《漏洞利用及渗透测试基础》实验报告

林晖鹏 2312966

March 2025

1 实验名称

格式化字符串漏洞

2 实验要求

以第四章示例 4-7 代码，完成任意地址的数据获取，观察 Release 模式和 Debug 模式的差异，并进行总结。

3 实验原理

C 语言中，格式化函数允许在传入的字符串中获知可变参数的个数和类型，并在栈中相应位置读取参数。比如‘%x’就会以十六进制数形式输出参数，由于参数与格式化字符串相邻，所以就会直接读取字符串后面的地址。此时，如果我们在格式化字符串中加了格式化符号，而没有写入参数，就会将字符串地址栈后面的内容以相应格式输出，来达到泄漏内存的目的。

4 实验过程

4.1 实验流程

1. 我们用 4-7 代码生成的执行程序，输入’AAAA%x%x%x%x’，观察输出结果。
2. 在 Release 模式下和 Debug 模式下分别进行实验，观察并分析结果的差异。

4.2 实验代码解析

```
#include<stdio.h>
int main(int argc, char * argv[])
{
    char str[200];
    fgets(str,200,stdin);     \\输入'AAAA%x%x%x%x'
    printf(str);              \\输出结果
    return 0;
}
```

4.3 Release 模式下执行结果及分析

图 1: Release 模式

为了分析输出的东西，我们来利用 Olydbg 来分析栈中的信息。

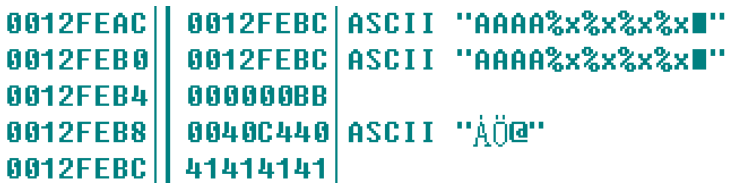

图 2: Release 栈

可以看到，在栈中，当我们输出完 “AAAA%x%x%x%x” 的时候，由于含有格式化符号，它会继续读取后面四个位置的地址，以 16 进制输出，而由于我们这里没有输入格式化参数，所以它直接读取了栈中后面四个元素的内容。分别为 “0012febc”、“000000bb”、“0040c400”、“41414141”。而十六进制数字前缀的 0 会省略，所以得到图 1 所示的输出结果。

4.4 Debug 模式下执行结果及分析

4.4.1 执行结果

图 3: Debug 模式

同样，为了分析输出的东西，我们来利用 Olydbg 来分析栈中的信息。

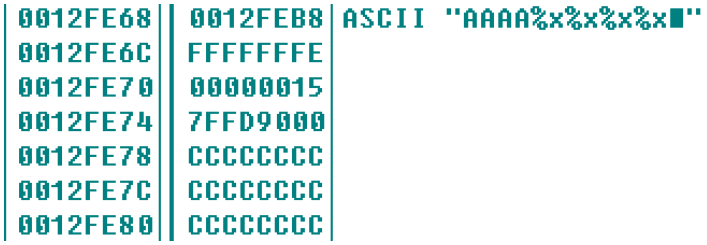

图 4: Debug 栈

可以看到，后面四个元素分别为“ffffffe”、“00000015”、“7ffd9000”、“ccccccc”，与得到的结果一致。

4.4.2 汇编代码分析

下面我们来分析这个结果。我们先来观察 printf 的汇编代码：观察图 11，这里把传入的 str 格式化字符串的地址传给来寄存器 ECX，我们检查 ECX 地址的内容也可以印证这点 (图 6)，并且此时把 exc 的地址压入栈中。

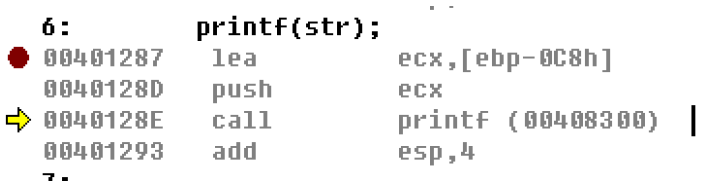

图 5: printf 函数汇编代码 (外层)

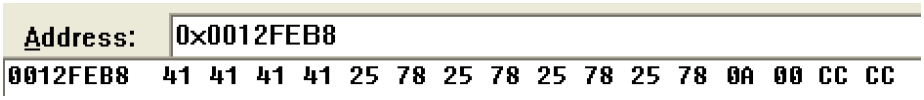

图 6: EXC 地址内容 (就是 str 的内容)

然后我们进入内层的函数，先是依次把 ebp、ebx、esi、edi 压入栈

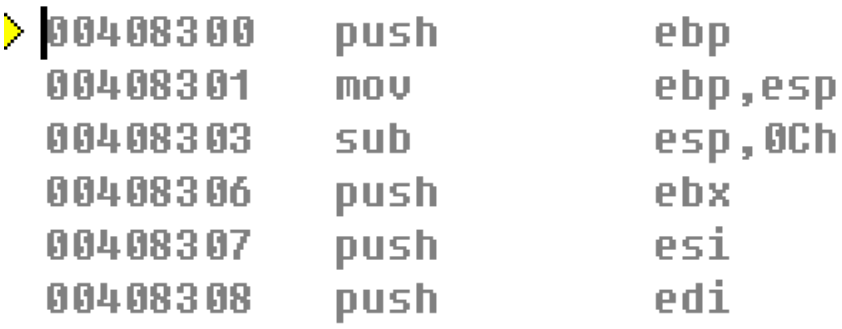

图 7: Enter Caption

所以实际上在 debug 模式中，后面四个元素对应的应该是 edi、esi、ebx 以及系统预留的大片内存位置。

5 额外尝试

下面我们使用%s 来尝试再次打印字符串。我们从前面的栈可以知道，字符串后面存的第一个元素还是自己，如果此时我们将这个元素以字符串格式输出，就能在此打印自己。

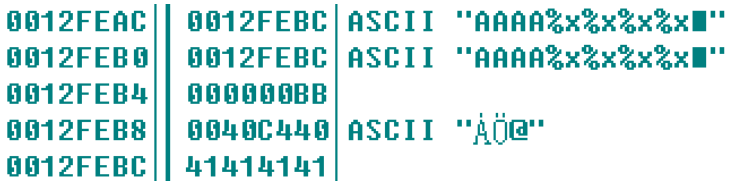

图 8: Release 栈

所以这里我们输入 “AAAA%s%x%x%x”，得到下图结果

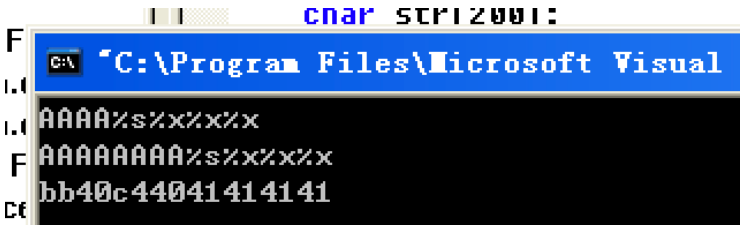

图 9: Release 尝试结果

可以看到,除了先打印 AAAA 之外,重新打印了一次 AAAA%s%x%x%x。

而当我尝试去用 debug 模式下尝试的时候，发现会将第一个元素识别成 @，可能是该元素刚好能被识别成 ASCII 码中的 @。

图 10: debug 尝试

而我们尝试将%s 放到别的位置的时候，程序显示报错，应该是别的位置无法被识别成 ASCII 码，导致无法按字符串输出。

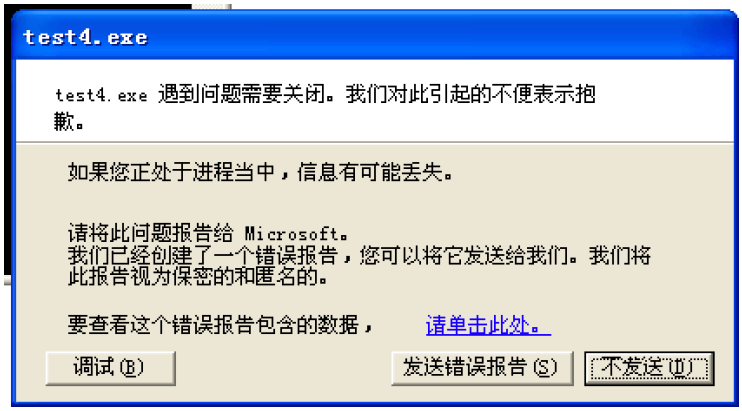

图 11: 程序报错

6 Release 模式与 Debug 模式的区别总结

- **栈结构不同**:

  - **Release 模式**：栈更紧凑，变量按最小开销优化排列，有效信息更集中，容易泄露有用数据。
  - **Debug 模式**：栈中添加了调试辅助信息，如 0xCCCCCCCC（未初始化内存标记），变量布局较为松散。

- **输出结果不同**：
  - **Release 模式**：%x 能打印出看似有意义的地址或数据（如变量值、指针地址等）。
  - **Debug 模式**：%x 打印出的往往是调试填充值或寄存器值，如 0xCCCCCCCC、0xFFFFFFFE 等。

- **可利用性差异**：
  - **Release 模式**：更容易构造攻击，例如通过%s 打印自身字符串实现信息泄漏。
  - **Debug 模式**：由于填充值或不可读内存，格式化字符串利用更难，甚至可能导致程序崩溃。

- **函数调用行为差异**：
  - **Debug 模式**中，调用 printf 前会先保存多个寄存器（如 EDI、ESI、EBX 等），因此栈中这些寄存器的值会被输出。
  - **Release 模式**中，由于优化，这些中间变量不会保存，输出的则是更“靠近用户数据”的内容。

- **调试与追踪的便利性**：
  - **Debug 模式**：更容易用工具（如 OllyDbg）观察变量和函数调用细节。
  - **Release 模式**：优化后可能省略部分变量信息，调试不便。

7 心得体会

通过本次实验，我深入理解了格式化字符串漏洞的原理及其在不同编译模式下的表现差异。在 Release 模式下，由于编译器优化，栈空间布置更加紧凑，能够较容易地通过格式化字符串泄露有效的内存信息。而在 Debug 模式下，编译器为了调试方便，会在栈中加入一些额外信息和填充值，如 0xcccccccc，虽然更易于调试，但也改变了内存布局，影响了漏洞利用的效果。

实验中还尝试使用%s 读取字符串内容，进一步验证了我们对栈中参数排列的理解。这种从理论到实践的过程，加深了我对漏洞原理的掌握，也提醒我编程中应避免类似的危险写法，如直接将用户输入作为 printf 的格式化字符串。

本次实验不仅提升了我对格式化字符串漏洞的认识，也增强了我对底层内存结构和调试工具使用的能力，为后续深入学习漏洞利用与防护打下了坚实基础。