October 17, 2019

Jared Wright
Software Engineer/Researcher

ais

WWW.AINFOSEC.COM

# Scapula

## An Open-Source Toolkit for Model Based Fuzzing and Verification of ARM CPUs

# Who am I?

- Software Engineer/Researcher
  - Assured Information Security, Inc (AIS)
  - Funded in part by AFRL/DARPA: FA8750-17-C-0260
  - All expressed opinions are my own, and do not necessarily reflect those of the United States Department of Defense or AIS

- Interests
  - Hypervisors
  - Embedded Systems

- Outdoor Enthusiast
  - Rock Climber

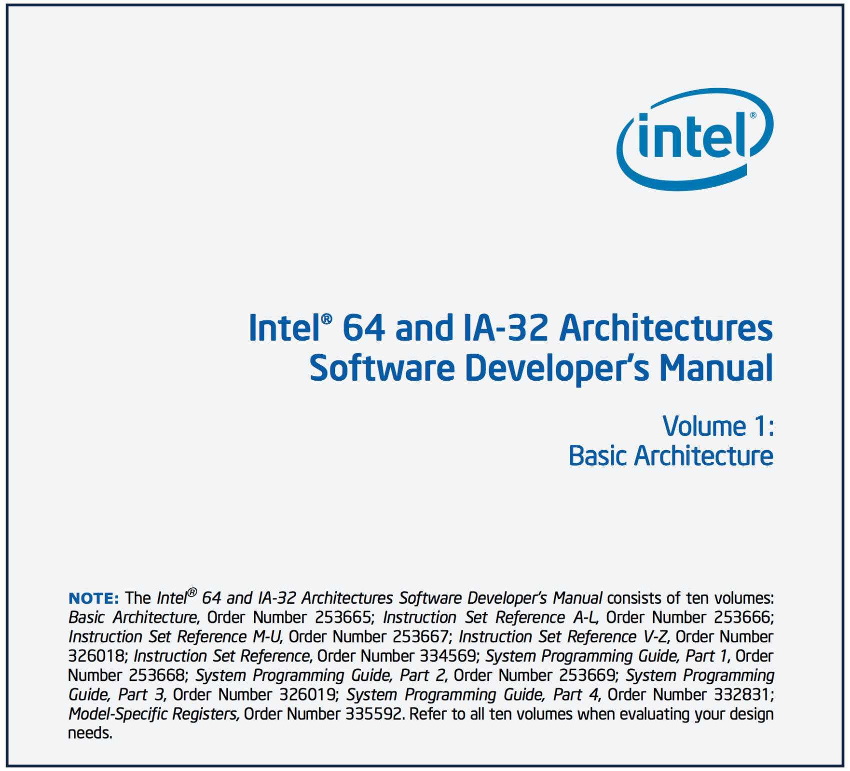
## Background

- I spend a lot of time working on:
  - Open-source, MIT license
  - https://github.com/Bareflank/hypervisor

- Bareflank™ allows you to:
  - Build new bare-metal hypervisors in C++
  - Turn Linux or Windows into a virtual machine while it is running
  - Use Linux or Windows as a "control domain" for other virtual machines

# Background

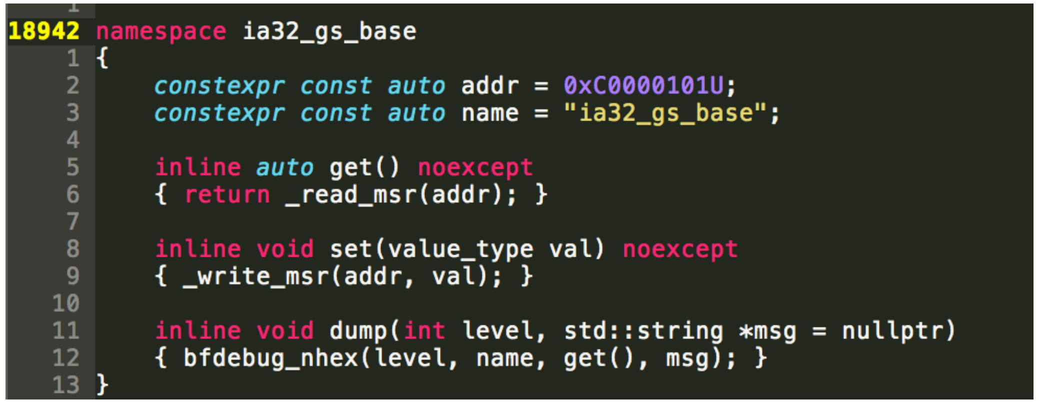
- Much of the Bareflank™ project revolves around this:



Intel® 64 and IA-32 Architectures Software Developer's Manual

Volume 1: Basic Architecture

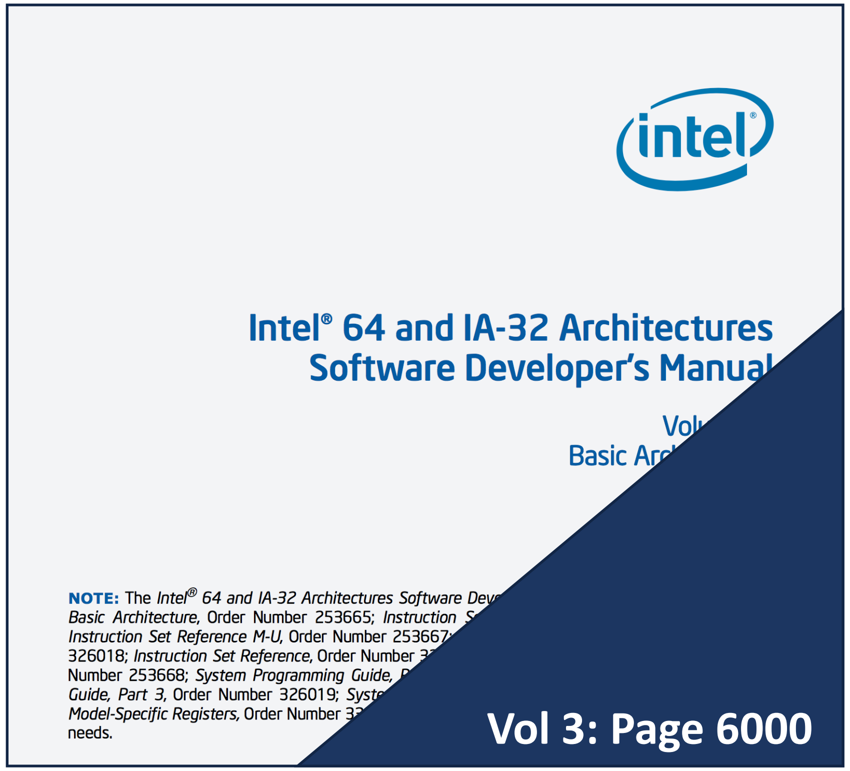
**NOTE:** The *Intel® 64 and IA-32 Architectures Software Developer's Manual* consists of ten volumes: *Basic Architecture*, Order Number 253665; *Instruction Set Reference A-L*, Order Number 253666; *Instruction Set Reference M-U*, Order Number 253667; *Instruction Set Reference V-Z*, Order Number 326018; *Instruction Set Reference*, Order Number 334569; *System Programming Guide, Part 1*, Order Number 253668; *System Programming Guide, Part 2*, Order Number 253669; *System Programming Guide, Part 3*, Order Number 326019; *System Programming Guide, Part 4*, Order Number 332831; *Model-Specific Registers*, Order Number 335592. Refer to all ten volumes when evaluating your design needs.



```
namespace ia32_gs_base
{
    constexpr const auto addr = 0xC0000101U;
    constexpr const auto name = "ia32_gs_base";

    inline auto get() noexcept
    { return _read_msr(addr); }

    inline void set(value_type val) noexcept
    { _write_msr(addr, val); }

    inline void dump(int level, std::string *msg = nullptr)
    { bfdebug_nhex(level, name, get(), msg); }
}
```

# Background

- Much of the Bareflank™ project revolves around this:

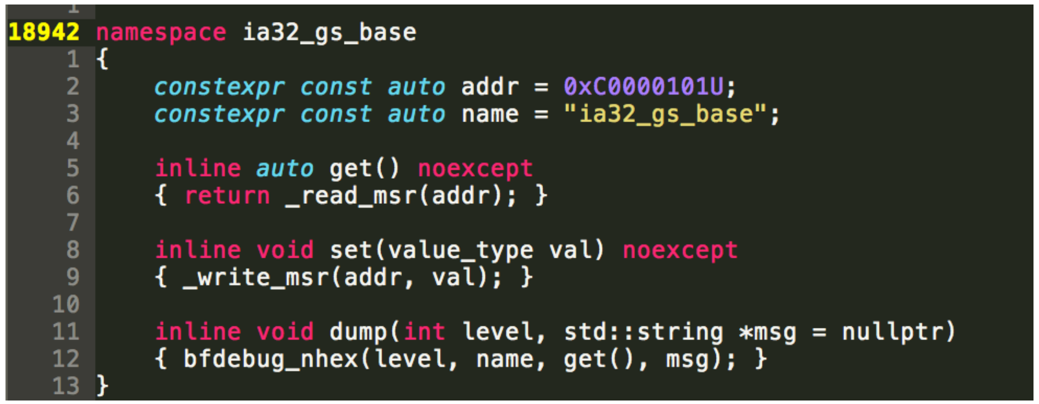


Intel® 64 and IA-32 Architectures
Software Developer's Manual

Vol
Basic Arc

**NOTE:** The *Intel® 64 and IA-32 Architectures Software Dev*
*Basic Architecture*, Order Number 253665; *Instruction S*
*Instruction Set Reference M-U*, Order Number 253667;
326018; *Instruction Set Reference*, Order Number 3
Number 253668; *System Programming Guide, P*
*Guide, Part 3*, Order Number 326019; *Syst*
*Model-Specific Registers*, Order Number 3
needs.

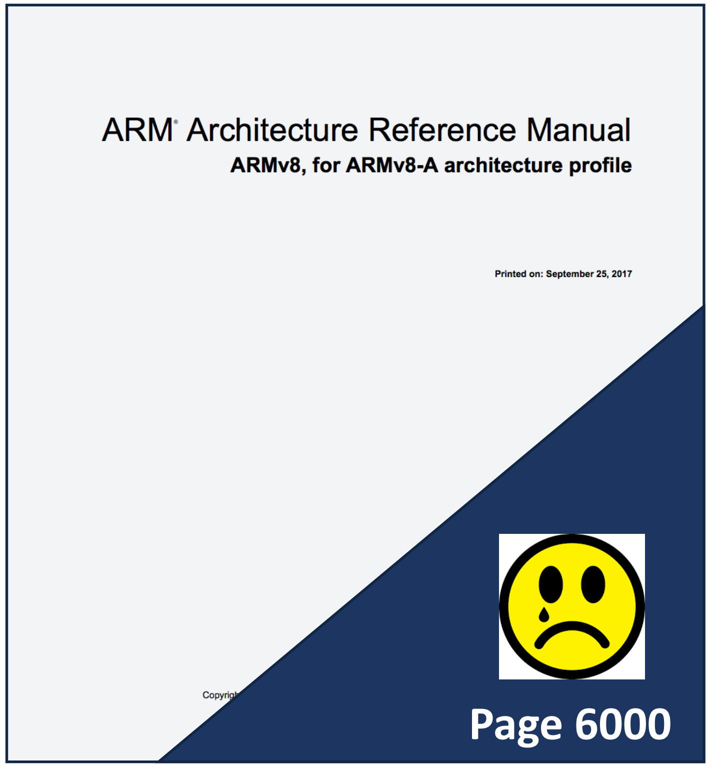
Vol 3: Page 6000



```
18942 namespace ia32_gs_base
{
    constexpr const auto addr = 0xC0000101U;
    constexpr const auto name = "ia32_gs_base";

    inline auto get() noexcept
    { return _read_msr(addr); }

    inline void set(value_type val) noexcept
    { _write_msr(addr, val); }

    inline void dump(int level, std::string *msg = nullptr)
    { bfdebug_nhex(level, name, get(), msg); }
}
```

# Background

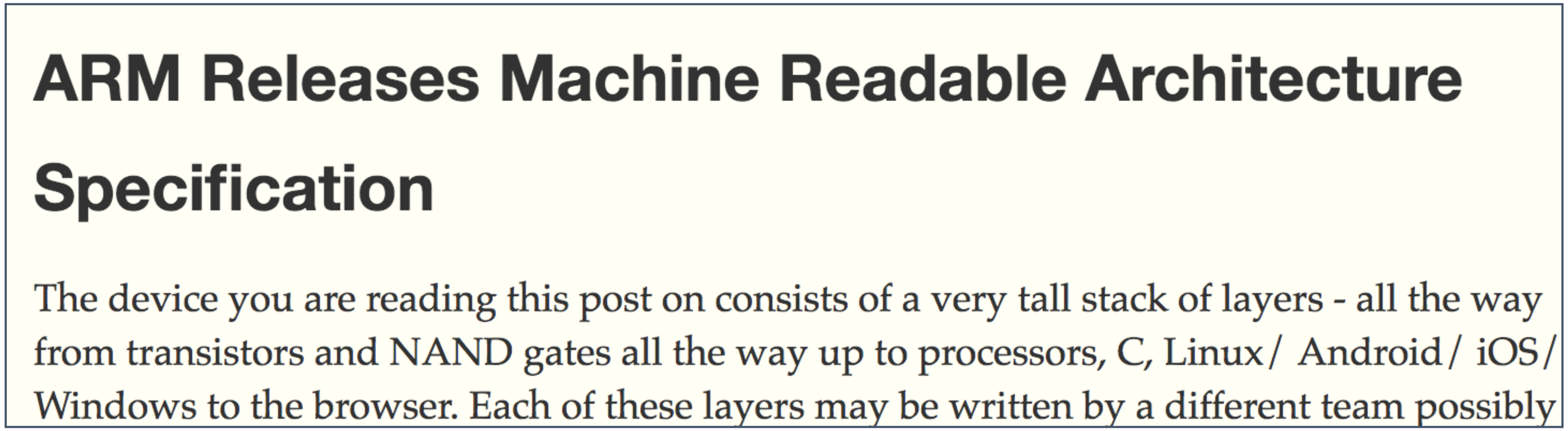
- Expanding to new architectures means…



ARM® Architecture Reference Manual
ARMv8, for ARMv8-A architecture profile

Printed on: September 25, 2017

Copyrig


**ARM Releases Machine Readable Architecture Specification**

The device you are reading this post on consists of a very tall stack of layers - all the way from transistors and NAND gates all the way up to processors, C, Linux/ Android/ iOS/ Windows to the browser. Each of these layers may be written by a different team possibly

⋮

- Alastair Reid's Blog
  - https://alastairreid.github.io/ARM-v8a-xml-release/

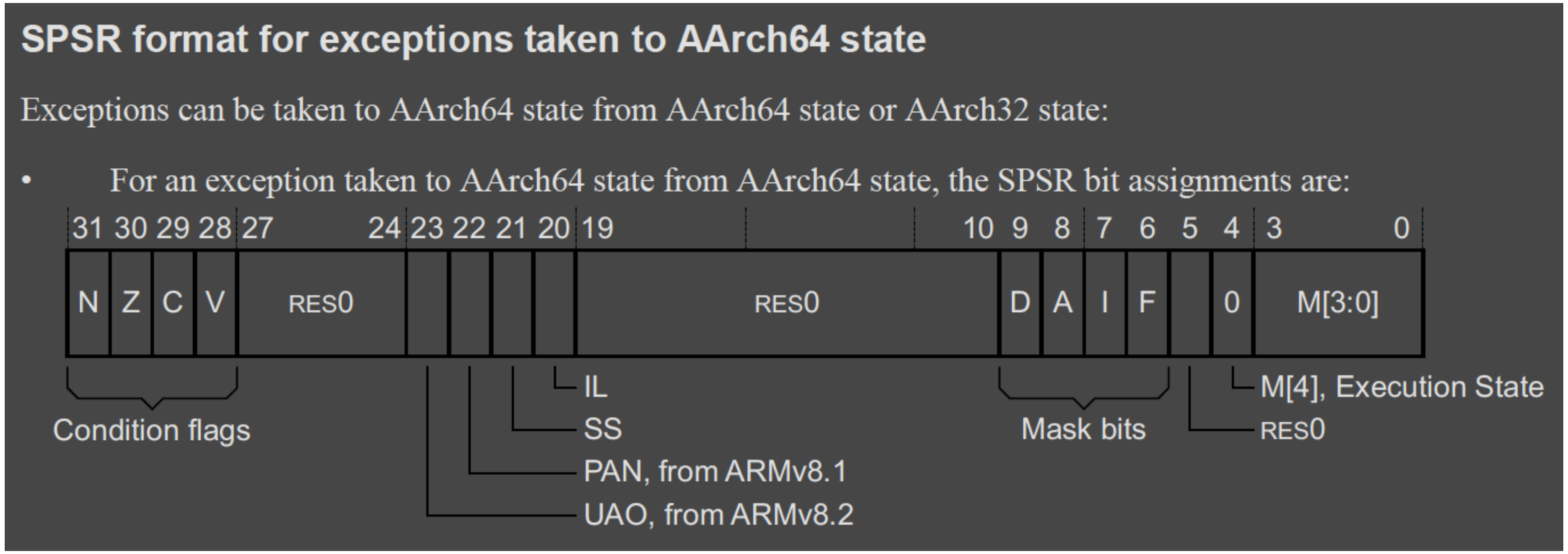
# "Machine Readable": What's That?

- Consists of:
  - XML – structured English prose
  - ASL – a specification language
  - Easy for computer programs *and* humans to read it

- Developed by ARM Holdings
  - First public release in 2017
  - https://developer.arm.com/products/architecture/a-profile/exploration-tools

# Human Readable Format

- An example register looks like this:



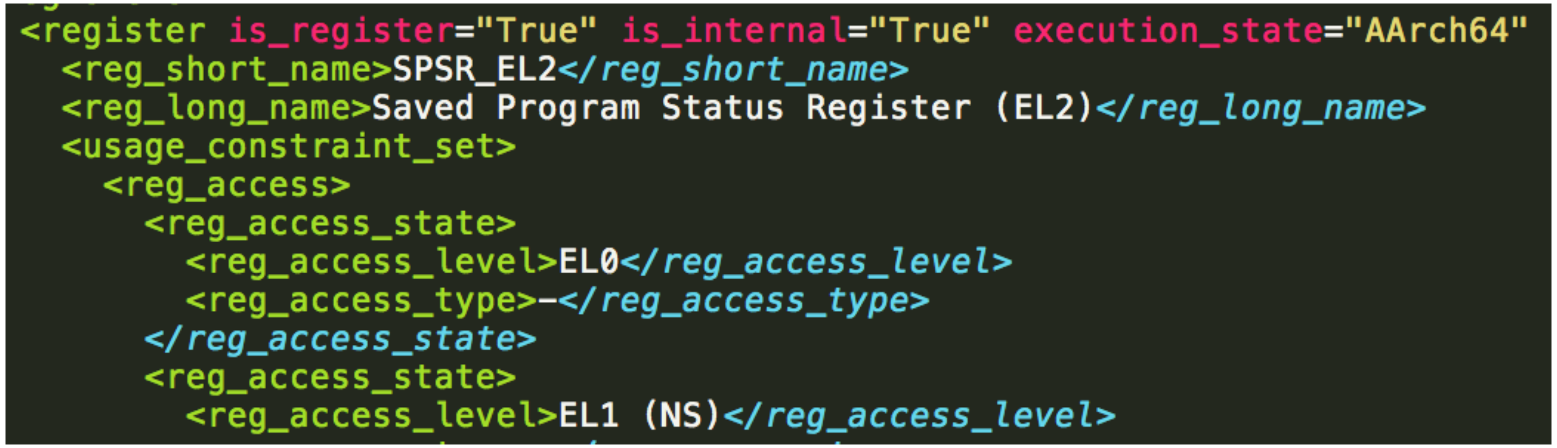
**SPSR format for exceptions taken to AArch64 state**

Exceptions can be taken to AArch64 state from AArch64 state or AArch32 state:

- For an exception taken to AArch64 state from AArch64 state, the SPSR bit assignments are:

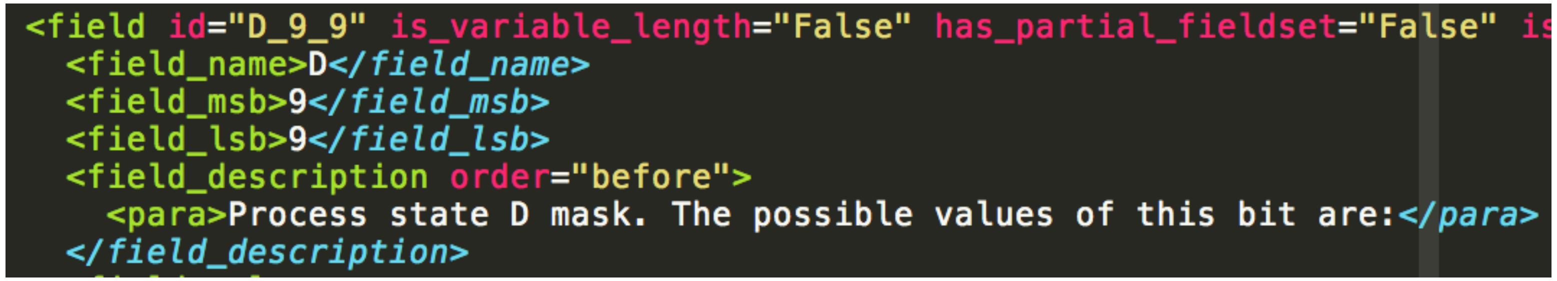
# Machine Readable Format

- The same register looks like this:



```xml
<register is_register="True" is_internal="True" execution_state="AArch64"
  <reg_short_name>SPSR_EL2</reg_short_name>
  <reg_long_name>Saved Program Status Register (EL2)</reg_long_name>
  <usage_constraint_set>
    <reg_access>
      <reg_access_state>
        <reg_access_level>EL0</reg_access_level>
        <reg_access_type>-</reg_access_type>
      </reg_access_state>
      <reg_access_state>
        <reg_access_level>EL1 (NS)</reg_access_level>
```

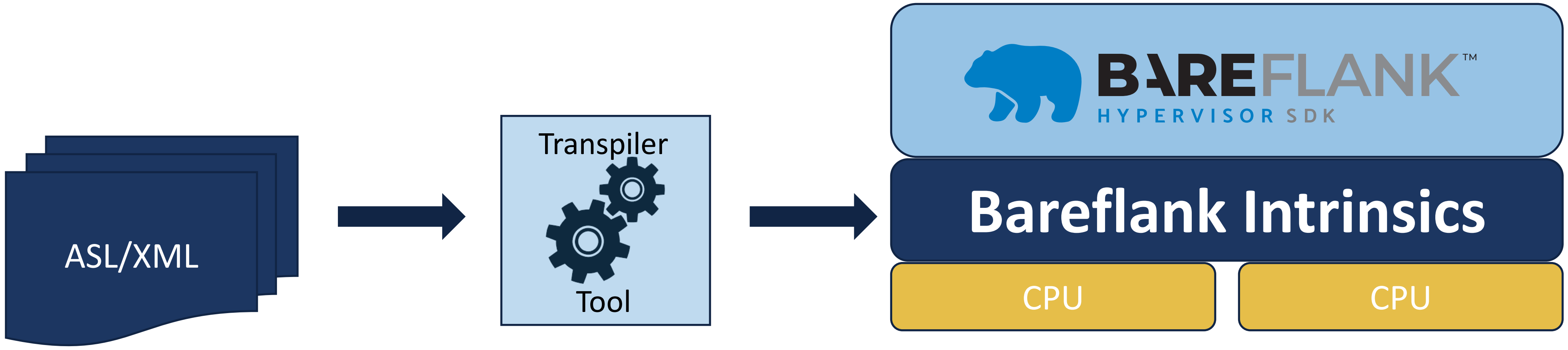
⋮



```xml
<field id="D_9_9" is_variable_length="False" has_partial_fieldset="False" is
  <field_name>D</field_name>
  <field_msb>9</field_msb>
  <field_lsb>9</field_lsb>
  <field_description order="before">
    <para>Process state D mask. The possible values of this bit are:</para>
  </field_description>
```

# This Could Be Useful

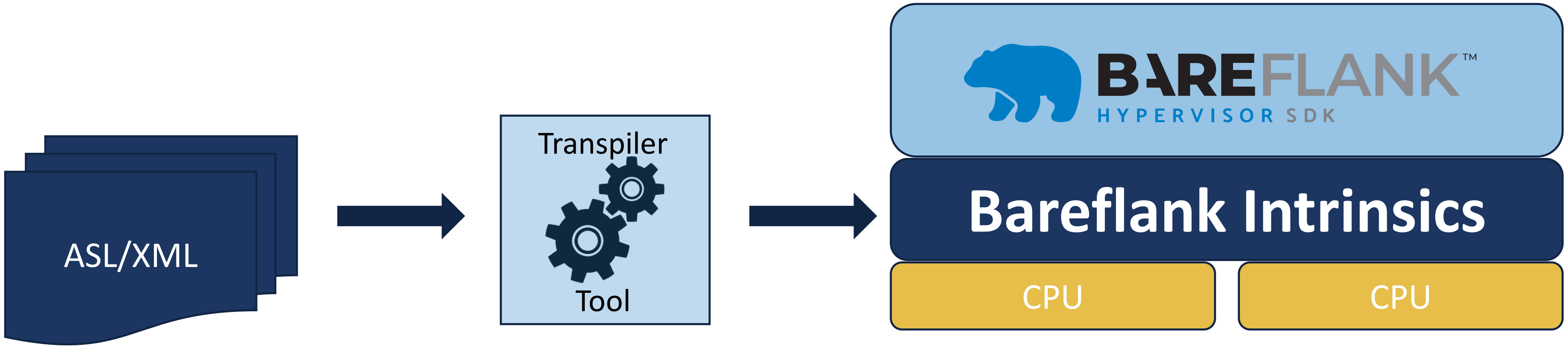
- ☐ Is it possible to generate Bareflank™ support for ARM from the XML/ASL manuals?



## This ~~Could Be~~ Is Useful

✓ Is it possible to generate Bareflank™ support for ARM from the XML/ASL manuals?

- https://github.com/Bareflank/shoulder



“But the specification can be used for much, much more – so download it and do something surprising with it.

*- Alastair Reid*

## What Else Is Possible?

- ☐ Can you generate a CPU emulator?
- ☐ Can you generate an assembler or disassembler?
- ☐ Can you verify that a physical CPU matches its abstract specification?

# What Else Is Possible?

- ☑ Can you generate a CPU emulator?
  - https://github.com/rems-project/sail-arm
- ☑ Can you generate an assembler or disassembler?
  - https://github.com/nspin/hs-arm
- ☐ Can you verify that a physical CPU matches its abstract specification?
  - Let’s find out!

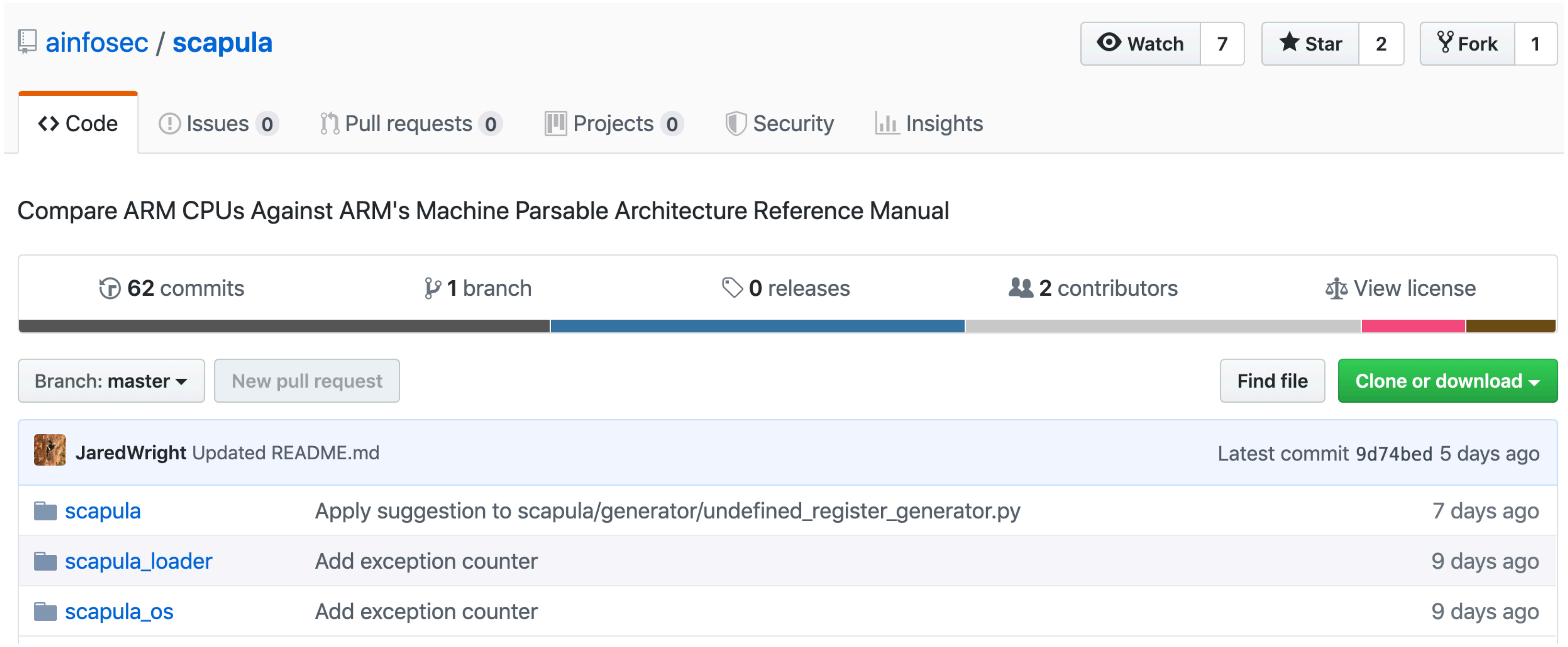
## Why Bother?

- Big.LITTLE problems
    - https://medium.com/@jadr2ddude/a-big-little-problem-a-tale-of-big-little-gone-wrong-e7778ce744bb
- Closed source designs are difficult to verify
    - https://github.com/xoreaxeaxeax/sandsifter
- In the ARM world: design != implementation

## Introducing…
## Scapula

# Scapula

- https://github.com/ainfosec/scapula



Compare ARM CPUs Against ARM's Machine Parsable Architecture Reference Manual

JaredWright Updated README.md Latest commit 9d74bed 5 days ago

| | | |
|---|---|---|
| scapula | Apply suggestion to scapula/generator/undefined_register_generator.py | 7 days ago |
| scapula_loader | Add exception counter | 9 days ago |
| scapula_os | Add exception counter | 9 days ago |

# What Does It Do?

- Verification:
  - Create a list of assumptions about how a CPU *should* work
  - Test to see if those assumptions hold

- Fuzzing
  - Run through many iterations of inputs (instructions) on a CPU
  - Watch to see if inputs cause unexpected side-effects

- Focused on testing behaviors that concern system registers

## Architecture

### Scapula

A Python package for reading the manuals and generating assumptions as executable test cases

### Scapula OS

A light-weight operating system that provides an execution environment for Scapula test cases

### Scapula Loader

A bootloader for loading test cases onto a CPU

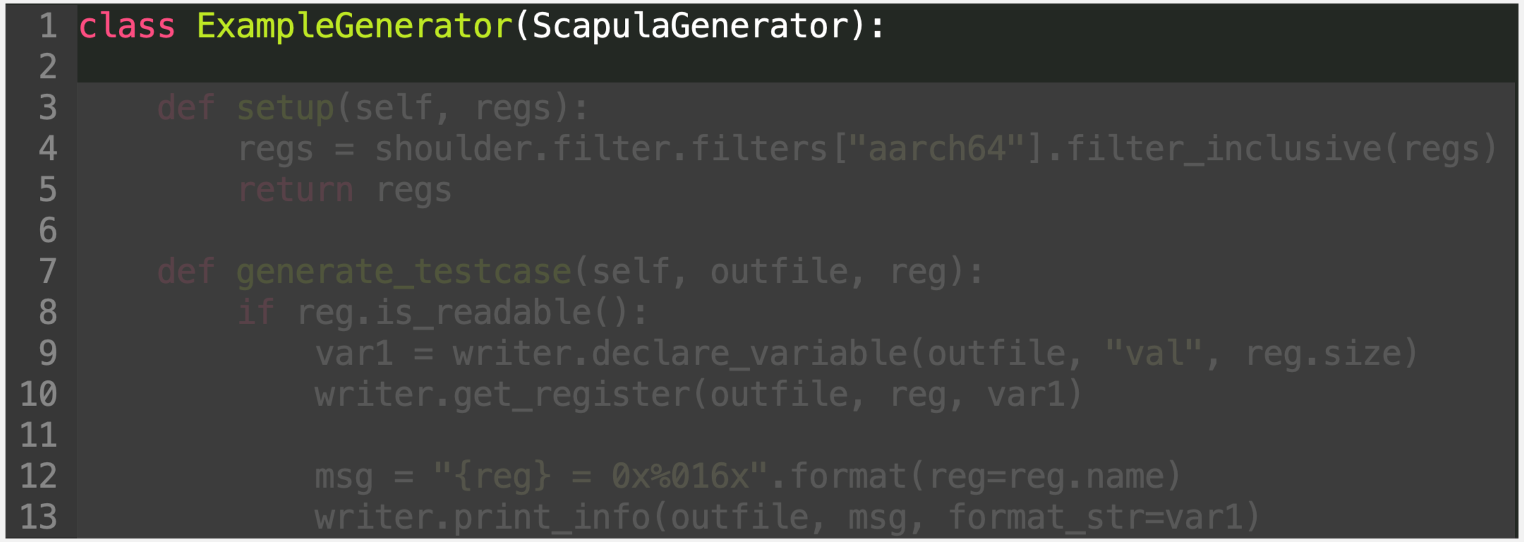
# Generating Test Cases

```python
class ExampleGenerator(ScapulaGenerator):

    def setup(self, regs):
        regs = shoulder.filter.filters["aarch64"].filter_inclusive(regs)
        return regs

    def generate_testcase(self, outfile, reg):
        if reg.is_readable():
            var1 = writer.declare_variable(outfile, "val", reg.size)
            writer.get_register(outfile, reg, var1)

            msg = "{reg} = 0x%016x".format(reg=reg.name)
            writer.print_info(outfile, msg, format_str=var1)
```

**Scapula**

*Your* job in this framework is to write some Python code that generates test cases.

# Generating Test Cases

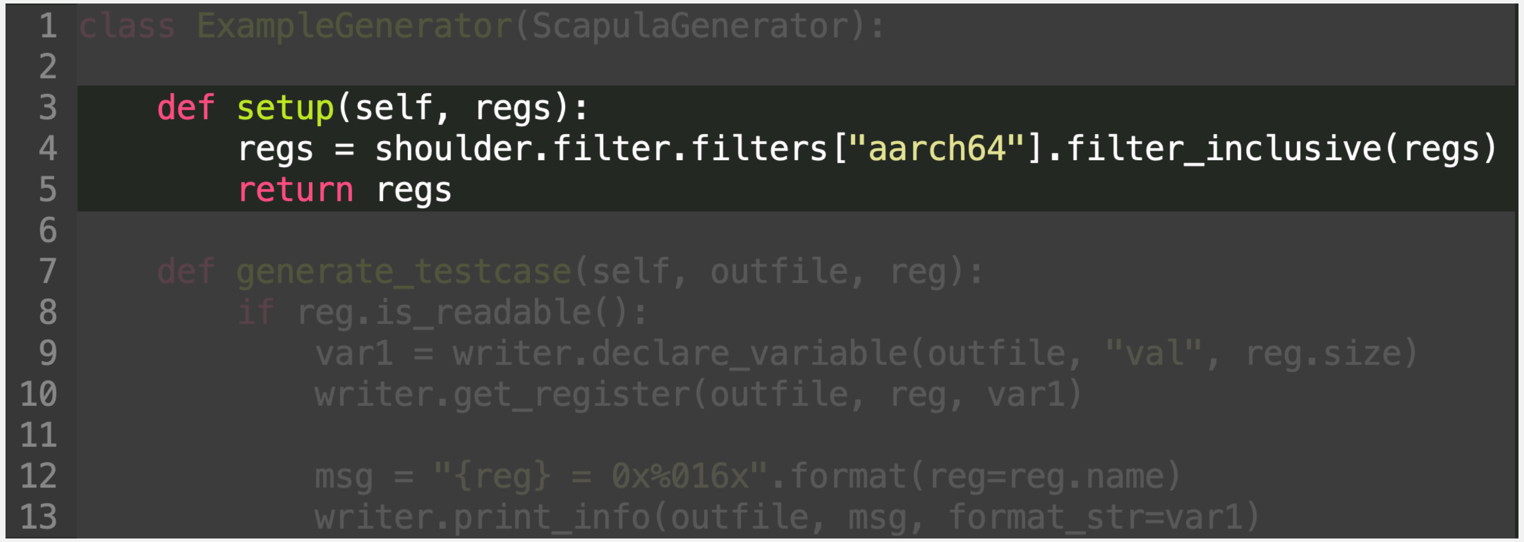


```
class ExampleGenerator(ScapulaGenerator):

    def setup(self, regs):
        regs = shoulder.filter.filters["aarch64"].filter_inclusive(regs)
        return regs

    def generate_testcase(self, outfile, reg):
        if reg.is_readable():
            var1 = writer.declare_variable(outfile, "val", reg.size)
            writer.get_register(outfile, reg, var1)

            msg = "{reg} = 0x%016x".format(reg=reg.name)
            writer.print_info(outfile, msg, format_str=var1)
```

**Scapula**

Add your own module of functionality by defining a Scapula Generator.

## Generating Test Cases

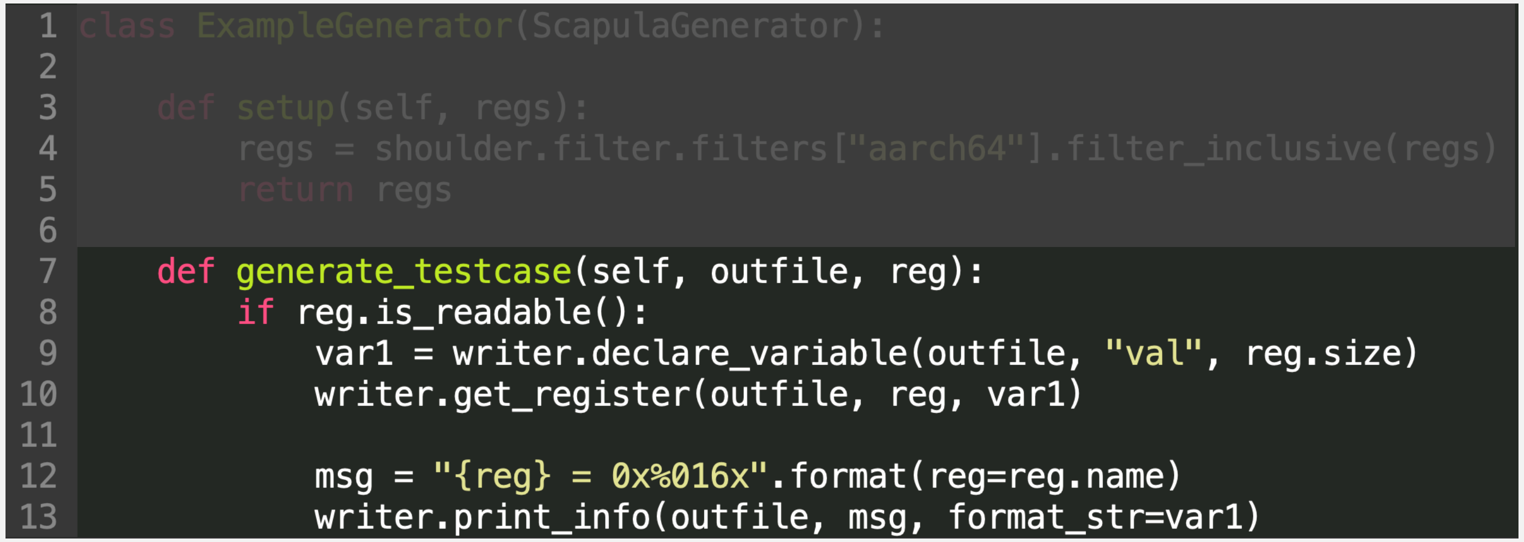


```
class ExampleGenerator(ScapulaGenerator):

    def setup(self, regs):
        regs = shoulder.filter.filters["aarch64"].filter_inclusive(regs)
        return regs

    def generate_testcase(self, outfile, reg):
        if reg.is_readable():
            var1 = writer.declare_variable(outfile, "val", reg.size)
            writer.get_register(outfile, reg, var1)

            msg = "{reg} = 0x%016x".format(reg=reg.name)
            writer.print_info(outfile, msg, format_str=var1)
```

**Scapula**

Use filters and transforms to sift through data in the manuals, and focus on a sub-set of interest.

# Generating Test Cases

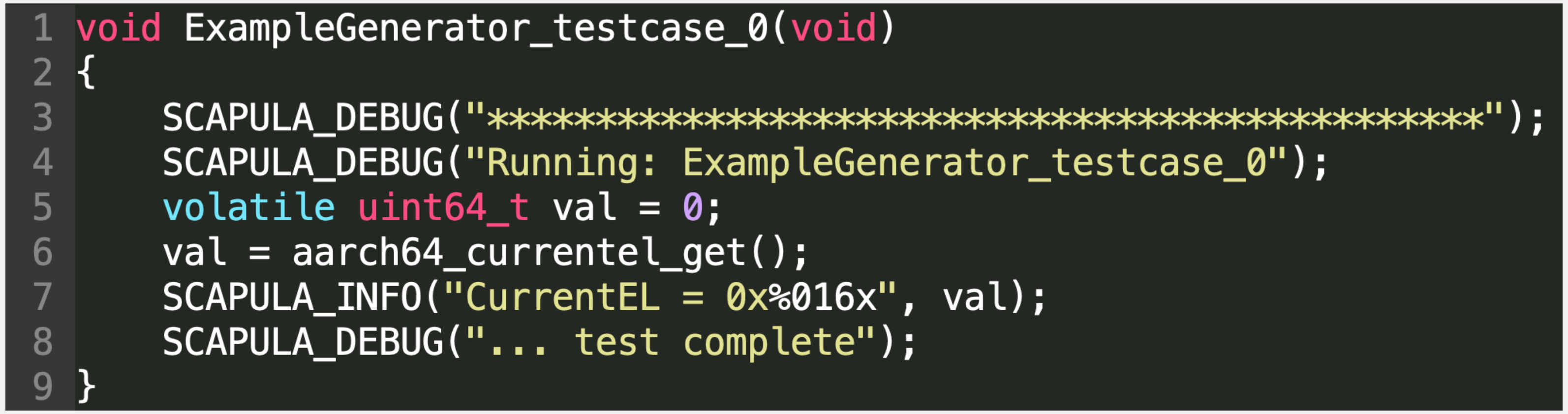


```
class ExampleGenerator(ScapulaGenerator):

    def setup(self, regs):
        regs = shoulder.filter.filters["aarch64"].filter_inclusive(regs)
        return regs

    def generate_testcase(self, outfile, reg):
        if reg.is_readable():
            var1 = writer.declare_variable(outfile, "val", reg.size)
            writer.get_register(outfile, reg, var1)

            msg = "{reg} = 0x%016x".format(reg=reg.name)
            writer.print_info(outfile, msg, format_str=var1)
```

## Scapula

Use a writer utility to generate bits and pieces of a test case, and interact with the Scapula OS environment.

# Generating Test Cases

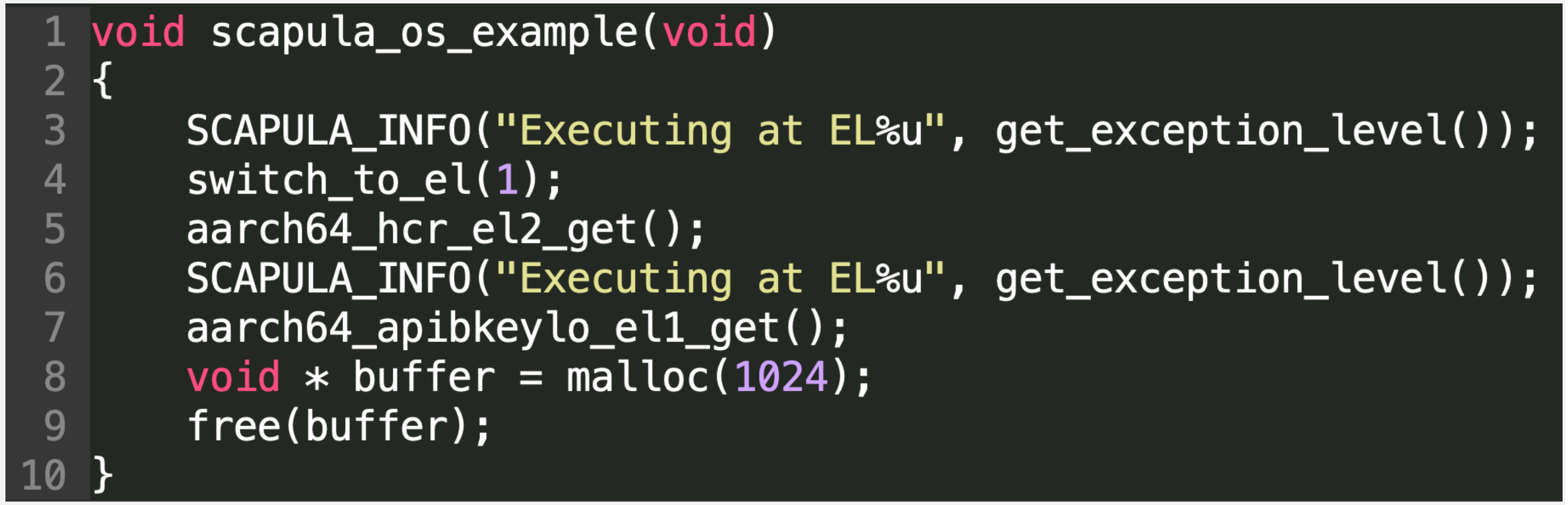


```c
void ExampleGenerator_testcase_0(void)
{
    SCAPULA_DEBUG("**********************************************");
    SCAPULA_DEBUG("Running: ExampleGenerator_testcase_0");
    volatile uint64_t val = 0;
    val = aarch64_currentel_get();
    SCAPULA_INFO("CurrentEL = 0x%016x", val);
    SCAPULA_DEBUG("... test complete");
}
```

**Result**

Test cases are generated as C code (one test per register).

## Running Test Cases

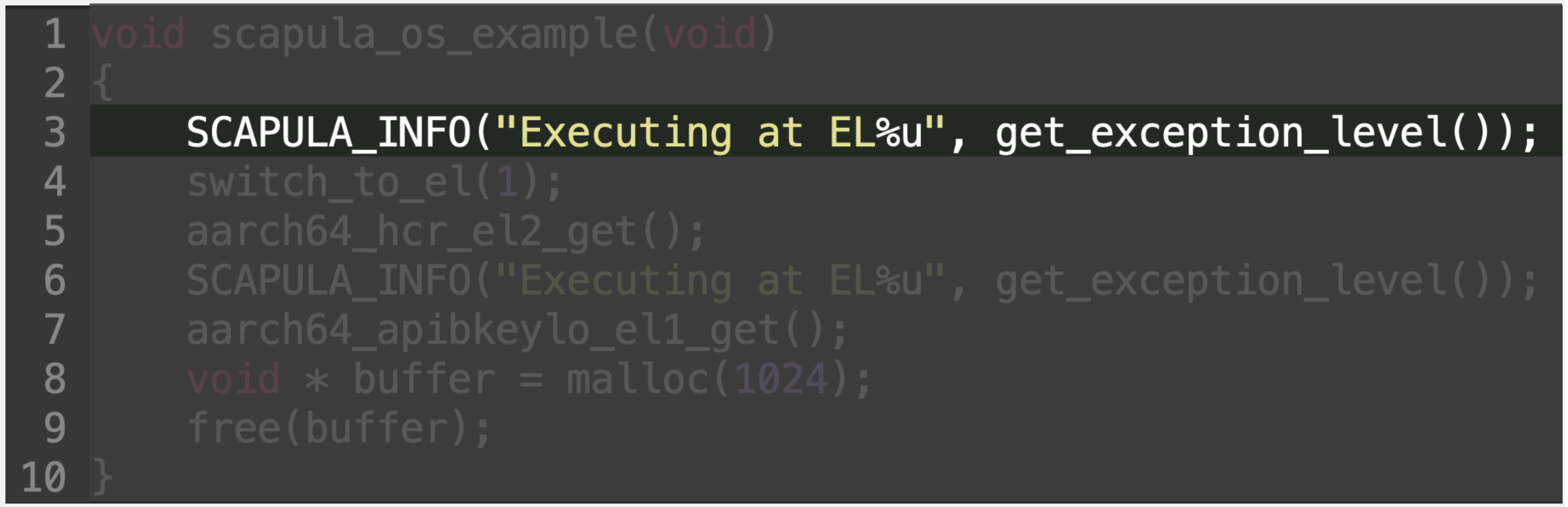


```
void scapula_os_example(void)
{
    SCAPULA_INFO("Executing at EL%u", get_exception_level());
    switch_to_el(1);
    aarch64_hcr_el2_get();
    SCAPULA_INFO("Executing at EL%u", get_exception_level());
    aarch64_apibkeylo_el1_get();
    void * buffer = malloc(1024);
    free(buffer);
}
```

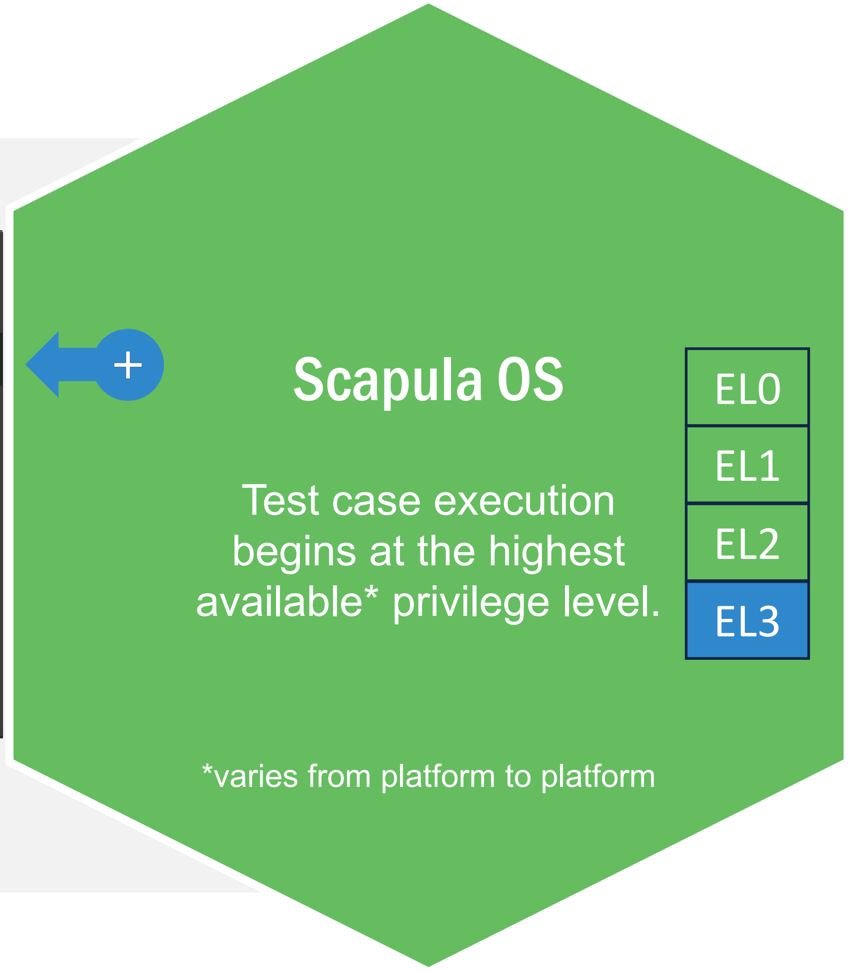
### Scapula OS

A traditional OS aims to segregate and isolate computing resources. Scapula OS makes them all accessible.

# Running Test Cases



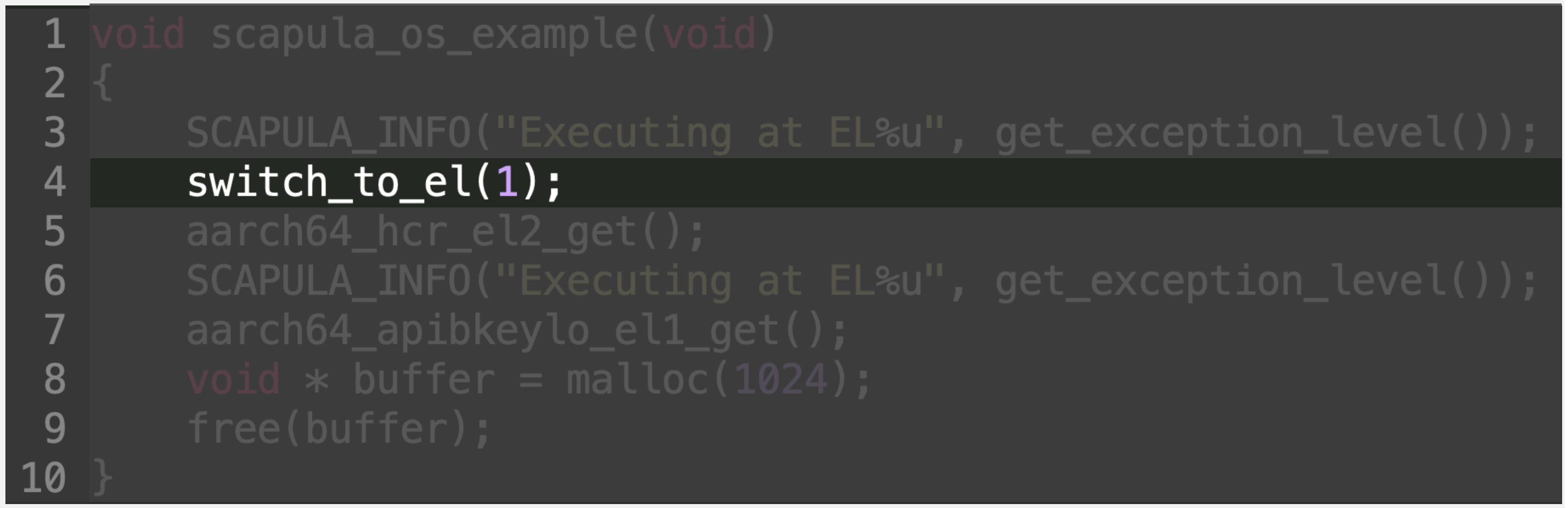
```
void scapula_os_example(void)
{
    SCAPULA_INFO("Executing at EL%u", get_exception_level());
    switch_to_el(1);
    aarch64_hcr_el2_get();
    SCAPULA_INFO("Executing at EL%u", get_exception_level());
    aarch64_apibkeylo_el1_get();
    void * buffer = malloc(1024);
    free(buffer);
}
```



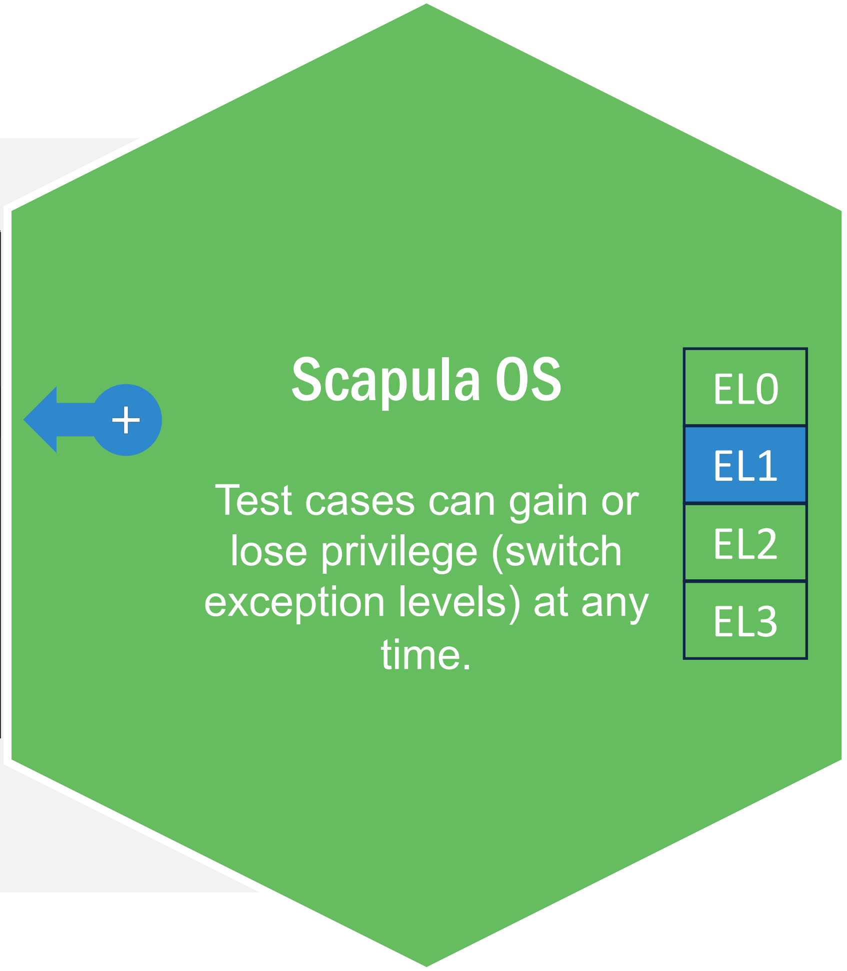
Test case execution begins at the highest available* privilege level.

*varies from platform to platform

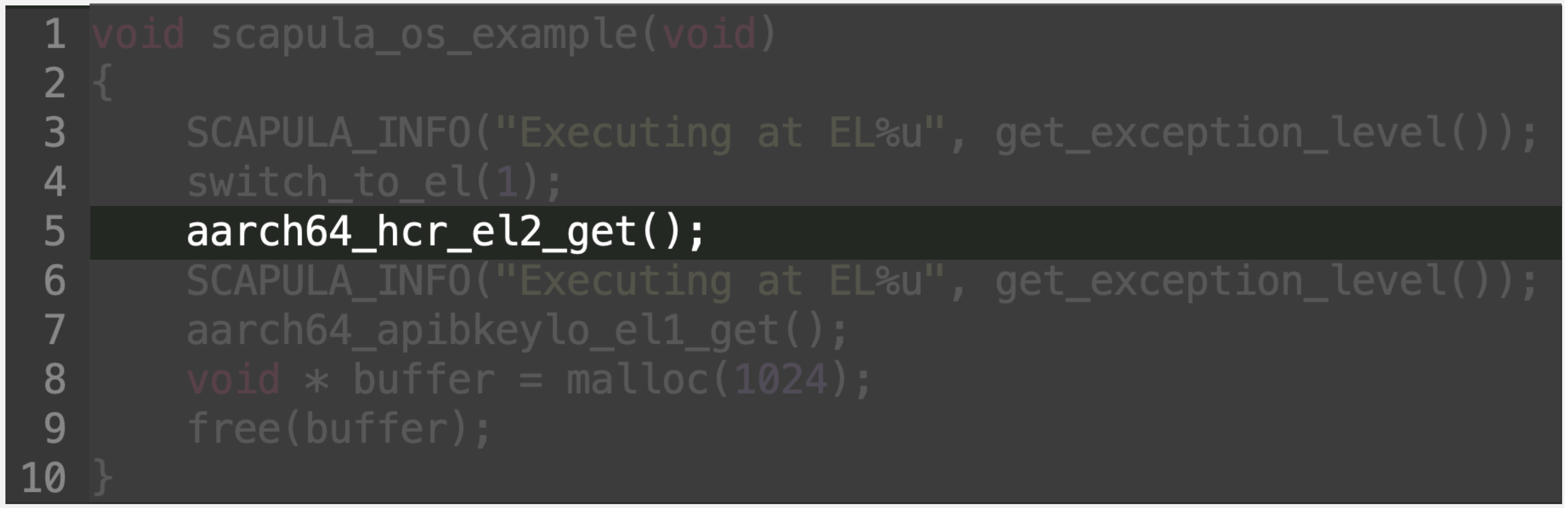
## Running Test Cases



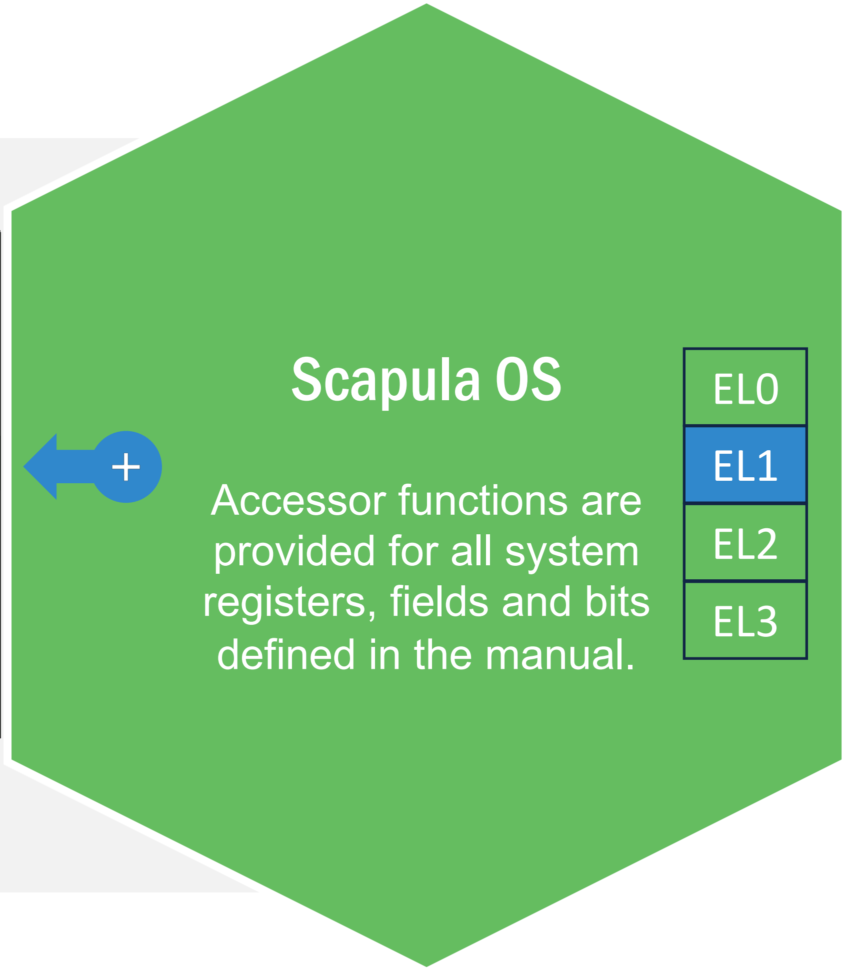
```
void scapula_os_example(void)
{
    SCAPULA_INFO("Executing at EL%u", get_exception_level());
    switch_to_el(1);
    aarch64_hcr_el2_get();
    SCAPULA_INFO("Executing at EL%u", get_exception_level());
    aarch64_apibkeylo_el1_get();
    void * buffer = malloc(1024);
    free(buffer);
}
```


Scapula OS
Test cases can gain or lose privilege (switch exception levels) at any time.
EL0
EL1
EL2
EL3

# Running Test Cases

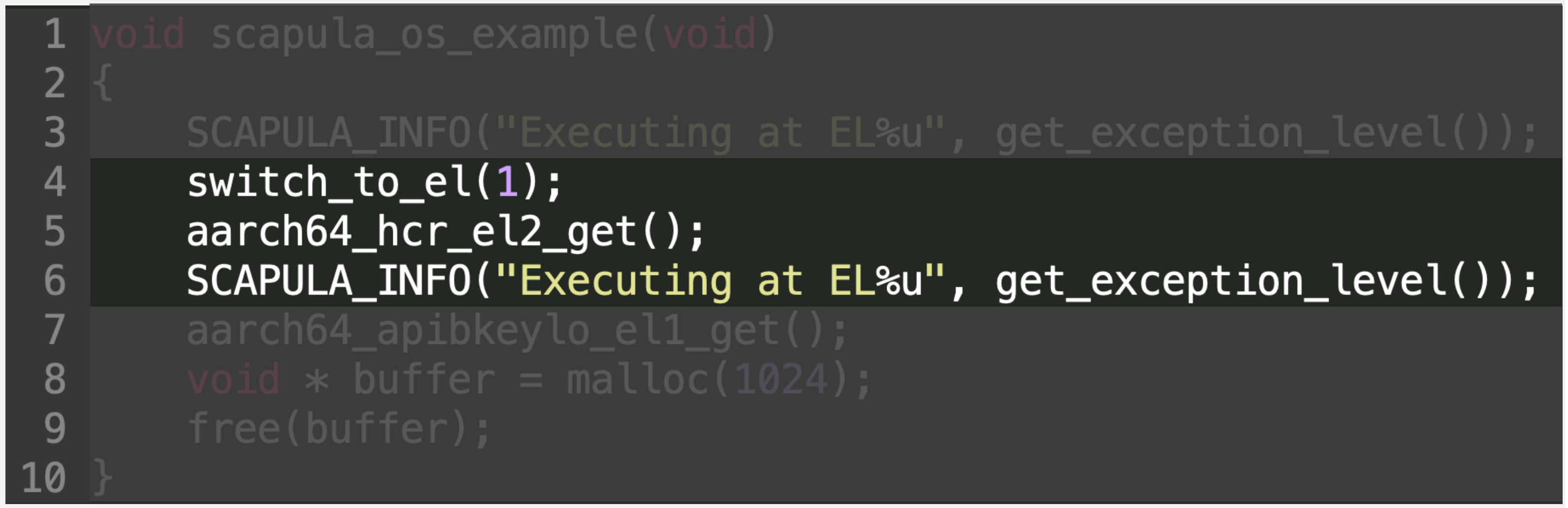


```
void scapula_os_example(void)
{
    SCAPULA_INFO("Executing at EL%u", get_exception_level());
    switch_to_el(1);
    aarch64_hcr_el2_get();
    SCAPULA_INFO("Executing at EL%u", get_exception_level());
    aarch64_apibkeylo_el1_get();
    void * buffer = malloc(1024);
    free(buffer);
}
```

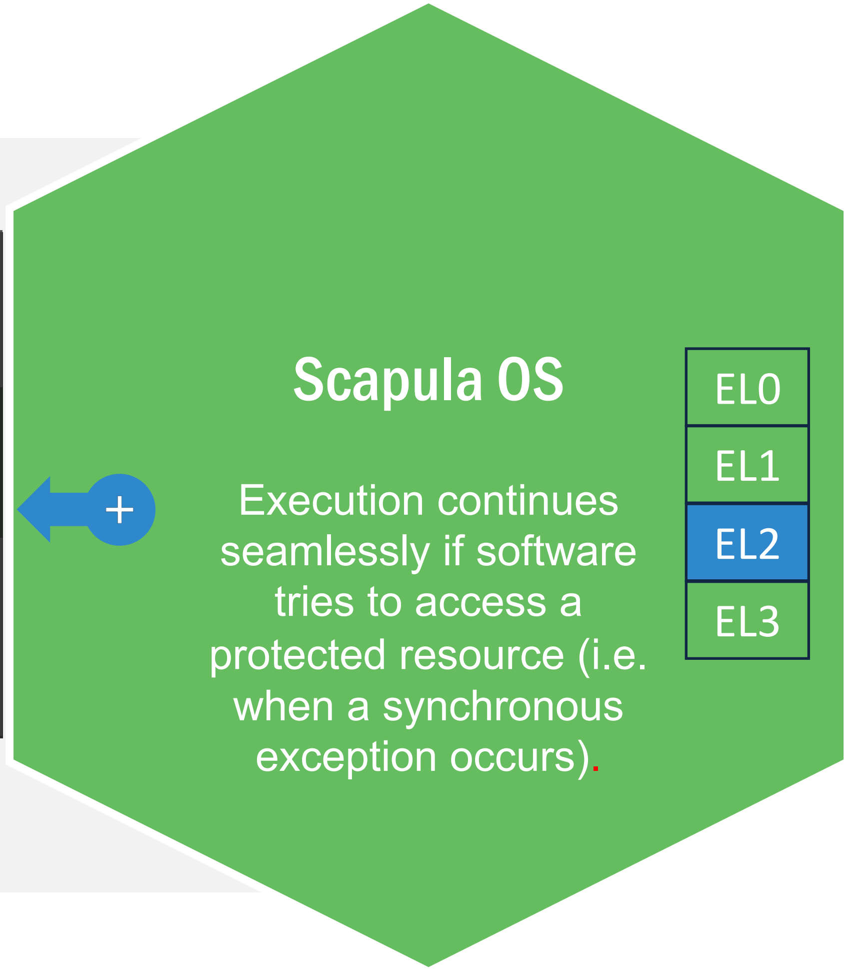


Scapula OS
Accessor functions are provided for all system registers, fields and bits defined in the manual.
EL0
EL1
EL2
EL3

# Running Test Cases

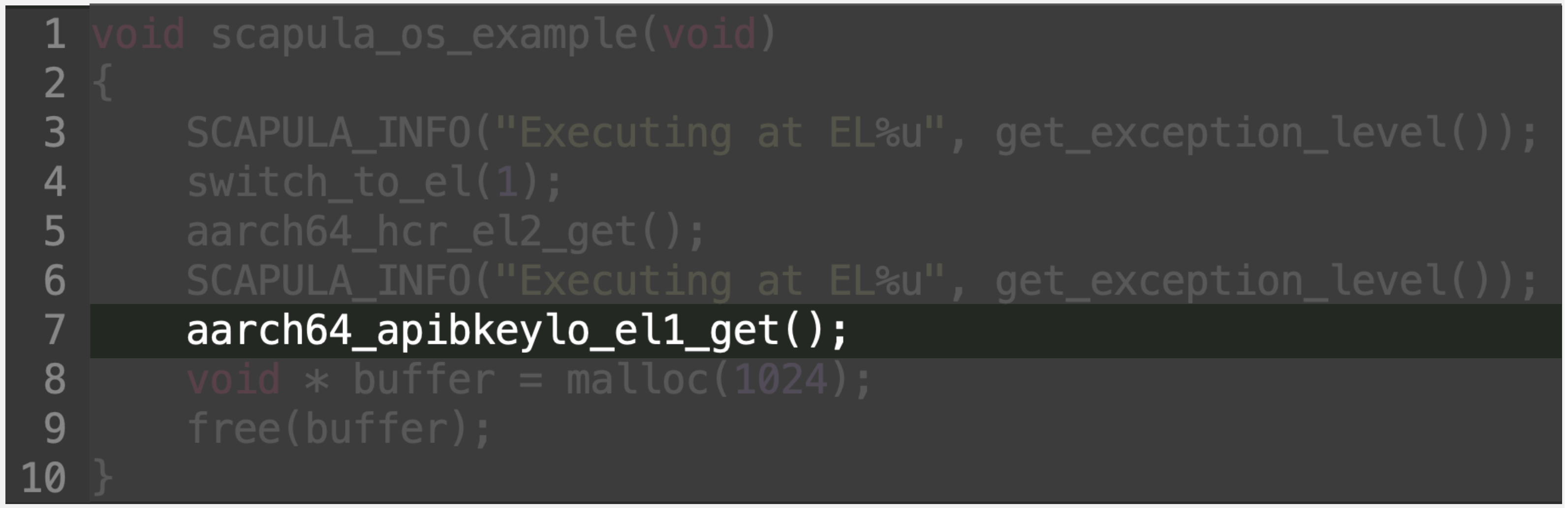


```c
void scapula_os_example(void)
{
    SCAPULA_INFO("Executing at EL%u", get_exception_level());
    switch_to_el(1);
    aarch64_hcr_el2_get();
    SCAPULA_INFO("Executing at EL%u", get_exception_level());
    aarch64_apibkeylo_el1_get();
    void * buffer = malloc(1024);
    free(buffer);
}
```

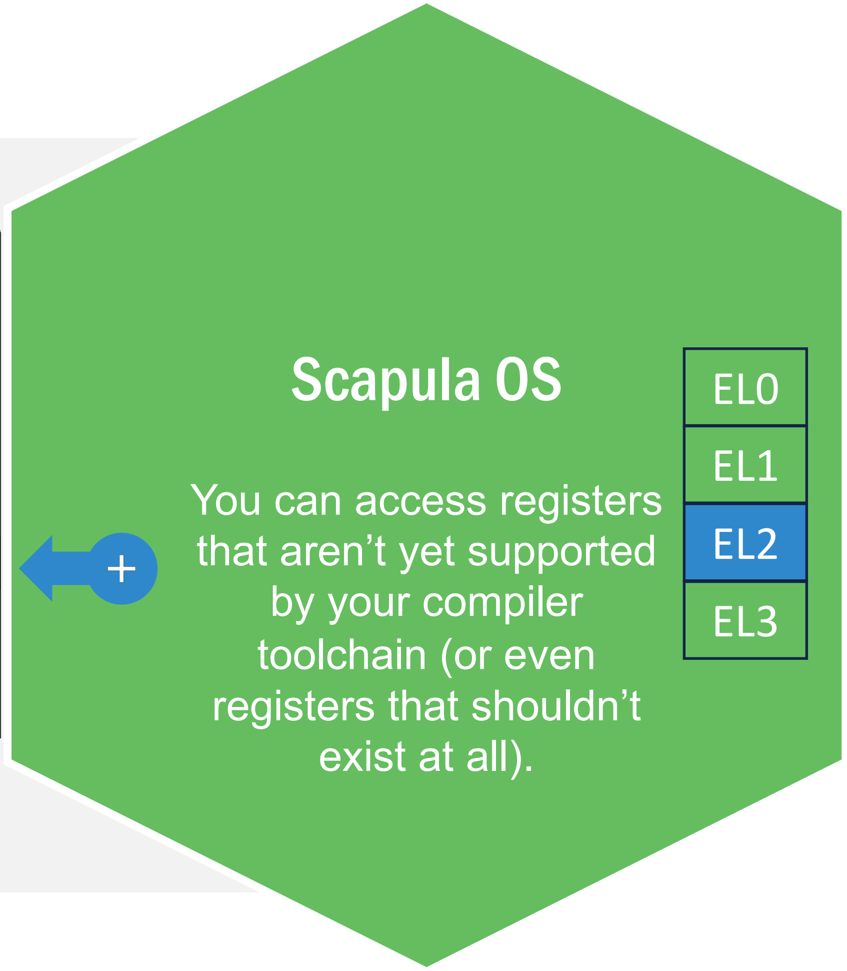


Execution continues seamlessly if software tries to access a protected resource (i.e. when a synchronous exception occurs).

# Running Test Cases

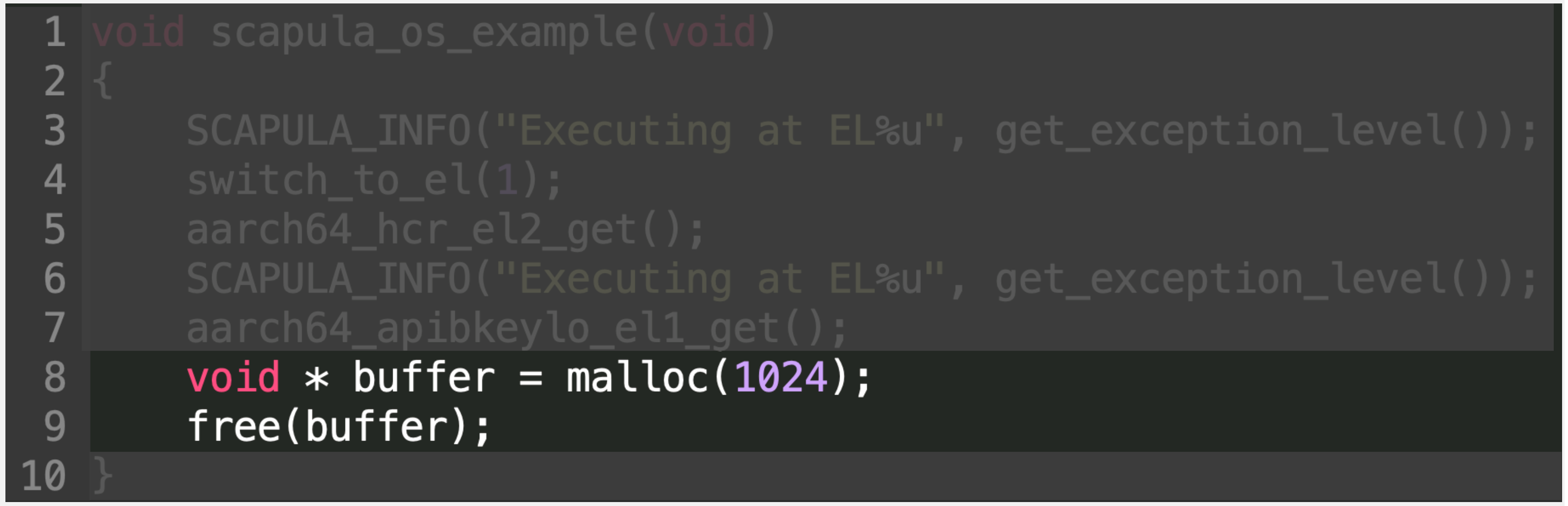


```
void scapula_os_example(void)
{
    SCAPULA_INFO("Executing at EL%u", get_exception_level());
    switch_to_el(1);
    aarch64_hcr_el2_get();
    SCAPULA_INFO("Executing at EL%u", get_exception_level());
    aarch64_apibkeylo_el1_get();
    void * buffer = malloc(1024);
    free(buffer);
}
```

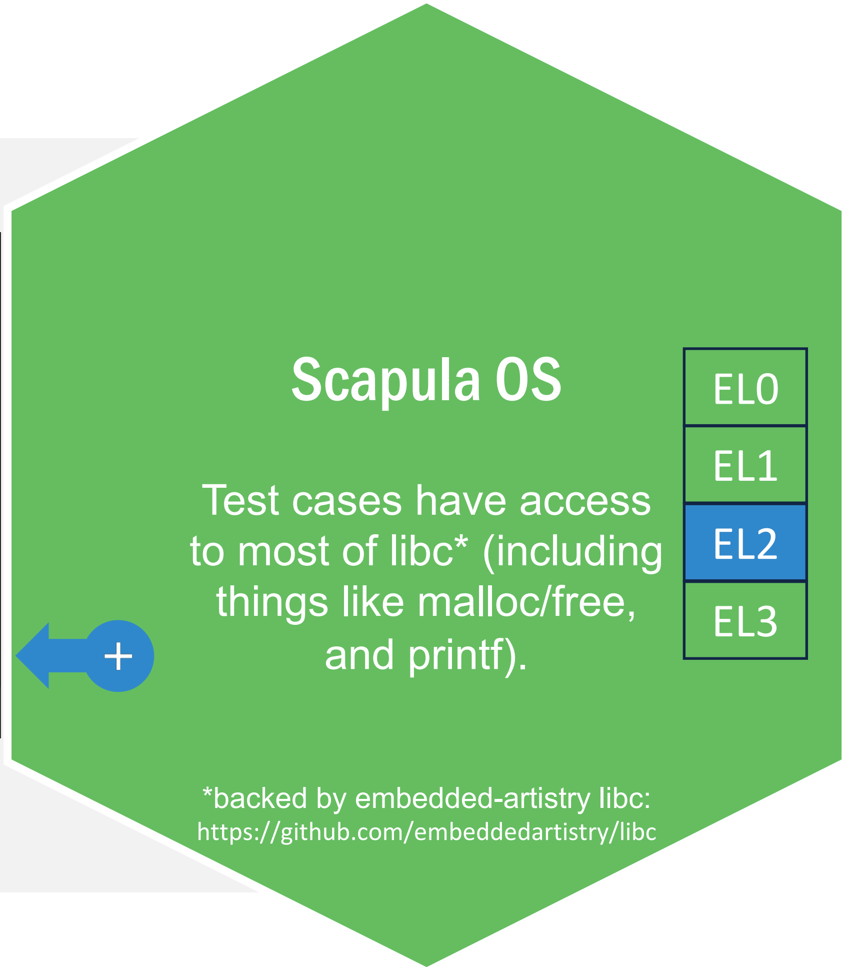


You can access registers that aren't yet supported by your compiler toolchain (or even registers that shouldn't exist at all).

# Running Test Cases



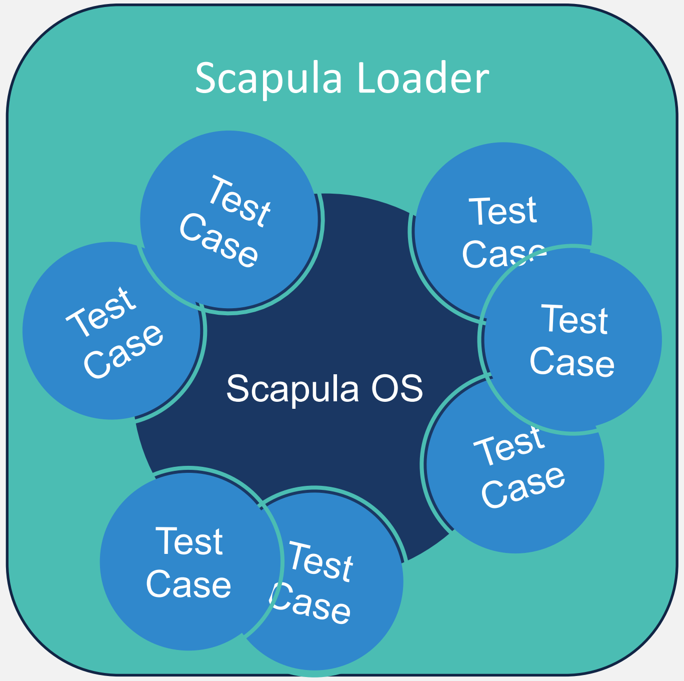
```
void scapula_os_example(void)
{
    SCAPULA_INFO("Executing at EL%u", get_exception_level());
    switch_to_el(1);
    aarch64_hcr_el2_get();
    SCAPULA_INFO("Executing at EL%u", get_exception_level());
    aarch64_apibkeylo_el1_get();
    void * buffer = malloc(1024);
    free(buffer);
}
```



Test cases have access to most of libc* (including things like malloc/free, and printf).

*backed by embedded-artistry libc: https://github.com/embeddedartistry/libc

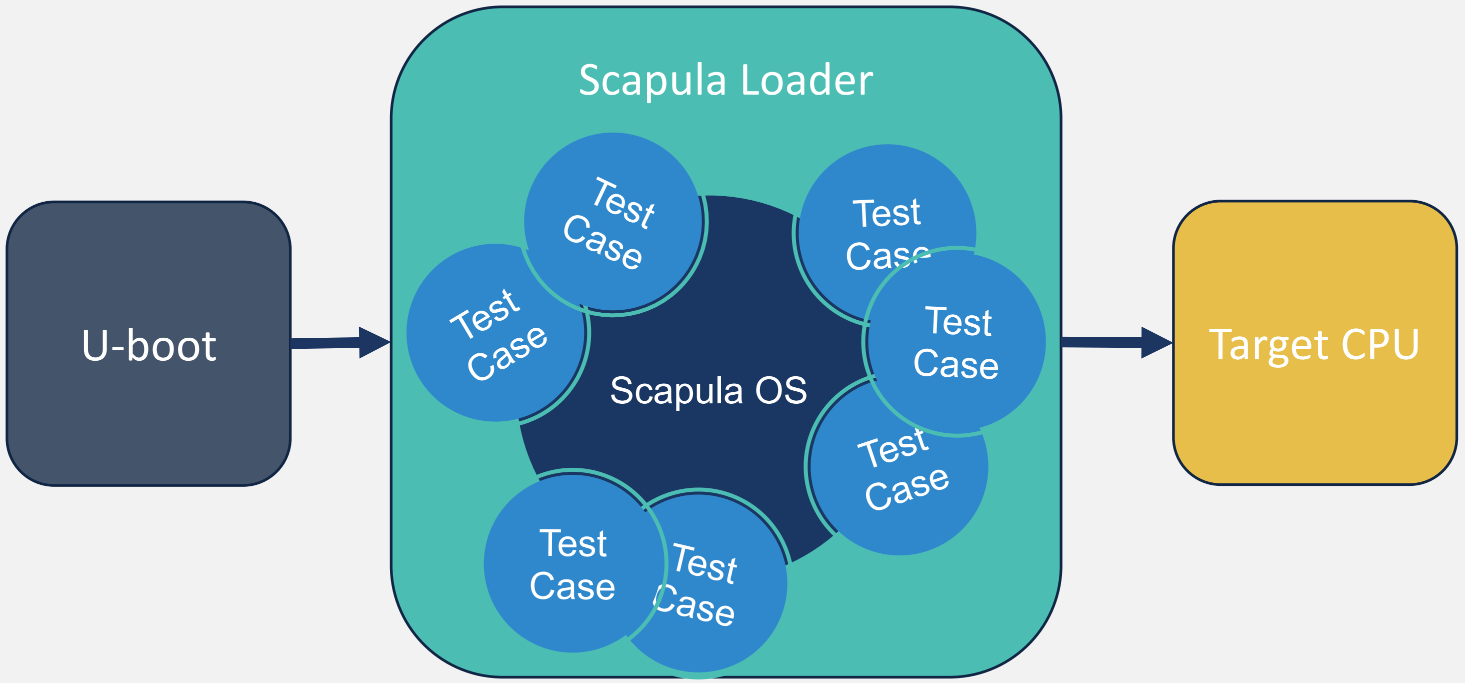
## Loading Test Cases



### Scapula Loader

Bundles up Scapula OS with all of the generated test cases into a single deployable binary (.bin or .elf).

# Loading Test Cases



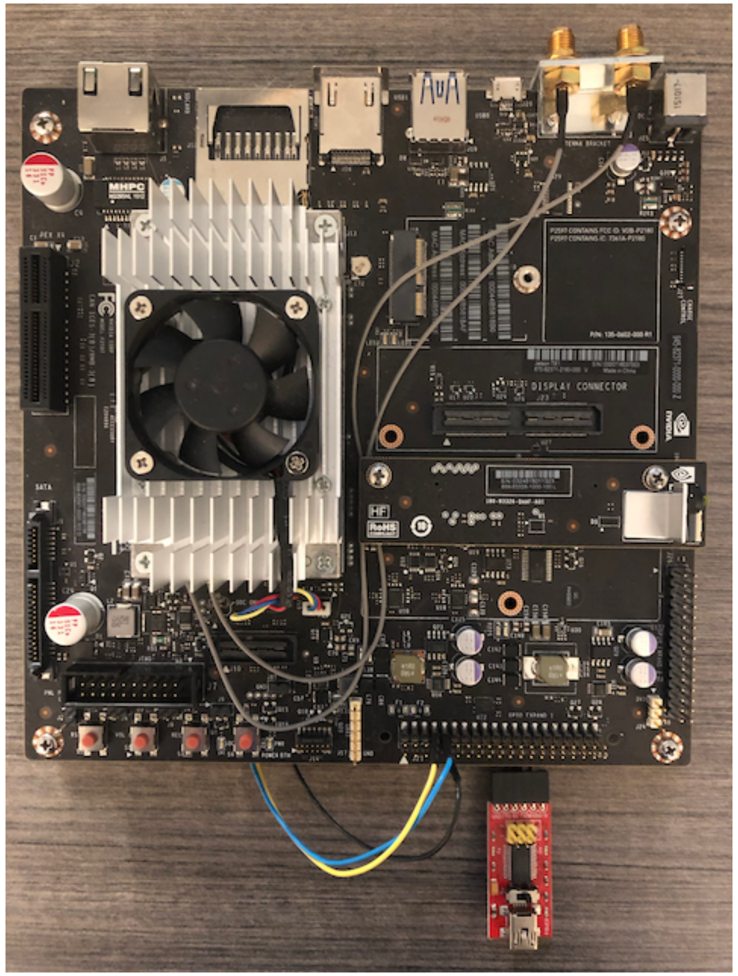
**Scapula Loader**

Compatible with U-boot + extlinux. Official hardware support includes NVIDIA Jetson TX1/TX2 developer kits, and Sail emulators.

## Let’s Run Some Tests!

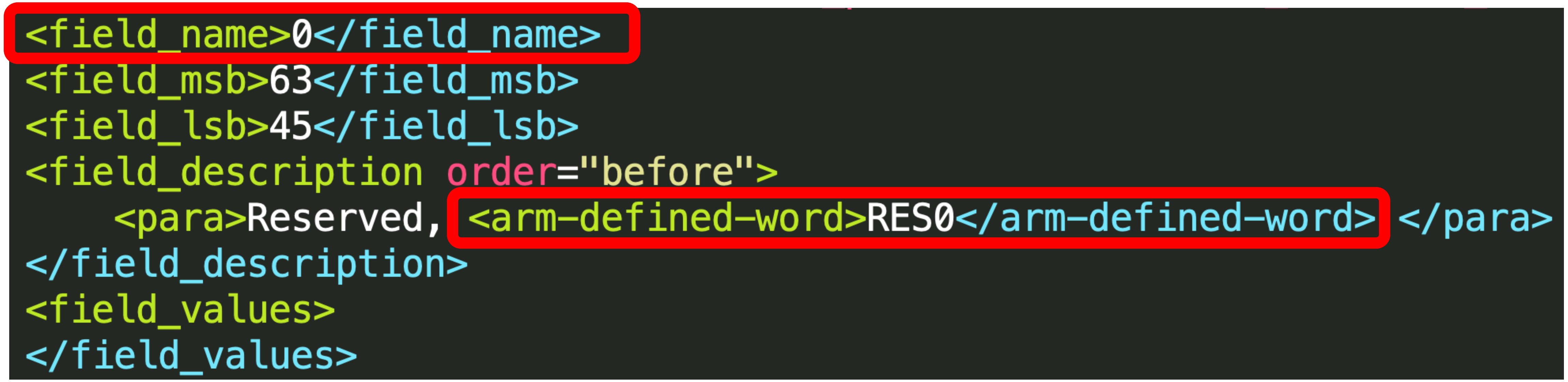
# The Target

- NVIDIA Jetson TX1 Developer Kit
  - Tegra X1 Processor
  - Cortex-A57
  - ARMv8-A
  - Released in 2015
- Used in:
  - Nintendo Switch
  - Google Pixel-C
  - Nvidia Shield Android TV



# Test 1

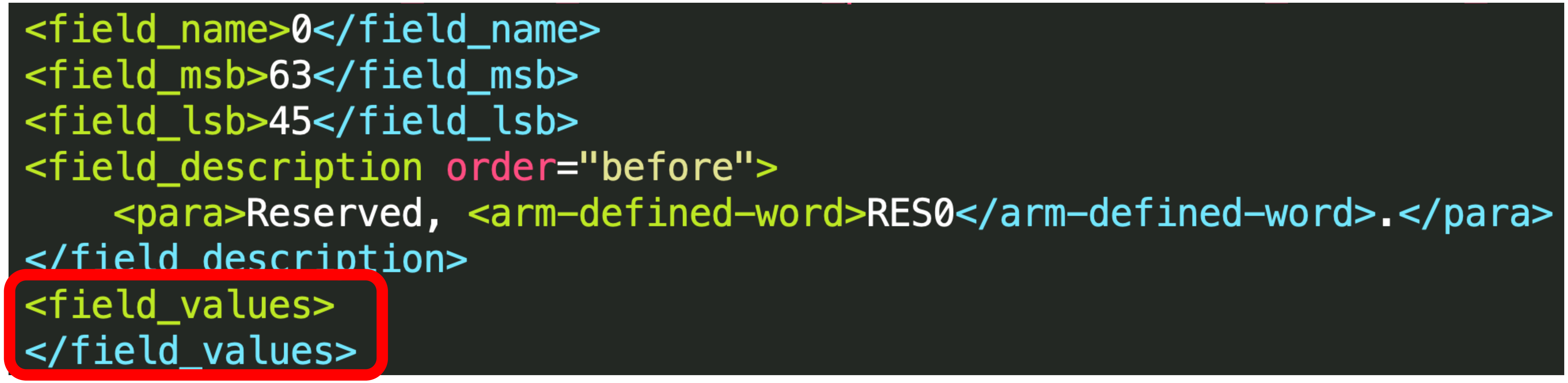
- What happens when you read “RES0” bits?



```
<field_name>0</field_name>
<field_msb>63</field_msb>
<field_lsb>45</field_lsb>
<field_description order="before">
    <para>Reserved, <arm-defined-word>RES0</arm-defined-word> </para>
</field_description>
<field_values>
</field_values>
```

# Test 2

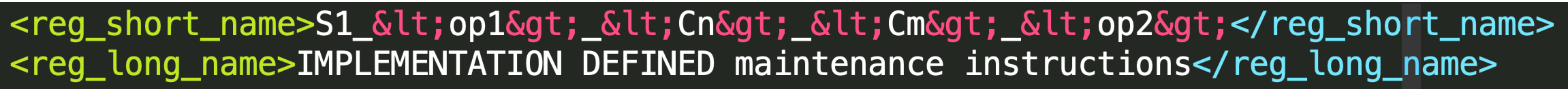
- Can you reliably toggle any “RES0” bits?



```
<field_name>0</field_name>
<field_msb>63</field_msb>
<field_lsb>45</field_lsb>
<field_description order="before">
    <para>Reserved, <arm-defined-word>RES0</arm-defined-word>.</para>
</field_description>
<field_values>
</field_values>
```

# Test 3

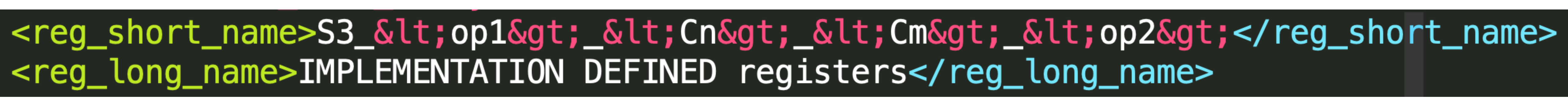
- Are there any hidden instructions?



```
<reg_short_name>S1_&lt;op1&gt;_&lt;Cn&gt;_&lt;Cm&gt;_&lt;op2&gt;</reg_short_name>
<reg_long_name>IMPLEMENTATION DEFINED maintenance instructions</reg_long_name>
```

## Test 4

- Are there any hidden system registers?



```
<reg_short_name>S3_&lt;op1&gt;_&lt;Cn&gt;_&lt;Cm&gt;_&lt;op2&gt;</reg_short_name>
<reg_long_name>IMPLEMENTATION DEFINED registers</reg_long_name>
```

# Results

- Looking for togglable RES0 bits reveals hidden features
  - Pointer authentication control bits (defined in 2016) are present in the Tegra X1 (released in 2015)
  - Others may provide a basis for a covert storage or communications channel

- NVIDIA has added their own “secret sauce” to the Tegra X1
  - At least 1 custom instruction
  - At least 17 system registers

- Two versions of the same processor (Tegra X1 vs. Tegra X2) are slightly different

## Technical Challenges and Room for Improvement

## Future Improvements: Scapula OS

- Support for switching between security states (secure world)
- Support for aarch32/A32 (32-bit) execution
- More supported platforms
- Expanded model of assumptions that includes
    - All A64/A32 instructions
    - Behaviors defined in ASL

## Future Hopes: The Spec

- Definition of the ASL language
    - tools for interpreting it
- Open-source ASL for instruction semantics
    - Currently, only registers and “shared pseudo-code” are publicly available
- Manuals for -M (microcontroller) and -R (real-time) architecture profiles
- Pressure on Intel/AMD to release similar specs

# Conclusion

- The current version of the XML/ASL manuals are useful
  - Thanks ARM!
- Software security is rooted in assumptions about hardware
  - Need more openly available tools for verifying these assumptions
- Abstract design != concrete implementation

Scapula: An Open-Source Toolkit For Fuzzing and Verification of ARM CPUs

October 17, 2019

## Any Questions

Jared Wright

*Software Engineer/ Researcher*

wrightj@ainfosec.com

github.com/jaredwright

ais

WWW.AINFOSEC.COM