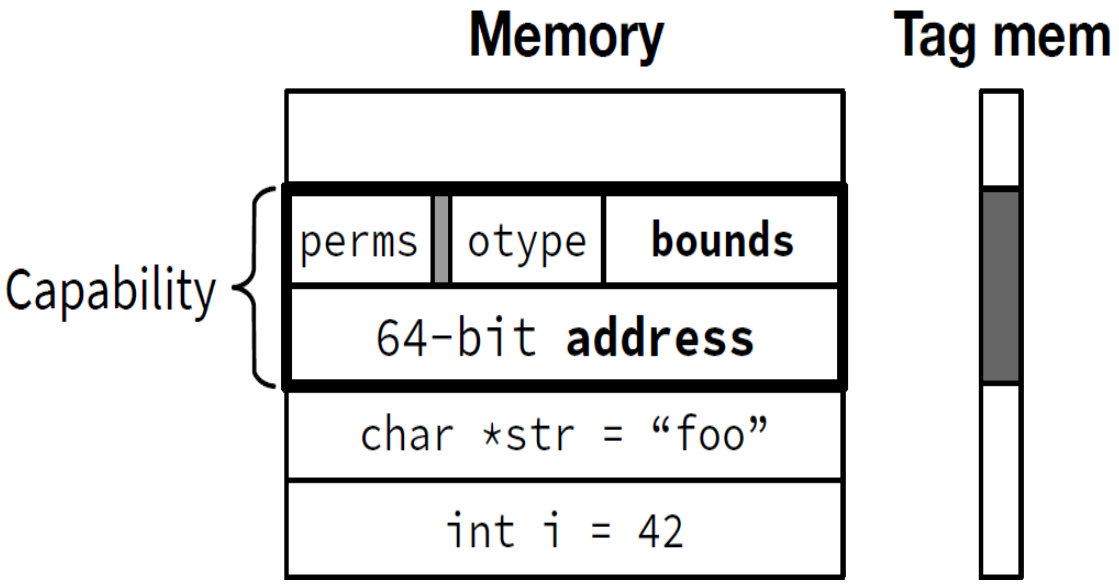
Cherifying Linux: A Practical View on Using CHERI

Kui Wang[1], Dmitry Kasatkin[1], Vincent Ahlrichs[2], Lukas Auer[2],
Konrad Hohentanner[2], Julian Horsch[2], Jan-Erik Ekberg[1]

[1]Huawei Technologies, Helsinki, Finland
[2]Fraunhofer AISEC, Garching near Munich, Germany

C Language, Memory Safety, and CHERI

- C is a low-level language, is absent of bound-checking when accessing memory
- Pointer exploitation lead memory safety issues
- Capability Hardware Enhanced RISC Instructions (CHERI) introduces hardware capability to enforce memory safety (spatial) on C program
- A pointer is represented as a 128 bits capability, which contains 64 bits address and metadata
- Bounds are packed together with address, bound-checking is enforced by architecture
- A capability can only be manipulated with CHERI instructions, added to the base instruction set, as an extension to ISA
- Overwrite the capability, *e.g.*, arithmetically manipulate its address clears the out-of-band tag bit, invalid the capability
- CHERI – constrain pointer, extend ISA, update compiler, OS and C Runtime change, C programs change. PAC, MTE comparison



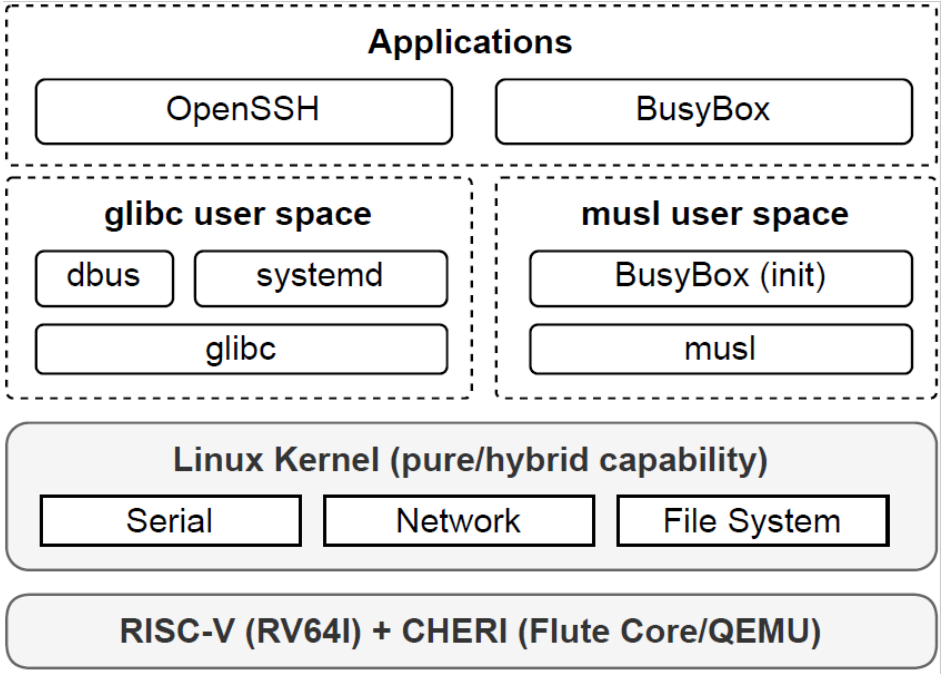
Practical perspective to use CHERI

- C programs need to be recompiled for CHERI ABI, requires compiler support, *e.g.*, LLVM
- CHERI ABI in turn needs CHERI extended ISA, requires hardware support, *e.g.*, MIPS, RISCV, ARM (Morello)
- Recompile a C program completely for CHERI, *i.e.*, using pure capability ABI, all pointers use 128 bits representation, including PC, SP
- Recompile a C program using the base ABI, add a new type to support 128 bits pointer representation, *e.g.*, int* p __capability
- **Recompile Linux kernel, C library, Busybox with RISC-V pure capability ABI to build a working system that starts to a shell, *i.e.*, Cherifying Linux[1]**
- Our contributions:
- **Summary identified issues and provide suggested patterns of changes**
- **Evaluate the memory safety properties and performance**

[1]https://github.com/cheri-linux

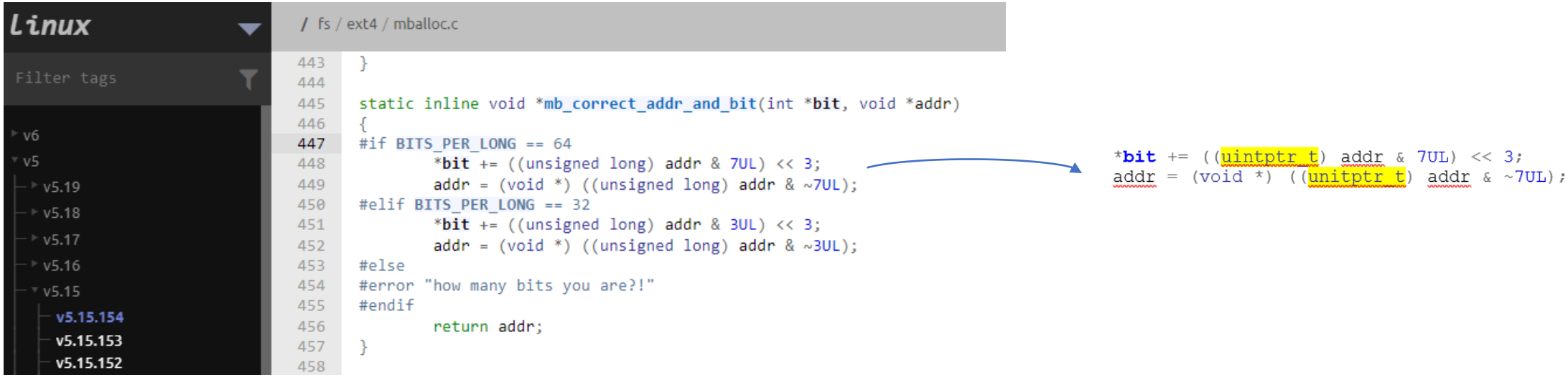
System Architecture

- RISC-V as hardware platform due to its mature CHERI support
- The RISC-V + CHERI hardware can be either emulated by QEMU or be FPGA-based
- A minimal viable software stack consisting of the Linux kernel, Musl C library and Busybox to realize a basic shell environment
- A slight complex stack to replace Musl C library with GNU C library, also added dbus and systemd
- Linux kernel can be compiled
  - either in CHERI hybrid mode, where the kernel supports applications with capability protection
  - or in CHERI pure-capability mode where also kernel memory accesses are protected.



Issue 1: Interchangeable use integer and pointer cause pointer missing metadata

- C program use pointer and integer interchangeable, rather common, not an issue for RISCV64
- For purecap CHERI-RISCV64, casting causes missing pointer provenance. Running the program causes an runtime exception when dereference the pointer
- The necessary change is to use *uintptr*_t, which can hold a capability, not drop its provenance



Issue 1: How does compiler handle Integer <-> Pointer casts?

C source #1

```
1  /* Type your code here, or load an example. */
2  void* cast(void* num) {
3      unsigned long unsafe = (unsigned long) num;
4      unsafe += 1 ;
5      return (void*) unsafe;
6  }
```

RISCV64 (without CHERI) (Editor #1)

RISCV64 (without CHERI) -O1

```
1  cast:                                   # @cast
2          addi    a0, a0, 1
3          ret
```

Purecap CHERI-RISCV64 (Editor #1)

Purecap CHERI-RISCV64 -O1

```
1  cast:                                   # @cast
2          cgetaddr        a0, ca0
3          addi    a0, a0, 1
4          cincoffset      ca0, cnull, a0
5          cret
```

```
1  /* Type your code here, or load an example. */
2  #include <stdint.h>
3
4  void* safecast(void* num) {
5      uintptr_t safe = (uintptr_t) num;
6      safe += 1;
7      return (void*) safe;
8  }
```

RISCV64 (without CHERI) (Editor #1)

RISCV64 (without CHERI) -O1

```
1  safecast:                               # @safecast
2          addi    a0, a0, 1
3          ret
```

Purecap CHERI-RISCV64 (Editor #1)

Purecap CHERI-RISCV64 -O1

```
1  safecast:                               # @safecast
2          cincoffset      ca0, ca0, 1
3          cret
```

```
<source>:5:12: warning: cast from provenance-free integer type to pointer type will give pointer that can not be dereferenced [-Wcheri-capability-misuse]
    return (void*) unsafe;
```

Issue 1: Create a capability from an integer?

C source #1

Save/Load + Add new... Vim

```
1  void* integertocap(int n) {
2      void* base = __builtin_cheri_global_data_get();
3      return __builtin_cheri_address_set(base, n);
4  }
```

Purecap CHERI-RISCV64 (Editor #1)

Purecap CHERI-RISCV64 -O1

Output... Filter... Libraries Overri

```
1  integertocap:
2          cspecialr       ca1, ddc
3          csetaddr        ca0, ca1, a0
4          cret
5
```

- Special CHERI register ddc (default data capability) used to give provenance to integer address
- For legacy C code that is difficult to establish provenance
- During early boot set ddc to cnull

```
464 + #if 0
465 +          /* FIXCHERI
466 +           * adjust permissions, boundaries
467 +           * burn ddc
468 +           */
469 +          cmove ct0, cnull
470 +          cspecialw ddc, ct0
471 + #endif
```

Issue 1: Walkaround by creating a capability

/ mm / vmalloc.c

```
2369  static inline void setup_vmalloc_vm_locked(struct vm_struct *vm,
2370          struct vmap_area *va, unsigned long flags, const void *caller)
2371  {
2372          vm->flags = flags;
2373          vm->addr = (void *)va->va_start;
2374          vm->size = va->va_end - va->va_start;
2375          vm->caller = caller;
2376          va->vm = vm;
2377  }
```

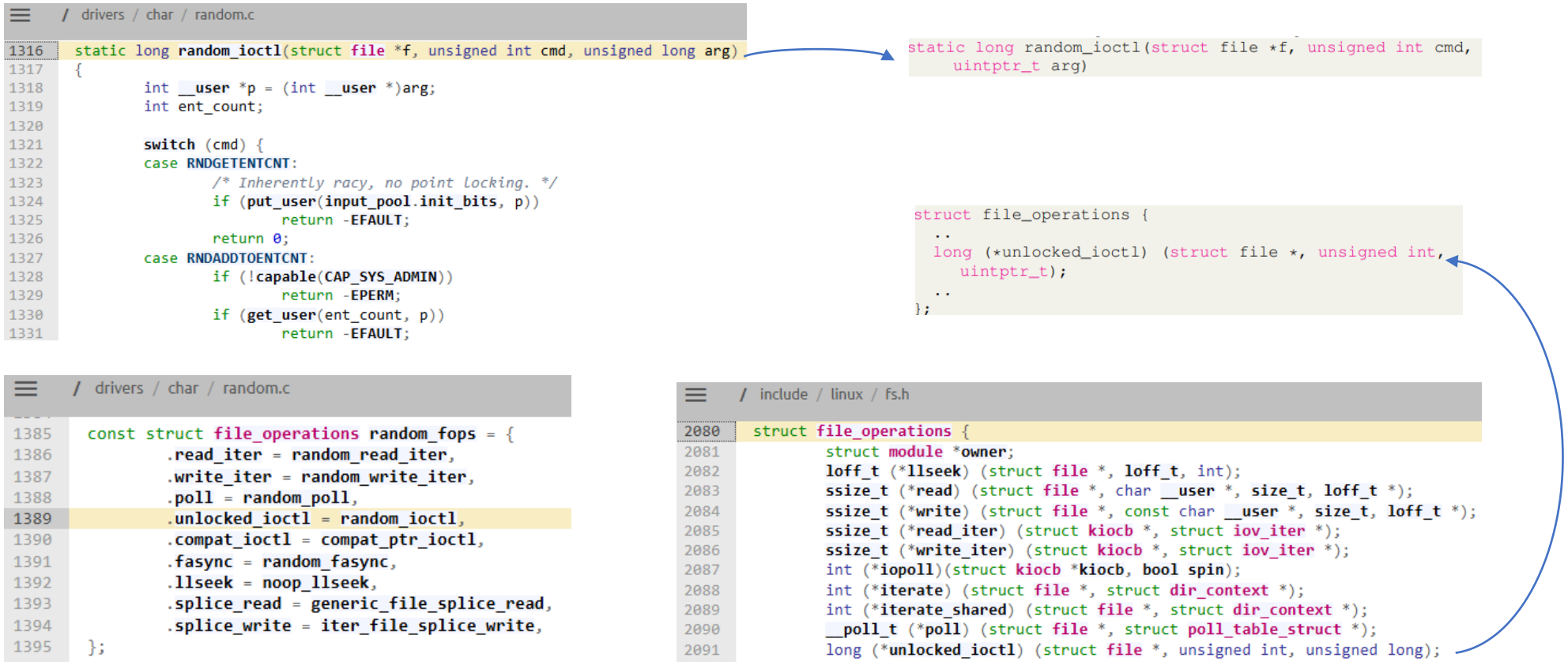
/ include / linux / vmalloc.h

```
73  struct vmap_area {
74          unsigned long va_start;
75          unsigned long va_end;
76
77          struct rb_node rb_node;         /* address sorted rbtree */
78          struct list_head list;          /* address sorted list */
79
```

- Numerous cases where capabilities need to be constructed using *ddc*, to accommodate legacy code
- We use compiler macros to walkaround these issues

```
vm->addr = (void *)cheri_long_data(va->va_start);
```

Issue 1: Propagate fix to multiple files



/ drivers / char / random.c

```
1316  static long random_ioctl(struct file *f, unsigned int cmd, unsigned long arg)
1317  {
1318          int __user *p = (int __user *)arg;
1319          int ent_count;
1320
1321          switch (cmd) {
1322          case RNDGETENTCNT:
1323                  /* Inherently racy, no point locking. */
1324                  if (put_user(input_pool.init_bits, p))
1325                          return -EFAULT;
1326                  return 0;
1327          case RNDADDTOENTCNT:
1328                  if (!capable(CAP_SYS_ADMIN))
1329                          return -EPERM;
1330                  if (get_user(ent_count, p))
1331                          return -EFAULT;
```

```
static long random_ioctl(struct file *f, unsigned int cmd,
    uintptr_t arg)
```

```
struct file_operations {
  ..
  long (*unlocked_ioctl) (struct file *, unsigned int,
    uintptr_t);
  ..
};
```

/ drivers / char / random.c

```
1385  const struct file_operations random_fops = {
1386          .read_iter = random_read_iter,
1387          .write_iter = random_write_iter,
1388          .poll = random_poll,
1389          .unlocked_ioctl = random_ioctl,
1390          .compat_ioctl = compat_ptr_ioctl,
1391          .fasync = random_fasync,
1392          .llseek = noop_llseek,
1393          .splice_read = generic_file_splice_read,
1394          .splice_write = iter_file_splice_write,
1395  };
```

/ include / linux / fs.h

```
2080  struct file_operations {
2081          struct module *owner;
2082          loff_t (*llseek) (struct file *, loff_t, int);
2083          ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
2084          ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
2085          ssize_t (*read_iter) (struct kiocb *, struct iov_iter *);
2086          ssize_t (*write_iter) (struct kiocb *, struct iov_iter *);
2087          int (*iopoll)(struct kiocb *kiocb, bool spin);
2088          int (*iterate) (struct file *, struct dir_context *);
2089          int (*iterate_shared) (struct file *, struct dir_context *);
2090          __poll_t (*poll) (struct file *, struct poll_table_struct *);
2091          long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
```

- *random_ioctl* uses *arg* as a pointer to read data from user or write data to user
- Function prototype is changed to use *uintptr_t*
- Due to the change on definition of *struct file_operations*, many other files are changed as well

Issue 2: Move a capability not as a whole clears its tag

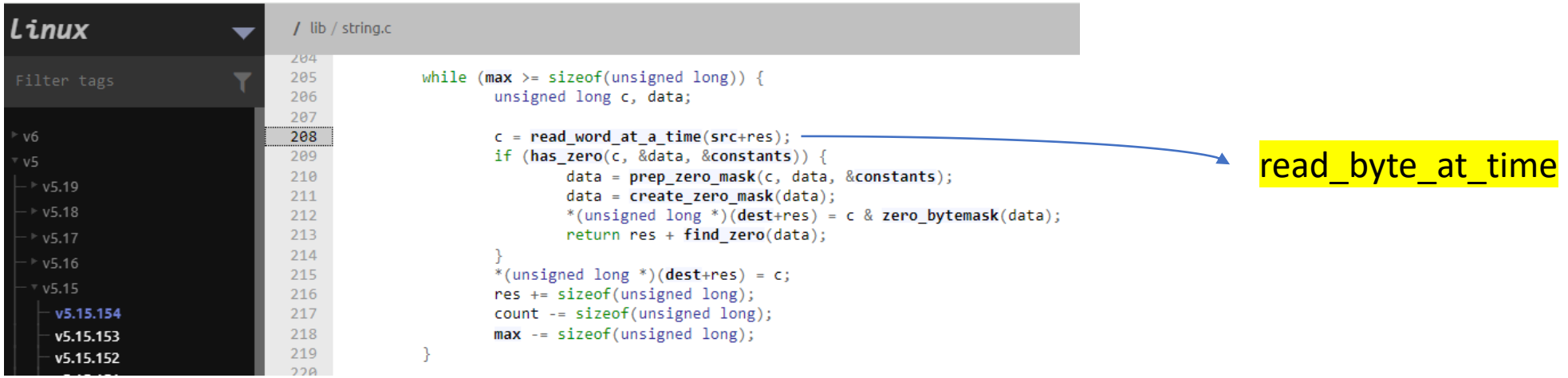
- If a function deals with moving data, alignment to capability size must be considered
- When a capability is moved not as a whole, its tag bit is cleared, rendering it invalid
- The fix is to move any heading and trailing data in smaller granularity[1], leaving the middle region as 16-byte aligned, and move data in 16-byte granularity[2]

```
1:
clbu t2, (ca1)
csb t2, (ca0)
cincoffset ca1, ca1, 1
cincoffset ca0, ca0, 1
bltu a1, t0, 1b
```

```
2:
clc ct2, (ca1)
csc ct2, (ca0)
cincoffset ca1, ca1, CHERICAP_SIZE
cincoffset ca0, ca0, CHERICAP_SIZE
bltu a1, t1, 2b
```

Issue 3: Functions intentionally overreading (for performance optimization) fail bound-checking

- Function can intentionally read or write beyond boundaries of a pointer, often for optimizing performance to reduce memory access
- String manipulation function checks the ending *'\0'* by reading a bigger chunk each time and scan the *'\0'* char, which fails bound-checking
- The fix is to disable the optimization and retreat to reading / writing one byte at a time



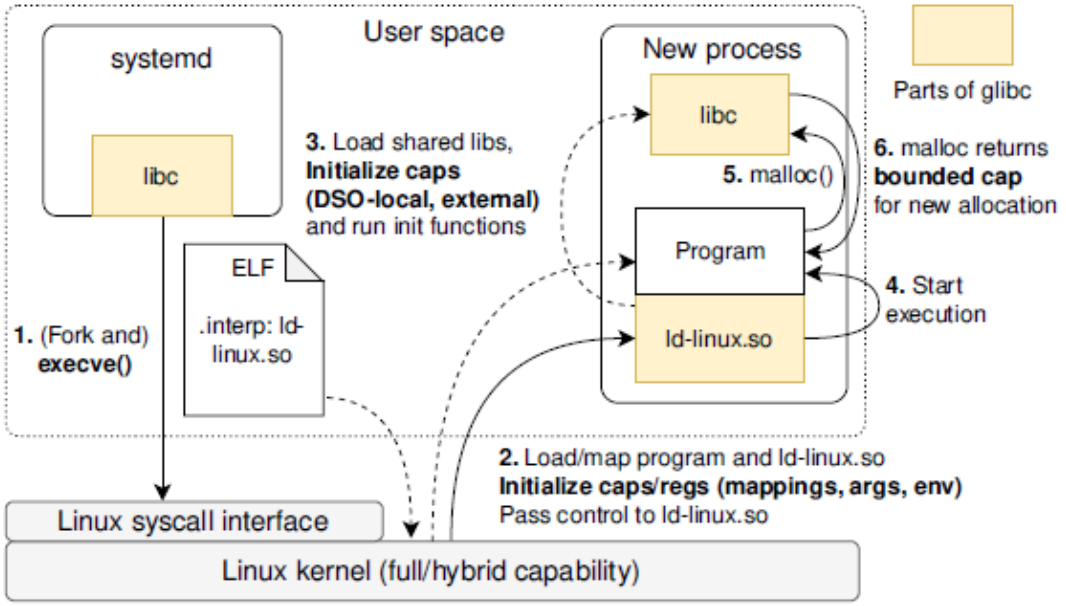
Cherification issue types

- Missing Pointer Provenance (MPP)
- Raw Copy (RCP)
- Intentional Overflow (IOF)
- Assembler Instructions (ASE)
  - *e.g.* in assembly file and inline assembly ld/st instruction change to clc/csc
- Heap Allocator (HAC)
  - Set bounds for dynamically allocated memory
- Global Data (GD)
  - Initialize correct bounds for data pointers and code pointers in capability table, replace GOT
- Pointer Size Assumption (PSZ)
  - Pointer size should not be hardcoded in source file

Project	MPP	RCP	IOF	ASE	HAC	GD	PSZ	LoC
Linux Kernel 5.15	✓	✓	✓	✓	✓	✓	✓	5942
MUSL libc 1.2.0	✓	-	-	✓	✓	✓	✓	2030
glibc 2.27	✓	✓	✓	✓	✓	✓	✓	2268
Busybox	-	-	-	-	-	-	-	21
OpenSSH	-	-	✓	-	-	-	-	6
OpenSSL	✓	-	-	-	-	-	-	24
systemd	-	-	✓	-	-	-	✓	52
dbus	-	-	-	-	✓	-	✓	29

Start a user program in CHERI Linux

- Linux kernel prepares arguments and environment variables as capabilities on stack for interpreter, *i.e.,* dynamic linker and pass control to it
- Capabilities are initialized, *e.g.*, function pointers for procedure calls
- Dynamic memory allocation such as malloc need to return bounded capability



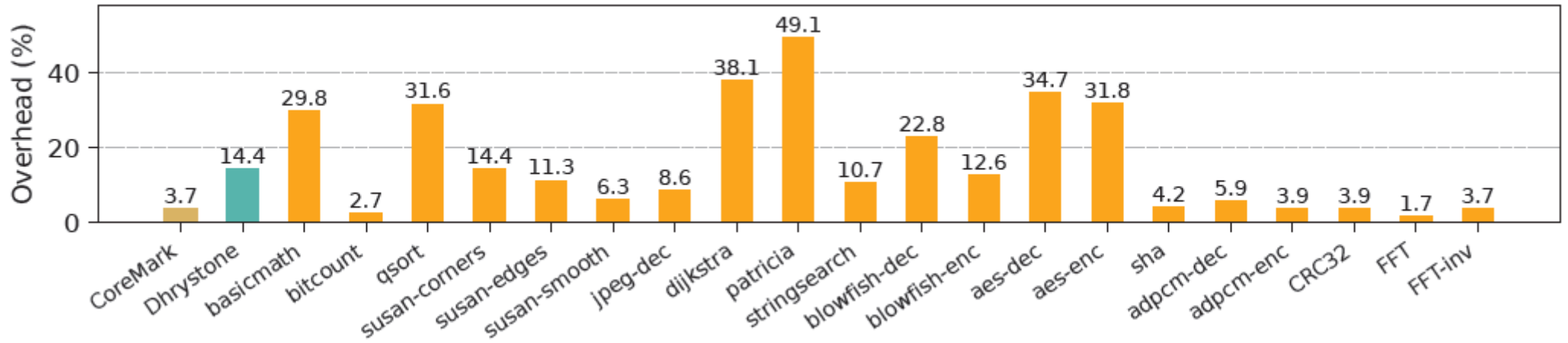
Memory safety evaluation

- Juliet Test Suite for security evaluation
- According to types of flaws, test cases are categorized to Common Weakness Enumerations (CWEs)
- Each test case exhibits a flaw, Normal exit means flaw is not detected
- CHERI can detect more spatial violation, *i.e.*, successfully exposes the flaws by triggering runtime CHERI exceptions, reducing the Normal exit counts, reporting CHERI violations instead of Segfaults
- CHERI do not improve protections against other weakness, *e.g.*, temporal violations

Category (CWEs)	Exit Status	Plain	CHERI
Spatial Violations (121, 122, 124, 126, 127)	Normal	111	6
	Segfaults	41	0
	CHERI Violations	0	146
Temporal Violations (401, 415, 416, 562, 590)	Normal	29	29
	Segfaults	1	9
	Aborts	19	11
	CHERI Violations	0	0
Others (...)	Normal	279	276
	Timeouts	37	37
	Explicit Error	1	1
	Allocation Error	3	3
	Segfaults	21	2
	Aborts	4	4
	CHERI Violations	0	22
Summary	Normal	419	311
	Segfaults	63	11
	Aborts	23	15

Performance evaluation

- CoreMark, Dhrystone, and MiBench for performance evaluation
- Evaluation were conducted on the Flute CPU, a 5-stage in-order RISC-V core, extended with CHERI, synthesized to run at 94MHz on a Xilinx Virtex UltraScale+ FPGA
- Compare to a non-CHERI system, CoreMark has a 3.7% overhead, Dhrystone 14.4%, MiBench 16.4%
- Overhead of individual MiBench varies from 1.7% to 49.1%
- Remove optimization in glibc to comply with CHERI potentially impacted some benchmark results
- Due to increased size of pointer, cache pressure increases, may negatively affect the performance



Conclusion

- C does not have built-in bound-checking, causing memory safety issue
- CHERI introduces hardware capabilities to enforce bound-checking on C programs
- Recompile Linux to CHERI purecap ABI, on CHERI extended RISC-V ISA
- Setup a cherified system[1] including Linux kernel, C library, busybox to realize a shell environment
- Categorized issues during cherifying Linux, analysis, and provided patterns of changes
- CHERI improves memory spatial safety
- The incurred performance overhead is about 15% (on our setup, not generalizable)

[1]https://github.com/cheri-linux