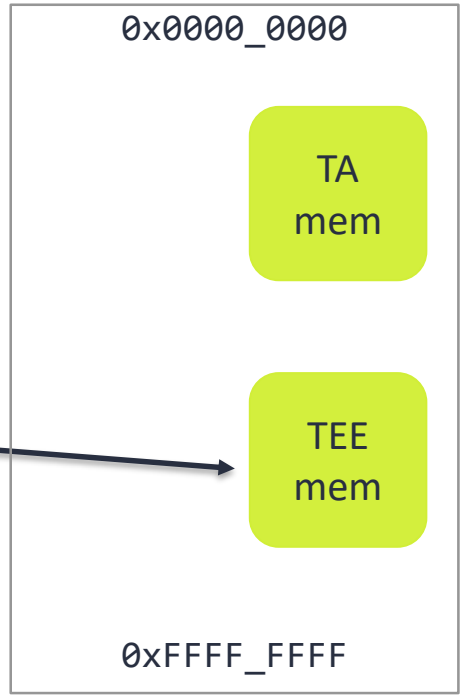
```
27  void syscall_log(const void *buf __maybe_unused, size_t len __maybe_unused)
28  {
29  #ifdef CFG_TEE_CORE_TA_TRACE
30          char *kbuf;
31
32          if (len == 0)
33                  return;
34
35          kbuf = malloc(len + 1);
36          if (kbuf == NULL)
37                  return;
38
39          if (tee_svc_copy_from_user(kbuf, buf, len) == TEE_SUCCESS) {
40                  kbuf[len] = '\0';
41                  trace_ext_puts(kbuf);
42          }
43
44          free(kbuf);
45  #endif
46  }
```



OP-TEE-2018-0005

utee_log("Hello Nullcon!", 0xd); -> svc -> syscall_log()

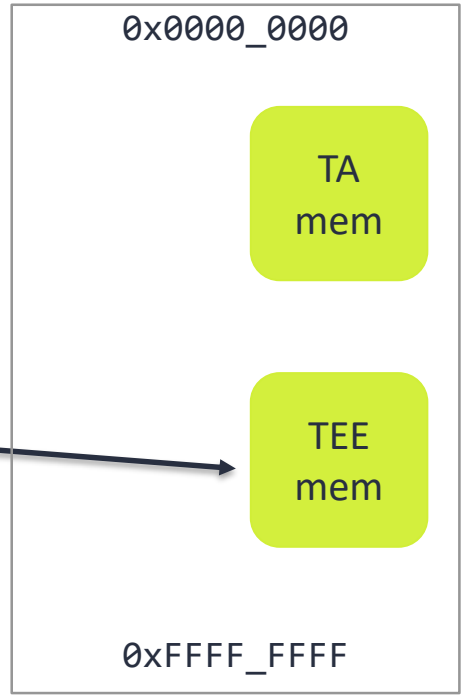
```
27  void syscall_log(const void *buf __maybe_unused, size_t len __maybe_unused)
28  {
29  #ifdef CFG_TEE_CORE_TA_TRACE
30          char *kbuf;
31
32          if (len == 0)
33                  return;
34
35          kbuf = malloc(len + 1);
36          if (kbuf == NULL)
37                  return;
38
39          if (tee_svc_copy_from_user(kbuf, buf, len) == TEE_SUCCESS) {
40                  kbuf[len] = '\0';
41                  trace_ext_puts(kbuf);
42          }
43
44          free(kbuf);
45  #endif
46  }
```



OP-TEE-2018-0005

utee_log("Hello Nullcon!", 0xd); -> svc -> syscall_log()

```
27  void syscall_log(const void *buf __maybe_unused, size_t len __maybe_unused)
28  {
29  #ifdef CFG_TEE_CORE_TA_TRACE
30          char *kbuf;
31
32          if (len == 0)
33                  return;
34
35          kbuf = malloc(len + 1);
36          if (kbuf == NULL)
37                  return;
38
39          if (tee_svc_copy_from_user(kbuf, buf, len) == TEE_SUCCESS) {
40                  kbuf[len] = '\0';
41                  trace_ext_puts(kbuf);
42          }
43
44          free(kbuf);
45  #endif
46  }
```



OP-TEE-2018-0005

```
27  void syscall_log(const void *buf __maybe_unused, size_t len __maybe_unused)
28  {
29  #ifde
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45  #endif
46  }
```

```
897  TEE_Result tee_svc_copy_from_user(void *kaddr, const void *uaddr, size_t len)
898  {
899          TEE_Result res;
900          struct tee_ta_session *s;
901
902          res = tee_ta_get_current_session(&s);
903          if (res != TEE_SUCCESS)
904                  return res;
905
906          res = tee_mmu_check_access_rights(to_user_ta_ctx(s->ctx),
907                                            TEE_MEMORY_ACCESS_READ |
908                                            TEE_MEMORY_ACCESS_ANY_OWNER,
909                                            (uaddr_t)uaddr, len);
910          if (res != TEE_SUCCESS)
911                  return res;
912
913          memcpy(kaddr, uaddr, len);
914          return TEE_SUCCESS;
915  }
```

OP-TEE-2018-0005

```
TEE_Result tee_mmu_check_access_rights(const struct user_ta_ctx *utc,
                                       uint32_t flags, uaddr_t uaddr, size_t len) {
    uaddr_t a;
    size_t addr_incr = MIN(CORE_MMU_USER_CODE_SIZE,
                           CORE_MMU_USER_PARAM_SIZE);

    if (ADD_OVERFLOW(uaddr, len, &a))
        return TEE_ERROR_ACCESS_DENIED;

    for (a = uaddr; a < (uaddr + len); a += addr_incr) {
        res = tee_mmu_user_va2pa_attr(utc, (void *)a, NULL, &attr);
        if (res != TEE_SUCCESS)
            return res;

        // check attributes of the page
        [..]
    }

    return TEE_SUCCESS;
}
```

OP-TEE-2018-0005

```
for (a = uaddr; a < (uaddr + len); a += addr_incr) {
        res = tee_mmu_user_va2pa_attr(utc, (void *)a, NULL, &attr);
        if (res != TEE_SUCCESS)
            return res;

        // check attributes of the page
        [..]
    }
```

OP-TEE-2018-0005

```
for (a = uaddr; a < (uaddr + len); a += addr_incr) {
        res = tee_mmu_user_va2pa_attr(utc, (void *)a, NULL, &attr);
        if (res != TEE_SUCCESS)
            return res;

        // check attributes of the page
        [..]
    }
```

OP-TEE-2018-0005

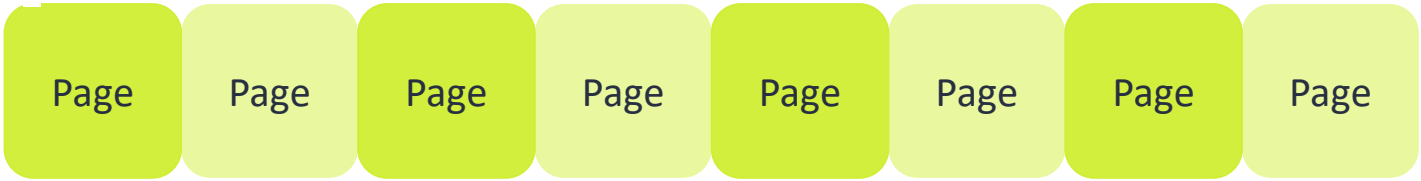
4 KiB (recent versions)
1 MiB (older versions)

```
for (a = uaddr; a < (uaddr + len); a += addr_incr) {
        res = tee_mmu_user_va2pa_attr(utc, (void *)a, NULL, &attr);
        if (res != TEE_SUCCESS)
            return res;

        // check attributes of the page
        [..]
    }
```

OP-TEE-2018-0005

4 KiB (recent versions)
1 MiB (older versions)

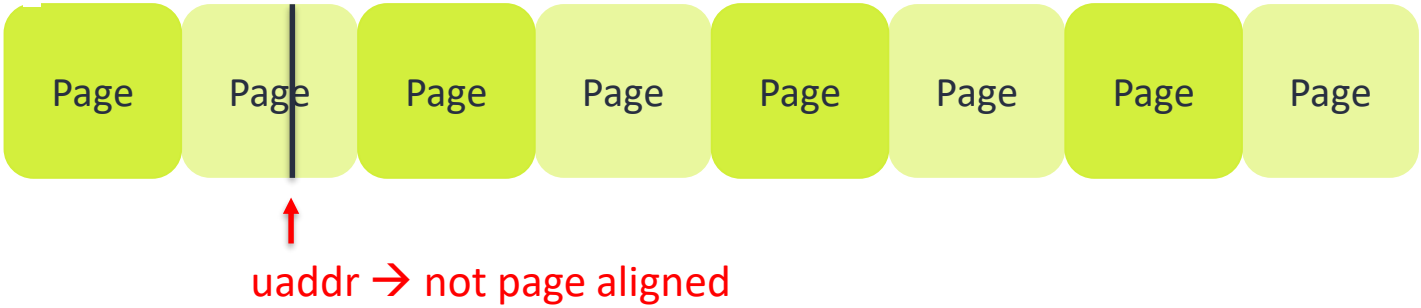
```
for (a = uaddr; a < (uaddr + len); a += addr_incr) {
        res = tee_mmu_user_va2pa_attr(utc, (void *)a, NULL, &attr);
        if (res != TEE_SUCCESS)
            return res;

        // check attributes of the page
        [..]
    }
```



OP-TEE-2018-0005

4 KiB (recent versions)
1 MiB (older versions)

```
for (a = uaddr; a < (uaddr + len); a += addr_incr) {
        res = tee_mmu_user_va2pa_attr(utc, (void *)a, NULL, &attr);
        if (res != TEE_SUCCESS)
            return res;

        // check attributes of the page
        [..]
    }
```

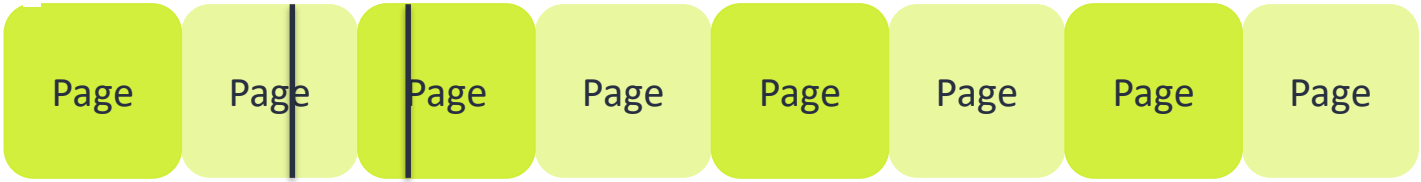


OP-TEE-2018-0005

4 KiB (recent versions)
1 MiB (older versions)

```
for (a = uaddr; a < (uaddr + len); a += addr_incr) {
        res = tee_mmu_user_va2pa_attr(utc, (void *)a, NULL, &attr);
        if (res != TEE_SUCCESS)
            return res;

        // check attributes of the page
        [..]
    }
```

spans 2+ pages

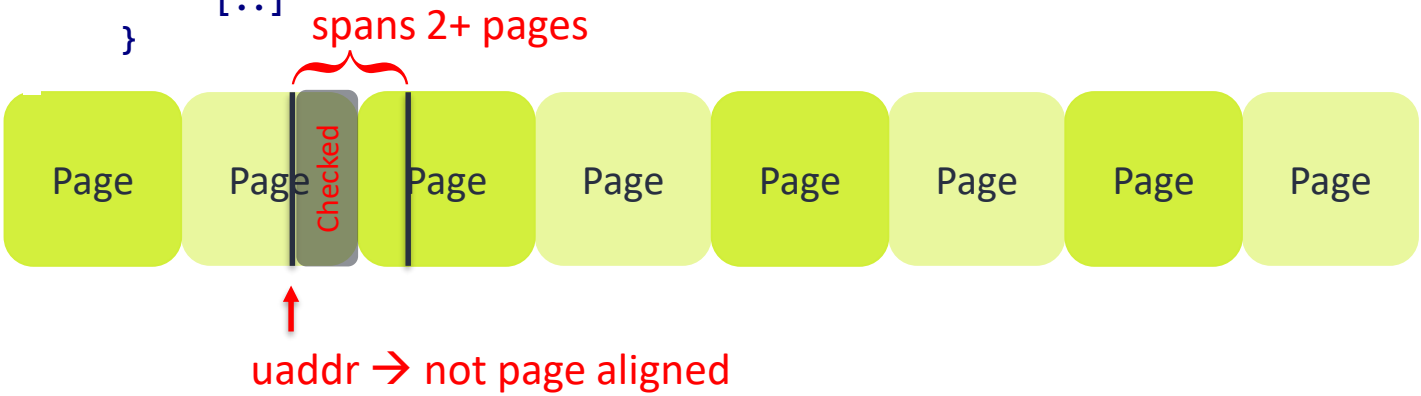


uaddr → not page aligned

OP-TEE-2018-0005

4 KiB (recent versions)
1 MiB (older versions)

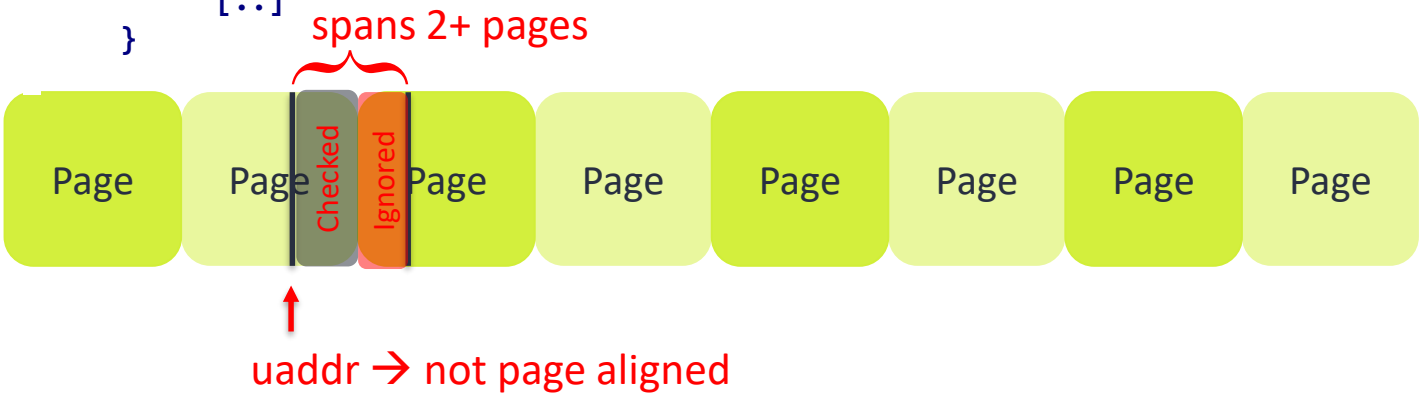
```
for (a = uaddr; a < (uaddr + len); a += addr_incr) {
        res = tee_mmu_user_va2pa_attr(utc, (void *)a, NULL, &attr);
        if (res != TEE_SUCCESS)
            return res;

        // check attributes of the page
        [..]
    }
```



OP-TEE-2018-0005

4 KiB (recent versions)
1 MiB (older versions)

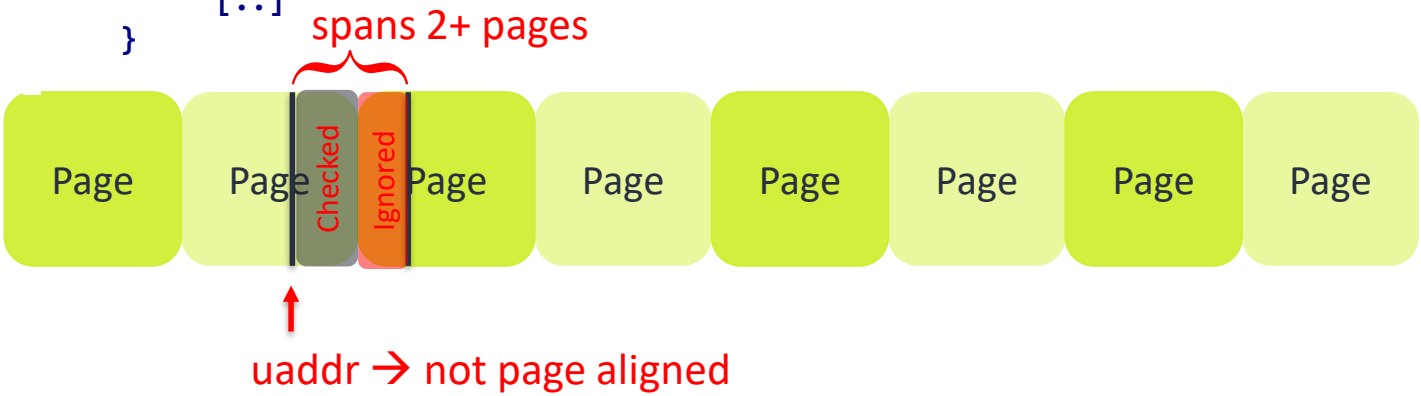
```
for (a = uaddr; a < (uaddr + len); a += addr_incr) {
        res = tee_mmu_user_va2pa_attr(utc, (void *)a, NULL, &attr);
        if (res != TEE_SUCCESS)
            return res;

        // check attributes of the page
        [..]
    }
```



OP-TEE-2018-0005

4 KiB (recent versions)
1 MiB (older versions)

```
for (a = uaddr; a < (uaddr + len); a += addr_incr) {
        res = tee_mmu_user_va2pa_attr(utc, (void *)a, NULL, &attr);
        if (res != TEE_SUCCESS)
            return res;

        // check attributes of the page
        [..]
    }
```



Fixed in OP-TEE 3.4.0

But which parts did we fuzz?

We know already which parts are covered by each input.
Can we aggregate and visualize this information?

Coverage tracking

IDA - tee.elf C:\Users\Martijn\Desktop\tee.elf

File Edit Jump Search View Debugger Lumina Options Windows Help

No debugger

Library function | Regular function | Instruction | Data | Unexplored | External symbol | Lumina function

Functions window

Function name

- __text_start
- unhandled_cpu
- **init**
- thread_excp_vect
- reset_vect_table
- **init_user_ta**
- **resolve_symbol**
- to_user_ta_ctx
- **user_ta_get_instance_id**
- user_ta_enter_close_session
- user_ta_dump_state
- free_utc
- user_ta_ctx_destroy
- user_ta_enter_invoke_cmd
- user_ta_enter_open_session
- is_user_ta_ctx
- **tee_ta_register_ta_store**
- **tee_ta_init_user_ta_session**
- **ta_get_size**
- **register_supplicant_user_ta**
- **ta_read**
- **ta_open**
- **ta_close**
- copy_in_param
- unmap_mapped_param
- **verify_pseudo_tas_conformance**
- to_pseudo_ta_ctx
- **pseudo_ta_destroy**
- **pseudo_ta_enter_close_session**
- **pseudo_ta_enter_open_session**
- **pseudo_ta_enter_invoke_cmd**

Line 1032 of 1110

AU: idle | Down | Disk: 207GB

IDA View-A | Coverage Overview | Hex View-1 | Structures | Enums | Imports | Exports

Coverage %	Function Name	Address	Blocks Hit	Instructions Hit	Function Size	Complexity
10.03	tee_fs_htree_o…	0xE14CA10	33 / 323	159 / 1585	6352	207
26.68	__gcm_update_p…	0xE12B640	29 / 89	107 / 401	1604	54
25.19	omac_init	0xE17DDA0	23 / 88	101 / 401	1604	47
11.40	crypto_rng_read	0xE12DCF8	21 / 162	97 / 851	3432	97
26.92	syscall_cryp_s…	0xE13A968	35 / 83	91 / 338	1356	49
2.27	tee_fs_htree_s…	0xE149458	16 / 712	78 / 3436	13752	497
36.71	syscall_cryp_o…	0xE1396D0	17 / 41	76 / 207	828	22
25.99	__gcm_init	0xE12BC88	18 / 47	72 / 277	1108	27
29.66	tee_svc_cryp_c…	0xE134880	18 / 53	70 / 236	944	31
22.33	crypto_cipher_…	[illegible]	[illegible]	[illegible]	1260	36
5.54	rijndael_setup	[illegible]	21 / [illegible]	[illegible] / 173	4692	134
47.76	tee_fs_fek_cry…	[illegible]	[illegible]	[illegible] / 34	544	12
28.44	tee_obj_set_ty…	[illegible]	[illegible]	[illegible]	928	28
17.28	hmac_done	0xE17CC54	26 / 111	61 / 353	1412	64
13.24	ctr_start	0xE180E7C	22 / 134	58 / 438	1752	77
15.98	hmac_init	0xE17D1D8	24 / 116	58 / 363	1452	65
35.03	authenc_init	0xE1447A0	14 / 25	55 / 157	640	14
20.38	syscall_cipher…	0xE13BE20	16 / 51	54 / 265	1060	28
10.62	tee_svc_cryp_o…	0xE134C30	17 / 107	53 / 499	2008	62
13.61	syscall_storag…	0xE13FE48	18 / 77	52 / 382	1536	43
40.98	ree_fs_open_pr…	0xE142458	11 / 19	50 / 122	488	9
30.49	crypto_mac_init	0xE16F670	12 / 39	50 / 164	676	22
25.65	syscall_hash_i…	0xE13B460	15 / 42	49 / 191	764	22
39.20	tee_fs_rpc_cac…	0xE151600	16 / 26	49 / 125	500	15
17.56	syscall_hash_f…	0xE13B9C0	14 / 56	49 / 279	1116	34
31.79	cryp_state_free	0xE135BA8	15 / 33	48 / 151	604	16
22.01	crypto_aes_gcm…	0xE12C410	11 / 38	46 / 209	836	22
30.26	syscall_hash_u…	0xE13B760	15 / 34	46 / 152	608	19
20.55	syscall_storag…	0xE13FAD0	14 / 41	45 / 219	884	23

DEMO

Composer

A - 3.50% - Aggregate

Lighthouse plugin for IDA Pro - https://github.com/gaasedelen/lighthouse

Ongoing work

- Fix remaining issues
  - Setting multiple crypto props in 1 call not supported
  - Thread support remains buggy

- Upstream patches
  - Repository: https://github.com/MartijnB/optee_fuzzer

- Generalize framework beyond OP-TEE / syscalls
  - Separate platform specific information **(mostly done)**
  - Arbitrarily nested structures / arrays
  - Context aware mutations (AST, ...)

riscure

We are hiring!

Thank you! Any questions?

Or come visit our booth to talk!

Martijn Bogaard

Senior Security Analyst

martijn@riscure.com / @jmartijnb