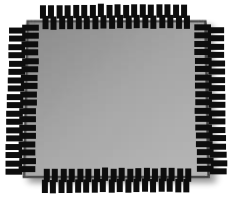
UNIVERSITY OF BIRMINGHAM

Collaboration project with Hewlett-Packard



CHERI – Zephyr

Dr Jennifer Jackson

CHERITech'24

UNIVERSITY OF BIRMINGHAM | CENTRE FOR CYBER SECURITY AND PRIVACY

What will be covered

- **CHERI-Zephyr project with HP**
  - Zephyr and memory vulnerabilities
  - Modifications to Zephyr
  - Known Zephyr CVE case studies
  - Live Demo – Buffer overflow

- **Open-source operating system**
- **Small footprint** designed for **embedded** devices.
- **Highly configurable** - supports architectures and boards.
- **Some memory protection**:
  - stack overflow protection, thread-level memory protection (plus others), but not universal.

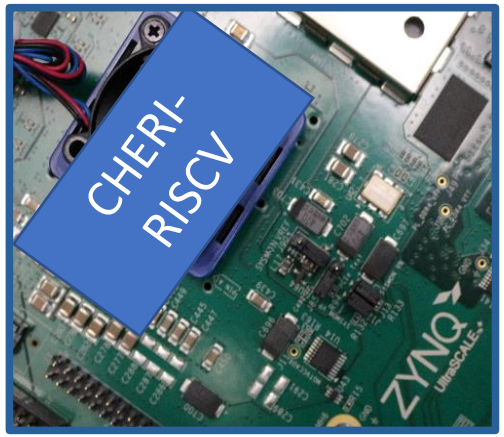
Large C-code base

- Large amount of **C code** & architecture specific **assembly**, leading to memory safety issues.
- **Memory safety vulnerabilities** such as traditional
  - buffer overflows (CVE-2020-10064),
  - out of bounds issues (CVE-2021-3330)
  - NULL pointer dereferences (CVE-2021-3319/3320/3322)

Not memory safe!

Modifications to Zephyr

- Tool Chain CMake Support
- Code Modifications
- Board support



CHERI-RISC-V 64 bit processors:

QEMU CHERI-RISC-V 64 bit
FPGA CHERI-Flute RISC-V 64 bit

Toolchain CMake support for LLVM-CHERI (1/2)

- CHERI SDK

LLVM-CHERI tool chain

QEMU –> CHERI-RISCV64

- Zephyr build

west

west build -p always -b qemu_riscv64 samples/hello_world

cmake

gcc - default

llvm-cheri

```
export ZEPHYR_TOOLCHAIN_VARIANT=llvm-cheri
export LLVM_CHERI_TOOLCHAIN_PATH=/path/cheri/output/sdk
export QEMU_BIN_PATH=/path/cheri/output/sdk/bin
```

Toolchain CMake support for LLVM-CHERI (2/2)

New set of CMake files:

Build zephyr

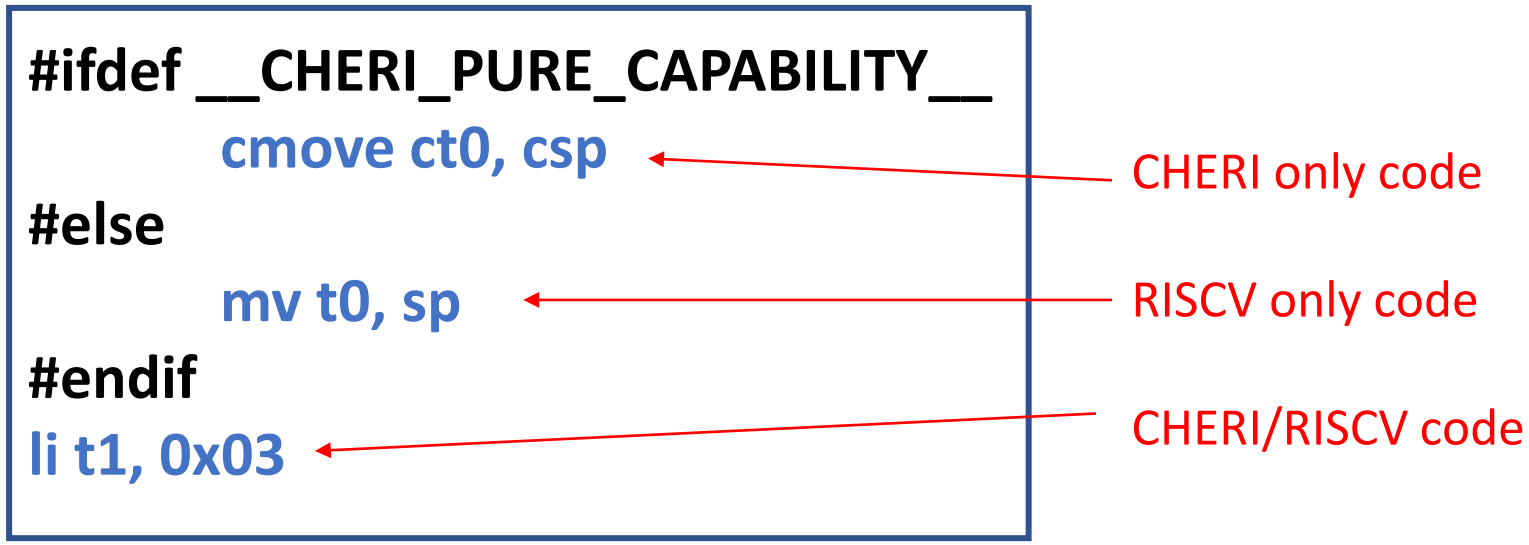
- **toolchain->llvm-cheri**
- compiler->llvm-cheri
- linker->lld-cheri
- bintools->llvm-cheri

```
set(COMPILER llvm-cheri)
set(LINKER lld-cheri)
set(BINTOOLS llvm-cheri)
```

```
if(CONFIG_CHERI)
   #if compiling for riscv64 CHERI-PURECAP
   string(PREPEND CMAKE_ASM_FLAGS "-march=rv64gcxcheri -mabi=l64pc128d ")
   string(PREPEND CMAKE_C_FLAGS   "-march=rv64gcxcheri -mabi=l64pc128d ")
   string(PREPEND CMAKE_CXX_FLAGS "-march=rv64gcxcheri -mabi=l64pc128d ")
else()
   #if compiling for normal riscv64
   string(PREPEND CMAKE_ASM_FLAGS "-march=rv64gc ")
   string(PREPEND CMAKE_C_FLAGS   "-march=rv64gc ")
   string(PREPEND CMAKE_CXX_FLAGS "-march=rv64gc ")
endif()
```

Code Modifications – assembly (1/8)

- cmake support not enough
- mods to architecture specific **assembly** and c code **in-line assembly**
- #ifdef __CHERI_PURE_CAPABILITY__



Code Modifications – macros and defines (2/8)

- CHERI-alternative macros

RISCV

```
.macro lr, rd, mem
ld \rd, \mem
.endm
```

integer load at addr

CHERI

```
.macro clr, rd, mem
ld.cap \rd, \mem
.endm
```

integer load at cap addr

arch/riscv/core/asm_macros.inc

- CHERI-alternative defines

RISCV

```
#define DO_CALLER_SAVED(op) \
   RV_E(op t0, __z_arch_esf_t_t0_OFFSET(sp));\
……
```

CHERI

```
#define DO_CALLER_SAVED(op) \
   RV_E(op ct0, __z_arch_esf_t_ct0_OFFSET(csp));\
   …….
```

arch/riscv/core/isr.S

Code Modifications – assembly boot code (3/8)

- Boot the riscv64 machine into a capability mode
  - Switch modes
- set up specific capability requirements such as the
  - global pointer table.
  - global capabilities
  - Bounding PCC
  - Zero out DDC (root capability)

Code Modifications –c code structures and alignment (4/8)

2. Structure alignment

} __packed; ⟶ } __aligned(16);

1. Types

```
struct __esf {
        #ifdef __CHERI_PURE_CAPABILITY__
        uintptr_t ca0;
        …
        #else
        unsigned long a0;
        ...
        #endif
```

Include/zephyr/arch/riscv/exp.h

3. Section alignment – in linker script

```
#ifdef CONFIG_CHERI
   ITERABLE_SECTION_RAM_GC_ALLOWED(k_sem, 16)
   …
#else
   ITERABLE_SECTION_RAM_GC_ALLOWED(k_sem, 4)
   ….
#endif
```

Include/zephyr/linker/common-ram.ld

Code Modifications – Automated generation of fixed offsets (5/8)

Offset code → **Fixed offsets generated for RISCV64 Automated header file** → **Offsets used in assembly**

```
…
#ifdef __CHERI_PURE_CAPABILITY__
GEN_OFFSET_SYM(z_arch_esf_t, mepcc);
#else
GEN_OFFSET_SYM(z_arch_esf_t, mepc);
#endif
….
```

arch/riscv/core/offsets/offsets.c

```
#ifndef __GEN_OFFSETS_H__
#define __GEN_OFFSETS_H__
…

#define __z_arch_esf_t_mepc_OFFSET 0x80

…

#endif
```

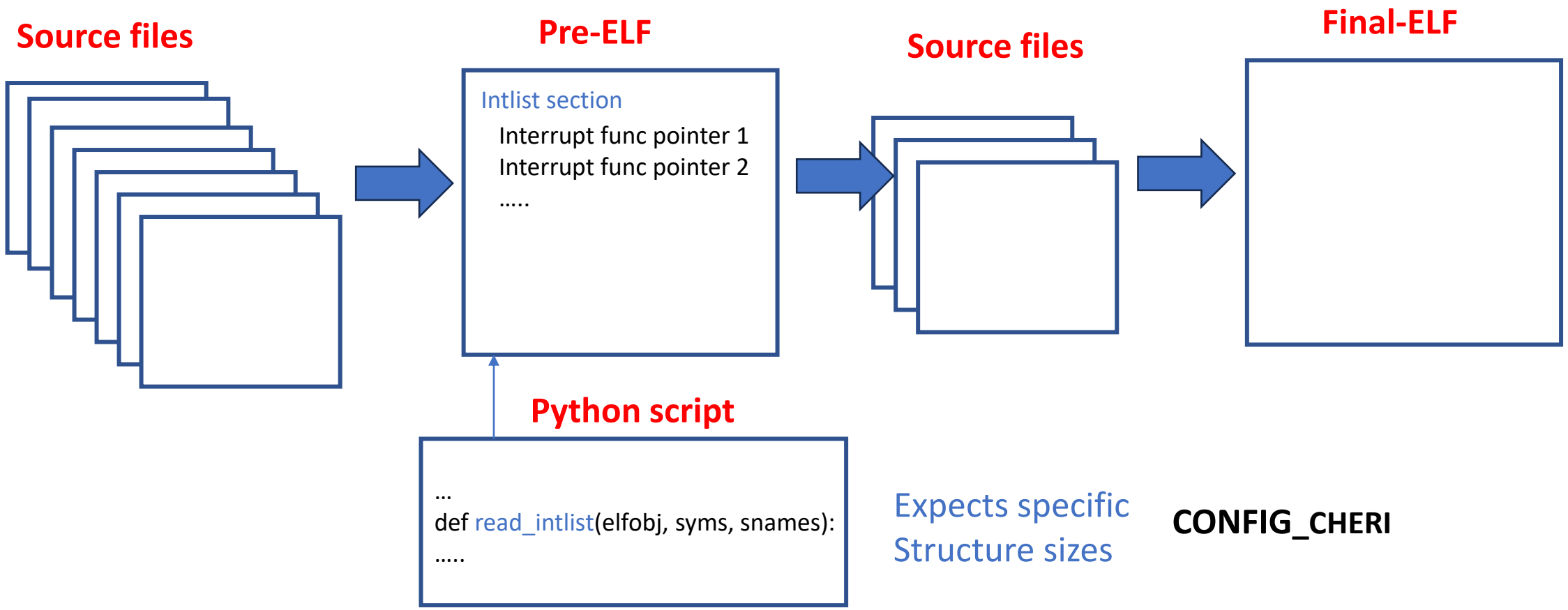
Build/zephyr/include/generated/offsets.h

```
/* Save MEPC register */
#ifdef __CHERI_PURE_CAPABILITY__
   cspecialr ct0, mepcc
   csc ct0, __z_arch_esf_t_mepcc_OFFSET(csp)
#else
   csrr t0, mepc
   sr t0, __z_arch_esf_t_mepc_OFFSET(sp)
#endif
```

arch/riscv/core/isr.S

Code Modifications – Pre-ELF processing Python Script (6/8)

Zephyr build process involves a multi-step approach



Expects specific Structure sizes

CONFIG_CHERI

Code Modifications – Interrupt tables (7/8)

Pre-ELF

Intlist section

Interrupt func pointer 1
Interrupt func pointer 2
…..

Python script

```
…
def read_intlist(elfobj, syms, snames):
…..
```

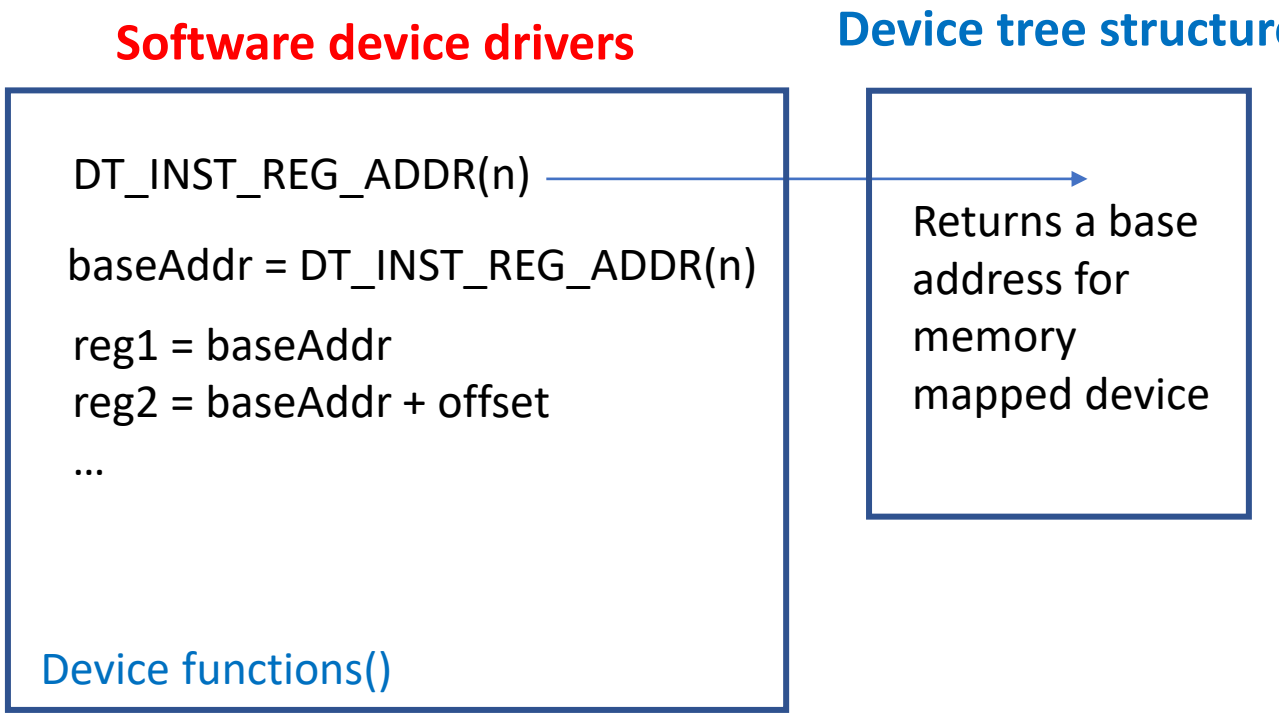
Interrupt table

```
 9 typedef void (* ISR)(const void *);
10 struct _isr_table_entry __sw_isr_table _sw_isr_table[1035] = {
11         {(const void *)0x0, (ISR)((uintptr_t)&z_irq_spurious)},
12         {(const void *)0x0, (ISR)((uintptr_t)&z_irq_spurious)},
13         {(const void *)0x0, (ISR)((uintptr_t)&z_irq_spurious)},
14         {(const void *)0x0, (ISR)((uintptr_t)&z_irq_spurious)},
15         {(const void *)0x0, (ISR)((uintptr_t)&z_irq_spurious)},
16         {(const void *)0x0, (ISR)((uintptr_t)&z_irq_spurious)},
17         {(const void *)0x0, (ISR)((uintptr_t)&z_irq_spurious)},
18         {(const void *)0x0, (ISR)0x8000159c},
19         {(const void *)0x0, (ISR)((uintptr_t)&z_irq_spurious)},
20         {(const void *)0x0, (ISR)((uintptr_t)&z_irq_spurious)},
21         {(const void *)0x0, (ISR)((uintptr_t)&z_irq_spurious)},
22         {(const void *)0x0, (ISR)0x800010f8},
23         /* Level 2 interrupts start here (offset: 12) */
24         {(const void *)0x0, (ISR)((uintptr_t)&z_irq_spurious)},
25         {(const void *)0x0, (ISR)((uintptr_t)&z_irq_spurious)},
26         {(const void *)0x0, (ISR)((uintptr_t)&z_irq_spurious)},
27         {(const void *)0x0, (ISR)((uintptr_t)&z_irq_spurious)},
28         {(const void *)0x0, (ISR)((uintptr_t)&z_irq_spurious)},
```

Fixed addresses
Not valid in the CHERI Architecture!

Code Modifications – Device tree mapping into valid capabilities(8/8)

No capabilities
Fixed address used for device base address
Returned from device tree structure



Board Configurations

QEMU CHERI-RISCV64

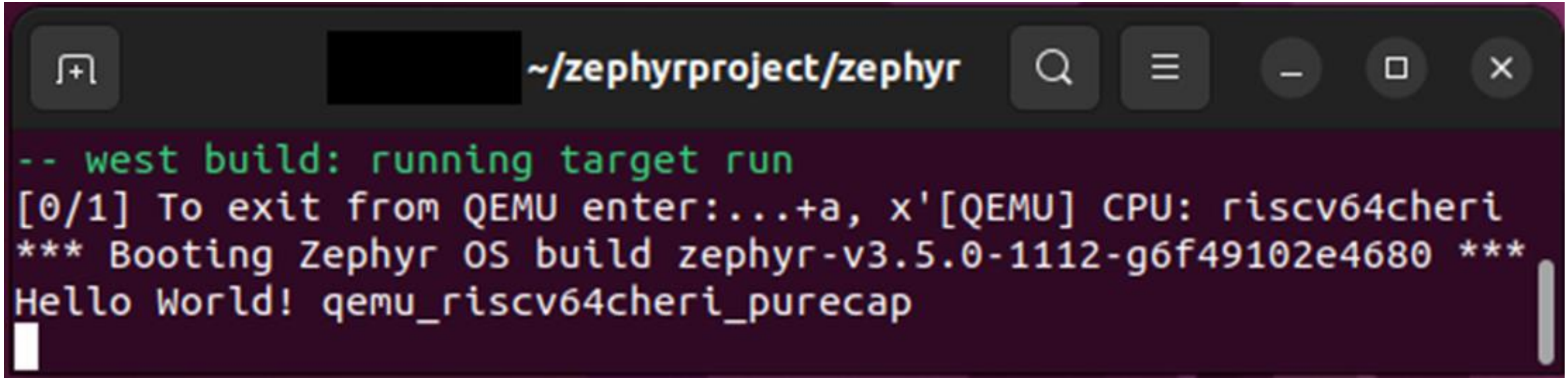
- Qemu_riscv64cheri
- Qemu_riscv64cheri_purecap
- Qemu_riscv64cheri_smp
- Qemu_riscv64cheri_smp_purecap

FPGA CHERI- Flute processor

- Zynq_besspinflutecheri
- Zynq_besspinflutecheri_purecap

```
west build -p always -b qemu_riscv64cheri_purecap samples/hello_world
```

Hello World! Zephyr App on CHERI-RISCV



CVE case studies

- Buffer overflow
  - CVE-2020-10065 Bluetooth
- NULL pointer de-reference
  - CVE-2020-10066 Bluetooth
- Out-of-bounds write
  - CVE-2021-3330 IEEE 802.15.4

CVE case studies

- Buffer overflow
  - CVE-2020-10065 Bluetooth — CHERI exception length violation!
- NULL pointer de-reference
  - CVE-2020-10066 Bluetooth — CHERI exception tag violation!
- Out-of-bounds write
  - CVE-2021-3330 IEEE 802.15.4 — CHERI exception length violation!

Demo – CHERI in-action – Buffer Overflow!

```
strcpy(buffer, printstring);
```

string copy into buffer without any string length checks

1

Normal Zephyr running on a RISC-V architecture:

The input string DOES NOT overflow the buffer.

2

Normal Zephyr running on a RISC-V architecture:

A BUFFER OVERFLOW causes the return address of the function to be overwritten in memory to run attacker code.

3

CHERI Zephyr running on a CHERI-RISC-V architecture:

A hardware exception occurs and the program halts before the buffer overflow can be exploited.

Demo – CHERI in-action – Buffer Overflow!

strcpy(buffer, printstring);

string copy into buffer without any string length checks

```
leo@office:~/Documents/cheri-processors$ ./uart_zephyr.sh zcu102
*** Booting Zephyr OS build zephyr-v3.5.0-1112-g6f49102e4680 ***
Zephyr Buffer Overflow Demo with CHERI! zynq_besspinflutecheri
Input string: ABCDEFGHIJKLMNO
buffer string: ABCDEFGHIJKLMNO
End of demo...!
^Cleo@office:~/Documents/cheri-processors$ ./uart_zephyr.sh zcu102
*** Booting Zephyr OS build zephyr-v3.5.0-1112-g6f49102e4680 ***
Zephyr Buffer Overflow Demo with CHERI! zynq_besspinflutecheri
Input string: ABCDEFGHIJKLMNOPQRSTUVW
X
buffer string: ABCDEFGHIJKLMNO
You have been HACKED!
^Cleo@office:~/Documents/cheri-processors$ ./uart_zephyr.sh zcu102
*** Booting Zephyr OS build zephyr-v3.5.0-1112-g6f49102e4680 ***
Zephyr Buffer Overflow Demo with CHERI! zynq_besspinflutecheri_purecap
Input string: ABCDEFGHIJKLMNOPQRSTUVW
X
```

1

2

3

UNIVERSITY OF BIRMINGHAM

Questions?

UNIVERSITY OF BIRMINGHAM | CENTRE FOR CYBER SECURITY AND PRIVACY