Homework 2

515021910384 潘亦晟

noise.c

```
#include <stdio.h>

void f1()
{
    int a=1, b=2, c=3;
};

void f2()
{
    int a, b, c;
    printf("%d, %d, %d\n", a, b, c);
};

int main()
{
    f1();
    f2();
};
```

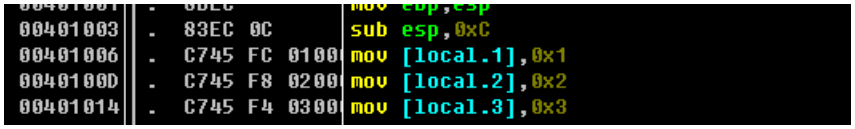
在 c++中，函数内部定义的局部变量（local variable）将会在函数调用结束后被销毁，它们对于函数是“局部的”，仅在函数的作用域内可见。

按照该特性，f1 中的三个局部变量：a=1，b=2，c=3 将会在 f1 调用完成后销毁，而 f2 定义了三个局部变量，而未对其初始化，按照正常理解，printf 函数将会输出三个随机的数，而运行可执行文件 exe 发现输出 1,2,3.

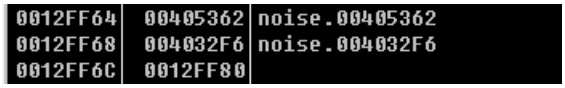
Step 1：寻找 main 函数调用 f1，f2 地址

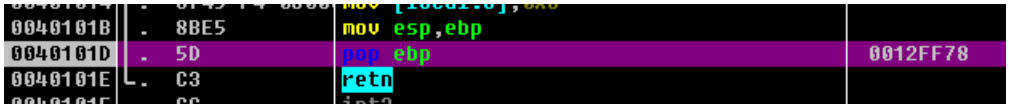
Step 2：调用 f1

在 00401053 处设置断点，开始进行 f1 的调用，此时先将程序中下一条指令（f2 的调用）的地址 00401058 压栈，然后将当前栈底地址（ebp 所指向的地址）0012FF78 压栈;



我们观察到接下来执行的反汇编代码为 sub esp，0xC。该指令表示在栈中开辟 12bit 的空间来存储三个局部变量。局部变量初始化对于栈的影响如下：



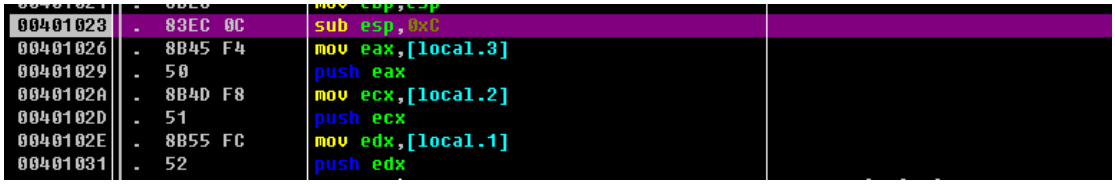


接下来观察到执行反汇编代码 mov esp，ebp； pop ebp；retn；
F1 的调用结束，eip 此时指向 00401058，同时观察栈中内容未发生改变。

Step 3： 调用 f2

观察反汇编代码，程序执行 call noise.00401020（f2 的调用）。

与 f1 类似，f2 同样没有参数传递，因此先将下一条指令的地址 0040105D 压栈，然后再将当前栈底地址（ebp 所指向的地址）0012FF78 压栈；



我们观察到接下来程序同样在栈中开辟 12bit 的空间（此空间中存储的数据与 f1 时相同），然后分别将 c，b，a 的值压入栈中。

最后再调用 printf 函数。

结论：
C++中局部变量的销毁在真正的机器码中其实只是对 ebp 指针的改变，栈中的历史数据其实保留下来。如果在函数体中未对局部变量进行初始化，可能会出现一些出人意料的错误。