Libpatch

A dynamic binary patcher

Olivier Dion

Polytechnique Montréal
Laboratoire Dorsal

Summary

Summary

What's libpatch?

- C library

What's libpatch?

- C library
- Specializes in insertion of probes at runtime

What's libpatch?

- C library
- Specializes in insertion of probes at runtime
- Tries to maximize coverage of probes

What's libpatch?

- C library
- Specializes in insertion of probes at runtime
- Tries to maximize coverage of probes
- Tries to minimize overhead of probes

Usage example

```
1   #include <stdlib.h>
2   #include <stdio.h>
3   #include <libpatch/patch.h>
4
5   static int exit_value = EXIT_FAILURE;
6
7   static void probe(struct patch_probe_context *ctx)
8   {
9     printf("probe:\tx=%d\n",
10           (int)ctx->gregs[PATCH_X86_64_RDI]);
11
12    exit_value ^= EXIT_FAILURE;
13  }
14
15  void func(int x)
16  {
17    printf("func:\tx=%d\n", x);
18  }
```

Usage example (continuation)

```
15      static void install_probe(void)
16      {
17        patch_op op = {
18          .type           = PATCH_OP_INSTALL,
19          .addr.func_sym  = "func",
20          .probe          = probe,
21        };
22
23        patch_result *results;
24        size_t results_count;
25
26        assert(PATCH_OK == patch_init(NULL, 0));
27        assert(PATCH_OK == patch_queue(PATCH_FENTRY, &op));
28        assert(PATCH_OK == patch_commit(&results, &results_count));
29        assert(0 == results_count);
30        patch_drop_results(results);
31      }
```

Usage example (continuation)

```
32      int main(void)
33      {
34        int x = random();
35
36        printf("main:\tx=%d\n", x);
37
38        func(x);
39
40        /* All probes are removed here. */
41        patch_fini();
42
43        func(x);
44
45        return exit_value;
46      }
```

Usage example (continuation)

```
gcc -O0 -g -o demo demo.c -lpatch && ./demo
```

```
main:   x=1804289383
probe:  x=1804289383
func:   x=1804289383
func:   x=1804289383
```

Public API

```
/* Library management. */
patch_err patch_init(const patch_opt *options, size_t count);
patch_err patch_fini(void);
patch_err patch_configure(const patch_opt *option);

/* Patch manipulation. */
patch_err patch_queue(uint64_t flags, patch_op *op);
patch_err patch_cancel(uint64_t cookie);
patch_err patch_commit(patch_result **results, size_t *count);

/* Memory cleanup. */
patch_err patch_drop_results(patch_result *results);

/* Error handling. */
const char *patch_error_string(void);
```

Summary

Comparison

Features	Uftrace	Libpatch
Runtime patching	Partial	Yes
Relative instructions	Partial	Yes
Handle CET	Partial	Partial
Handle W^X protection	No	Yes
Handle red zones	No	Yes
Handle indirect jumps	No	No
ARM support	Yes	No
Sub-five bytes patches	Yes	No
Per probe trampoline	No	No

What's missing?

- Handle CET's shadow stack (require hardware for testing)

What's missing?

- Handle CET's shadow stack (require hardware for testing)
- Per probe trampoline

What's missing?

- Handle CET's shadow stack (require hardware for testing)
- Per probe trampoline
- Add algorithms for better coverage

What's missing?

- Handle CET's shadow stack (require hardware for testing)
- Per probe trampoline
- Add algorithms for better coverage
- Add algorithms for handling indirect jumps

What's missing?

- Handle CET's shadow stack (require hardware for testing)
- Per probe trampoline
- Add algorithms for better coverage
- Add algorithms for handling indirect jumps
- Add support for ARM

Summary

Runtime patching

Install

- Lock patch region with trap

Runtime patching

Install

- Lock patch region with trap
- Synchronize cores if overlap cache lines

Runtime patching

Install

- Lock patch region with trap
- Synchronize cores if overlap cache lines
- Replace rest of region with jump offset

Runtime patching

Install

- Lock patch region with trap
- Synchronize cores if overlap cache lines
- Replace rest of region with jump offset
- Synchronize cores again if overlap cache lines

Runtime patching

Install

- Lock patch region with trap
- Synchronize cores if overlap cache lines
- Replace rest of region with jump offset
- Synchronize cores again if overlap cache lines
- Replace lock with jump

Runtime patching

Install

- Lock patch region with trap
- Synchronize cores if overlap cache lines
- Replace rest of region with jump offset
- Synchronize cores again if overlap cache lines
- Replace lock with jump

Uninstall

Runtime patching

Install

- Lock patch region with trap
- Synchronize cores if overlap cache lines
- Replace rest of region with jump offset
- Synchronize cores again if overlap cache lines
- Replace lock with jump

Uninstall

- Lock patch region with trap

Runtime patching

Install

- Lock patch region with trap
- Synchronize cores if overlap cache lines
- Replace rest of region with jump offset
- Synchronize cores again if overlap cache lines
- Replace lock with jump

Uninstall

- Lock patch region with trap
- Synchronize cores if overlap cache lines

Runtime patching

Install

- Lock patch region with trap
- Synchronize cores if overlap cache lines
- Replace rest of region with jump offset
- Synchronize cores again if overlap cache lines
- Replace lock with jump

Uninstall

- Lock patch region with trap
- Synchronize cores if overlap cache lines
- Restore rest of region with original bytes

Runtime patching

Install

- Lock patch region with trap
- Synchronize cores if overlap cache lines
- Replace rest of region with jump offset
- Synchronize cores again if overlap cache lines
- Replace lock with jump

Uninstall

- Lock patch region with trap
- Synchronize cores if overlap cache lines
- Restore rest of region with original bytes
- Synchronize cores again if overlap cache lines

Runtime patching

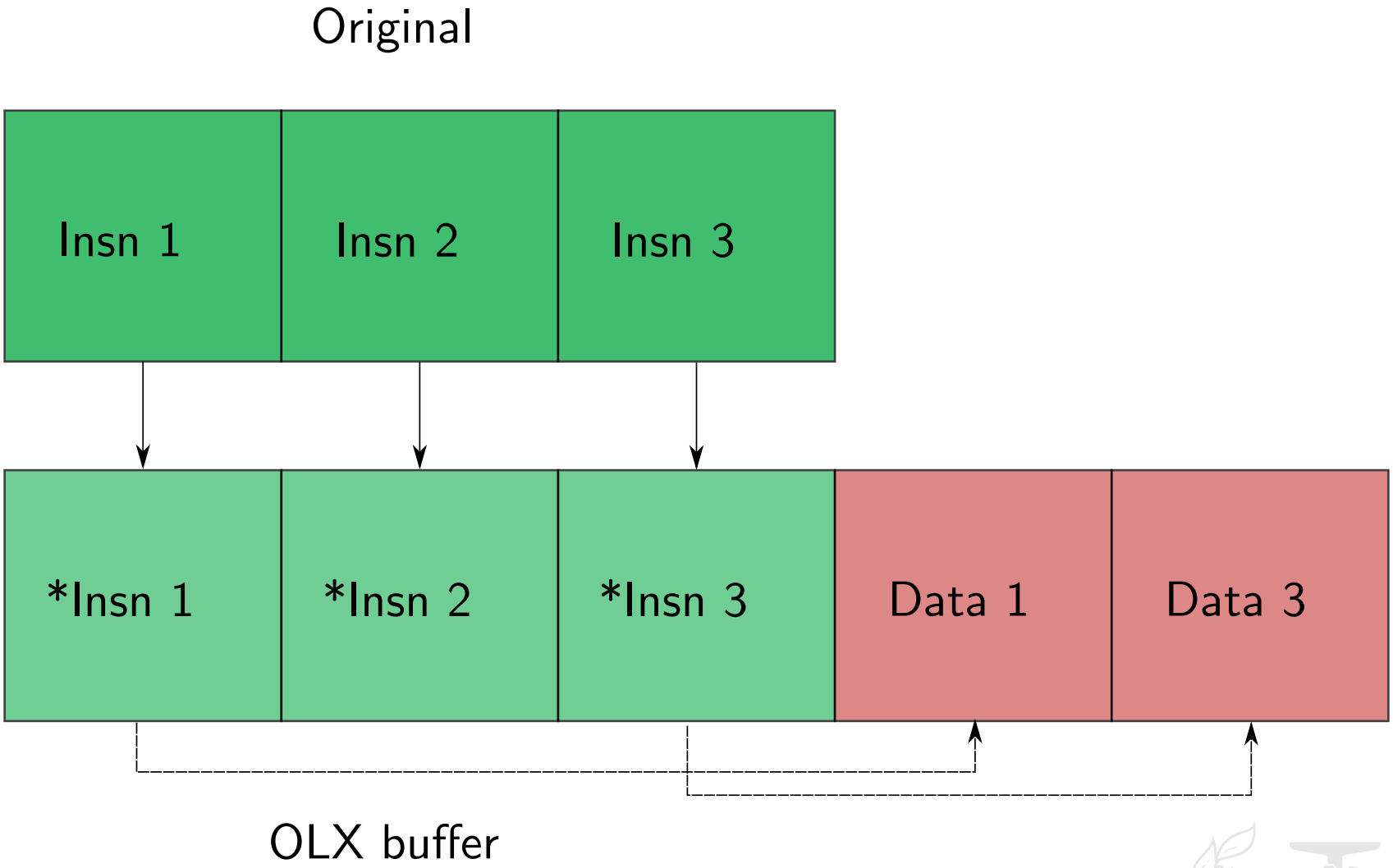
Install

- Lock patch region with trap
- Synchronize cores if overlap cache lines
- Replace rest of region with jump offset
- Synchronize cores again if overlap cache lines
- Replace lock with jump

Uninstall

- Lock patch region with trap
- Synchronize cores if overlap cache lines
- Restore rest of region with original bytes
- Synchronize cores again if overlap cache lines
- Restore original first byte

Relative instructions



* Emulated instruction

Relative instructions (continuation)

Generic case

```
push %R
movabs $ORIGINAL-RIP, %R
;; Do instruction with %R
pop %R

;; Example
mov %rax, DISP(%rip)
;; becomes
push %rbx
movabs $ORIGINAL-RIP, %rbx
mov %rax, DISP(%rbx)
pop %rbx
```

Relative instructions (continuation)

Generic case (continuation)

```
;; exception with
push DISP(%rip)
;; becomes
push %rax
push %rax
;; X = ORIGINAL-RIP + DISP
movabs $X, %rax
mov (%rax), %rax
mov %rax, 0x8(%rsp)
pop %rax
```

Relative instructions (continuation)

Call

```
call $REL
;; becomes
push %rax
push %rax
;; X = ORIGINAL-RIP + 5
movabs $X, %rax
mov %rax, 0x8(%rsp)
pop %rax
jump *0x0(%rip)
```

Relative instructions (continuation)

Conditional jump

```
je $REL
;; becomes
je 0x8
;; ALTERNATIVE = ORIGINAL-RIP + 6
;; TARGET      = ORIGINAL-RIP + REL
jmp *ALTERNATIVE(%rip)
jmp *TARGET(%rip)
```

Relative instructions (continuation)

Other cases

- `loop`
- Implicit registers?

W^X protection

Write XOR Execute

W^X protection

Write XOR Execute

- Security measure

W^X protection

Write XOR Execute

- Security measure
- Virtual pages can not be writable and executable at the same time

W^X protection

Write XOR Execute

- Security measure
- Virtual pages can not be writable and executable at the same time
- Problematic for JIT compiler but also for dynamic patching

W^X protection

Write XOR Execute

- Security measure
- Virtual pages can not be writable and executable at the same time
- Problematic for JIT compiler but also for dynamic patching
- Can be solved with `mprotect(2)`

W^X protection

Write XOR Execute

- Security measure
- Virtual pages can not be writable and executable at the same time
- Problematic for JIT compiler but also for dynamic patching
- Can be solved with `mprotect(2)`
- Can be solved with `memfd_create(2)`

W^X protection (continuation)

mprotect() method

- Slow
- Page granularity
- Possible huge jitter

W^X protection (continuation)

mprotect() method

- Slow
- Page granularity
- Possible huge jitter

memfd_create() method

- Fast
- No granularity
- No jitter
- Duplication of virtual pages
- Two cases to handle
- Little memory overhead

W^X protection (continuation)

Two virtual pages to a single physical anonymous page

```
1   int fd;
2   void *wr_only, *ex_only;
3   int wr_flags, ex_flags;
4
5   wr_flags = PROT_WRITE;
6   ex_flags = PROT_READ | PROT_EXEC;
7
8   fd = memfd_create("<libpatch>:anonymous", 0);
9
10  ftruncate(fd, PAGE_SIZE);
11
12  wr_only = mmap(NULL, PAGE_SIZE, wr_flags, MAP_SHARED, fd, 0);
13  ex_only = mmap(NULL, PAGE_SIZE, ex_flags, MAP_PRIVATE, fd, 0);
14
15  close(fd);
```

W^X protection (continuation)

Replacing an already mapped region (e.g. dlopen(3), ld-linux.so)

```
1  static void *hijack_map(void *begin, size_t size)
2  {
3    int fd = memfd_create("<libpatch>:file", 0);
4
5    write(fd, begin, size);
6
7    void *wr_only = mmap(NULL, size,
8                         PROT_WRITE, MAP_SHARED,
9                         fd, 0);
10   /* Swap physical pages. */
11   mmap(begin, size,
12        PROT_READ | PROT_EXEC, MAP_FIXED | MAP_PRIVATE,
13        fd, 0); close(fd);
14
15   return wr_only;
16 }
```

Red zone

- Part of x86-64 ABI

Red zone

- Part of x86-64 ABI
- First 128 bytes before `rsp` can be used by leaf functions

Red zone

- Part of x86-64 ABI
- First 128 bytes before `rsp` can be used by leaf functions
- Painful headache for dynamic instrumentation

Red zone

- Part of x86-64 ABI
- First 128 bytes before `rsp` can be used by leaf functions
- Painful headache for dynamic instrumentation
- Two methods to skip it

Red zone (continuation)

push method

```
push -0x8(%rsp)
push -0x8(%rsp)
push -0x8(%rsp)
push -0x8(%rsp)
push -0x8(%rsp)
push -0x8(%rsp)
push -0x8(%rsp)
push -0x8(%rsp)
push -0x8(%rsp)
push -0x8(%rsp)
push -0x8(%rsp)
push -0x8(%rsp)
push -0x8(%rsp)
push -0x8(%rsp)
pushf
sub $0x98, %rsp
```

Red zone (continuation)

exchange method

```
;; Prologue
push -0x8(%rsp)
pop -0x88(%rsp)
pushf
sub $0x118, %rsp
xchg %rax, 0x98(%rsp)
xchg %rax, 0x118(%rsp)
xchg %rax, 0x98(%rsp)
;; Epilogue
xchg %rax, 0x98(%rsp)
xchg %rax, 0x118(%rsp)
add $0x118, %rsp
popf
movq %rax, -0x8(%rsp)
movq -0x88(%rsp), %rax
```

Red zone (continuation)

Samples overhead of methods using perf(1) on a AMD Ryzen 9 5950X (10^8 loops)

Threads	Push overhead (%)	Exchange overhead (%)
1	56.15	43.02
2	56.04	43.02
4	56.11	43.02
8	56.20	42.92
16	56.22	42.89
32	67.09	29.05

Per probe trampoline

Current solution

- Pool of generic trampolines shared by all probes

Per probe trampoline

Current solution

- Pool of generic trampolines shared by all probes
- Simple but involves a search in an hash-map

Per probe trampoline

Current solution

- Pool of generic trampolines shared by all probes
- Simple but involves a search in an hash-map
- Minimal memory overhead

Next solution

- Individual trampoline per probe

Per probe trampoline

Current solution

- Pool of generic trampolines shared by all probes
- Simple but involves a search in an hash-map
- Minimal memory overhead

Next solution

- Individual trampoline per probe
- Much faster

Per probe trampoline

Current solution

- Pool of generic trampolines shared by all probes
- Simple but involves a search in an hash-map
- Minimal memory overhead

Next solution

- Individual trampoline per probe
- Much faster
- Requires a special memory allocator

Per probe trampoline

Current solution

- Pool of generic trampolines shared by all probes
- Simple but involves a search in an hash-map
- Minimal memory overhead

Next solution

- Individual trampoline per probe
- Much faster
- Requires a special memory allocator
- Trampoline placement is much more difficult

Per probe trampoline

Current solution

- Pool of generic trampolines shared by all probes
- Simple but involves a search in an hash-map
- Minimal memory overhead

Next solution

- Individual trampoline per probe
- Much faster
- Requires a special memory allocator
- Trampoline placement is much more difficult
- Work in progress

Per probe trampoline (continuation)

Layout of a shared generic trampoline

```
;; Where A is the address of the generic handler
jmp *0x0(%rip)
A
```

Layout of a specific trampoline

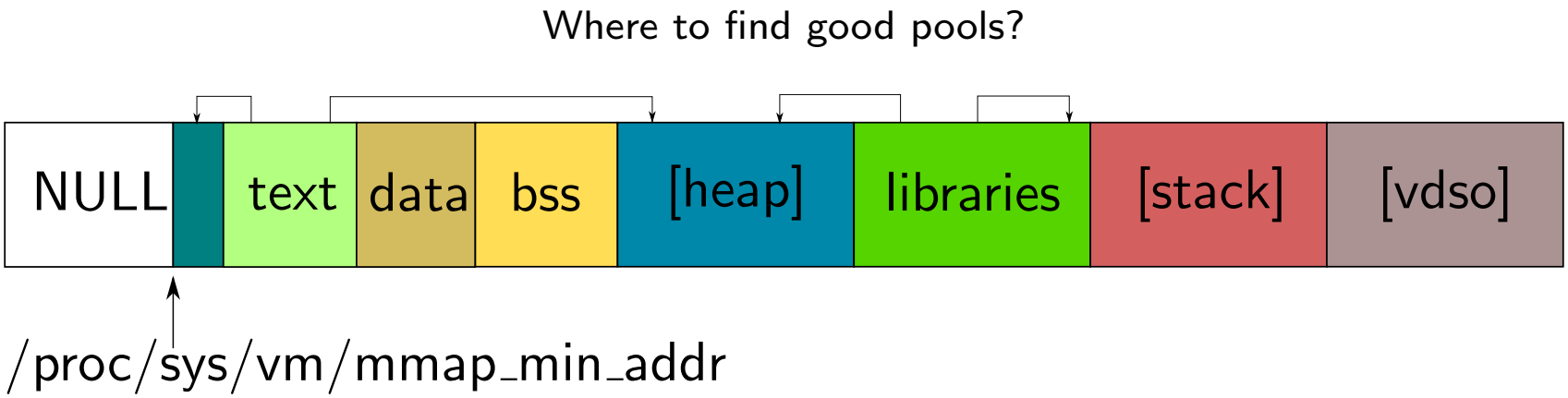
```
;; Where:
;; T is the tag of the trampoline.
;; O is the address of the OLX buffer.
;; S is the size of the OLX buffer.
;; C is the callback (instrumentation) to call.
;; A is the address of the generic handler.
push %rax
lea 0x6(%rip), %rax
jmp *0x20(%rip)
T
O
S
C
A
```

Per probe trampoline (continuation)

Layout of a specific trampoline of a leaf function

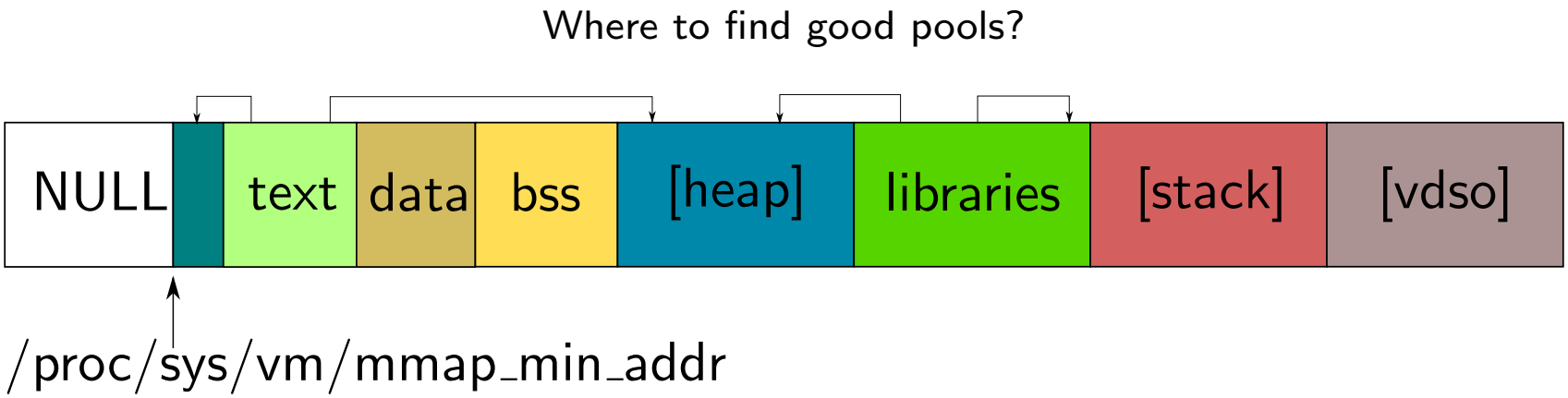
```
;; Where:
;; T is the tag of the trampoline.
;; O is the address of the OLX buffer.
;; S is the size of the OLX buffer.
;; C is the callback (instrumentation) to call.
;; A is the address of the generic handler for leaf function.
push -0x8(%rsp)
pop -0x88(%rsp)
push %rax
lea 0x6(%rip), %rax
jmp *0x20(%rip)
T
O
S
C
A
```

Per probe trampoline (continuation)



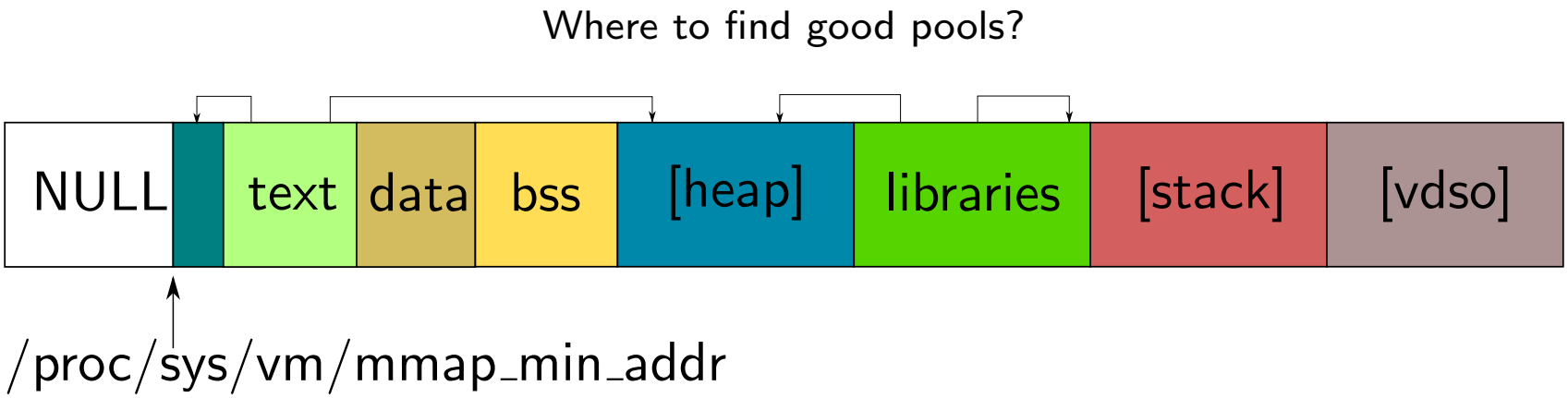
- Between NULL and first allocated page

Per probe trampoline (continuation)



- Between NULL and first allocated page
- Heap

Per probe trampoline (continuation)



- Between NULL and first allocated page
- Heap
- With libraries

Summary

Conclusion

- Libpatch can already

Conclusion

- Libpatch can already
  - insert probes on any instruction of at least 5 bytes

Conclusion

- Libpatch can already
  - insert probes on any instruction of at least 5 bytes
  - insert probes at runtime with very small overhead

Conclusion

- Libpatch can already
  - insert probes on any instruction of at least 5 bytes
  - insert probes at runtime with very small overhead
  - be used on system with W$\hat{\text{X}}$ protection

Conclusion

- Libpatch can already
  - insert probes on any instruction of at least 5 bytes
  - insert probes at runtime with very small overhead
  - be used on system with W$\hat{\text{X}}$ protection
- In a near future, libpatch will

Conclusion

- Libpatch can already
  - insert probes on any instruction of at least 5 bytes
  - insert probes at runtime with very small overhead
  - be used on system with W$\hat{\text{X}}$ protection
- In a near future, libpatch will
  - have less runtime overhead for probe execution

Conclusion

- Libpatch can already
  - insert probes on any instruction of at least 5 bytes
  - insert probes at runtime with very small overhead
  - be used on system with W$\hat{\text{X}}$ protection
- In a near future, libpatch will
  - have less runtime overhead for probe execution
  - be able to insert probes on more instructions

Conclusion

- Libpatch can already
  - insert probes on any instruction of at least 5 bytes
  - insert probes at runtime with very small overhead
  - be used on system with W$\hat{\text{X}}$ protection
- In a near future, libpatch will
  - have less runtime overhead for probe execution
  - be able to insert probes on more instructions
  - handle indirect jumps

Conclusion

- Libpatch can already
  - insert probes on any instruction of at least 5 bytes
  - insert probes at runtime with very small overhead
  - be used on system with W$\hat{\text{X}}$ protection
- In a near future, libpatch will
  - have less runtime overhead for probe execution
  - be able to insert probes on more instructions
  - handle indirect jumps
  - support for ARM

Questions

Questions?