南開大學
Nankai University

计算机学院

软件安全实验报告

实验四：格式化字符串漏洞

姓名：林盛森

学号：2312631

专业：计算机科学与技术

2025 年 4 月 9 日

目录

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。实验代码如下所示：

```
#include <stdio.h>
int main(int argc, char *argv[])
{
char str[200];
fgets(str,200,stdin);
printf(str);
return 0;
}
```

3 实验过程

3.1 debug 模式

首先我们在 vc6 中填入代码并在 debug 模式下进行编译运行，然后再 debug 文件夹下找到 exe 文件，并将 exe 文件拖入 ollydbg 中进行逆向分析。进入 ollydbg 中得到如下的汇编代码，我们找到函数入口点 00401005 这个位置，右键点击 breakpoint->run to selection 即可将程序运行到此句代码处。

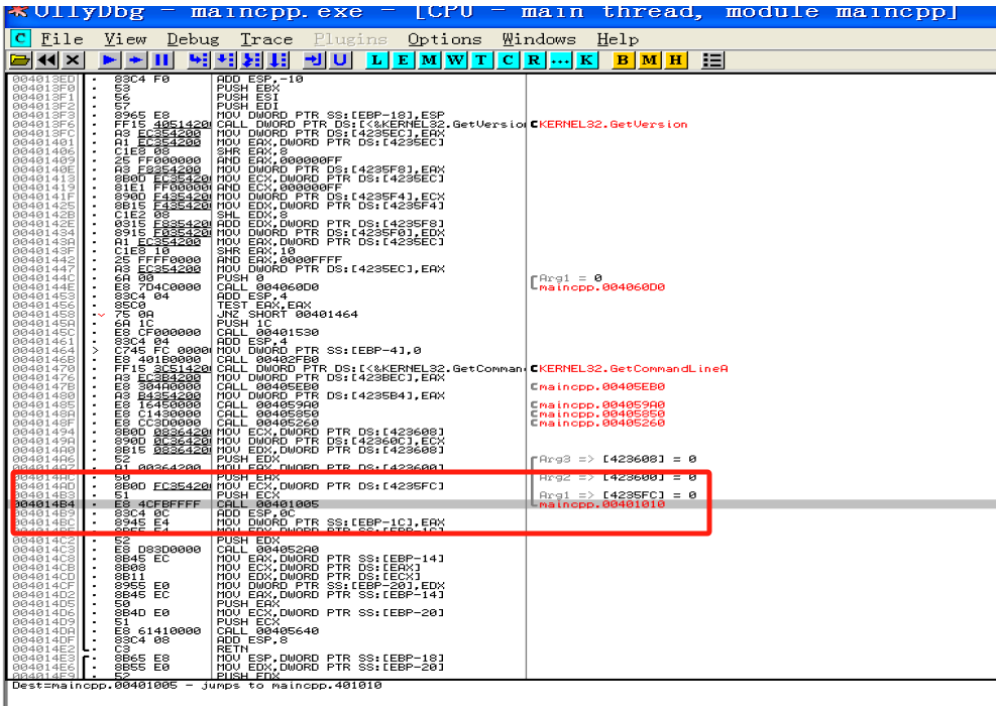

图 3.1: debug 模式下 ollydbg 视窗

然后我们点击 f7 进行单步步入调试，进入到 main 函数中，得到如下的汇编代码：

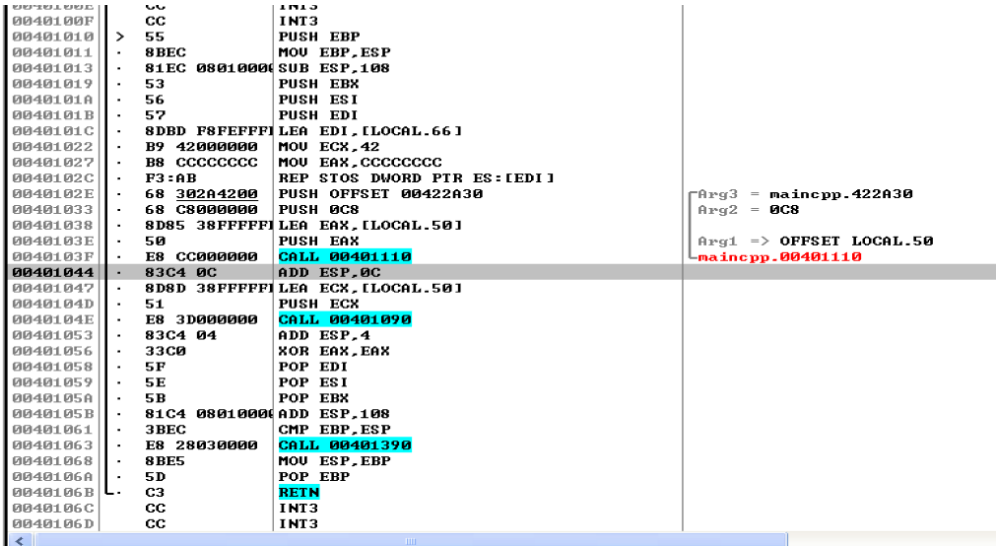

图 3.2: main 函数

首先先看前三句汇编代码，这三句分别是保存旧 ebp，赋新 ebp，抬高栈空间；再看后面三句代码，这三句的的作用是保存 ebx，esi，edi 寄存器中的值，可能我们在主函数中需要使用到这三个寄存器所以要把旧值保存起来。通过观察右下角的堆栈窗口以及右上角的寄存器窗口，我们发现，ebx，esi，edi 的值分别被压入到了栈顶。

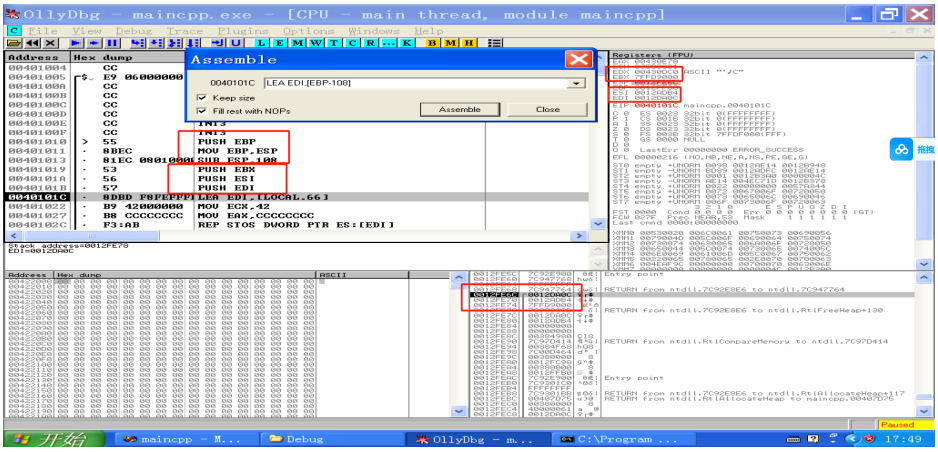

图 3.3: 栈结构初始化

接下来的四句意思是，对开辟出的栈空间全部赋值为 ccccccch，lea 取出的是栈顶的位置，所以我们依次根据循环从栈顶至栈底依次赋值，在这里可以点 f8 直接单步步过。可以看到开辟的空间全部被赋值成了 ccccccch。

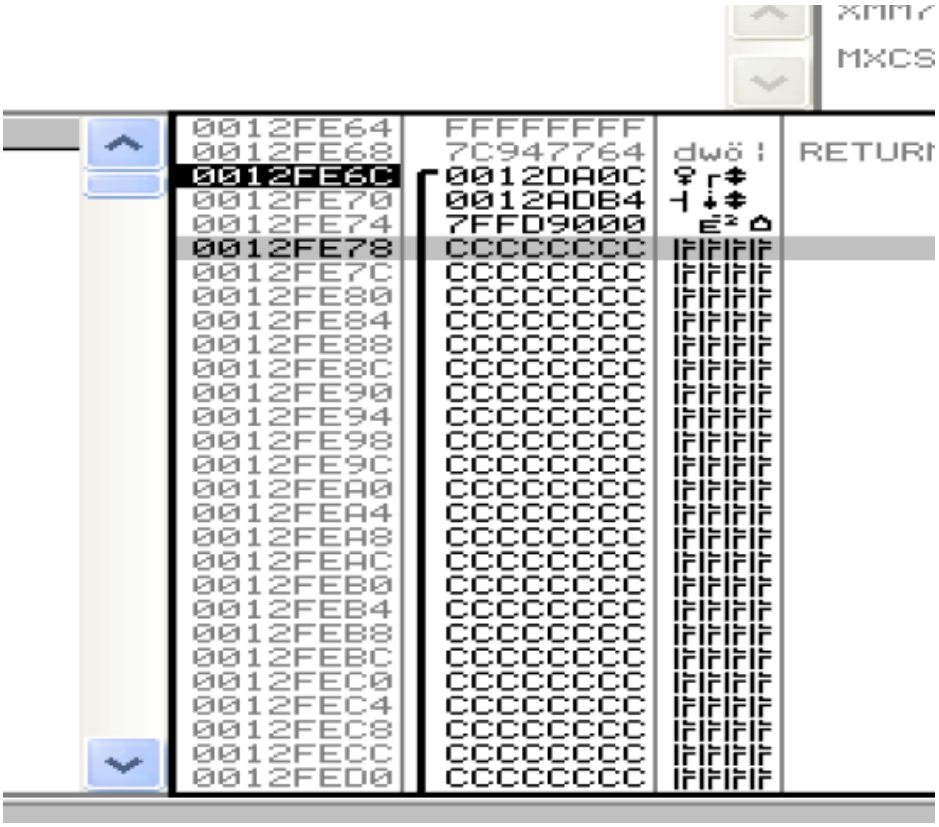

图 3.4: 初始化为 cccccccch

继续单步步过调试，再之后三个 push 为 fgets 函数传入参数，然后我们调用了 fgets 这个函数。ebp-0c8 就是写入 str 的起始地址。我们在窗口中输入 aaaa%x%x%x%x，观察右下角堆栈窗口，发现传入的字符串被保存在栈中。

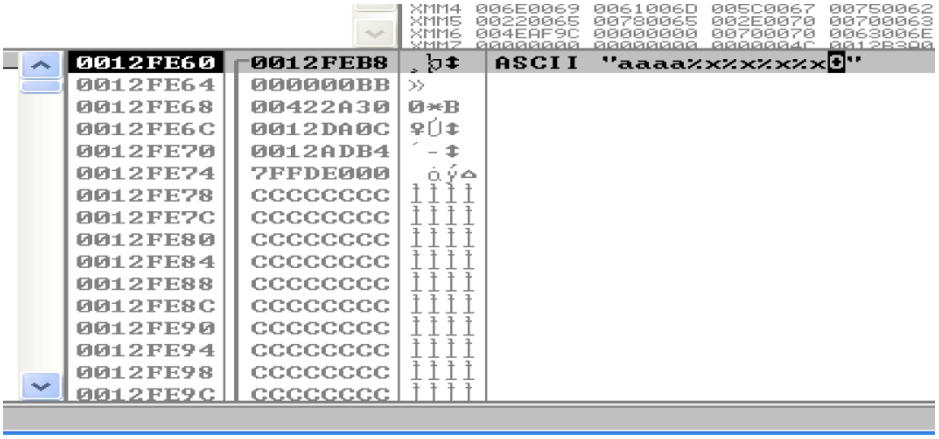

图 3.5: fgets 输入

继续运行，执行 print 函数，输出如下所示：

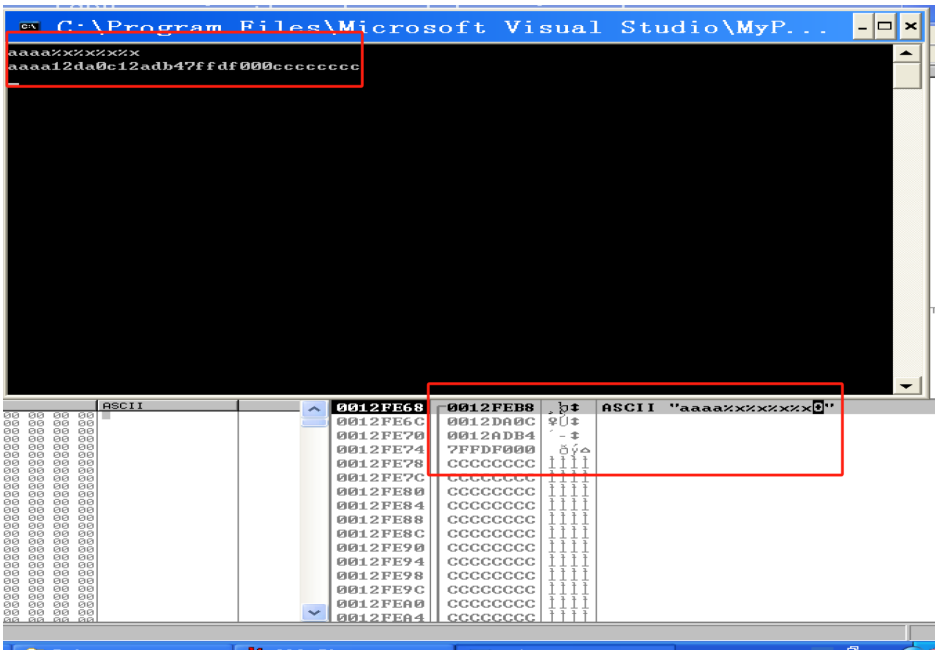

图 3.6: printf 输出

通过观察堆栈窗口，可以看到，首先程序正常输出了 4 个 a，但是当程序检测到后面的%x 时，由于没有参数,故会通过读取栈中之后的地址中的内容作为填充实现输出,于是就输出了 12da0c、12adb4、7ffdf000、cccccccc,作为 4 个%x 的格式化目标,所以程序总的输出为 aaaa12da0c12adb47ffdf000cccccccc。另外，我们发现一个问题，图 3.6 和图 3.5 中栈中内容不同，我们回看之前代码，发现在调用 print 之前，通过 add esp，0c 清除了 fgets 函数的相关参数，所以得到了如图 3.6 所示的栈结构。

3.2 release 模式

在 vc6 中切换为 release 模式，这里可以通过 build->set active configuration 进行切换。编译运行之后，在 release 文件夹下找到 exe 文件拖入 ollydbg 中，得到汇编代码。

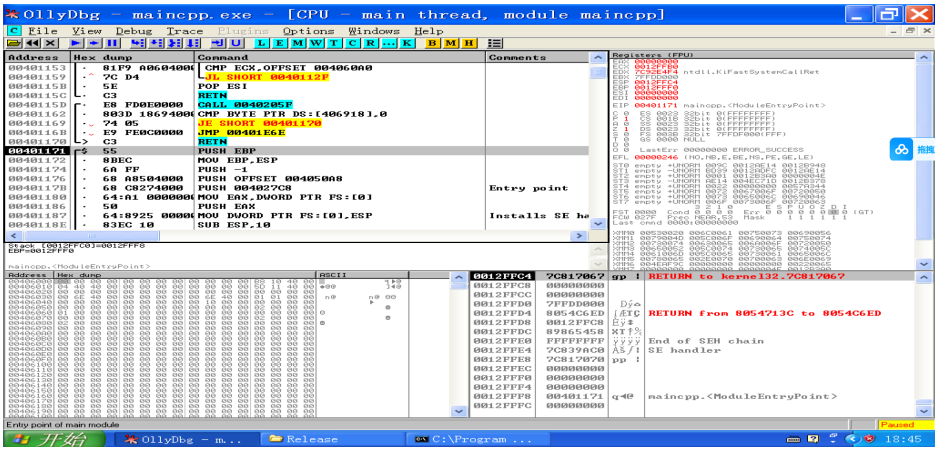

图 3.7: release 模式下 ollydbg 视窗

找到 main 函数入口位置，运行进入其中。

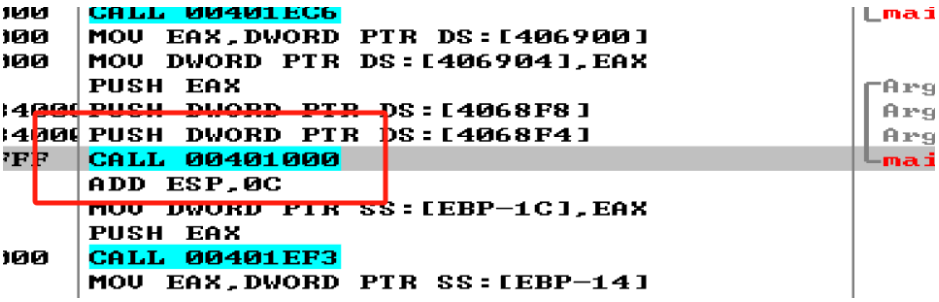

图 3.8: main 入口处

进入 main 函数后得到如下的汇编代码。不难发现，与 debug 模式相比，release 模式代码明显减少，可以看到，在 release 模式中，main 函数不进行严格的栈帧切换，也不会 push 保存旧寄存器的值，也不会对栈空间进行初始化操作。sub esp，0c8 为输入的字符串提供了 20 字节的空间，可以看到 release 模式下栈空间更小，然后 eax 存了 esp 的地址，也就是字符串的起始地址。

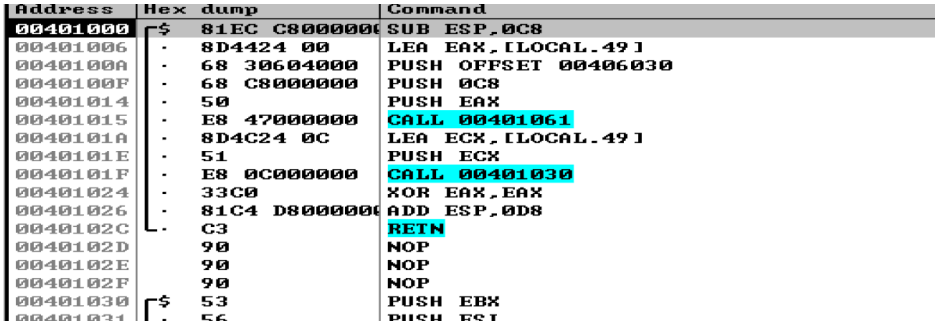

图 3.9: main 函数

接着传入三个参数，并调用 fgets 函数，我们在窗口中输入 aaaa%x%x%x%x，可以看到已被保存在栈中。

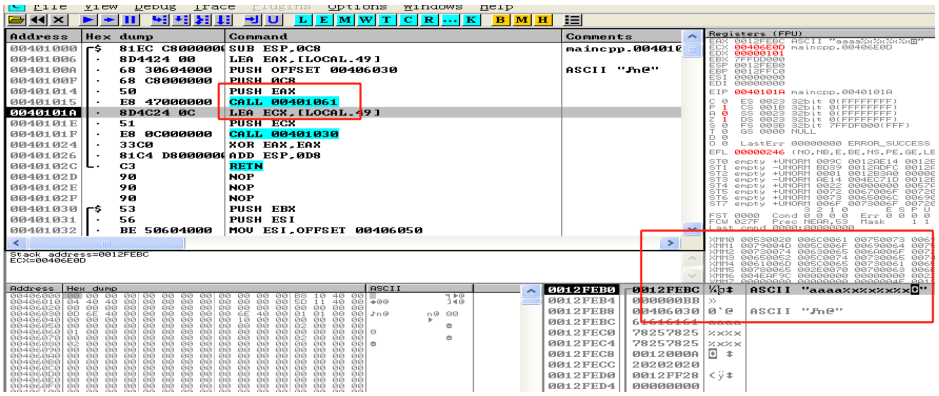

图 3.10: fgets 输入

随后调用 print 函数，输出信息如下所示：可以看到，先输出正常的 aaaa，4 个%x 再依次格式化读出后面的内容，分别是 12febc，bb，406030，61616161，所以最后的输出为 aaaa12febcbb40603061616161，并且 61616161 表示的就是 aaaa，这说明我们读到了字符串的起始处，也进一步说明了 release 模式下栈结构的紧凑性。

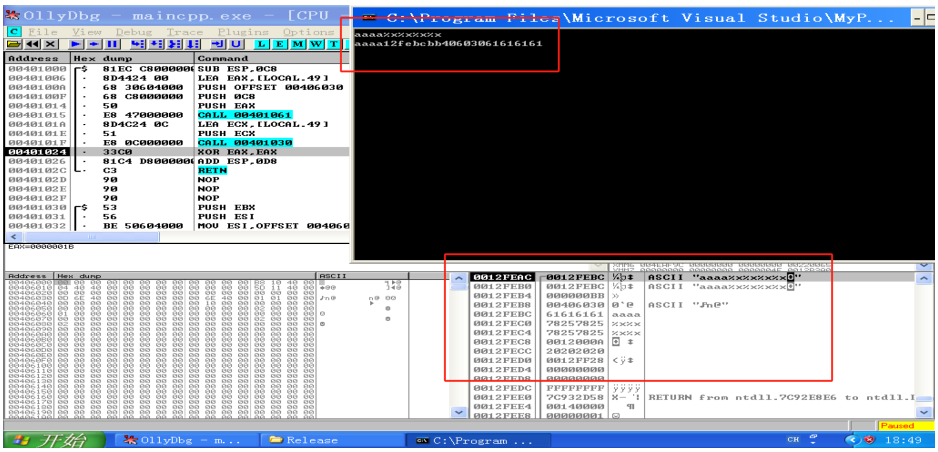

图 3.11: printf 输出

3.3 debug 模式与 release 模式的差异

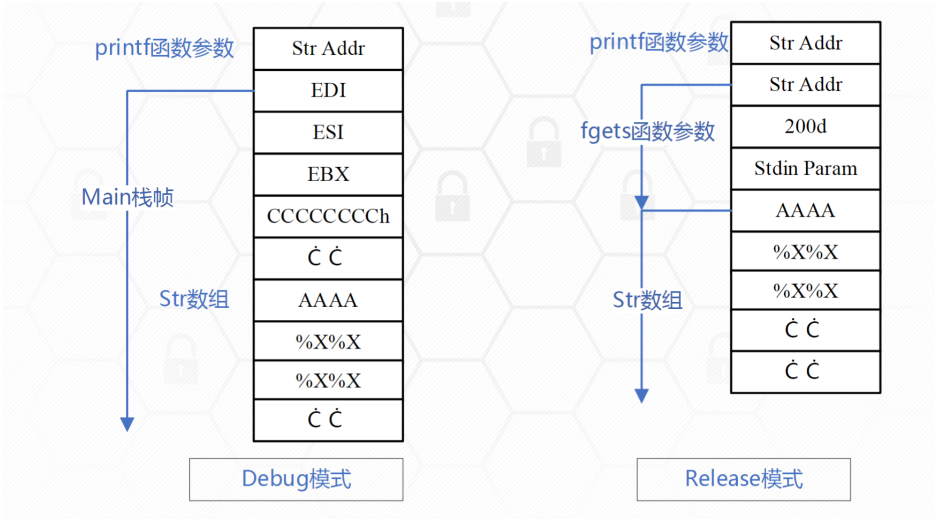

图 3.12: debug 模式与 release 模式栈帧结构示意图

1. 在 debug 模式下，在 main 函数中首先会进行栈帧切换，即保存旧的 ebp 并把新的 ebp 赋值为 esp，而且会通过 sub esp 分配很大的栈空间。并且会通过 push 保存原来寄存器的值，还会对栈空间进行初始化的操作。由于栈空间很大，由上图左侧 debug 模式示意图所示，如果要读取到字符串的起始地址处，需要更多的格式化字符。

2. 相应的，在 release 模式下，如上图右表所示，其空间很紧凑，所以只需要很少的格式化字符即可读取到字符串的起始地址处，另外，在 release 模式中，main 函数不进行严格的栈帧切换，也不会进行 push 保存旧寄存器的值，也不会对栈空间进行初始化操作。

无论是哪种模式，如果处理不当，都会造成向任意地址读写数据的 bug，在设计代码时一定要注意这个漏洞问题，降低代码的脆弱性。

4 心得体会

1. 了解了格式化字符溢出的原理，并且能够模拟漏洞实现的过程；

2. 感受到了漏洞溢出的危害，要是我们把%x 换成%s，再精心构造我们输入的字符串，就可以实现向任意地址读取数据，要是 sprintf 配合%n 使用，就会实现向任意地址写入数据，这种 bug 是不能接受的！所以我们在设计代码时，一定要考虑到代码的安全性，要对用户输入进行过滤，或者使用安全函数代替危险函数。