《软件安全》实验报告

姓名：王众 学号：2313211 班级：计算机科学卓越班

实验名称：

格式化字符串漏洞

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：

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

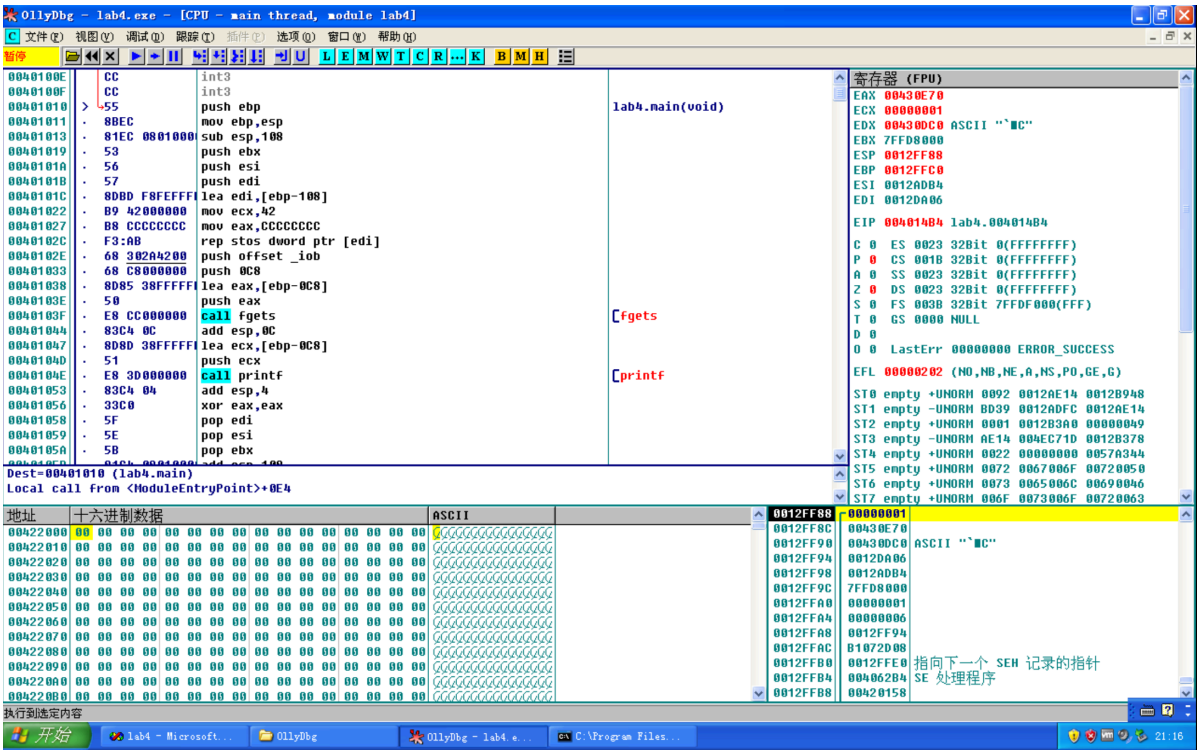
实验过程：

（1）Debug模式观察

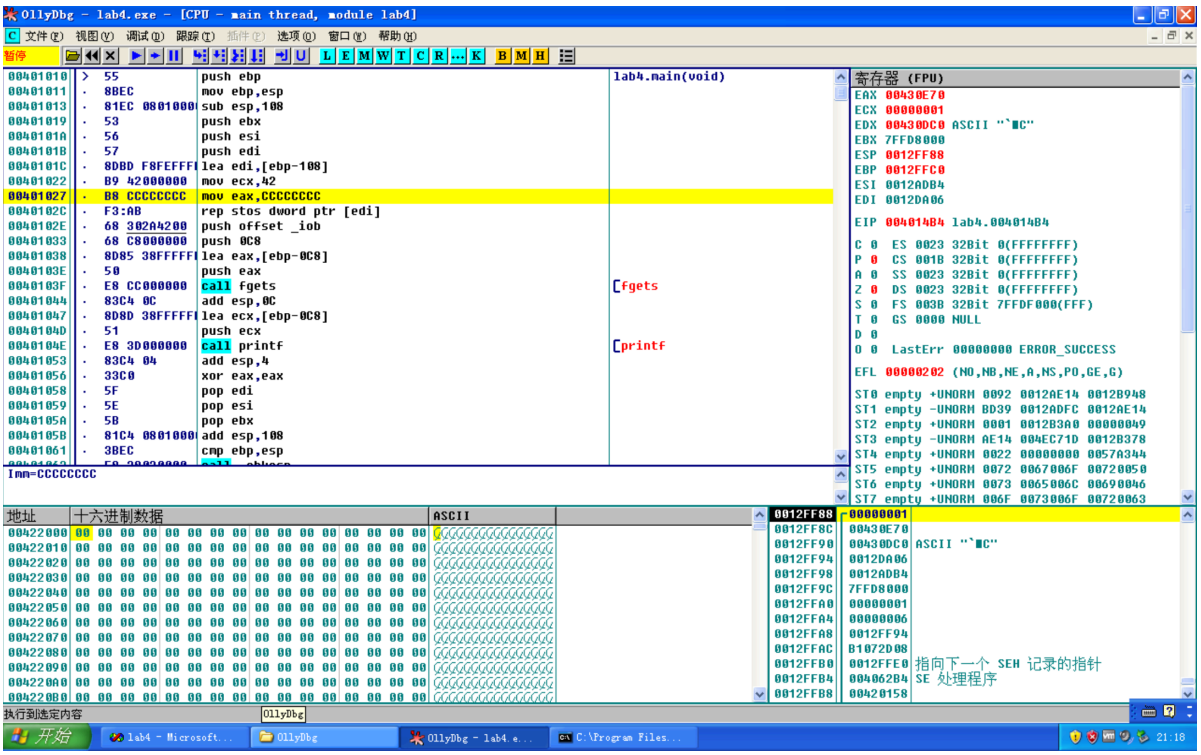
首先，我们在vc6中输入源代码，进行debug模式的调试，并且将exe文件导出，并导入到ollydbg中进行调试。

首先我们进入主界面，找到`call 0041005`这条指令（代表着主函数的开始），并进行跳转，得到汇编代码。

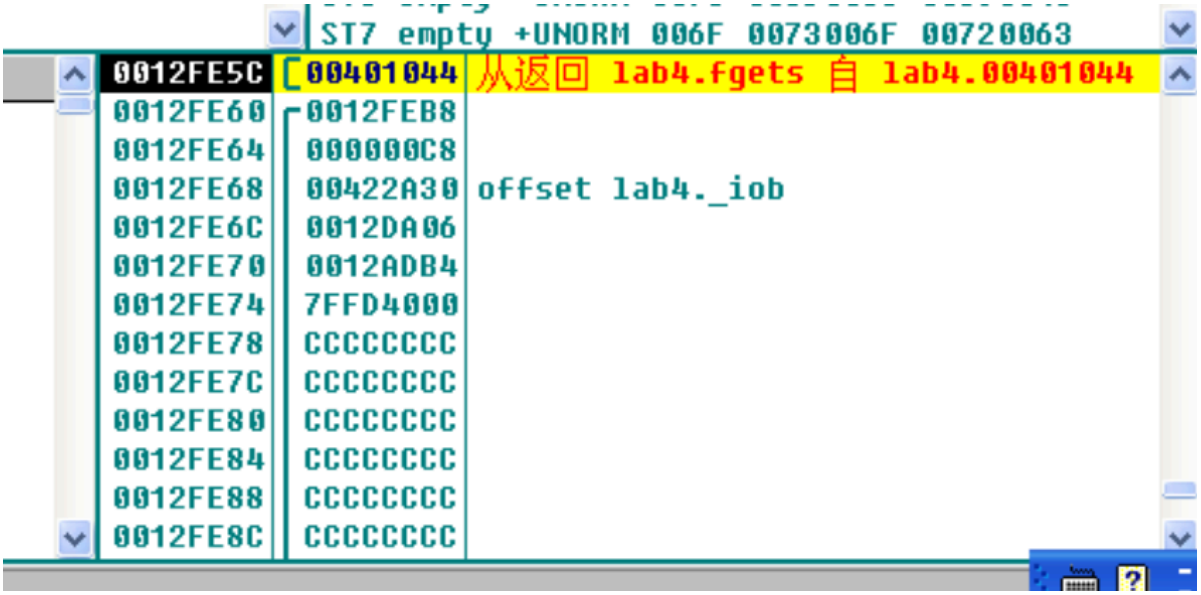
前六句的主要指令用于将`ebp`这个寄存器压入栈，完成栈底与栈顶的变换，并将108的空间压入栈中。再将三个寄存器压入栈中。



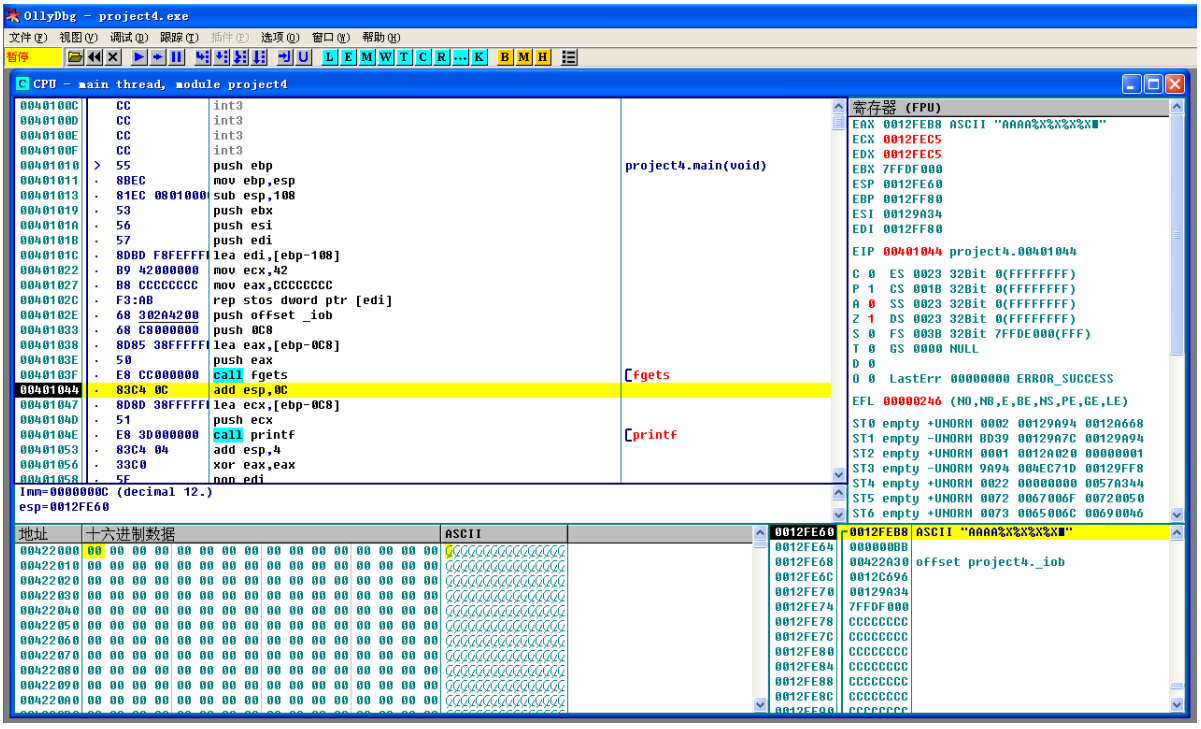
我们可以发现三个寄存器按顺序都被压入了栈中，分别是`EDI`,`ESI`,`EBX`。



然后对开辟出来的新空间进行赋值操作。全部赋值为`CCCCCCCCh`，经观察我们发现整个栈都被赋值成了`CCCCCCCCh`。

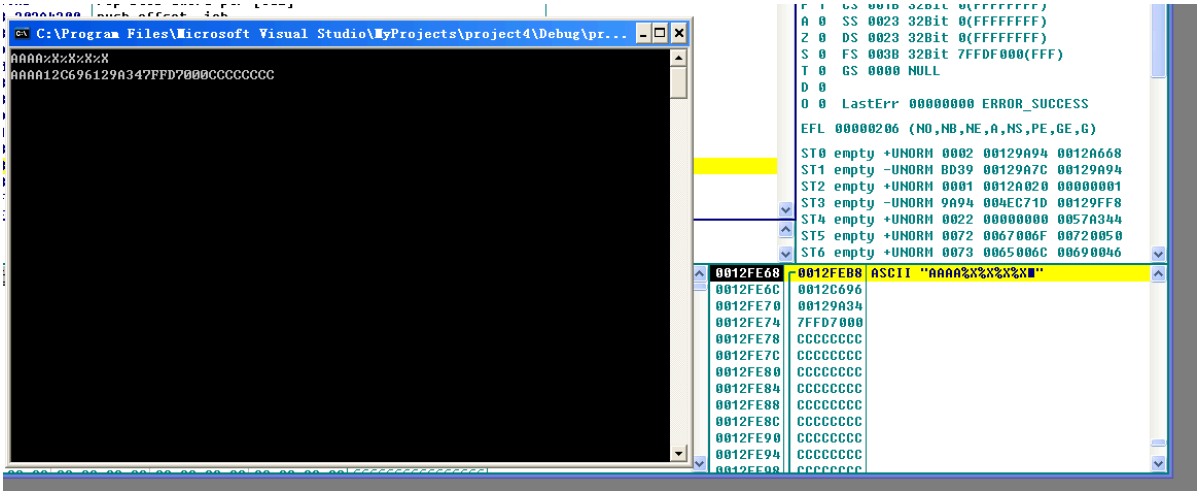


接下来，我们将三个参数入栈，调用函数 `fgets` 函数， `lea eax,[ebp-0C8]` 就是我们的 `str` 的起 始地址，在 `str` 的起始地址后压入 `eax` 寄存器。 分析完成后，我们进行内容的输入，输入 `AAAA%x%x%x%x` ，观察栈区，确认已经将其压入栈中。



我们可以发现字符串 `AAAA%X%X%X%X` 已经被输入到了变量中。我们打开命令行查看输出结果

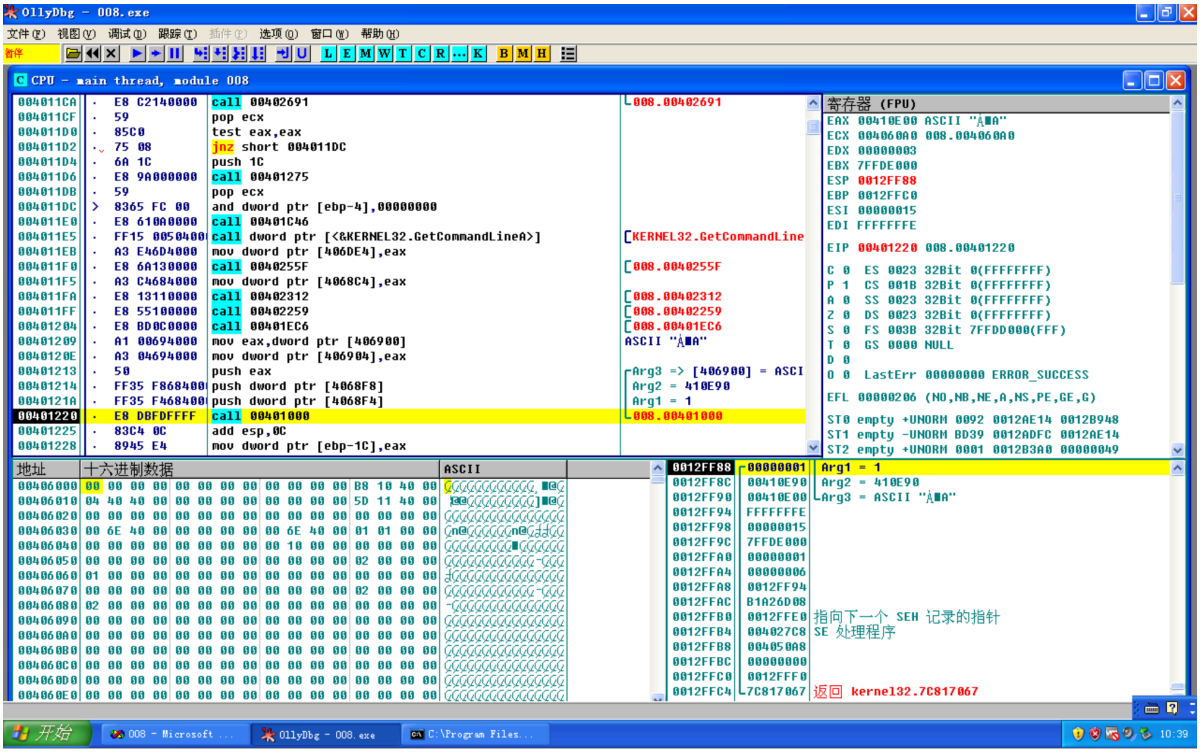
`AAAA12C696129A347FFD7000CCCCCCCC`



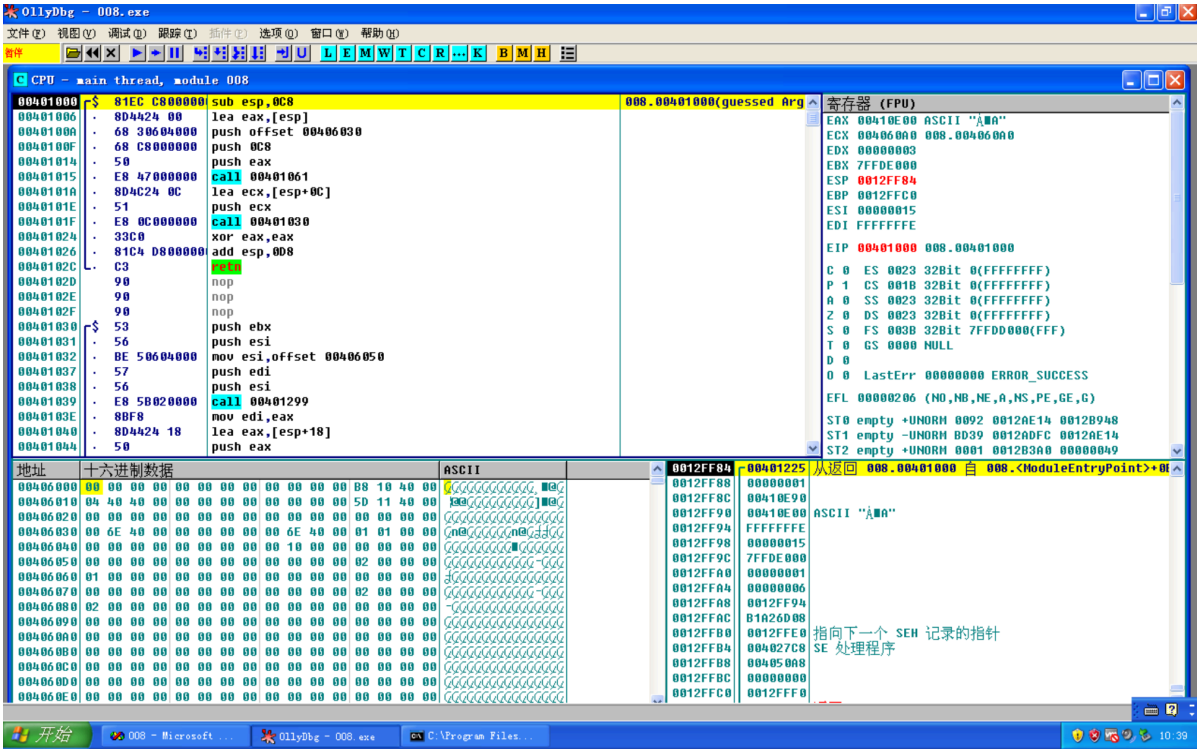
我们在输入`AAAA`时，程序正常输出四个`A`；但是当程序遇到后面的`%x`时， 自动读取后面的地址作为输出，因此四个`%x`对应的输出分别是， `00000000(0)`、`00000000(0)` 、 `7FFD8000` 、`cccccccc`。所以，对应的程序输出为： `AAAA007ffd8000cccccccc` 。这样我们就完成了 在debug模式下的调试运行。

（2）Release模式

接下来我们开始进行`release`模式下的分析。在CV6中将执行模式切换为`release`模式，也将生成的可执行模式导入到`ollydebug`模式中，同时通过三个`push`操作来定位到`main`函数中。

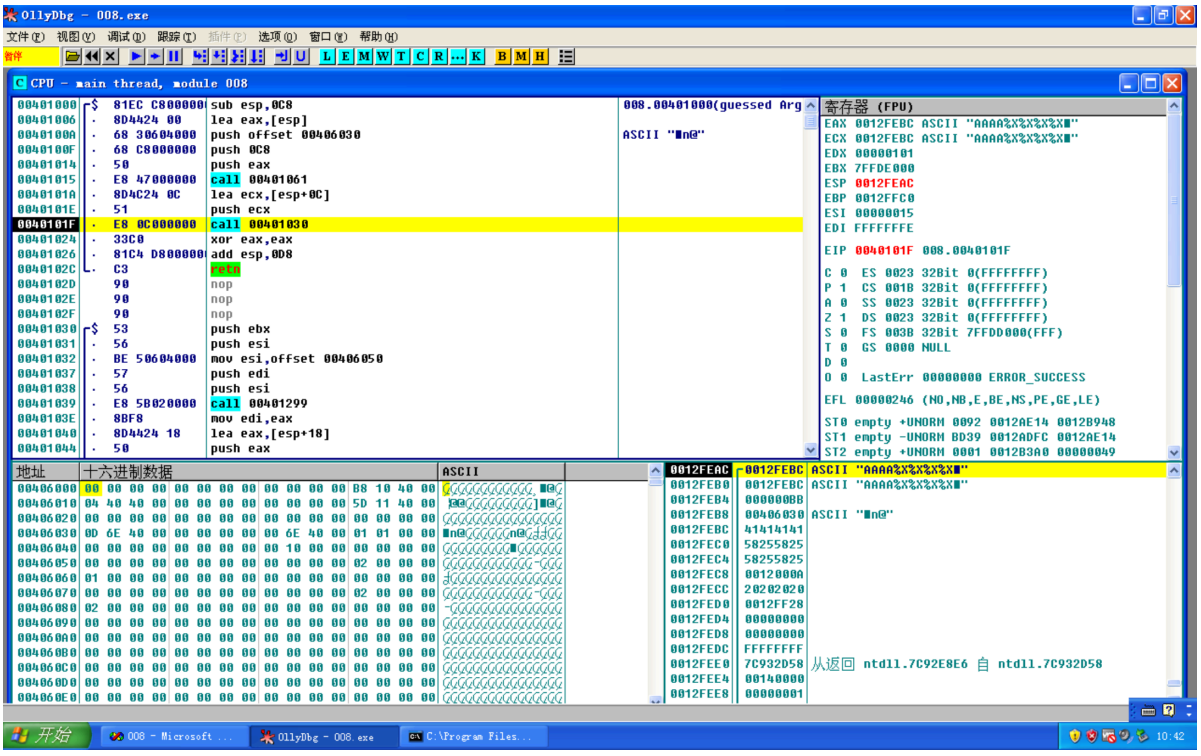


这时我们按`F7`进入主函数，发现在`release`模式下，栈的分配明显变得更加紧凑了，每个变量都变得靠在了一起，接下来我们对代码进行详细的分析。

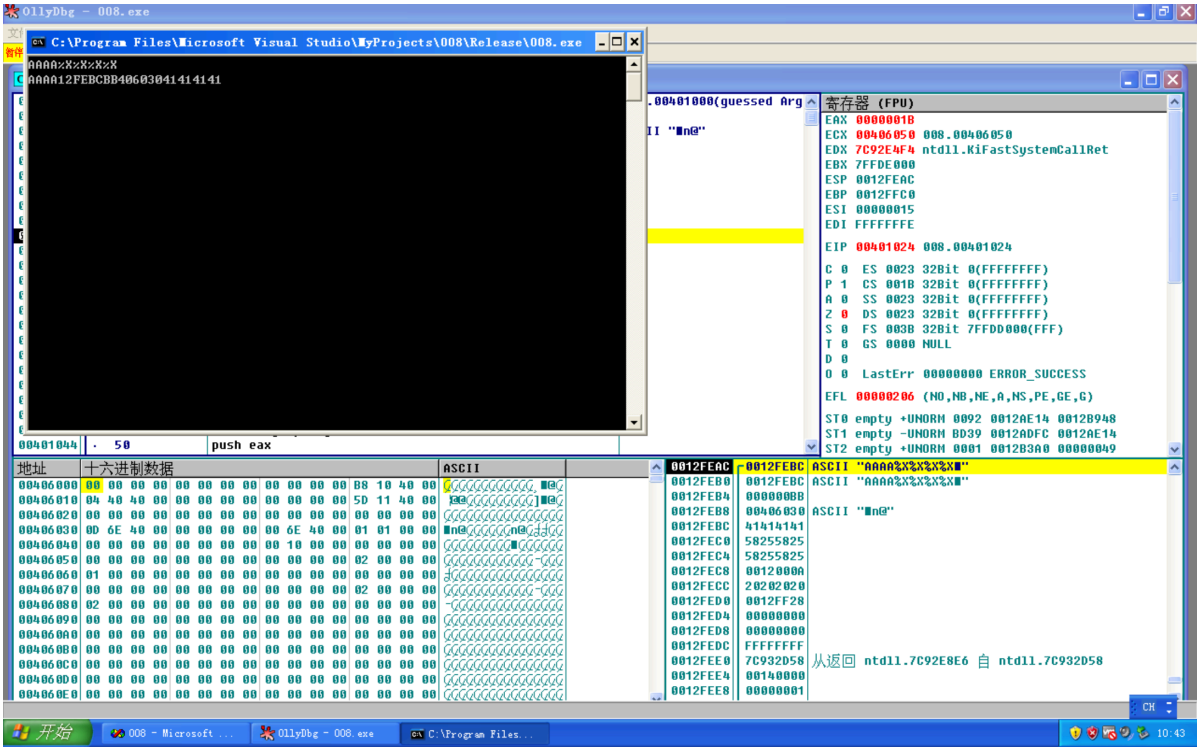


`sub esp 0c8` 是直接为 `str` 字符串分配20字节的内存空间，而后通过三个 `push` 操作将 `fgets` 函数的三个参数进行入栈操作，为后续的调用做准备。这里我们可以看出在此模式下与debug模式的一个区别是无过多的栈内存空间分配，无寄存器的旧制保存。

然后，我们执行 `fgets` 函数，输入与debug模式相同的 `AAAA5X%X%X%X` 字符串，对比之后发现，`fgets` 函数的地址与 `str` 的地址是紧挨着的。这进一步说明在该模式下，栈分配与debug模式的不同。

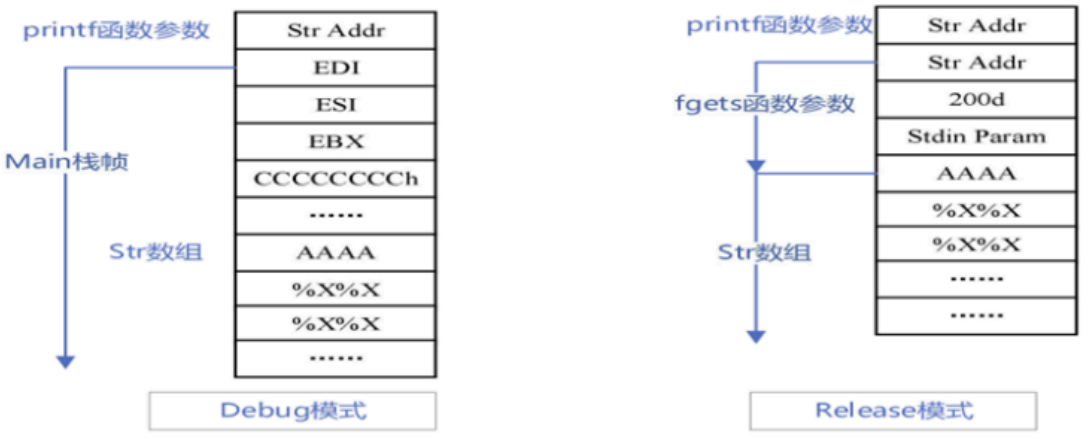


接着我们执行 `print` 函数，第一行为 `print` 的参数，即 `str` 的地址 `0012FEBC`.然后执行是先输出 `AAAA`,接着四个 `%X` 格式化操作符读取后四行的内容作为四个参数进行输出，输出结果为 `AAAA12FEBCBB40603041414141`.



在程序函数调用完之后，还有两句汇编语言代码，含义是，将 `eax` 寄存器置零，然后将 `esp` 寄存器 的位置恢复到原始的位置，完成栈帧改的恢复。

（3）Debug模式与Release模式的差异



对比模式图我们可以发现两个不同点：

Debug模式 `main` 函数一开始 `sub esp` 会分配更大的栈空间， `char str[200]` 是从靠近 `EBP` 的地 址分配空间，因此在`DEBUG`模式下如果要读到 `str`的地址，需要很多的格式化字符。

Release模式下， `main` 函数不执行严格的栈帧转换(即 push ebp,mov ebp,esp)，也不对栈空间进行统一初始化(即 `rep stos` 指令)，也不通过`push` 保存寄存器原来的值。会在程序的最后处完成栈帧的恢复。

心得体会：

通过本次实验，我学会了对格式化字符溢出的使用，对这个漏洞有了更深的了解。

首先我通过理论课的学习，发现一些看似没有问题的程序中竟然存在这么多漏洞，比如说SQL注入就是 一个很严重的漏洞，轻则进入系统，重则更改账户中的内容，而破解方式也很简单易懂，这说明了在一些常用的代码中，还是存在不少问题，需要我们去发现与解决。在本次实验中我通过对程序 的两种调试方法，了解了两种模式下的具体栈差异，了解了具体的利用格式化字符去攻击代码的方式。

不仅仅是 `%x` 符号可以攻击程序，理论课上讲的许多 `%` 型字符都可以造成一些的攻击性，这告诉我们，在编写代码的时候，不能只考虑代码段正确性，还必须要考虑代码的安全性，目前我了解到的 就是关于栈溢出与堆溢出两种常见的攻击方式，我们需要防范，比如说使用一些输入限制，或者在 程序堆或者栈异常时直接中断调试，这样就可以避免一些问题。