how to pwn lol

Nathan Huckleberry

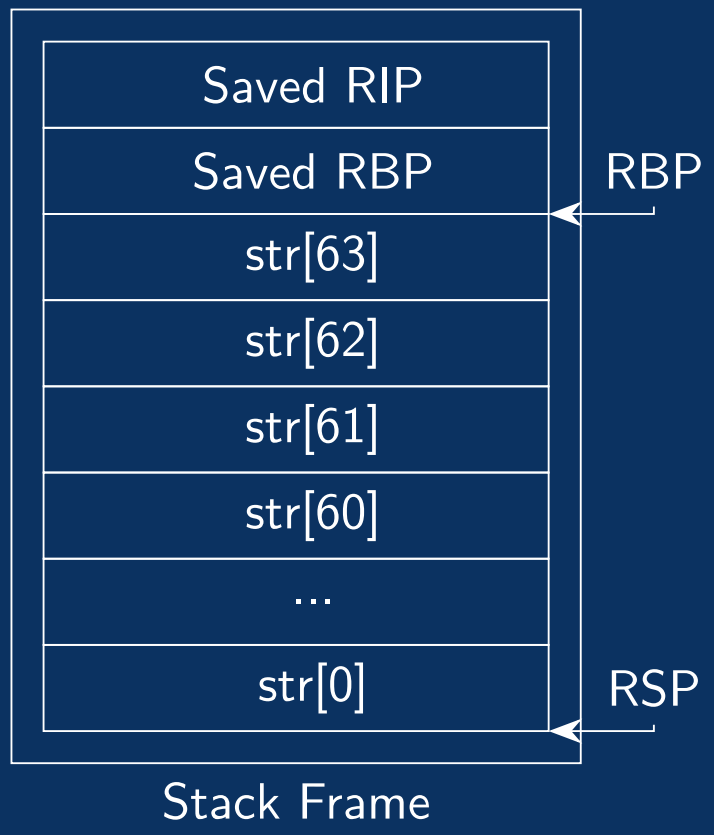
table of contents

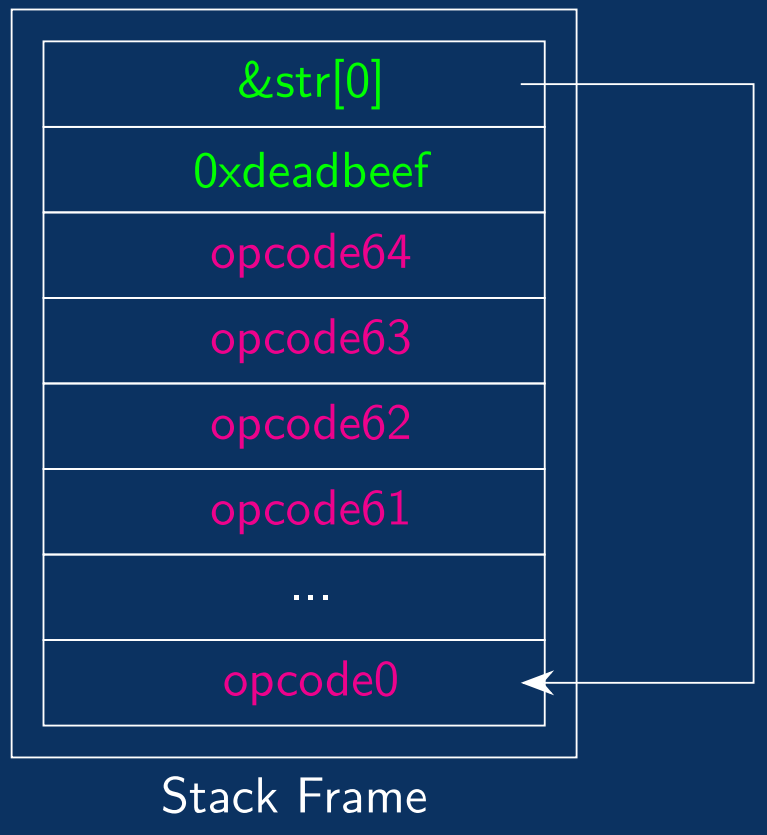
table of contents

shellcoding ⭐⭐

- 😎 What if we put x86 opcodes in a string and called it like a function
- 🔥 Really easy method of arbitrary code execution
- 🗿 Will probably hurt security mitigation engineers' feelings

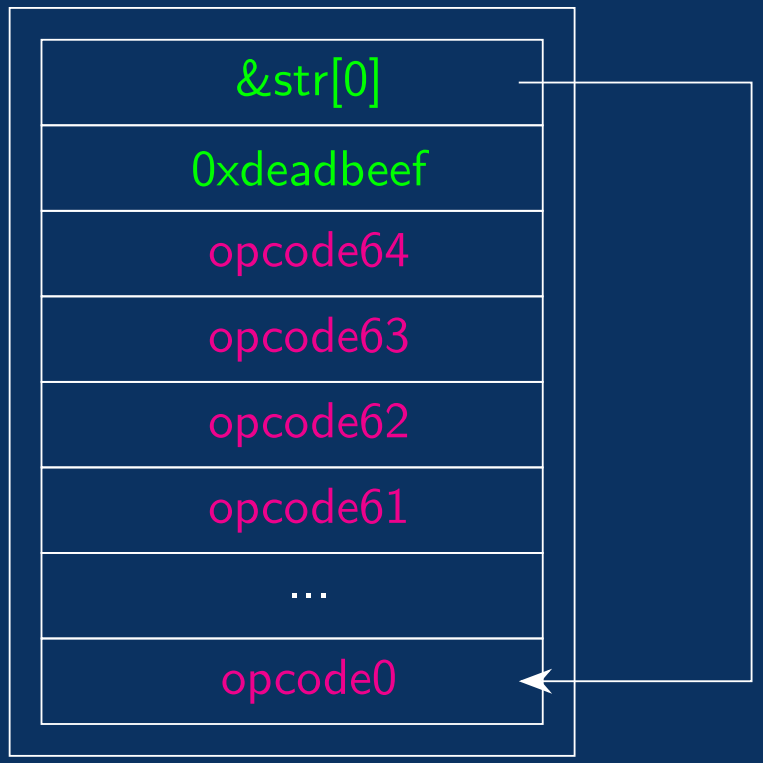
shellcoding





shellcoding

- Allows us to run an arbitrary 64 byte program



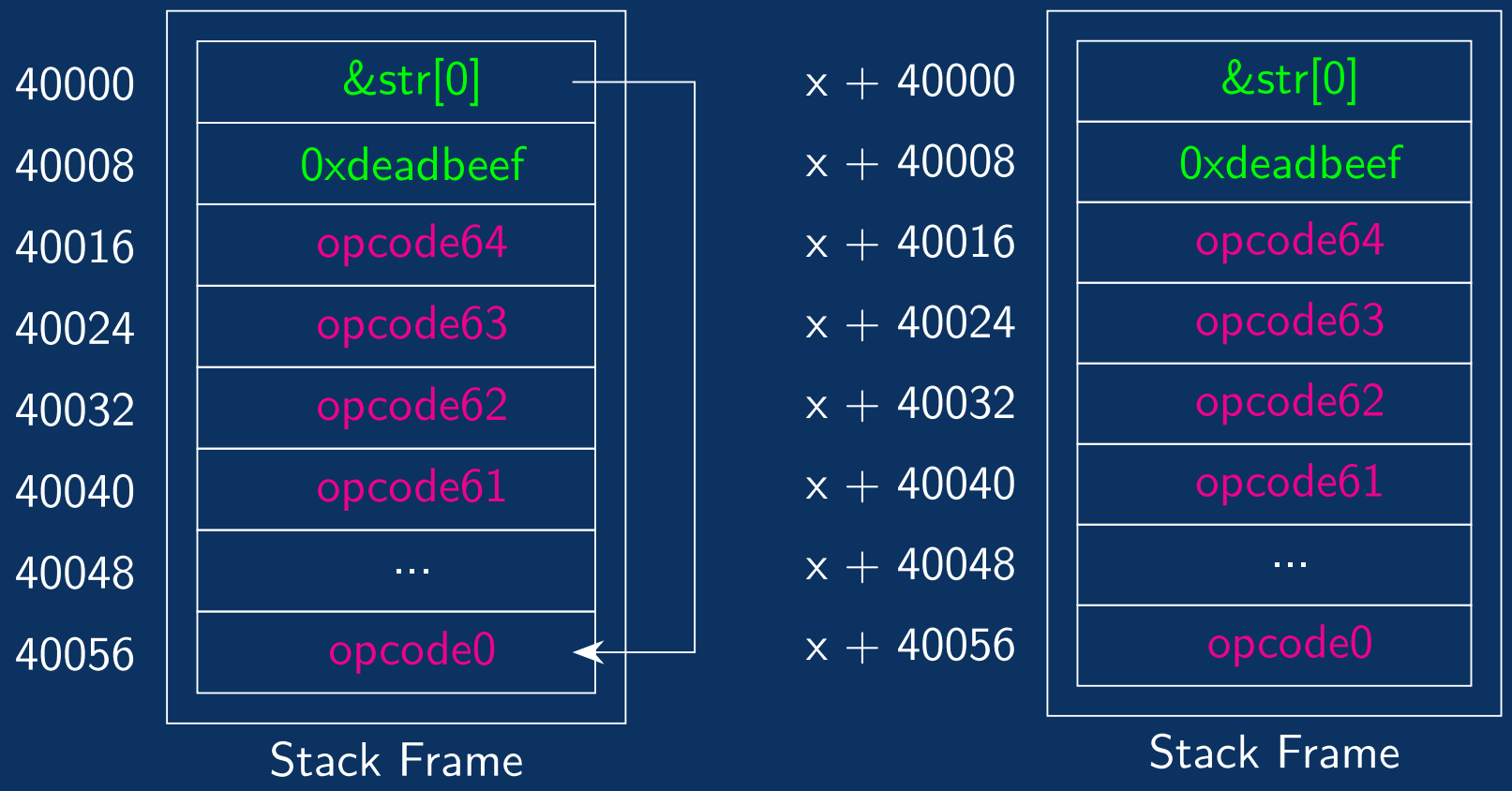
Stack Frame

table of contents

aslr

- Mitigation engineers feelings were hurt
  - They decided to add a constant to stack, heap and dynamic library addresses
  - Each constant is randomized per program run
- Now we can't jump to shellcode because we can't predict its address
- Doesn't randomize code addresses or global variable addresses

aslr


40000
40008
40016
40024
40032
40040
40048
40056
&str[0]
0xdeadbeef
opcode64
opcode63
opcode62
opcode61
...
opcode0
Stack Frame
x + 40000
x + 40008
x + 40016
x + 40024
x + 40032
x + 40040
x + 40048
x + 40056
&str[0]
0xdeadbeef
opcode64
opcode63
opcode62
opcode61
...
opcode0
Stack Frame

aslr

- For our exploit to work we need to jump to x + &str[0] instead of &str[0]
- Since we can't predict x, our exploit is broken

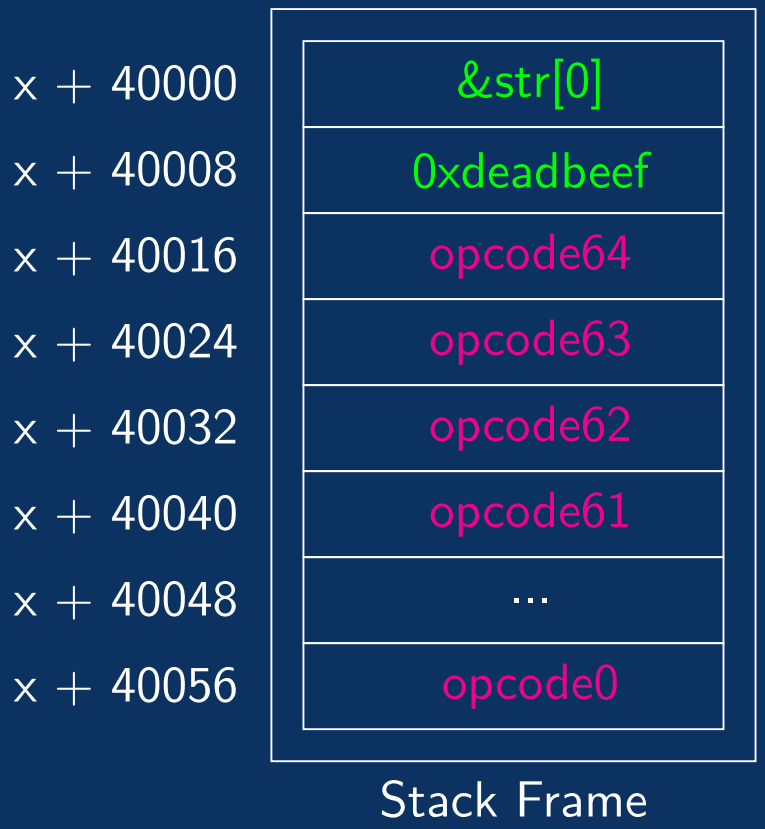


table of contents

stack info leaks ⭐

- 🗿 Mitigation engineers really forgot that we can print things
- 🔥 If we can print a stack address, we can learn x
- 😞 Unfortunately, stack info leaks are kinda hard
  - More on other info leaks later

table of contents

rop ⭐⭐⭐

- 🔥 Instead of shellcoding, we can just reuse the code in the binary
- 🗿 Unfortunately binaries don't usually include exploit payloads
- 😢 Not useful on its own

wtf is a rop

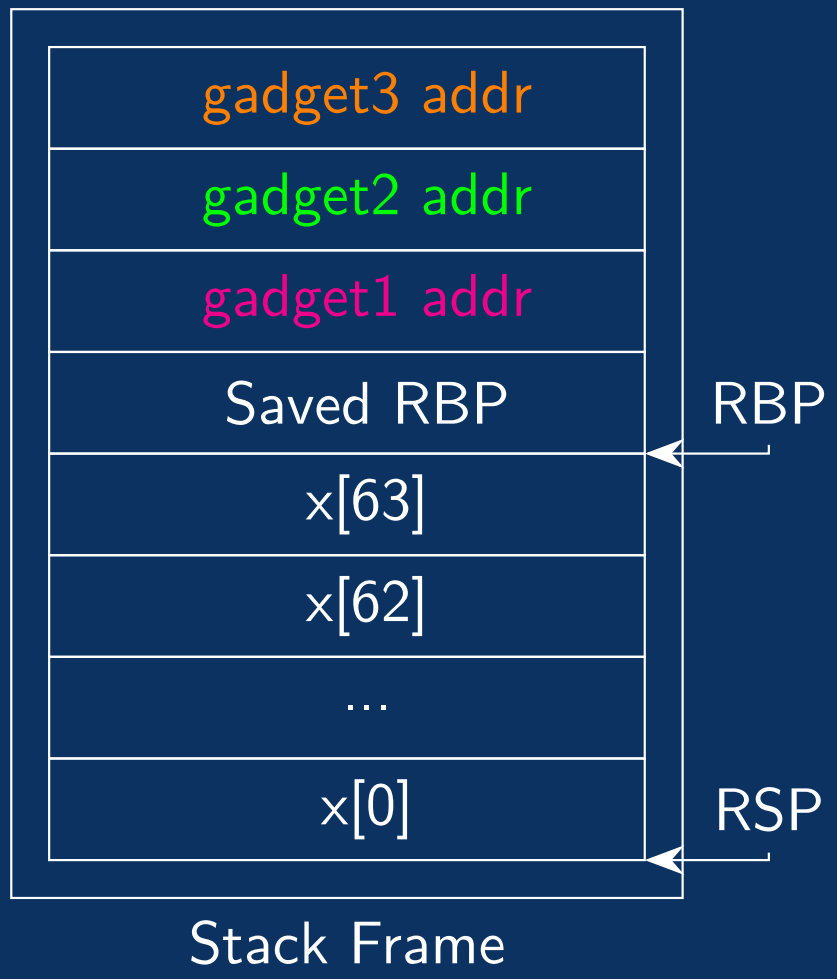
- ROP is done by chaining instruction sequences called gadgets
- gadgets are short instruction sequences ending in a ret
- ROP gadgets can be chained by putting them in a row in a buffer overflow

Listing 1: pop rdi gadget

```
pop rdi;
ret;
```

how 2 rop

- This is known as a ROP-chain
- gadget1 will be called, then gadget2, then gadget3



why do i care about rop

- 😎 ROP *is* useful for running a few instructions before jumping to shellcode
- 🧠 This gives us way more flexibility in our exploits

table of contents

is shellcoding dead?

- 😢 Without a stack info leak we can't determine where our shellcode is on the stack
- 😢 aslr completely destroys shellcoding by randomizing addresses

.bss ⭐⭐⭐⭐

- 🧠 What if we just use a fixed address lol
- 💻 Uninitialized file scope global variables live in a segment called .bss
- 🔥 The address of .bss is not randomized at runtime
- 🔥 There's usually a big unused space in .bss

.bss ⭐⭐⭐⭐

- 🔥 Just write shellcode into .bss
- 🔥 Then jump to .bss instead of stack

.bss ⭐⭐⭐⭐

- Mitigation engineers really forgot that fixed addresses exist

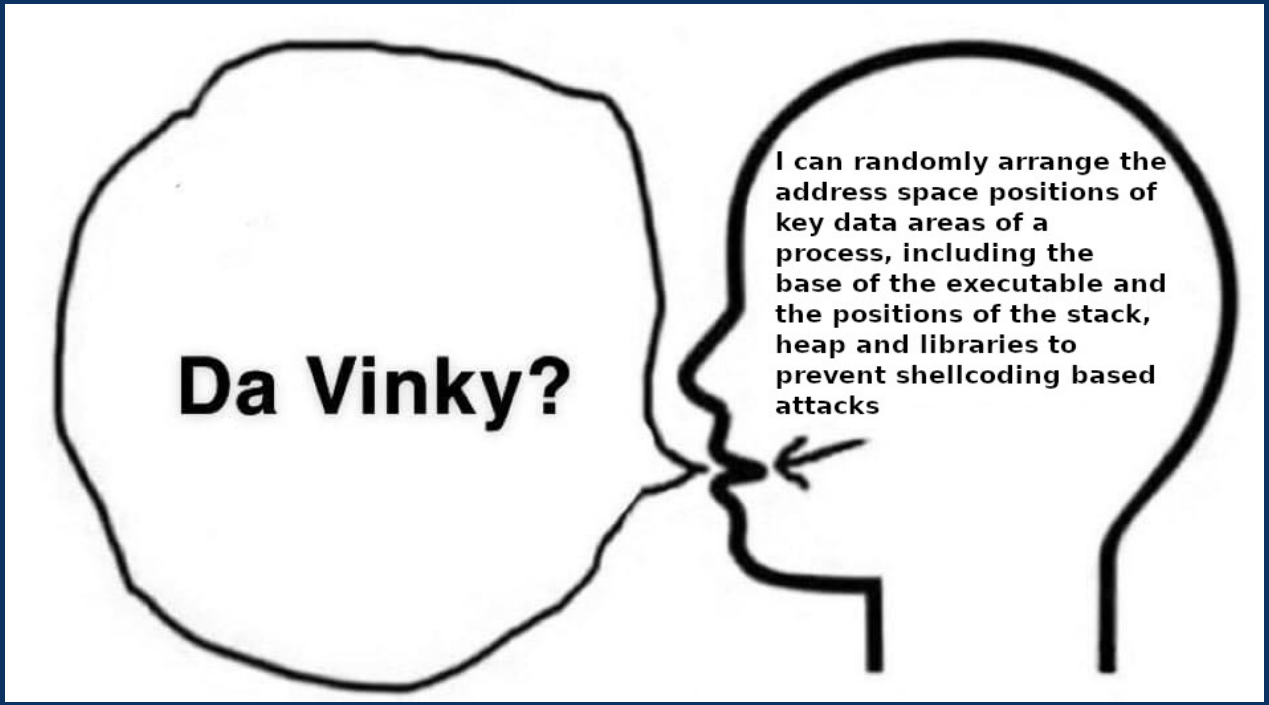


Figure 1: Mitigation Engineers

.bss

Prev Stackframe
Prev Stackframe
Prev Stackframe
Saved RIP
Saved RBP
str[63]
...
str[0]

Stack Frame

.bss addr
gets
.bss addr
pop rdi gadget
Saved RBP
str[63]
...
str[0]

Stack Frame

.bss

- We put .bss into rdi
- We call gets with rdi as .bss
- The program reads shellcode from us
- We call .bss and our shellcode runs

Stack Frame

table of contents

nx

- Mitigation engineers are determined to get it right this time
- Prevent execution of writeable data
- Jumping to writable data causes a segfault
- We can't use shellcode anymore because it's writeable

nx

Figure 2: well that sucks

is nx on

- Pwntools has a builtin tool called checksec

```
[huck@manjaro rop_pwn]$ checksec rop
[*] '/home/huck/Downloads/rop_pwn/rop'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

Figure 3: checksec

now is shellcoding dead?

- 😡 Shellcoding is mostly dead
- 🧠 If we *really* wanted to we could use mprotect to mark shellcode as read-only and execute
- 🗿 This is usually too much work lol

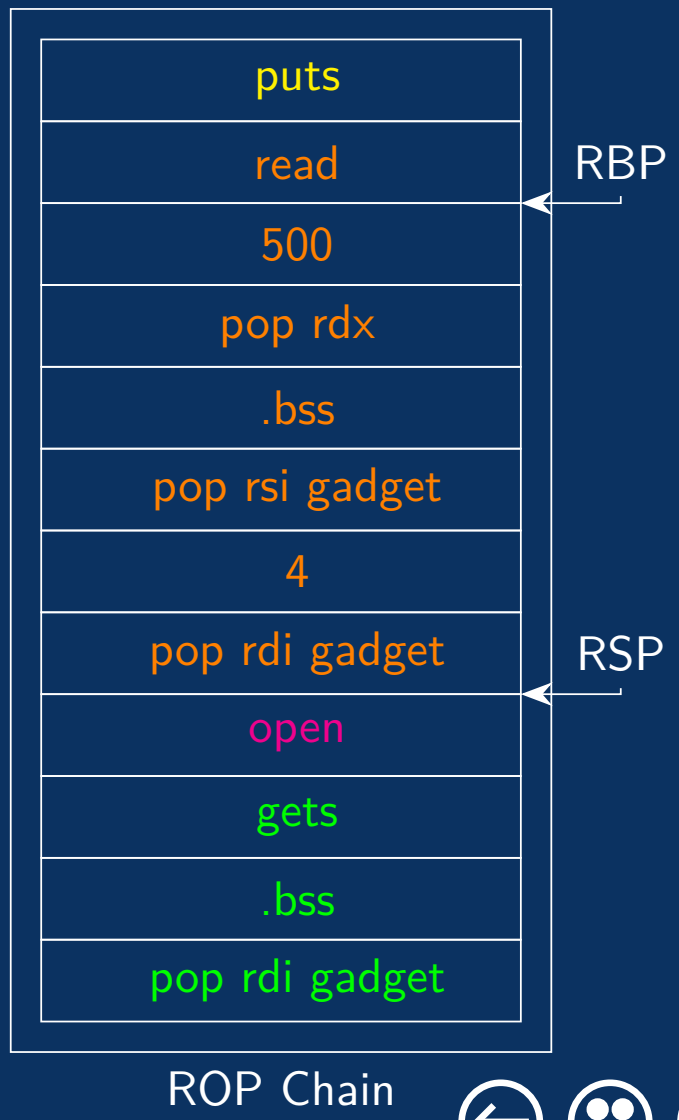
table of contents

rop but smart ⭐⭐⭐⭐

- 🧠 Functions are just big ROP gadgets
- 😎 Sometimes we can do interesting things by calling libc functions as big ROP gadgets

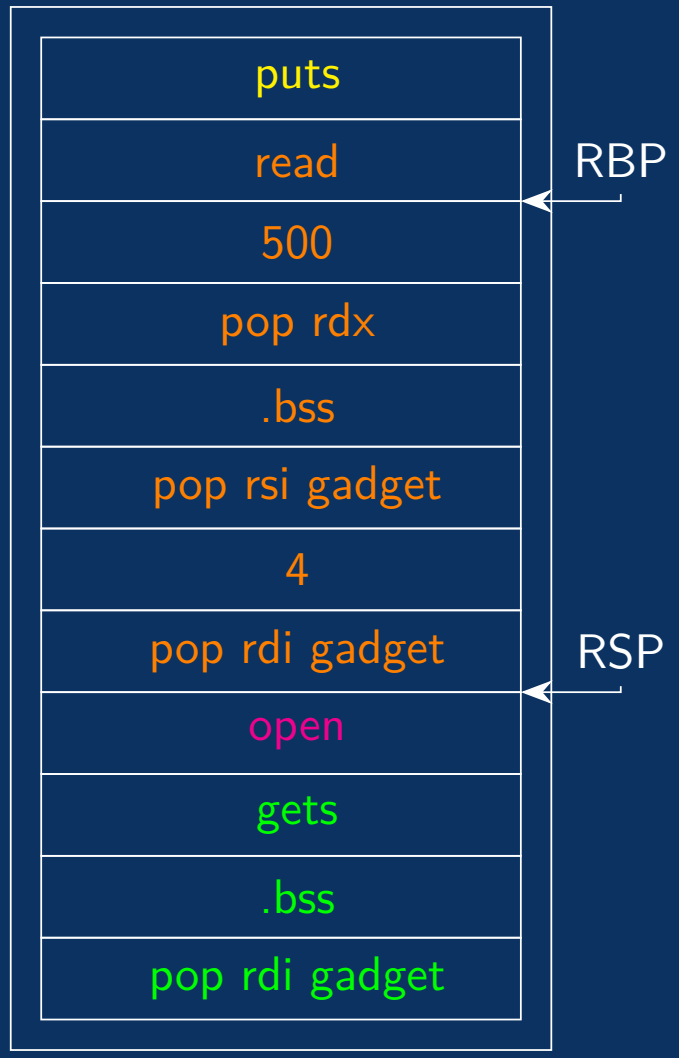
rop but smart example

- Call gets(.bss)
- We input /flag.txt



rop but smart example

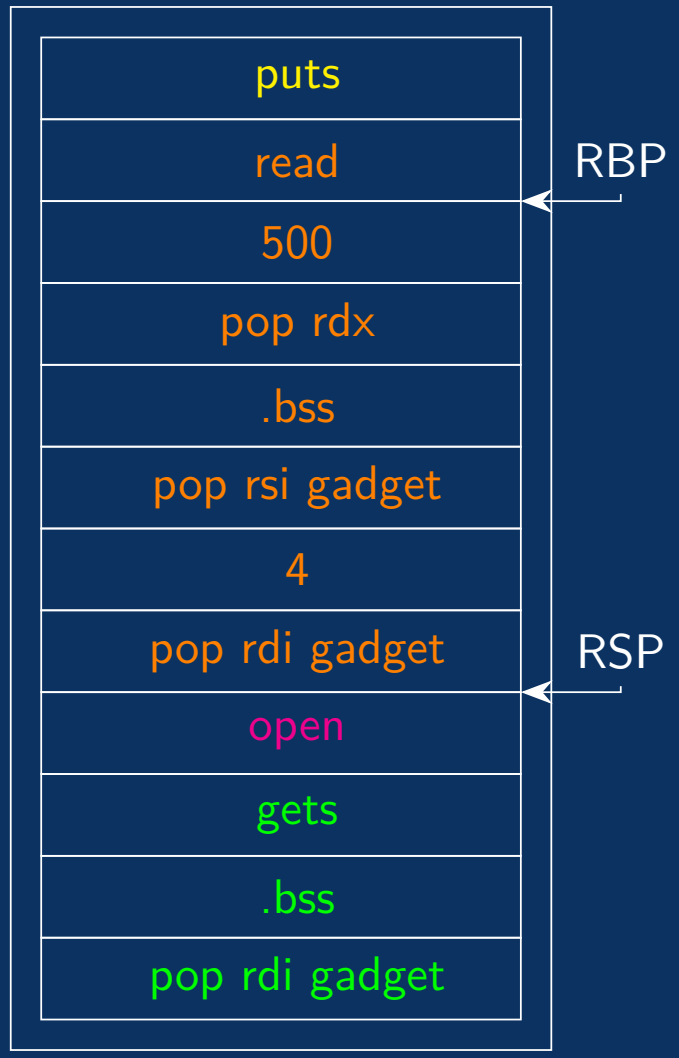
- Call gets(.bss)
- We input /flag.txt
- "/flag.txt" is written to .bss



ROP Chain

rop but smart example

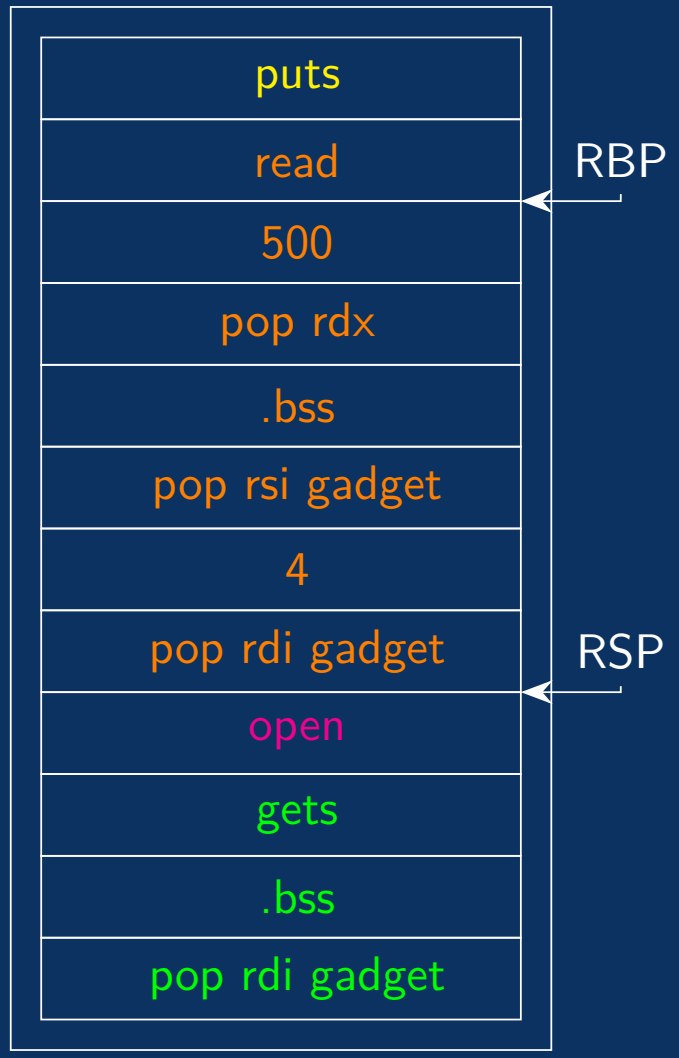
- Call gets(.bss)
- We input /flag.txt
- "/flag.txt" is written to .bss
- Call open(.bss, junk)
- /flag.txt is opened with (probably) file descriptor 4



ROP Chain

rop but smart example

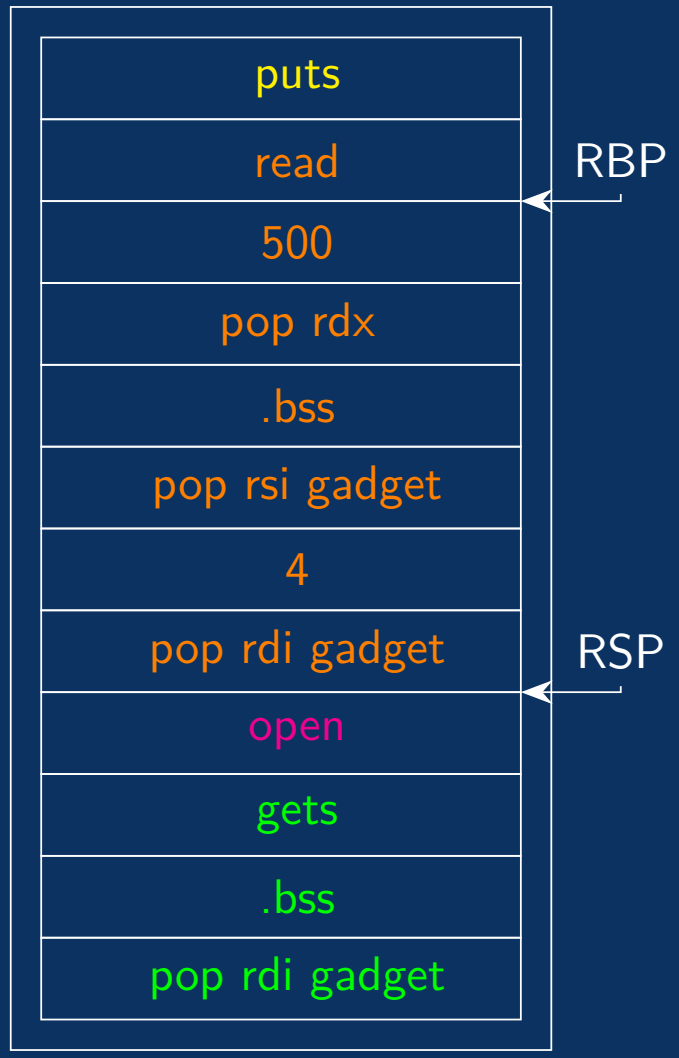
- Call gets(.bss)
- We input /flag.txt
- "/flag.txt" is written to .bss
- Call open(.bss, junk)
- /flag.txt is opened with (probably) file descriptor 4
- Call read(4, .bss, 500)
- Read 500 bytes from file into .bss



ROP Chain

rop but smart example

- Call gets(.bss)
- We input /flag.txt
- "/flag.txt" is written to .bss
- Call open(.bss, junk)
- /flag.txt is opened with (probably) file descriptor 4
- Call read(4, .bss, 500)
- Read 500 bytes from file into .bss
- Call puts(.bss)
- Print file contents



ROP Chain

interesting functions

- The previous example requires interesting functions (gets, puts, read, open) in the binary
- If the binary doesn't have any interesting functions we need to do something else

table of contents

ret2libc ⭐⭐⭐⭐⭐

- 💻 Libc calls are made through a wrapper
- 🗿 The entirety of libc is loaded somewhere in memory
- 😡 Libc has randomized base address due to aslr

ret2libc ⭐⭐⭐⭐⭐

- 🔥 If we can find libc, we can call any libc function
- 💻 Calling system is a common target

libc wrapper

- Addresses to libc functions are stored in a table called the global offset table (GOT)
- When calling a libc function, we look up the GOT entry and jump there
- The code responsible for looking up GOT entries is stored in the procedure linkage table (PLT)

proof by example

Proof.

- We call printf in our source code

□

proof by example

Proof.

- We call printf in our source code
- The compiler expands printf to printf@plt

□

proof by example

Proof.

- We call printf in our source code
- The compiler expands printf to printf@plt
- We jump to the printf entry of the PLT

□

proof by example

Proof.

- We call printf in our source code
- The compiler expands printf to printf@plt
- We jump to the printf entry of the PLT
- The printf@plt function reads the GOT entry for printf

□

proof by example

Proof.

- We call printf in our source code
- The compiler expands printf to printf@plt
- We jump to the printf entry of the PLT
- The printf@plt function reads the GOT entry for printf
- This entry tells us where the actual printf function is

□

proof by example

Proof.

- We call printf in our source code
- The compiler expands printf to printf@plt
- We jump to the printf entry of the PLT
- The printf@plt function reads the GOT entry for printf
- This entry tells us where the actual printf function is
- We jump to this address

□

why do i care about the GOT

- Compiler engineers really put a table of libc address leaks in our binary
- They're also at fixed addresses that aren't randomized by aslr

why do i care about the GOT

- 🗿 Compiler engineers really put a table of libc address leaks in our binary
- 🗿 They're also at fixed addresses that aren't randomized by aslr
- 🔥 If we leak a GOT entry, we can easily find the libc offset and call any libc function we want

why do i care about the GOT

- 🗿 Compiler engineers really put a table of libc address leaks in our binary
- 🗿 They're also at fixed addresses that aren't randomized by aslr
- 🔥 If we leak a GOT entry, we can easily find the libc offset and call any libc function we want
- 🤬 Mitigation engineers BTFO

leaking libc

Figure 4: Exploit devs probably

libc info leaks

- 🔥 Much more useful than stupid stack info leaks
- 🔥 Easy since the GOT exists

leaking libc

Prev Stackframe
Prev Stackframe
Prev Stackframe
Saved RIP
Saved RBP
str[63]
...
str[0]

Stack Frame

main
puts
got.puts
pop rdi gadget
Saved RBP
str[63]
...
str[0]

Stack Frame

leaking libc

- Call puts(got.puts) to print the libc address of puts
- Call main to "restart" the program without changing libc offset
- Now we can compute the base address of libc

Stack Frame

ret2libc

Prev Stackframe
Prev Stackframe
Saved RIP
Saved RBP
str[63]
str[62]
...
str[0]

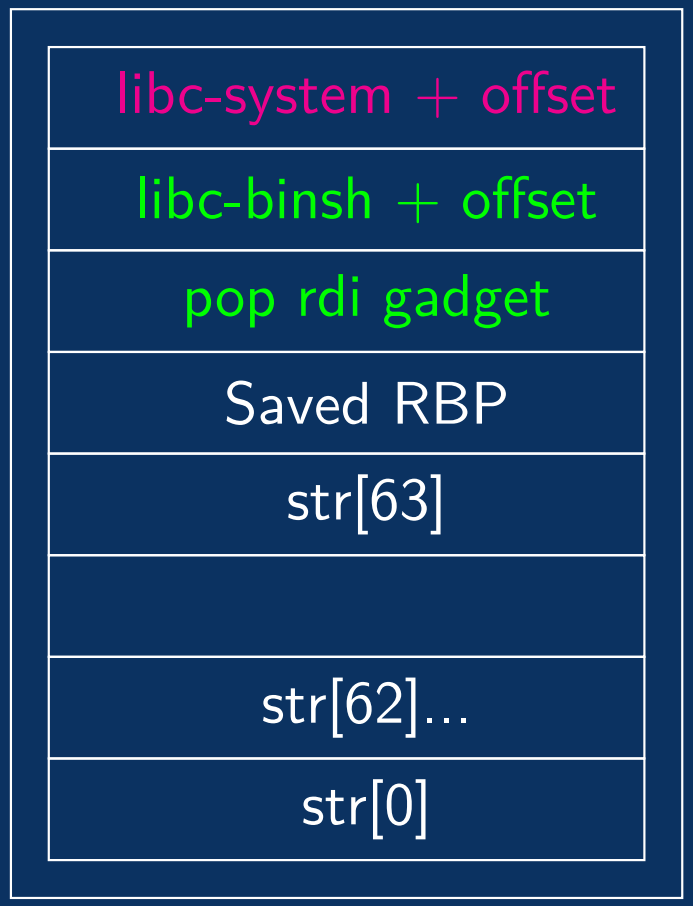
Stack Frame

libc-system + offset
libc-binsh + offset
pop rdi gadget
Saved RBP
str[63]
str[62]...
str[0]

Stack Frame

ret2libc

- Libc contains the string "/bin/sh" somewhere
- We put this string into rdi then call system
- Now we have a shell



Stack Frame

ret2libc

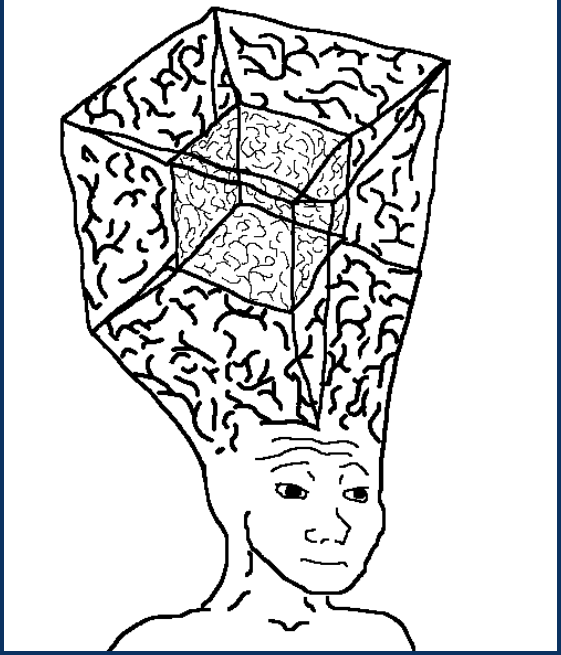


Figure 5: People who can ret2libc

table of contents

got overwrites ⭐⭐⭐⭐

- 🗿 The GOT is writable
- 🧠 If we overwrite a GOT entry, all future calls will go to our address instead
- 🔥 If we can find a way to overwrite GOT entries, we don't even need buffer overflows

Questions