`pwnlib.rop.srop` — Sigreturn Oriented Programming

Sigreturn ROP (SROP)

Sigreturn is a syscall used to restore the entire register context from memory pointed at by ESP.

We can leverage this during ROP to gain control of registers for which there are not convenient gadgets. The main caveat is that *all* registers are set, including ESP and EIP (or their equivalents). This means that in order to continue after using a sigreturn frame, the stack pointer must be set accordingly.

i386 Example:

Let's just print a message out using SROP.

```
>>> message = "Hello, World\\n"
```

First, we'll create our example binary. It just reads some data onto the stack, and invokes the `sigreturn` syscall. We also make an `int 0x80` gadget available, followed immediately by `exit(0)`.

```
>>> context.clear(arch='i386')
>>> assembly =  'setup: sub esp, 1024\n'
>>> assembly += 'read:'      +
shellcraft.read(constants.STDIN_FILENO, 'esp', 1024)
>>> assembly += 'sigreturn:' + shellcraft.sigreturn()
>>> assembly += 'int3:'      + shellcraft.trap()
>>> assembly += 'syscall: '  + shellcraft.syscall()
>>> assembly += 'exit: '     + 'xor ebx, ebx; mov eax, 1; int 0x80;'
>>> assembly += 'message: '  + ('.asciz "%s"' % message)
>>> binary = ELF.from_assembly(assembly)
```

Let's construct our frame to have it invoke a `write` syscall, and dump the message to stdout.

```
>>> frame = SigreturnFrame(kernel='amd64')
>>> frame.eax = constants.SYS_write
>>> frame.ebx = constants.STDOUT_FILENO
>>> frame.ecx = binary.symbols['message']
>>> frame.edx = len(message)
>>> frame.esp = 0xdeadbeef
>>> frame.eip = binary.symbols['syscall']
```

Let's start the process, send the data, and check the message.

```
>>> p = process(binary.path)
>>> p.send(bytes(frame))
>>> p.recvline()
b'Hello, World\n'
>>> p.poll(block=True)
0
```

amd64 Example:

```
>>> context.clear()
>>> context.arch = "amd64"
>>> assembly =  'setup: sub rsp, 1024\n'
>>> assembly += 'read:'      + shellcraft.read(constants.STDIN_FILENO,
'rsp', 1024)
>>> assembly += 'sigreturn:' + shellcraft.sigreturn()
>>> assembly += 'int3:'      + shellcraft.trap()
>>> assembly += 'syscall: '  + shellcraft.syscall()
>>> assembly += 'exit: '     + 'xor rdi, rdi; mov rax, 60; syscall;'
>>> assembly += 'message: '  + ('.asciz "%s"' % message)
>>> binary = ELF.from_assembly(assembly)
>>> frame = SigreturnFrame()
>>> frame.rax = constants.SYS_write
>>> frame.rdi = constants.STDOUT_FILENO
>>> frame.rsi = binary.symbols['message']
>>> frame.rdx = len(message)
>>> frame.rsp = 0xdeadbeef
>>> frame.rip = binary.symbols['syscall']
>>> p = process(binary.path)
>>> p.send(bytes(frame))
>>> p.recvline()
b'Hello, World\n'
>>> p.poll(block=True)
0
```

arm Example:

```
>>> context.clear()
>>> context.arch = "arm"
>>> assembly =  'setup: sub sp, sp, 1024\n'
>>> assembly += 'read:'      + shellcraft.read(constants.STDIN_FILENO,
'sp', 1024)
>>> assembly += 'sigreturn:' + shellcraft.sigreturn()
>>> assembly += 'int3:'      + shellcraft.trap()
>>> assembly += 'syscall: '  + shellcraft.syscall()
>>> assembly += 'exit: '     + 'eor r0, r0; mov r7, 0x1; swi #0;'
>>> assembly += 'message: '  + ('.asciz "%s"' % message)
>>> binary = ELF.from_assembly(assembly)
>>> frame = SigreturnFrame()
>>> frame.r7 = constants.SYS_write
>>> frame.r0 = constants.STDOUT_FILENO
>>> frame.r1 = binary.symbols['message']
>>> frame.r2 = len(message)
>>> frame.sp = 0xdead0000
>>> frame.pc = binary.symbols['syscall']
>>> p = process(binary.path)
>>> p.send(bytes(frame))
>>> p.recvline()
b'Hello, World\n'
>>> p.wait_for_close()
>>> p.poll(block=True)
0
```

Mips Example:

```
>>> context.clear()
>>> context.arch = "mips"
>>> context.endian = "big"
>>> assembly =  'setup: sub $sp, $sp, 1024\n'
>>> assembly += 'read:'      + shellcraft.read(constants.STDIN_FILENO,
'$sp', 1024)
>>> assembly += 'sigreturn:' + shellcraft.sigreturn()
>>> assembly += 'syscall: '  + shellcraft.syscall()
>>> assembly += 'exit: '     + shellcraft.exit(0)
>>> assembly += 'message: '  + ('.asciz "%s"' % message)
>>> binary = ELF.from_assembly(assembly)
>>> frame = SigreturnFrame()
>>> frame.v0 = constants.SYS_write
>>> frame.a0 = constants.STDOUT_FILENO
>>> frame.a1 = binary.symbols['message']
>>> frame.a2 = len(message)
>>> frame.sp = 0xdead0000
>>> frame.pc = binary.symbols['syscall']
>>> p = process(binary.path)
>>> p.send(bytes(frame))
>>> p.recvline()
b'Hello, World\n'
>>> p.poll(block=True)
0
```

Mipsel Example:

```
>>> context.clear()
>>> context.arch = "mips"
>>> context.endian = "little"
>>> assembly =  'setup: sub $sp, $sp, 1024\n'
>>> assembly += 'read:'      + shellcraft.read(constants.STDIN_FILENO,
'$sp', 1024)
>>> assembly += 'sigreturn:' + shellcraft.sigreturn()
>>> assembly += 'syscall: '  + shellcraft.syscall()
>>> assembly += 'exit: '     + shellcraft.exit(0)
>>> assembly += 'message: '  + ('.asciz "%s"' % message)
>>> binary = ELF.from_assembly(assembly)
>>> frame = SigreturnFrame()
>>> frame.v0 = constants.SYS_write
>>> frame.a0 = constants.STDOUT_FILENO
>>> frame.a1 = binary.symbols['message']
>>> frame.a2 = len(message)
>>> frame.sp = 0xdead0000
>>> frame.pc = binary.symbols['syscall']
>>> p = process(binary.path)
>>> p.send(bytes(frame))
>>> p.recvline()
b'Hello, World\n'
>>> p.poll(block=True)
0
```

class **pwnlib.rop.srop.SigreturnFrame**(***kw*) [source]

Crafts a sigreturn frame with values that are loaded up into registers.

Parameters: **arch** (*str*) – The architecture. Currently `i386` and `amd64` are supported.

Examples

Crafting a SigreturnFrame that calls mprotect on amd64

```
>>> context.clear(arch='amd64')
>>> s = SigreturnFrame()
>>> unpack_many(bytes(s))
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
51, 0, 0, 0, 0, 0, 0, 0]
>>> assert len(s) == 248
>>> s.rax = 0xa
>>> s.rdi = 0x00601000
>>> s.rsi = 0x1000
>>> s.rdx = 0x7
>>> assert len(bytes(s)) == 248
>>> unpack_many(bytes(s))
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6295552, 4096, 0, 0, 7, 10,
0, 0, 0, 0, 51, 0, 0, 0, 0, 0, 0, 0]
```

Crafting a SigreturnFrame that calls mprotect on i386

```
>>> context.clear(arch='i386')
>>> s = SigreturnFrame(kernel='i386')
>>> unpack_many(bytes(s))
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 115, 0, 0, 123, 0]
>>> assert len(s) == 80
>>> s.eax = 125
>>> s.ebx = 0x00601000
>>> s.ecx = 0x1000
>>> s.edx = 0x7
>>> assert len(bytes(s)) == 80
>>> unpack_many(bytes(s))
[0, 0, 0, 0, 0, 0, 0, 0, 6295552, 7, 4096, 125, 0, 0, 0, 115, 0, 0,
123, 0]
```

Crafting a SigreturnFrame that calls mprotect on ARM

```
>>> s = SigreturnFrame(arch='arm')
>>> unpack_many(bytes(s))
[0, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1073741840, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1447448577, 288]
>>> s.r0 = 125
>>> s.r1 = 0x00601000
>>> s.r2 = 0x1000
>>> s.r3 = 0x7
>>> assert len(bytes(s)) == 240
>>> unpack_many(bytes(s))
[0, 0, 0, 0, 0, 6, 0, 0, 125, 6295552, 4096, 7, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1073741840, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1447448577,
288]
```

Crafting a SigreturnFrame that calls mprotect on MIPS

```
>>> context.clear()
>>> context.endian = "big"
>>> s = SigreturnFrame(arch='mips')
>>> unpack_many(bytes(s))
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0]
>>> s.v0 = 0x101d
>>> s.a0 = 0x00601000
>>> s.a1 = 0x1000
>>> s.a2 = 0x7
>>> assert len(bytes(s)) == 296
>>> unpack_many(bytes(s))
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4125, 0, 0, 0, 6295552,
0, 4096, 0, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Crafting a SigreturnFrame that calls mprotect on MIPSel

```
>>> context.clear()
>>> context.endian = "little"
>>> s = SigreturnFrame(arch='mips')
>>> unpack_many(bytes(s))
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0]
>>> s.v0 = 0x101d
>>> s.a0 = 0x00601000
>>> s.a1 = 0x1000
>>> s.a2 = 0x7
>>> assert len(bytes(s)) == 292
>>> unpack_many(bytes(s))
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4125, 0, 0, 0, 6295552, 0,
4096, 0, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0]
```

Crafting a SigreturnFrame that calls mprotect on Aarch64

```
>>> context.clear()
>>> context.endian = "little"
>>> s = SigreturnFrame(arch='aarch64')
>>> unpack_many(bytes(s))
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1179680769, 528]
>>> s.x8 = 0xe2
>>> s.x0 = 0x4000
>>> s.x1 = 0x1000
>>> s.x2 = 0x7
>>> assert len(bytes(s)) == 600
>>> unpack_many(bytes(s))
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 16384, 0, 4096, 0, 7, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 226, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1179680769, 528]
```

`__init__`(***kw*) [source]

x.__init__(...) initializes x; see help(type(x)) for signature

`__len__`() <==> *len(x)* [source]

`__setattr__`(*attr*, *value*) [source]

x.__setattr__('name', value) <==> x.name = value

`__setitem__`(*item*, *value*) [source]

x.__setitem__(i, y) <==> x[i]=y

`__str__`() <==> *str(x)* [source]

`set_regvalue`(*reg*, *val*) [source]

Sets a specific `reg` to a `val`

`__weakref__` [source]

list of weak references to the object (if defined)