SMC Fuzzing

Overview of fuzzing in TF-A

Mark Dykes

6-4-2025

Agenda:

- Rundown of Fuzzer fundamentals
- Show features of SMC fuzzer and current status
- Bias tree explanation and examples
- SMC call invocations
- Sanity level description
- Argument constrains
- Error Handling
- References

What is Fuzzing?

Various points of interest:

- Fuzzing is the ability to generate random inputs for a given software or hardware module
- Has capability to control the amount of randomness provided to a module
- Is used industry wide for both software and hardware
- Is used in tandem with directed testing to find bugs
- Currently utilized in SDEI testing and SPM(Hafnium)

TF-A SMC Fuzzer

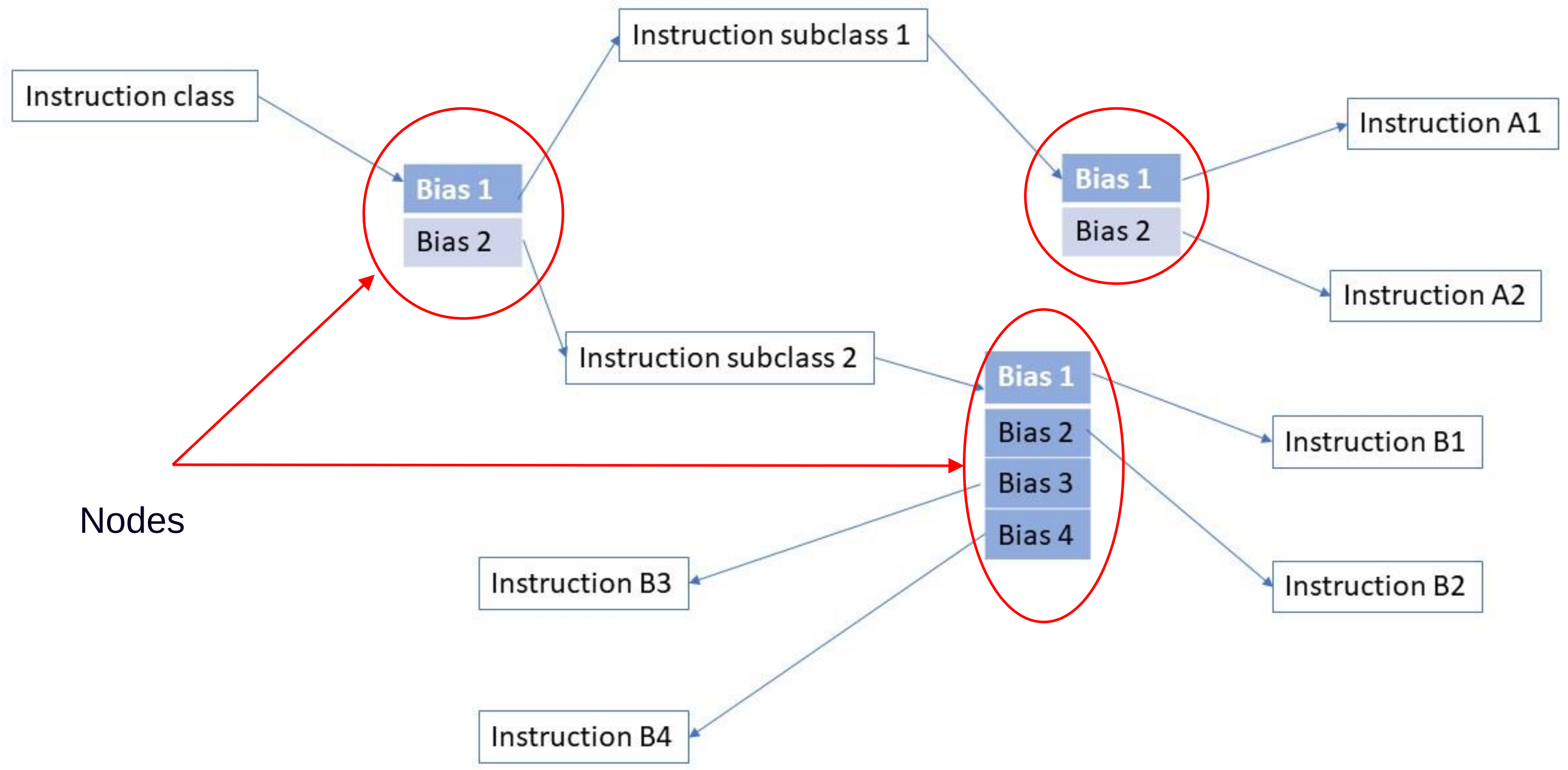
- Integrated into TF-A tests as an explicit tool for fuzzing SMC calls
- Uses bias tree model to choose amongst flavors of SMC that can be layered based on many criteria
- Testing types can be flexible where combinations of categories can be individually emphasized
- Sanity levels give the user capability to apply constrained randomness at higher values
- Granularities of single SMC calls to multiple groups is feasible
- Highest sanity level gives the user the ability to apply differing constraints types to input arguments
- At present no constraint solver

Current status:

- Fuzzing feature sets for SDEI and FFA
- Found several bugs in SDEI, running more extensive tests for FFA
- Evaluation for usage in other feature sets

Diagram of Bias Tree

- All bias references are numerical and are only meaningful when cast against the other biases in the given node. It is possible to have a bias of zero which disables that option(instruction or class)



Generic example of Bias tree

Uses Device Tree format

- In this example the fuzzer chooses the first level where smc_var1 is weighed against the other SMC top level calls and then to smc_var1_var1, smc_var1_var2, and smc_var1_var3 and then finally to smc_var1_var3_var1 and smc_var1_var3_var2 if smc_var1_var3 is selected

```
smc_var1 {
    bias = <65>;
    smc_var1_var1 {
        bias = <30>;
        functionname = "smc_var1_var1";
    };
    smc_var1_var2 {
        bias = <30>;
        functionname = "smc_var1_var2";
    };
    smc_var1_var3 {
        bias = <35>;
        smc_var1_var3_var1 {
            bias = <30>;
            functionname = "smc_var1_var3_var1";
        };
        smc_var1_var3_var2 {
            bias = <30>;
            functionname = "smc_var1_var3_var2";
        };
```

Real World example of Bias tree

SDEI focus

- Here is the representation of a bias tree where all SMC calls have an equal chance of being selected provided that SDEI is chosen. The functionname is used by the fuzzer to call the actual SMC in another c file. It is bound to an integer automatically assigned.

```
sdei {
        bias = <30>;
        sdei_version {
            bias = <7>;
            functionname = "sdei_version_funcid";
        };
        sdei_pe_unmask {
            bias = <7>;
            functionname = "sdei_pe_unmask_funcid";
        };
        sdei_pe_mask {
            bias = <7>;
            functionname = "sdei_pe_mask_funcid";
        };
        sdei_event_status {
            bias = <7>;
            functionname = "sdei_event_status_funcid";
        };
```

Implementing SMC Calls issued from Fuzzer

Continued SDEI example

- Typically in a file devoted to that specific feature set the calls are referenced from the fuzzer output that was created in the bias tree file. Each call has a corresponding funcid designation.

```
void run_sdei_fuzz(int funcid, struct memmod *mmod, bool inrange, int cntid)
{
        long long ret;
        if (funcid == sdei_version_funcid) {
                ret = sdei_version();

                if (ret != MAKE_SDEI_VERSION(1, 0, 0)) {
                        tftf_testcase_printf("Unexpected SDEI version: 0x%llx\n", ret);
                }
        }
        } else if (funcid == sdei_pe_unmask_funcid) {
                ret = sdei_pe_unmask();
        }
        } else if (funcid == sdei_pe_mask_funcid) {
                ret = sdei_pe_mask;
        }
```

Specifying arguments and Sanity Level

- Each SMC call has a setup of registers and fields that can be fuzzed in a constrained random fashion or fully randomized depending on the sanity level. Only sanity level 3 offers the constraint capability.

- Sanity Level 0 – Most random. All field boundaries are ignored in the registers and each register is given a fully random value.
- Sanity Level 1 – Same as level 0 but one register is randomly chosen where the field values are observed and randomization is applied only to those widths.
- Sanity Level 2 – All of the fields in the registers are honored and randomization on those is applied leaving the reserved bits zero(or one)
- Sanity Level 3- Most constrained. The user can specify a single value, a range, or a vector or some combination of the three to the fields

How to specify the format of an SMC call

- A separate text file containing the feature set calls would be created where each SMC would have the following format:

- smc: <name of call>
- arg<register number 1-17>: <name of register>
- field:<name of field>:[<starting bit>,<ending bit>] = <default value decimal or 0x>

- Each register argument must be specified and the fields contained within. A default value must also be supplied for reserved bits and/or bits that are to be static at sanity level 3. This information will be leveraged when constraints are applied in the previous system call c file.

Constraint application for Sanity Level 3

- When applying constraints the user can customize the randomization based upon factors that might be known only at the run time of the SMC call. This could be the state after previous calls or system state. The most ideal would be to have SMC invocations that are as independent of each other as possible so as to limit the complexity of how/when each call is executed and the required register values. However it is possible to apply calls that are interdependent in a stricter fashion but that works against the purpose of the tool to find bugs outside the scope of "desired behavior". Here is the makeup of a constraint specification:

- setconstraint(<constraint type>,<constraint>,<length of constraint>,<field>,
- <memory pointer(user just passed without consideration)>,<constraint mode>)

The constraint type is one of:

Single value
Range of values
Vector of values

Multiple of these can be specified and the fuzzer will choose amongst them in an equal random fashion

The other field of note is the constraint mode which can be one of:

Accumulation
Exclusive

- Accumulation is the mode that gathers all the constraints applied to be chosen randomly by the fuzzer. This could be any number of any of the constraint types mentioned. The user has no control of the biases within since each has equal chance of being selected.
- Exclusive only allows one constraint to be applied at a time and erases all previous constraints given before.

It should be noted that only sanity level 3 will allow constraint application.

Error handling from fuzzer output

- All sanity levels permit calling the generation of the arguments before the SMC in the test. This permits prediction of results from the call and subsequent handling if encountering an expected or unexpected result.

- Much of the testing will potentially prompt either assertion fails or hangs in the test which are not visible to the calling module. The error handling in the fuzzer can take the form:

```
ret = sdei_event_signal(inp.x2);
        if (ret < 0) {
            tftf_testcase_printf("sdei_event_signal failed: %lld\n", ret);
        }
```

- Where handling can manifest panics or print warning or errors. This allows finding errors that are less catastrophic but no less informative.

References:

Read the docs for further details on fuzzing capability:
https://trustedfirmware-a-tests.readthedocs.io/en/latest/fuzzing/index.html

Previous presentation on FFA fuzzing:
https://www.trustedfirmware.org/docs/Fuzzing-Tech-Forum-11Jul24.pdf

arm

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