UNIVERSITY OF BIRMINGHAM

CARDIS 2022, November 7 – 9

CHERI – Exercises Part 2

UNIVERSITY OF BIRMINGHAM | CENTRE FOR CYBER SECURITY AND PRIVACY

Help sheet

Work a bit more independently, based on what tools and commands learned so far

Help sheet with useful commands for reference

Cheri exercises help sheet

Virtualbox VM access

Username: cheri

Password: cheri

UTM VM access

Username: user

Password: user

Cross compiler script (from Ubuntu)

Run from ~/cheri/cheri-exercises/tools

target: RISCV (riscv64) or CHERI RISCV (riscv64-purecap).

./ccc <target> <inputpathandfile.c> -o <outputpathandfile>

Start QEMU

From ~/cheribuild

./cheribuild.py run-riscv64-purecap (Do not update!)

Login: root

Quit: Ctrl&a, then x

Samba Mount – shared directory (From QEMU)

mount_smbfs -I 10.0.2.4 -N //10.0.2.4/source_root /mnt

GDB (From QEMU)

gdb <binaryfile>

gdb <binaryfile> -c <coredumpfile>

info about register a0: info reg a0

quit: q

Dissasemble with obidump (from Ubuntu)

From ~/cheri/output/sdk/bin/

./llvm-objdump -dS ~/cheri/cheri-exercises/src/exercises/<exercise binary>

What we will cover

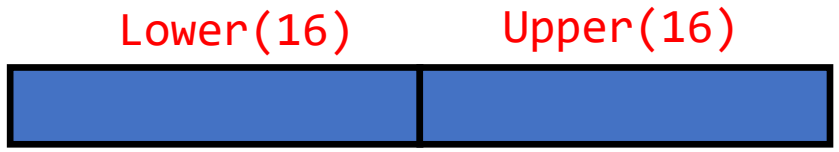
- **Agenda: Part 1 (before lunch 11.00am-12noon)**
- **Introduction to CHERI:** What is CHERI and what does it protect against? What is ARM Morello?
- **Introduction to CHERI exercises:** Understanding the build environment, toolchain, and debugger
- **Guided exercises using the VM:**
  - Compiling and running RISC-V and CHERI-RISC-V programs
  - Disassembling and debugging RISC-V and CHERI-RISC-V binaries
  - Understanding CHERI exceptions
- **Agenda: Part 2 (after lunch 13.30pm-15.30pm)**
- **Attacks (and mitigation through CHERI):**
  - Identifying and fixing a buffer overflow
  - Overwriting function pointers
  - Attempting arbitrary code execution through a buffer overflow

Exercise 4 – Buffer overflow

Exercise: ~/cheri/cheri-exercises/src/exercises/buffer-overflow-stack

Objective – Demonstrate a stack buffer overflow, and how CHERI can prevent an overflow.

Buffer-overflow-stack.c




```
char upper[0x10];
char lower[0x10];
```

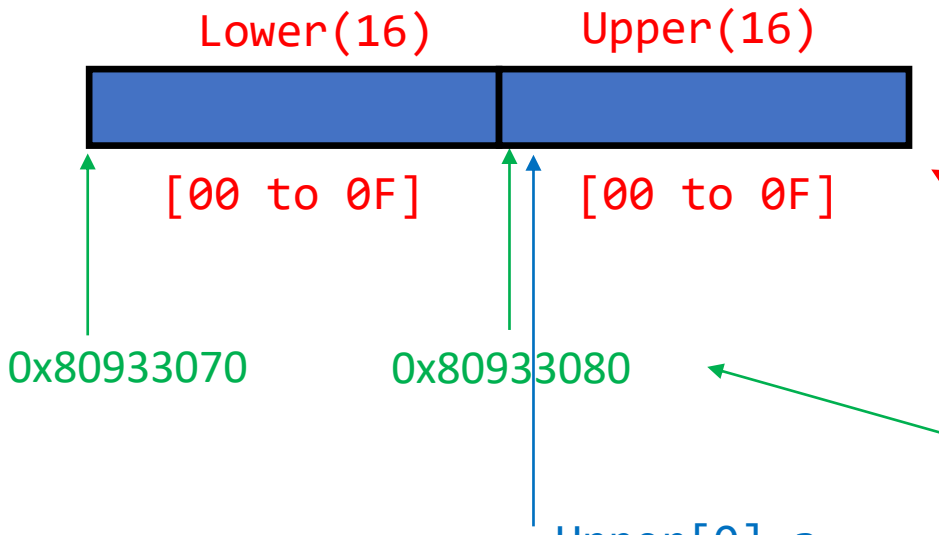
Exercise 4 – Buffer overflow

Exercise: ~/cheri/cheri-exercises/src/exercises/buffer-overflow-stack

1. Observe the code.
2. Compile and run **buffer-overflow-stack.c** for
   - a **RISCV** target and
   - a **CHERI – RISCV** target.
3. For **CHERI** use **GDB** to identify the fault.
4. What in the code is causing the overflow, how would you fix it?

Exercise 4 – Buffer overflow

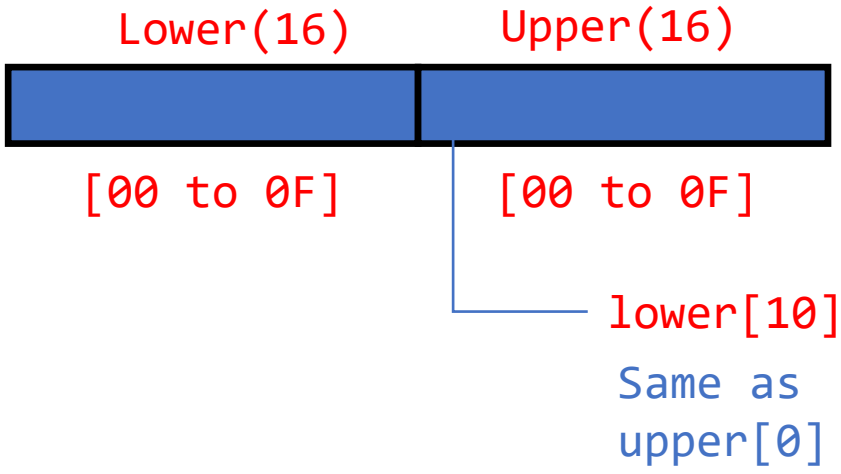
What is happening in the code?




```
void
write_buf(char *buf, size_t ix)
{
        buf[ix] = 'b';
}

int
main(void)
{
        char upper[0x10];
        char lower[0x10];

        printf("upper = %p, lower = %p, diff = %zx\n",
            upper, lower, (size_t)(upper - lower));

        /* Assert that these get placed how we expect */
        assert((ptraddr_t)upper == (ptraddr_t)&lower[sizeof(lower)]);

        upper[0] = 'a';
        printf("upper[0] = %c\n", upper[0]);

        write_buf(lower, sizeof(lower));

        printf("upper[0] = %c\n", upper[0]);

        return 0;
}
```

Exercise 4 – Buffer overflow

What is happening in the code?

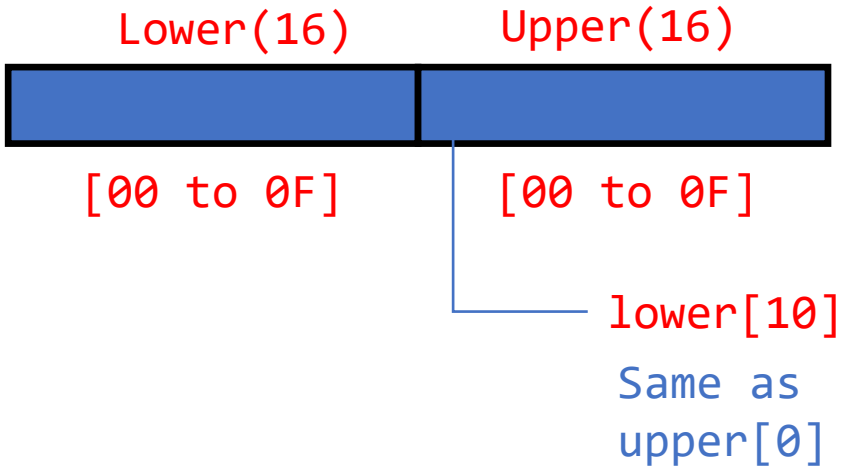



```
/* Assert that these get placed how we expect */
assert((ptraddr_t)upper == (ptraddr_t)&lower[sizeof(lower)]);

upper[0] = 'a';
printf("upper[0] = %c\n", upper[0]);

write_buf(lower, sizeof(lower));

printf("upper[0] = %c\n", upper[0]);
```

```
write_buf(char *buf, size_t ix)
{
        buf[ix] = 'b';
}
```

Exercise 4 – Buffer overflow

What is happening in the code?




```
/* Assert that these get placed how we expect */
assert((ptraddr_t)upper == (ptraddr_t)&lower[sizeof(lower)]);

upper[0] = 'a';
printf("upper[0] = %c\n", upper[0]);

write_buf(lower, sizeof(lower));

printf("upper[0] = %c\n", upper[0]);
```

```
write_buf(char *buf, size_t ix)
{
        buf[ix] = 'b';
}
```

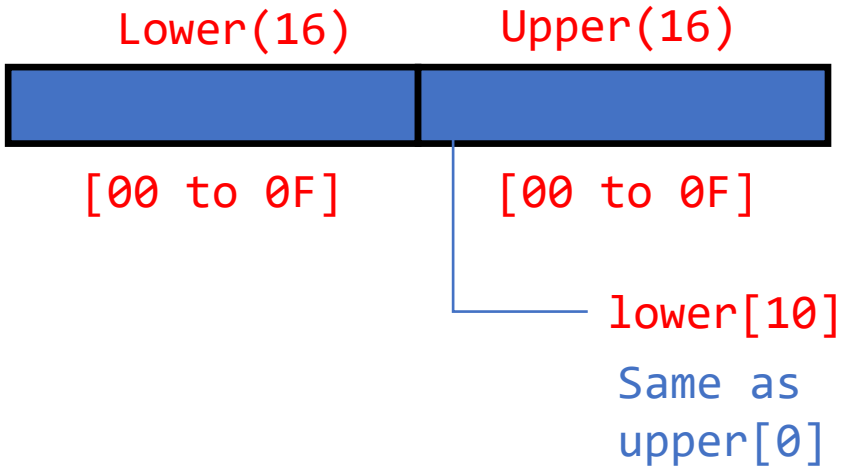
RISCV

upper = 0x80933080, lower = 0x80933070, diff = 10
upper[0] = a
upper[0] = b

CHERI - RISCV

upper = 0x3fffdfff50, lower = 0x3fffdfff40, diff = 10
upper[0] = a
In-address space security exception

Exercise 4 – Buffer overflow



CHERI - RISC-V – GDB output

Program terminated with signal SIGPROT, CHERI protection violation
Capability bounds fault caused by register ca0.
#0 0x0000000000101d30 in write_buf (buf=<optimized out>, ix=<optimized out>)
...
13 buf[ix] = 'b';

Write_buf function

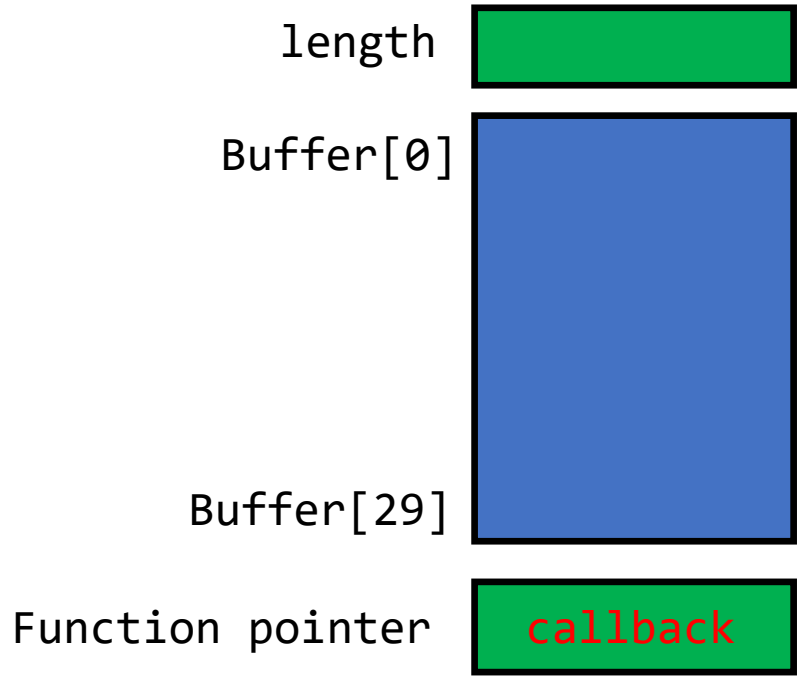
Exercise 5 – Corrupt a pointer

Exercise: ~/cheri/cheri-exercises/src/exercises/control-flow-pointer

Objective – use CHERI to prevent a corrupted function pointer from being used for further memory access.

buffer-overflow-fnptr.c

```
struct buf {
        size_t length;
        int buffer[30];
        size_t (*callback)(struct buf *);
};
```



Exercise 5 – Corrupt a pointer

Exercise: ~/cheri/cheri-exercises/src/exercises/control-flow-pointer

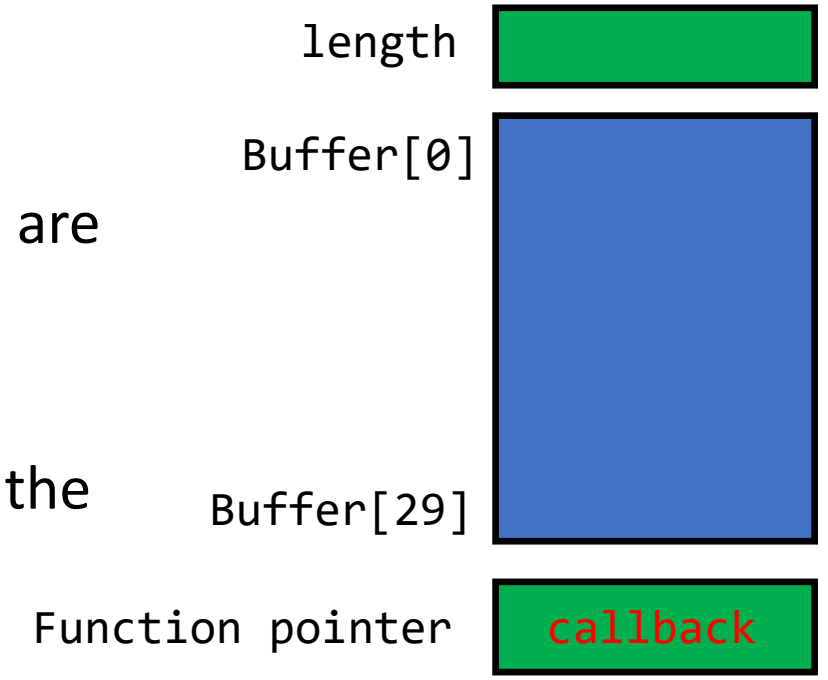
1 . Observe the code.

2. Compile and run **buffer-overflow-fnptr.c** for
   - a **RISCV** target and
   - a **CHERI – RISCV** target.

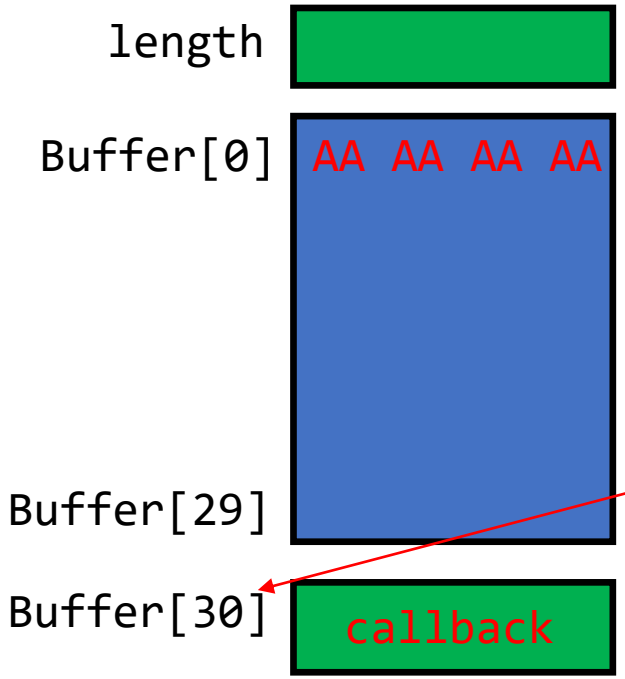
3. Run both binaries in **GDB** from **QEMU.** What faults are you seeing and why? Does CHERI prevent the corruption? If not, what does it prevent?

4. What is causing the corruption, how would you fix the bug?

```
struct buf {
        size_t length;
        int buffer[30];
        size_t (*callback)(struct buf *);
};
```



Exercise 5 – Corrupt a pointer




```
struct buf {
        size_t length;
        int buffer[30];
        size_t (*callback)(struct buf *);
};

void
fill_buf(struct buf *bp)
{
        bp->length = sizeof(bp->buffer)/sizeof(*bp->buffer);
        for (size_t i = 0; i <= bp->length; i++)
                bp->buffer[i] = 0xAAAAAAAA;
}

size_t
count_screams(struct buf *bp)
{
        int screams = 0;

        for (size_t i = 0; i < bp->length; i++)
                screams += bp->buffer[i] == 0xAAAAAAAA ? 1 : 0;
        return screams;
}

struct buf b = {.callback = count_screams};

int
main(void)
{
        fill_buf(&b);

        printf("Words of screaming in b.buffer %zu\n", b.callback(&b));

        return 0;
}
```

Function pointer

32 bit integer number

Bounds of b
Covers all buf
So overwrites
Even in CHERI

Exercise 5 – Corrupt a pointer

GDB output

RISC-V

Program received signal SIGSEGV, segmentation fault
0x00000000aaaaaaaa in ?? ()

CHERI - RISC-V

Program received signal SIGPROT, CHERI protection violation
Capability tag fault caused by register ca1
0x0000000000101cfa in main ()
...
38 printf("Words of screaming in b.buffer %zu\n", b.callback(&b));

Tag fault because have written an integer value to a capability pointer in memory (without using capability instructions) so clears the tag but doesn't fault until the printf

Exercise 6 – Code execution

Exercise: ~/cheri/cheri-exercises/src/missions/buffer-overflow-control-flow

Objective – Attempt arbitrary code execution through a buffer overflow

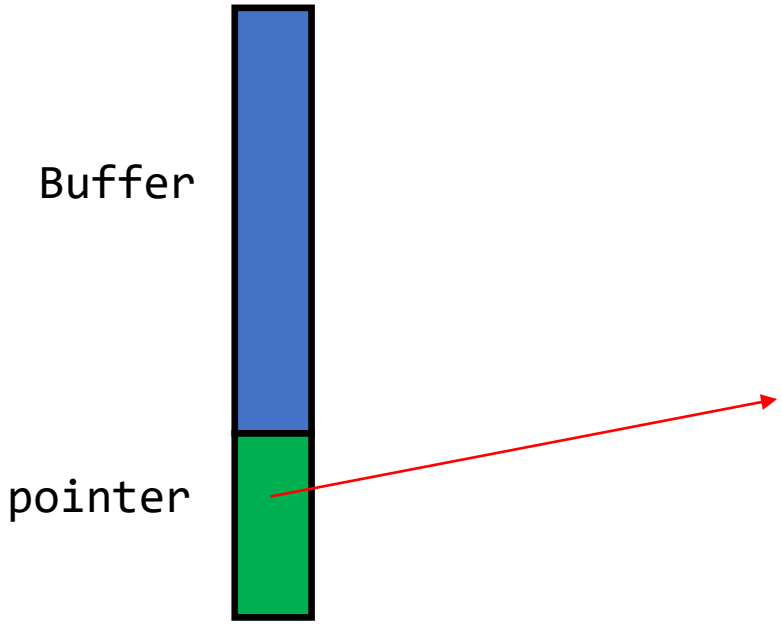
buffer-overflow.c **btpalloc.c**

3 Parts Attempt arbitrary code execution

1. RISCV
2. CHERI-RISCV
3. CHERI-RISCV weakened memory allocator

Exercise 6 – Code execution

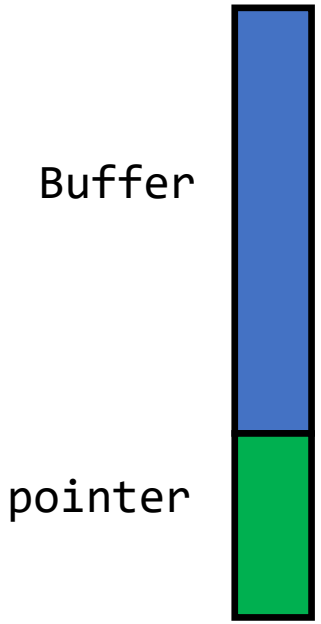
Exercise: ~/cheri/cheri-exercises/src/missions/buffer-overflow-control-flow




```
void
success(void)
{
        puts("Exploit successful!");
}

void
failure(void)
{
        puts("Exploit unsuccessful!");
}
```

Exercise 6 – Code execution

Exercise: ~/cheri/cheri-exercises/src/missions/buffer-overflow-control-flow



Memory allocator **btpalloc.c**

#ifdef __CHERI_PURE_CAPABILITY__

#if defined (__CHERI_PURE_CAPABILITY__)
&& !defined(CHERI_NO_ALIGN_PAD)

Exercise 6 – Code execution

Exercise: ~/cheri/cheri-exercises/src/missions/buffer-overflow-control-flow

Part 1 . **RISCV -** exploit the binary to execute the **success** function.

- Run / understand the code, how much memory is allocated for the buffer?
- Create an appropriate input to overflow the buffer

Part 2 . **CHERI RISCV** - attempt to exploit the binary. (Think **imprecision in the bounds** of **large** capabilities).

- Is the memory allocated for the buffer, the same as for riscv or different?
- Try to overflow the buffer, if it cannot be exploited, what is the fault?

Part 3 . **WEAKENED CHERI RISCV** – compile with -DCHERI_NO_ALIGN_PAD (Fail to pad allocations to account for capability bounds imprecision).

- How is the memory allocated different to previous? Can the buffer be overflowed?
- Try to overflow the buffer, if it cannot be exploited, explain why?

Exercise 6 – Hints

Part 1 . **RISCV -** exploit the binary to execute the **success** function.

- Run / understand the code, how much memory is allocated for the buffer?
- Create an appropriate input to overflow the buffer

Use objdump to find address of success function

```
12 success(void)
13 {
14         puts("Exploit successful!");
15 }
```

Check btpalloc.c, what size memory is allocated to the buffer?

```
/* RISC-V ABIs require 16-byte alignment */
allocsize = __builtin_align_up(size, 16);
```

One method to input characters and address values

perl -e 'print "a"x10 . "\xff\x1f\x01\x00"' > inputtest (Linux)

./buffer-overflow-riscv < inputtest (QEMU)

Exercise 6 – Hints

Part 2 . **CHERI RISCV** - attempt to exploit the binary. (**imprecision in the bounds**).

- Is the memory allocated for the buffer, the same as for riscv or different?
- Try to overflow the buffer, if it cannot be exploited, what is the fault?

Use objdump to find address of success function

```
12 success(void)
13 {
14         puts("Exploit successful!");
15 }
```

Check btpalloc.c, what size memory is allocated to the buffer with cheri?
Can we calculate it?

```
#if defined(__CHERI_PURE_CAPABILITY__) && !defined(CHERI_NO_ALIGN_PAD)

        allocsize = cheri_representable_length(allocsize);
        alloc = __builtin_align_up(alloc,
            ~cheri_representable_alignment_mask(allocsize) + 1);
        allocsize += (char *)alloc - (char *)btpmem;
#endif
```

One method to input characters and address values

perl -e 'print "a"x25024 ."\xff\x1f\x01\x00"' > inputtest (Linux)

./buffer-overflow-riscv < inputtest (QEMU)

Exercise 6 – Hints

Part 3 . **WEAKENED CHERI RISCV** – compile with -DCHERI_NO_ALIGN_PAD (Fail to pad allocations to account for capability bounds imprecision).

- How is the memory allocated different to previous? Can the buffer be overflowed?
- Try to overflow the buffer, if it cannot be exploited, explain why?

Use objdump to find address of success function

Check btpalloc.c, what size memory is allocated to the buffer? But what does the memory allocator pointer increment to, is it the same?

```
12 success(void)
13 {
14         puts("Exploit successful!");
15 }
```

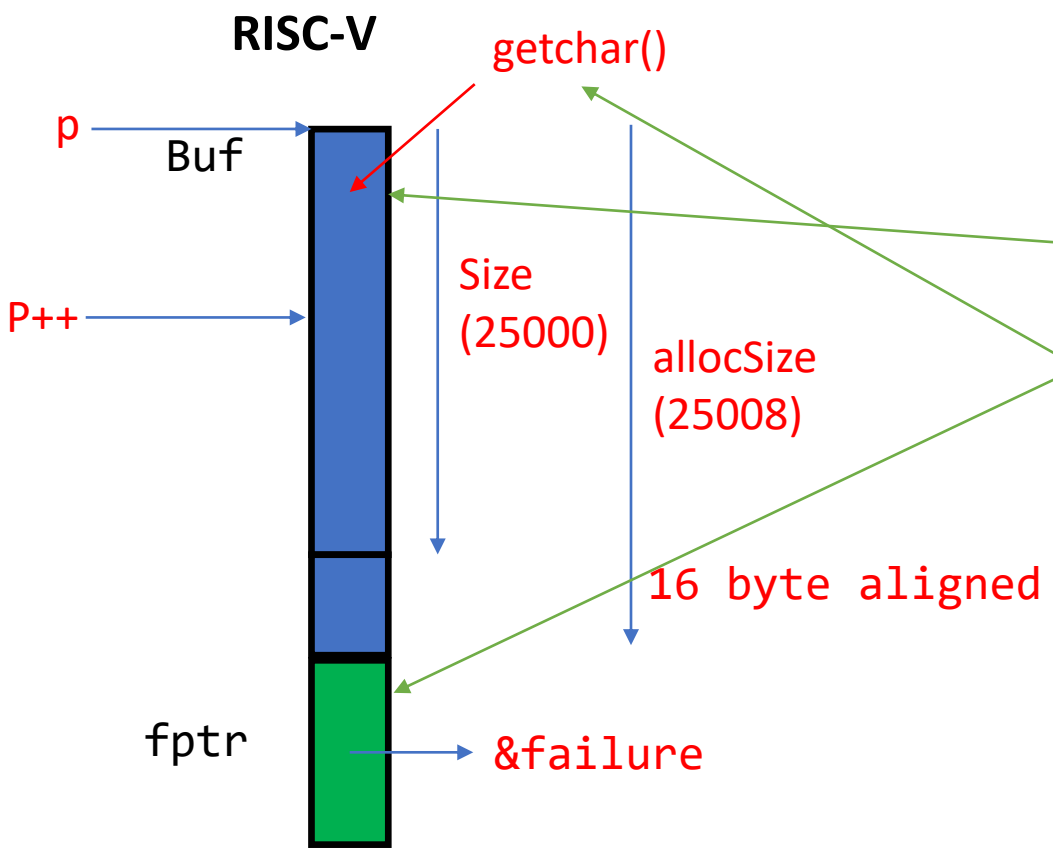
```
        btpmem = (char *)btpmem + allocsize;
        btpsize -= allocsize;
#ifdef __CHERI_PURE_CAPABILITY__
        alloc = cheri_bounds_set(alloc, size);
#endif
```

One method to input characters and address values

perl -e 'print "a"x10 ."\xff\x1f\x01\x00"' > inputtest (Linux)

./buffer-overflow-riscv < inputtest (QEMU)

Exercise 6 – Part 1

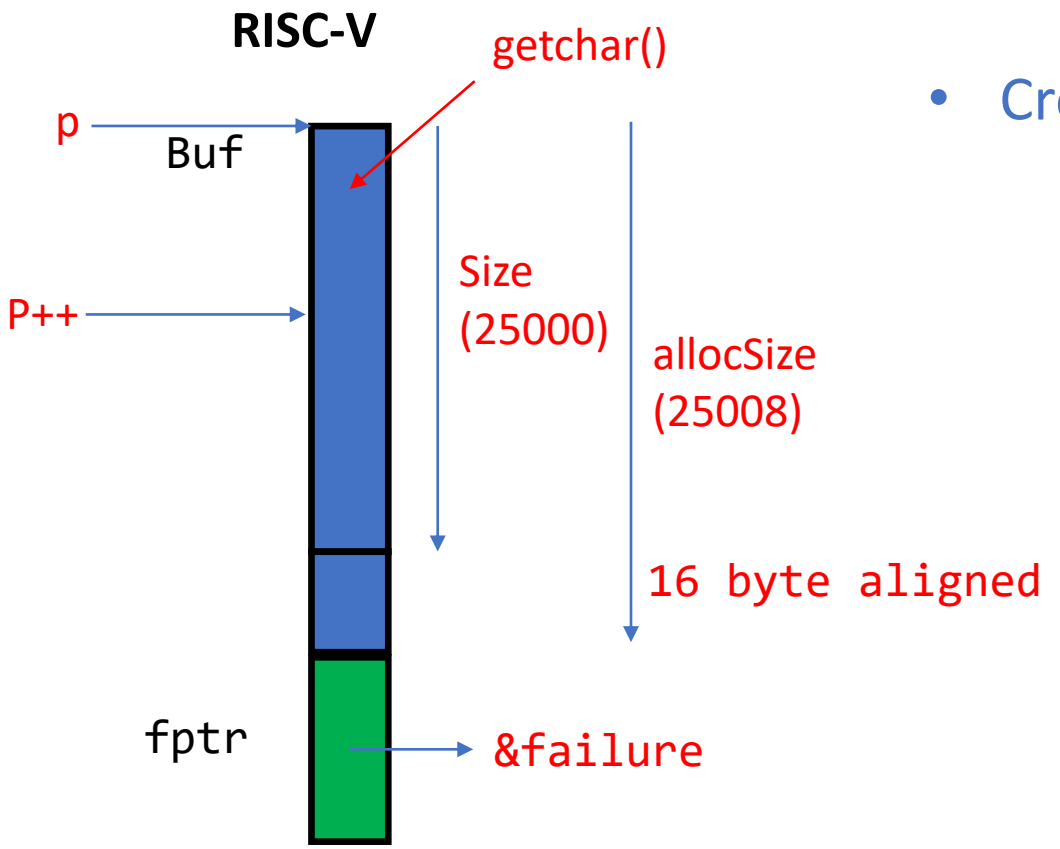



```
41 int
42 main(void)
43 {
44         int ch;
45         char *buf, *p;
46         uint16_t sum;
47         void (**fptr)(void);
48
49         buf = btpmalloc(25000);
50         fptr = btpmalloc(sizeof(*fptr));
51
52         main_asserts(buf, fptr);
53
54         *fptr = &failure;
55
56         p = buf;
57         while ((ch = getchar()) != EOF)
58                 *p++ = (char)ch;
59
60         if ((uintptr_t)p & 1)
61                 *p++ = '\0';
62
63         sum = ipv4_checksum((uint16_t *)buf, (p - buf) / 2);
64         printf("Checksum: 0x%04x\n", sum);
65
66         btpfree(buf);
67
68         (**fptr)();
69
70         btpfree(fptr);
71
72         return (0);
73 }
```

Call function

- Understand what the code is doing, how much memory is allocated for the buffer?

Exercise 6 – Part 1

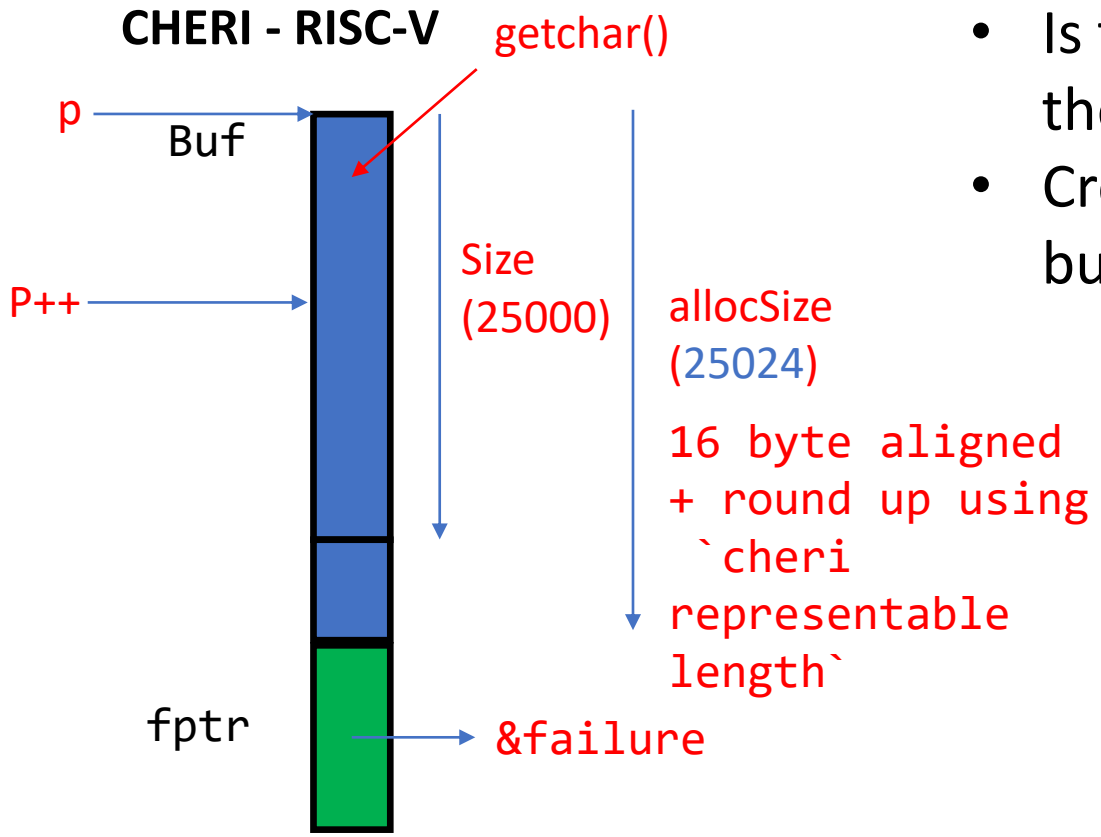


- Create an appropriate input to overflow the buffer

Use objdump to find address of success function
./llvm-objdump -dS ~/cheri/cheri-exercises/src/missions/buffer-overflow-control-flow/buffer-overflow-riscv
Create a file of characters to overflow the buffer, e.g
perl -e 'print "a"x25008 ."\xa8\x1f\x01\x00"' > inputtest

Input file to program in QEMU
./buffer-overflow-riscv < inputtest
Exploit successful!

Exercise 6 – Part 2



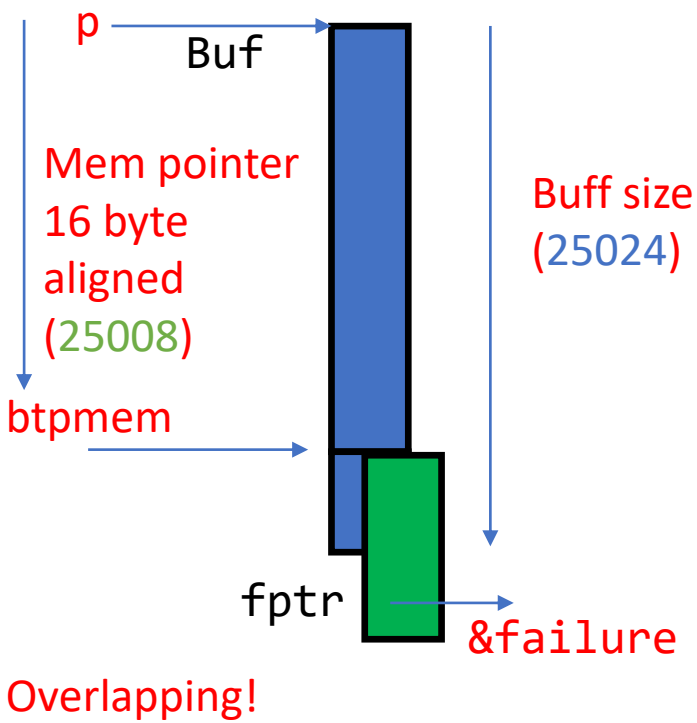
- Is the amount of memory allocated for the buffer, the same as for riscv or different?
- Create an appropriate input to try to overflow the buffer, if it cannot be exploited, what is the fault?

Use objdump to find address of success function
./llvm-objdump -dS ~/cheri/cheri-exercises/src/missions/buffer-overflow-control-flow/buffer-overflow-cheri
Create a file of characters to overflow the buffer, e.g
perl -e 'print "a"x25024 ."\x98\x27\x10\x00"' > inputtestcheri

Input file to program in QEMU (copy files to get core dump)
./buffer-overflow-cheri < inputtestcheri
In-address space security exception (core dumped)
GDB - Capability bounds fault at *p++ =(char)ch;

Exercise 6 – Part 3

CHERI RISC-V - weakened memory allocator



- How is the memory allocated different to previous? Can the buffer be overflowed?
- Create an appropriate input to try to overflow the buffer, if it cannot be exploited, explain why?

Buffer is allocated the representable size 25024
But memory allocator pointer increments to 25008
Creating memory bounds overlap, allowing **overflow!**
However, We **can't exploit** because changing the integer address value of the fptr capability **clears the tag bit**!

```
perl -e 'print "a"x25008 ."\x98\x27\x10\x00"' > inputcherinopad
Input file to program in QEMU (copy files to get core dump)
./buffer-overflow-cheri < inputcherinopad
In-address space security exception (core dumped)
GDB - Capability tag fault      at  (**fptr)()
```

More information

1. University of Cambridge CHERI:
- https://www.cl.cam.ac.uk/research/security/ctsrd/cheri/
- https://github.com/CTSRD-CHERI
2. Arm Morello:
- https://www.morello-project.org/
- https://git.morello-project.org/morello

Acknowledgements

Our project is funded by the Digital Security by Design (DSbD) Programme delivered by UKRI to support the DSbD ecosystem

https://www.dsbd.tech/

UNIVERSITY OF BIRMINGHAM

CARDIS 2022, November 7 – 9

End

UNIVERSITY OF BIRMINGHAM | CENTRE FOR CYBER SECURITY AND PRIVACY