Collaboration project with Hewlett-Packard

CHERI – Zephyr

Dr Jennifer Jackson
CHERITech’24
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
Zephyr

- 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 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)
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)

- LLVM-CHERI tool chain

  - CHERI SDK
  - QEMU → CHERI-RISCV64

- Zephyr build

  - west build -p always -b qemu_riscv64 samples/hello_world

  - gcc - default

  - cmake

  - `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:

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

```cmake
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()
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__`

```
#include <elfa.h>

CHERITech'24

#ifdef __CHERI_PURE_CAPABILITY__
cmove ct0, csp
#else
mv t0, sp
#endif
li t1, 0x03
```

CHERI only code
RISCV only code
CHERI/RISCV code
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/asmmacros.inc

• CHERI-alternative defines

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

 ........
```

arch/riscv/core/asm_macros.inc

```
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)
1. Types

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

#include/zephyr/arch/riscv/exp.h
```

2. Structure alignment

```
} __packed;        } __aligned(16);
```

3. Section alignment – in linker script

```
#include 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

```c
... #ifdef __CHERI_PURE__CAPABILITY__ GEN_OFFSET_SYM(z_arch_esf_t, mepcc);
#else GEN_OFFSET_SYM(z_arch_esf_t, mepc);
#endif ....
```

Fixed offsets generated for RISCV64

```c
#endif
```

Automated header file

```c
#ifndef __GEN_OFFSETS_H__
#define __GEN_OFFSETS_H__
...
#define __z_arch_esf_t_mepc_OFFSET 0x80
...
#endif
```

Offsets used in assembly

```c
/* Save MEPC register */
#ifndef __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/offsets/offsets.c Build/zephyr/include/generated/offsets.h arch/riscv/core/ isr.S
Code Modifications – Pre-ELF processing Python Script (6/8)

Zephyr build process involves a multi-step approach

Source files

Pre-ELF

Intlist section
Interrupt func pointer 1
Interrupt func pointer 2
.....

Source files

Python script

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

Final-ELF

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

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

Interrupt table

```
typedef void (* ISR)(const void *);
struct _isr_table_entry __sw_isr_table__sw_isr_table[1025] = {
    [0] 0x5, (ISR)((uintptr_t)0伊拉_qr_spurious),
    [1] 0x6, (ISR)((uintptr_t)0伊拉_qr_spurious),
    [2] 0x7, (ISR)((uintptr_t)0伊拉_qr_spurious),
    [3] 0x8, (ISR)((uintptr_t)0伊拉_qr_spurious),
    [4] 0x9, (ISR)((uintptr_t)0伊拉_qr_spurious),
    [5] 0xA, (ISR)((uintptr_t)0伊拉_qr_spurious),
    [6] 0xB, (ISR)((uintptr_t)0伊拉_qr_spurious),
    [7] 0xC, (ISR)((uintptr_t)0伊拉_qr_spurious),
    [8] 0xD, (ISR)((uintptr_t)0伊拉_qr_spurious),
    [9] 0xE, (ISR)((uintptr_t)0伊拉_qr_spurious),
    [10] 0xF, (ISR)((uintptr_t)0伊拉_qr_spurious),
    /* Level 2 interrupts start here (offset: 12) */
    [11] 0x0, (ISR)((uintptr_t)0伊拉_qr_spurious),
    [12] 0x1, (ISR)((uintptr_t)0伊拉_qr_spurious),
    [13] 0x2, (ISR)((uintptr_t)0伊拉_qr_spurious),
    [14] 0x3, (ISR)((uintptr_t)0伊拉_qr_spurious),
    [15] 0x4, (ISR)((uintptr_t)0伊拉_qr_spurious),
    [16] 0x5, (ISR)((uintptr_t)0伊拉_qr_spurious),
    [17] 0x6, (ISR)((uintptr_t)0伊拉_qr_spurious),
    [18] 0x7, (ISR)((uintptr_t)0伊拉_qr_spurious),
};
```

Fixed addresses
Not valid in the CHERI Architecture!
No capabilities
Fixed address used for device base address
Returned from device tree structure

Software device drivers

\[
\begin{align*}
\text{baseAddr} &= \text{DT_INST_REG_ADDR}(n) \\
\text{reg1} &= \text{baseAddr} \\
\text{reg2} &= \text{baseAddr} + \text{offset} \\
&\ldots
\end{align*}
\]

Device tree structure

Returns a base address for memory mapped device
Board Configurations

- Qemu_riscv64cheri
- Qemu_riscv64cheri_purecap
- Qemu_riscv64cheri_smp
- Qemu_riscv64cheri_smp_purecap
- 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
  - CVE-2020-10066 Bluetooth
- NULL pointer de-reference
  - CVE-2020-10066 Bluetooth
- Out-of-bounds write
  - CVE-2021-3330 IEEE 802.15.4
Demo – CHERI in-action – Buffer Overflow!

strncpy(buffer, printstring);

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!

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

string copy into buffer without any string length checks
Questions?