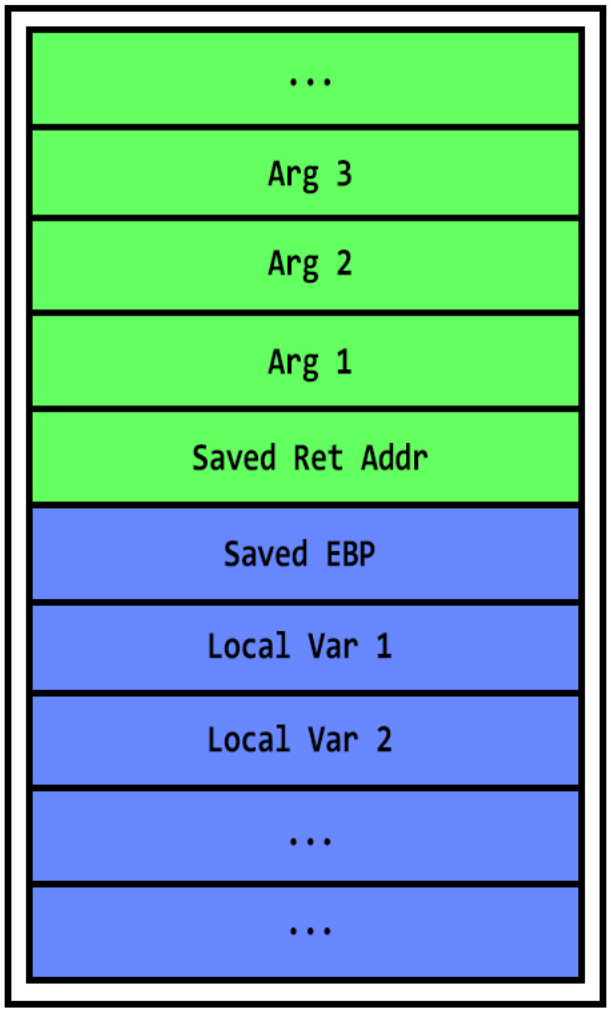
Return Oriented Exploitation (ROP)

Joshua Wang

What is ROP?

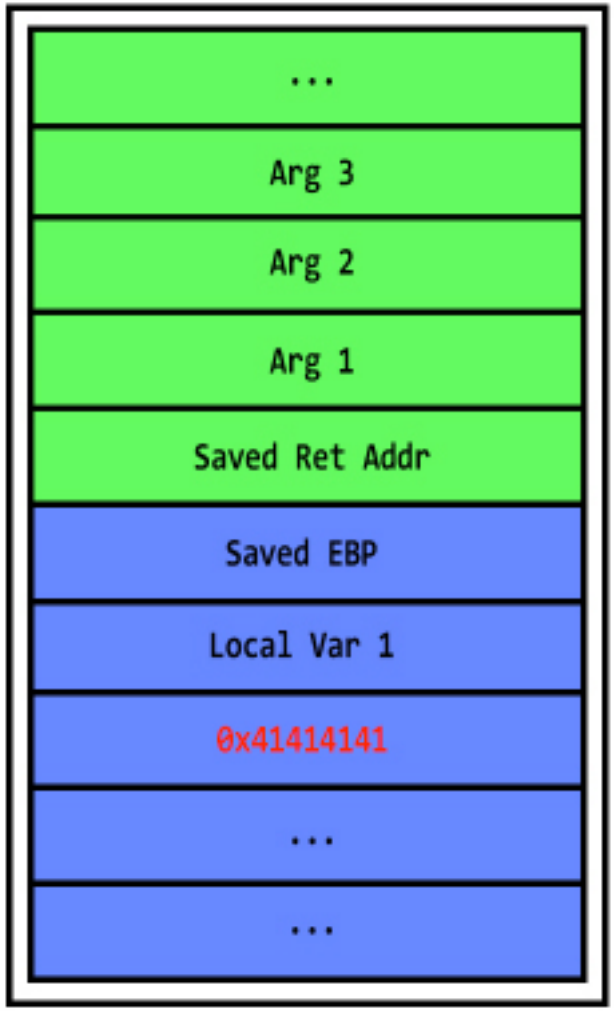
- Exploitation technique used to bypass non-executable stack and code signing

Stack Smashing 101



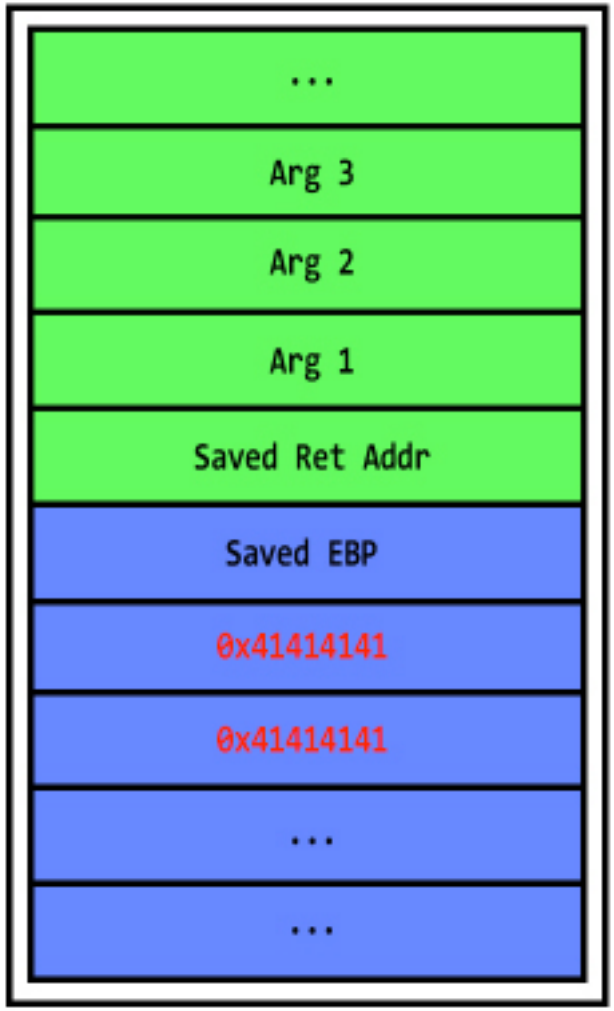

```
#include<stdio.h>
#include<stdlib.h>

void vuln(int, int, int);

int main(void){
        int arg1=1;
        int arg2=2;
        int arg3=3;

        vuln(arg1,arg2,arg3);

        return 0;
}

void vuln(int arg1, int arg2, int arg3){
 int localvar1;
 int localvar2;

 gets(&localvar2);
}
```

Stack Smashing 101



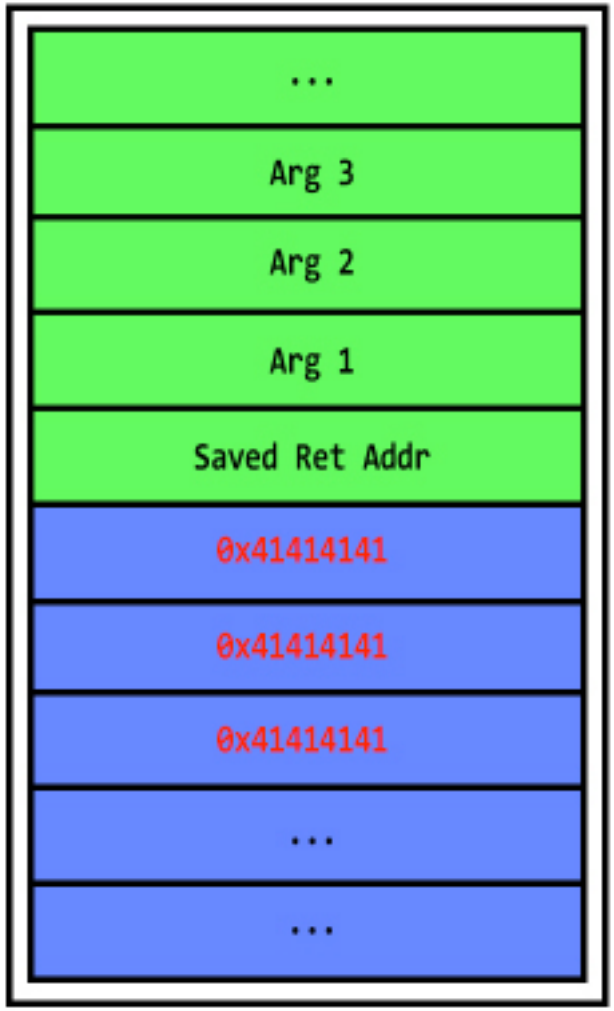

```
#include<stdio.h>
#include<stdlib.h>

void vuln(int, int, int);

int main(void){
        int arg1=1;
        int arg2=2;
        int arg3=3;

        vuln(arg1,arg2,arg3);

        return 0;
}

void vuln(int arg1, int arg2, int arg3){
 int localvar1;
 int localvar2;

 gets(&localvar2);
}
```

Stack Smashing 101




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

void vuln(int, int, int);

int main(void){
        int arg1=1;
        int arg2=2;
        int arg3=3;

        vuln(arg1,arg2,arg3);

        return 0;
}

void vuln(int arg1, int arg2, int arg3){
 int localvar1;
 int localvar2;

 gets(&localvar2);
}
```

Stack Smashing 101

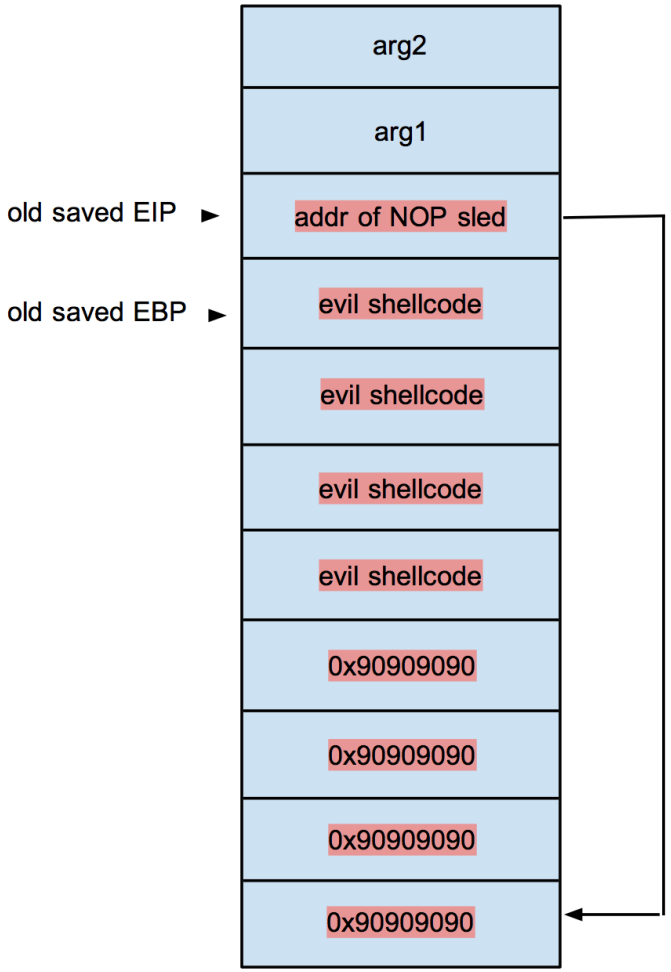



```
#include<stdio.h>
#include<stdlib.h>

void vuln(int, int, int);

int main(void){
        int arg1=1;
        int arg2=2;
        int arg3=3;

        vuln(arg1,arg2,arg3);

        return 0;
}

void vuln(int arg1, int arg2, int arg3){
 int localvar1;
 int localvar2;

 gets(&localvar2);
}
```

Stack Smashing 101

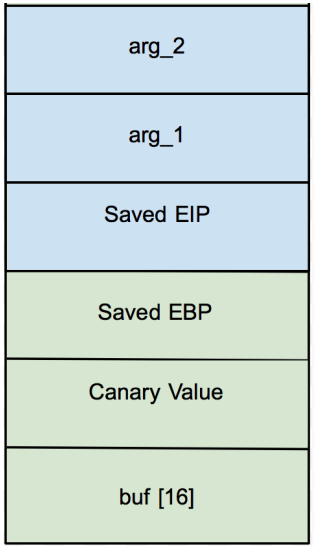

```
#include<stdio.h>
#include<stdlib.h>

void vuln(int, int, int);

int main(void){
        int arg1=1;
        int arg2=2;
        int arg3=3;

        vuln(arg1,arg2,arg3);

        return 0;
}

void vuln(int arg1, int arg2, int arg3){
 int localvar1;
 int localvar2;

 gets(&localvar2);
}
```

Pwned Stack

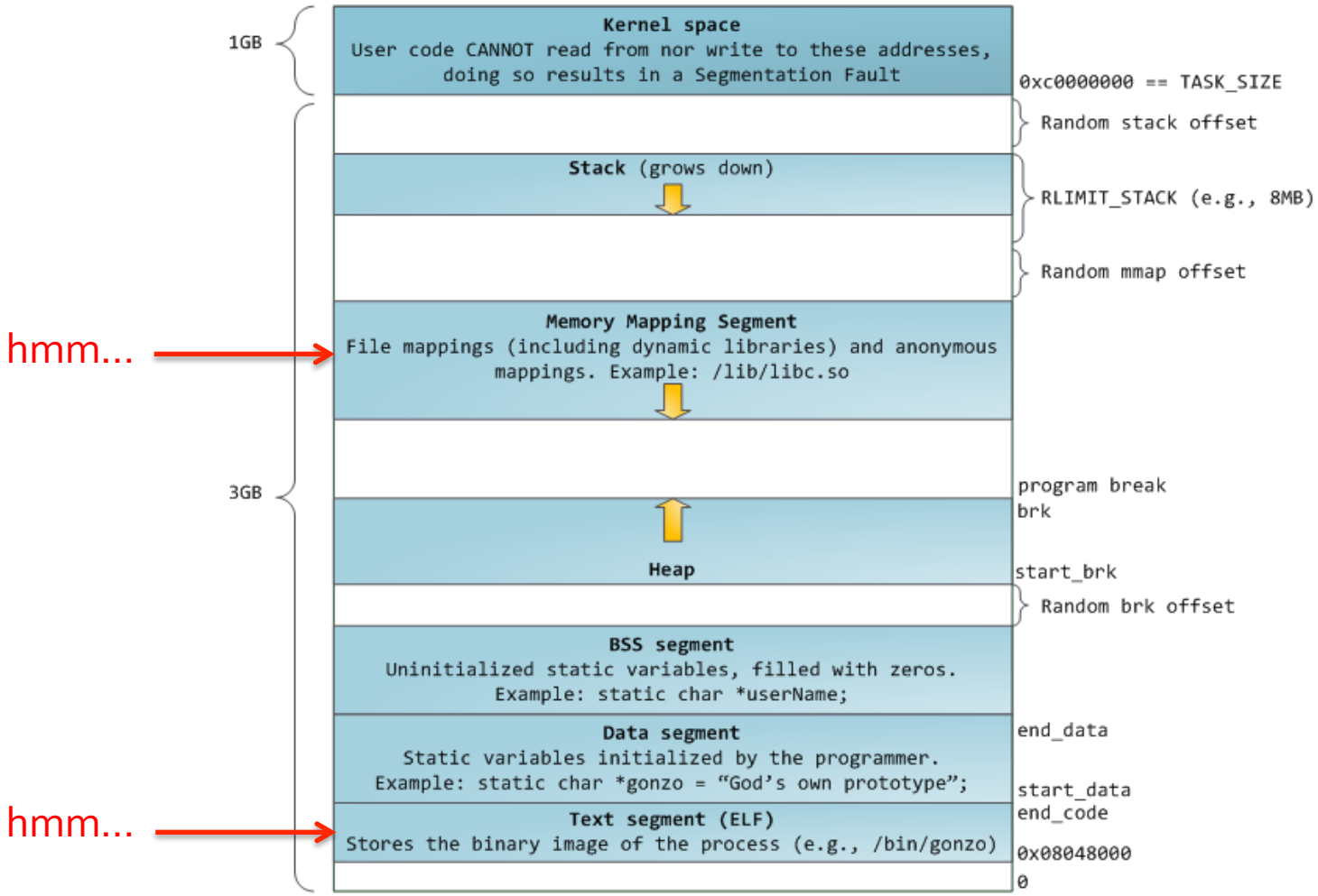


Stack Memory Protections

- Canaries
  - StackGuard [GCC level]
  - Stack Smashing Protection (SSP) [GCC level]
    - Formerly ProPolice
    - On by default since GCC 4.1
    - `-fno-stack-protector` disables SSP
- DEP/NX
  - GNU_STACK ELF markings [GCC level]
    - `-z execstack` disables NX
    - On by default since GCC 4.1
  - PaX [Kernel level]
- ASLR
  - PaX [Kernel level]
  - `/proc/sys/kernel/randomize_va_space` controls system-wide ASLR
  - `sudo bash -c "echo 0 > /proc/sys/kernel/randomize_va_space"` disables ASLR temporarily



How to execute code with DEP/NX on?

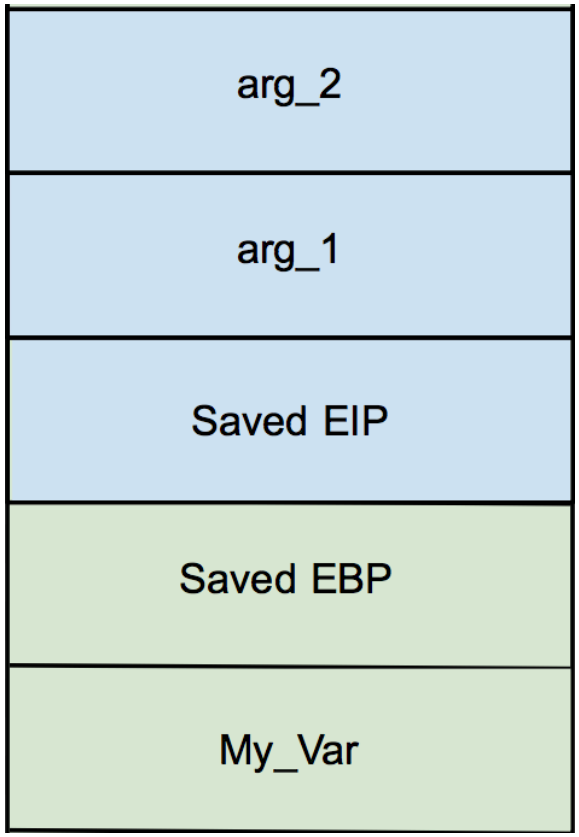


Src: http://duartes.org/gustavo/blog/post/anatomy-of-a-program-in-memory/

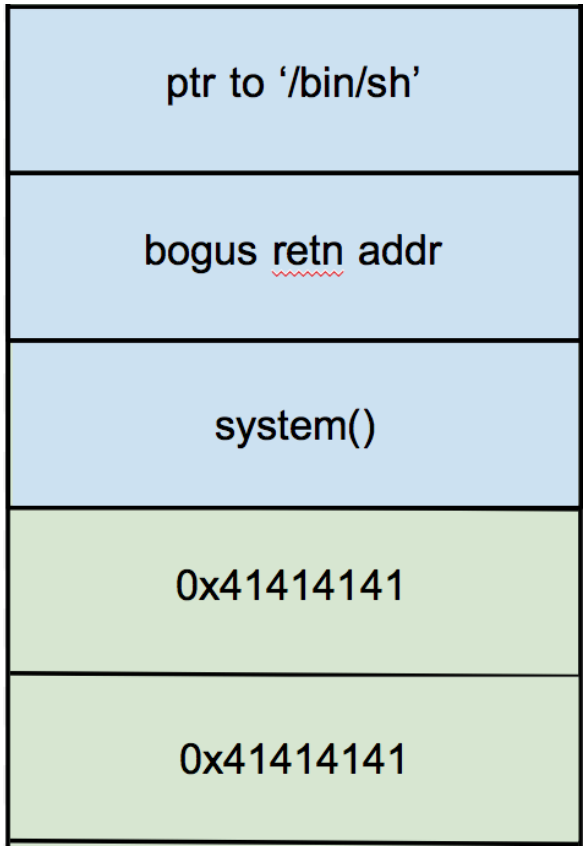
Ret-2-libc

- Goal: Execute `system("/bin/sh");`
- No shellcode required!
- Must find 3 key addresses
  1. Addr of `system()`
  2. Addr of `exit()`
  3. Addr of `/bin/sh` in libc

Ret-2-libc

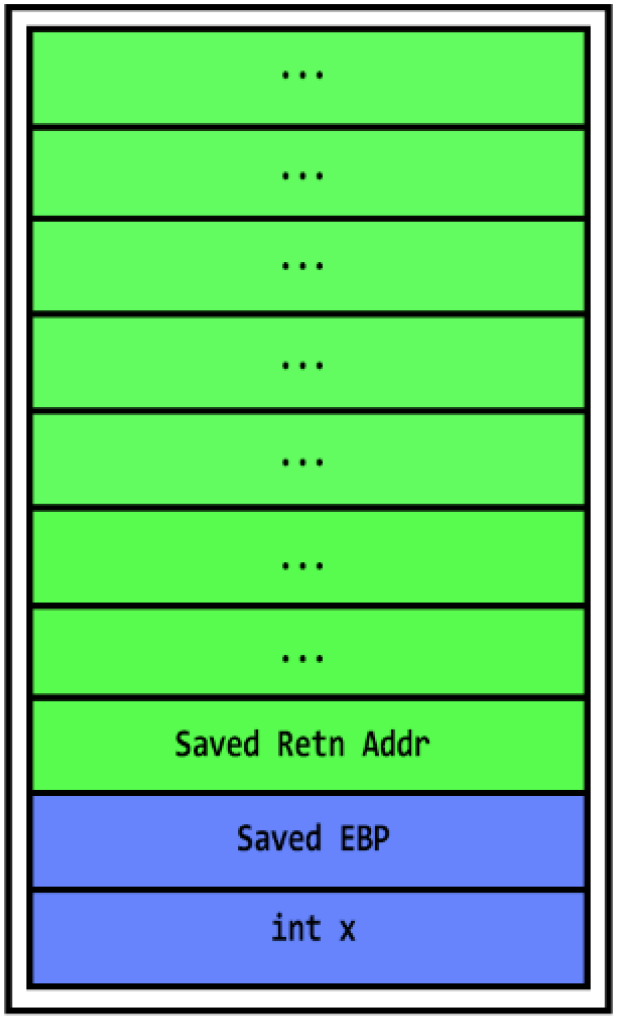


normal



system("/bin/sh")

ROP Chaining




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

void func_first(int);
void func_second (char);

int main(void){
 int x;
 gets(&x);
 return 0;
}

void func_first(int y){
   // call first
}

void func_second(char z){
   // call second
}
```

ROP Chaining


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

void func_first(int);
void func_second (char);

int main(void){
 int x;
 gets(&x);
 return 0;
}

void func_first(int y){
   // call first
}

void func_second(char z){
   // call second
}
```

ROP Chaining


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

void func_first(int);
void func_second (char);

int main(void){
  int x;
  gets(&x);
  return 0;
}

void func_first(int y){
    // call first
}

void func_second(char z){
    // call second
}
```

ROP Gadget

- Chunks of code from executable memory regions (typically ending in ret instruction)
- eg) pop-ret, pop-pop-ret, etc.

DEP is useless by itself!

But what if ASLR is enabled?

- Partial ASLR /PIE-disabled
  - Find a non-randomized memory segment
- Full ASLR /PIE-enabled
  - Brute forcing
  - Information disclosure vulnerability
    - leak address of a variable
    - use relative addressing to glean other addresses

Problems with ASLR

- Not truly randomized
  - Only 8 bits need to be brute-forced
- Only the starting address of libc changes
- Offsets remain the same

```
rh0gue@vexillum:~/Documents/ROP$ ldd vuln2
        linux-gate.so.1 =>  (0xf7778000)
        libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0xf75aa000)
        /lib/ld-linux.so.2 (0xf7779000)
rh0gue@vexillum:~/Documents/ROP$ ldd vuln2
        linux-gate.so.1 =>  (0xf76f9000)
        libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0xf752b000)
        /lib/ld-linux.so.2 (0xf76fa000)
rh0gue@vexillum:~/Documents/ROP$ ldd vuln2
        linux-gate.so.1 =>  (0xf7727000)
        libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0xf7559000)
        /lib/ld-linux.so.2 (0xf7728000)
```

Tools

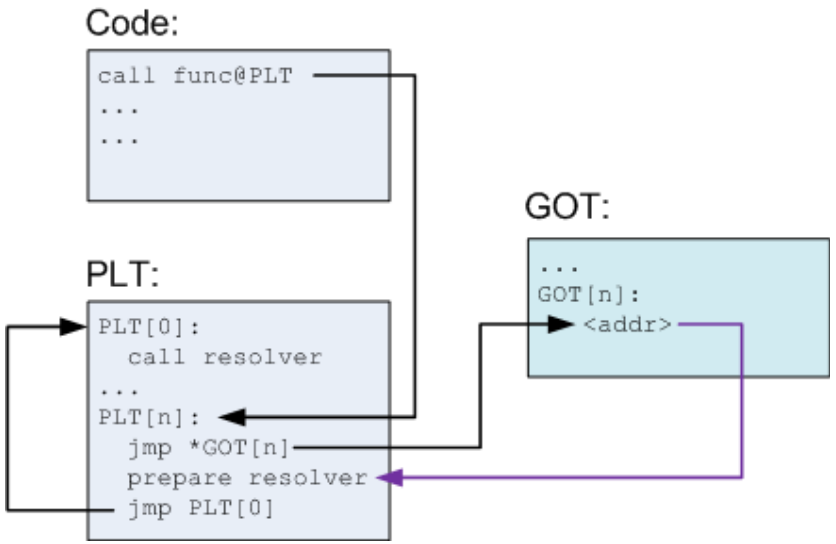
- Gdb-peda: https://github.com/longld/peda
- ROPGadget: https://github.com/JonathanSalwan/ROPgadget/tree/master
- GNU Binutils

Vuln2 Demo

Ropasaurus Demo

PLT & GOT

- first entry in PLT is special and used as a resolver routine
- ea/PLT entry contains
  - jmp to a corresponding GOT entry location
  - preparation for resolver
  - call to resolver routine
- when func() is called for the 1st time
  - compiler translates call to func@plt
    - N-th entry in PLT
  - jmps to GOT[n]
  - prepares arguments for resolver
  - jmps to PLT[0] / resolver routine
  - absolute address placed in GOT[n]



Source: http://eli.thegreenplace.net/2011/11/03/position-independent-code-pic-in-shared-libraries/

PLT & GOT

```
0804832c <read@plt>:
 804832c:       ff 25 1c 96 04 08       jmp    *0x804961c
 8048332:       68 18 00 00 00          push   $0x18
 8048337:       e9 b0 ff ff ff          jmp    80482ec <__gmon_start__@plt-0x10>
```

```
rh0gue@vexillum:~/Documents/ROP$ objdump -R ropasaurusrex

ropasaurusrex:     file format elf32-i386

DYNAMIC RELOCATION RECORDS
OFFSET   TYPE              VALUE
08049600 R_386_GLOB_DAT    __gmon_start__
08049610 R_386_JUMP_SLOT   __gmon_start__
08049614 R_386_JUMP_SLOT   write
08049618 R_386_JUMP_SLOT   __libc_start_main
0804961c R_386_JUMP_SLOT   read
```

```
gdb-peda$ x/a 0x0804961c
0x804961c <read@got.plt>:       0x8048332 <read@plt+6>
```

Resources

- https://cseweb.ucsd.edu/~hovav/dist/rop.pdf
- https://www.corelan.be/index.php/2010/06/16/exploit-writing-tutorial-part-10-chaining-dep-with-rop-the-rubikstm-cube/
- http://www.exploit-db.com/wp-content/themes/exploit/docs/28479.pdf
- http://www.fuzzysecurity.com/tutorials/expDev/7.html