Using the SMC Fuzzing Module in TF-A

Kathleen Capella

July 11, 2024

What this presentation is not

- A presentation on the merits of fuzzing
- An analysis of the SMC fuzzer’s bug-finding capabilities

What this presentation is

- Demonstration of
  - Basic fuzzer features
  - Adding SMC calls/test cases
  - Integration with tf-a-tests and platform-ci
- In other words, showing you how to get started with using the TF-A SMC fuzzing module

arm

Fuzzer Components

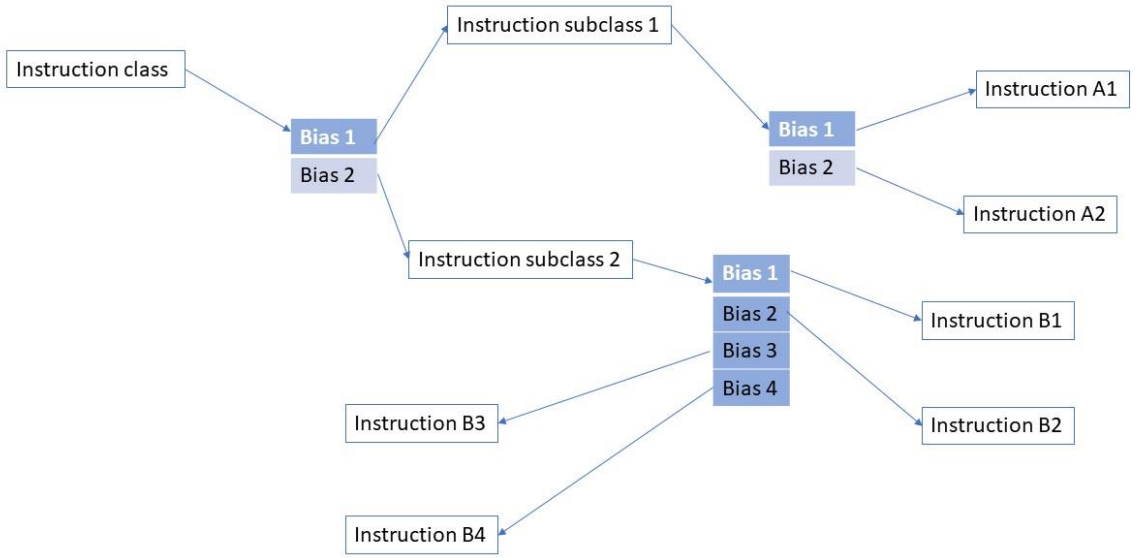
Fuzzing Definition

- What is Fuzzing?
  - Take random, invalid, or unexpected data → program → look for hangs, crashes, assertion fails, and memory leaks
- Why fuzz?
  - Generate unforeseen test cases via automation
- What should fuzzer input look like?
  - Structured
  - Varying degrees of validity

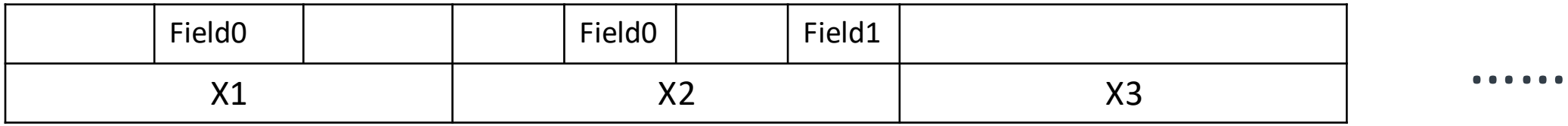
```
name = input(“Enter your username:”)
if name in users:
        password = input(“Password?”)
        if password == users[name].password
                command = input(“Command?”)
                os.popen(command)
```

SMC Fuzzer Components

- Q: How does the fuzzer choose which order to run SMC calls in?
  - A: Bias tree
- Q: How does the fuzzer generate input?
  - Q: How is input structured?
  - Q: How valid should input be?
    - A: Sanity level and Constraints



Structure of SMC calls



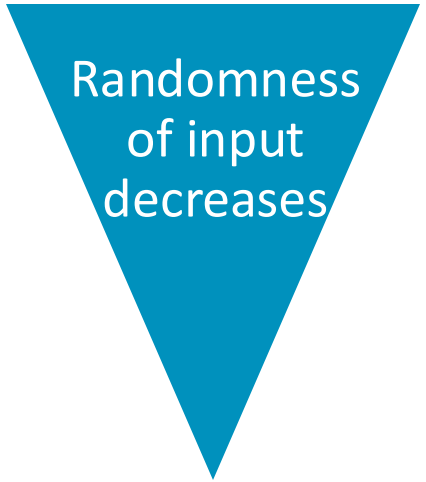
- Registers contain the arguments to SMC calls
- Each register may have one or more bitfields of varying widths
- Sanity level determines how the generated SMC arguments are randomized by the fuzzer

How valid should input be?

	Field0			Field0		Field1	
X1			X2				X3

.....

Sanity Level	Description
0	Registers fully randomized
1	One register (chosen at random) randomized based on fields
2	All registers randomized based on fields (rest of the register is 0 if not included in a field)
3	Fully constrained by the developer using constraints



How can we specify useful inputs?

- Add constraints to an input field
- Types of Constraints
  - Range of values
  - Single Value
  - Vector of Values
- Example:
  - A platform supports interrupts with IDs in the range 1-16
  - To constrain an interrupt_bind function you might
    - Restrict interrupt ID field to range 1-16
      - Or for negative testing, always 17-32 etc
    - Restrict interrupt ID field to always be 1
    - Restrict interrupt ID field to a vector of 1,3,5 etc

Adding SMC Calls to TF-A Tests

Setup

- Create bias tree with weights for each SMC function
  - Can have separate bias tree for a subclass of SMCs to be run by itself OR add to tree with another subclass to mix calls from different subclasses
    - FF-A calls only or mixed with SDEI, TSP, etc

```
dts-v1;

 {

        ffa {
                bias = <30>;
                ffa_msg_send_direct_req {
                        bias = <30>;
                        functionname = "ffa_msg_send_direct_req_funcid";
                };
                ffa_msg_send_direct_resp {
                        bias = <30>;
                        functionname = "ffa_msg_send_direct_resp_funcid";
                };
                ffa_features_feat_id {
                        bias = <15>;
                        functionname = "ffa_features_feat_id_funcid";
                };
                ffa_features_func_id {
                        bias = <15>;
                        functionname = "ffa_features_func_id_funcid";
                };
```

Setup continued – SMC descriptor file

SMC descriptor file

```
smc: FFA_NOTIFICATION_BIND_CALL
        arg1:sender_receiver
                field:sender_id:[16,31] = 0
                field:receiver_id:[0,15] = 0
        arg2:flags
                field:per_vcpu_notifications:[0,0] = 0
        arg3:notification_bitmap_lo
                field:bitmap:[0,31] = 0xAAAA
        arg4:notification_bitmap_hi
                field:bitmap:[0,31] = 0x5555
        arg5-arg17 = 0
```

FF-A specification

Table 16.11: FFA_NOTIFICATION_BIND function syntax

Parameter	Register	Value
Uint32 Sender/Receiver IDs	W1	Sender and Receiver endpoint IDs. – Bit[31:16]: Sender endpoint ID. – Bit[15:0]: Receiver endpoint ID.
uInt32 Flags	W2	Notification flags. – Bit[0]: Per-vCPU notification flag (see 10.4.2 Notification binding). * b'1: All notifications in the bitmap are per-vCPU notifications * b'0: All notifications in the bitmap are global notifications – Bit[31:1]: Reserved (SBZ).
Uint32 Notification bitmap Lo	W3	Bits[31:0] of a bitmap with one or more set bits to identify the notifications which the Sender endpoint is allowed to signal.
Uint32 Notification bitmap Hi	W4	Bits[63:32] of a bitmap with one or more set bits to identify the notifications which the Sender endpoint is allowed to signal.
Other parameter registers	W5-w7 X5-x17	Reserved (SBZ)

Steps for each fuzzing call

- Set the constraints
- Generate arguments
- Do SMC call
- Retrieve generated arguments (optional)
- Analyze/print results

Set the constraints

Generate arguments

SMC call

```
}else if (funcid == ffa_notification_bind_funcid) {
        uint64_t sps[] = {SP_ID(1), SP_ID(2), SP_ID(3)};
        uint64_t max_vm_id = (1 << 16) - 1;
        uint64_t vm_ids[] = {0, max_vm_id};
        uint64_t max_bitmap_value = 0xFFFFFFFF;
        uint64_t bitmap_range[] = {0, max_bitmap_value};

        setconstraint(FUZZER_CONSTRAINT_RANGE, vm_ids, 2, FFA_NOTIFICATION_BIND_CALL_ARG1_RECEIVER_ID, mmod,
            FUZZER_CONSTRAINT_ACCMODE);
        setconstraint(FUZZER_CONSTRAINT_RANGE, vm_ids, 2, FFA_NOTIFICATION_BIND_CALL_ARG1_SENDER_ID, mmod,
            FUZZER_CONSTRAINT_ACCMODE);
        setconstraint(FUZZER_CONSTRAINT_VECTOR, sps, 3, FFA_NOTIFICATION_BIND_CALL_ARG1_RECEIVER_ID, mmod,
            FUZZER_CONSTRAINT_ACCMODE);
        setconstraint(FUZZER_CONSTRAINT_VECTOR, sps, 3, FFA_NOTIFICATION_BIND_CALL_ARG1_SENDER_ID, mmod,
            FUZZER_CONSTRAINT_ACCMODE);
        setconstraint(FUZZER_CONSTRAINT_RANGE, bitmap_range, 2, FFA_NOTIFICATION_BIND_CALL_ARG3_BITMAP, mmod,
            FUZZER_CONSTRAINT_EXCMODE);
        setconstraint(FUZZER_CONSTRAINT_RANGE, bitmap_range, 2, FFA_NOTIFICATION_BIND_CALL_ARG4_BITMAP, mmod,
            FUZZER_CONSTRAINT_EXCMODE);

        struct inputparameters ip = generate_args(FFA_NOTIFICATION_BIND_CALL, SMC_FUZZ_SANITY_LEVEL);

        struct ffa_value ret = ffa_call_with_params(ip, FFA_NOTIFICATION_BIND);

        if(ret.fid == FFA_SUCCESS_SMC32 || ret.fid == FFA_SUCCESS_SMC64){
                printf("FFA_NOTIFICATION_BIND succeeded\n");
        }else if (ret.fid == FFA_ERROR){
                printf("FFA_NOTIFICATION_BIND returned with FFA_ERROR code %d\n", ffa_error_code(ret));
        }else{
                printf("FAIL FFA_NOTIFICATION_BIND returned with 0x%lx\n", ret.fid);
        }
```

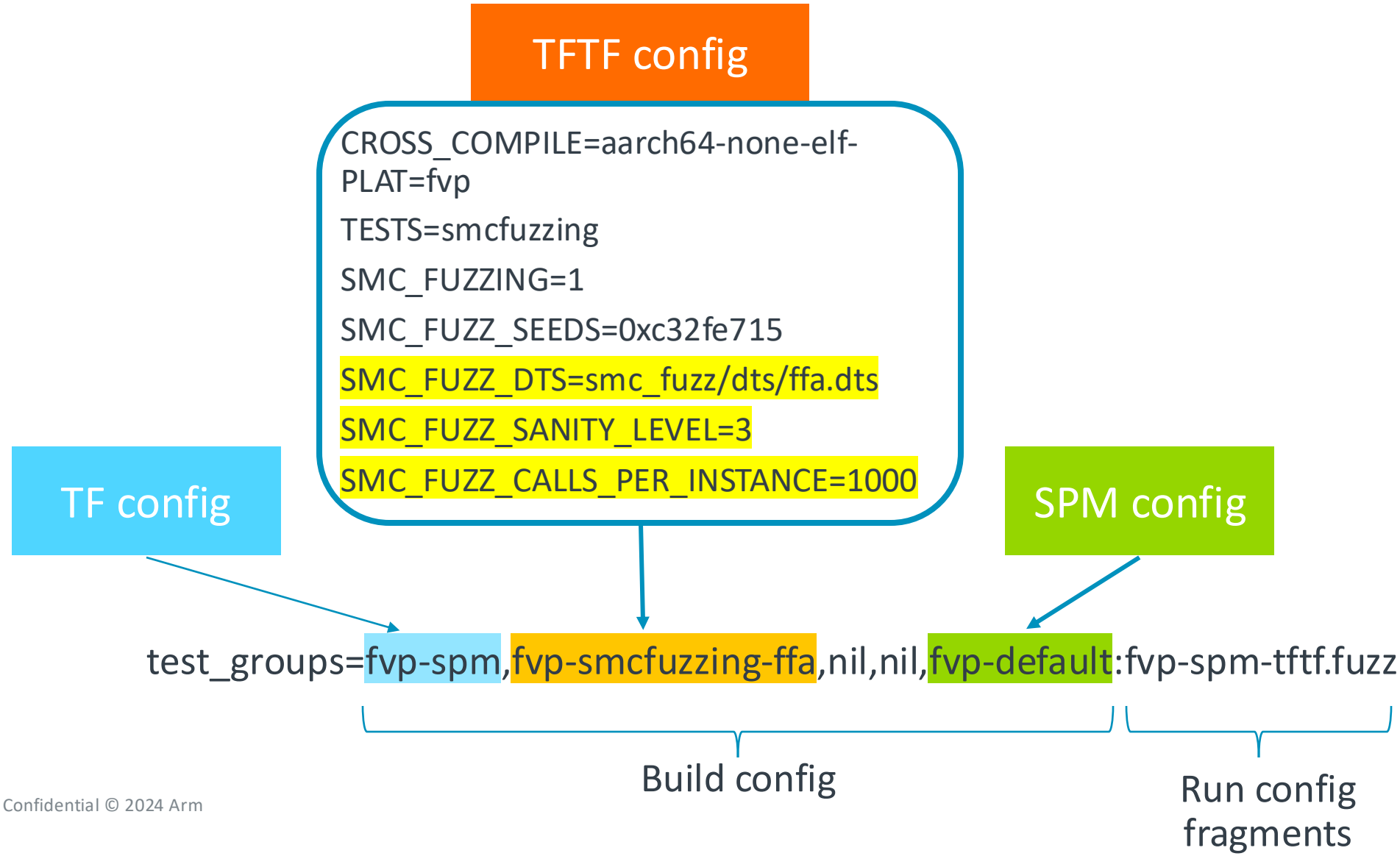
Running with CI

Integration with TF-A Tests and CI

- Can reuse existing build components
- Link with helpers from TF-A tests specific to your use case

```
}else if (funcid == ffa_msg_send_direct_req_funcid) {
                uint64_t receiver_ids[] = {SP_ID(1), SP_ID(2), SP_ID(3)};
                uint64_t sender_id[] = {HYP_ID};
                uint64_t message_type[] = {0, 1};
                uint64_t frmwrk_msg_type[] = {0,1};
                uint64_t msg_0_input[] = {CACTUS_ECHO_CMD, CACTUS_REQ_ECHO_CMD}; /* Cactus cmd */
                uint64_t msg_1_input[] = {100,200}; /* for echo cmds, echo_val */
```

Build Flow with Platform CI



Output

References

- https://en.wikipedia.org/wiki/Fuzzing

- Internal resources:
  - Confluence page on how to add SMC instructions to the Tf-A Fuzzer
    - https://confluence.arm.com/display/CESW/Adding+SMC+instructions+to+Fuzzer+module
  - Patches
    - FF-A https://gerrit.oss.arm.com/q/topic:%22kc%252Ffuzz%22+(status:open%20OR%20status:merged)
    - TF-A tests https://gerrit.oss.arm.com/c/trusted-firmware/tf-a-tests/+/285971/5
    - Platform-CI https://gerrit.oss.arm.com/c/pdswinf/ci/pdcs-platforms/platform-ci/+/280969

Thank You
Danke
Gracias
Grazie
谢谢
ありがとう
Asante
Merci
감사합니다
धन्यवाद
Kiitos
شكرًا
ধন্যবাদ
תודה
ధన్యవాదములు