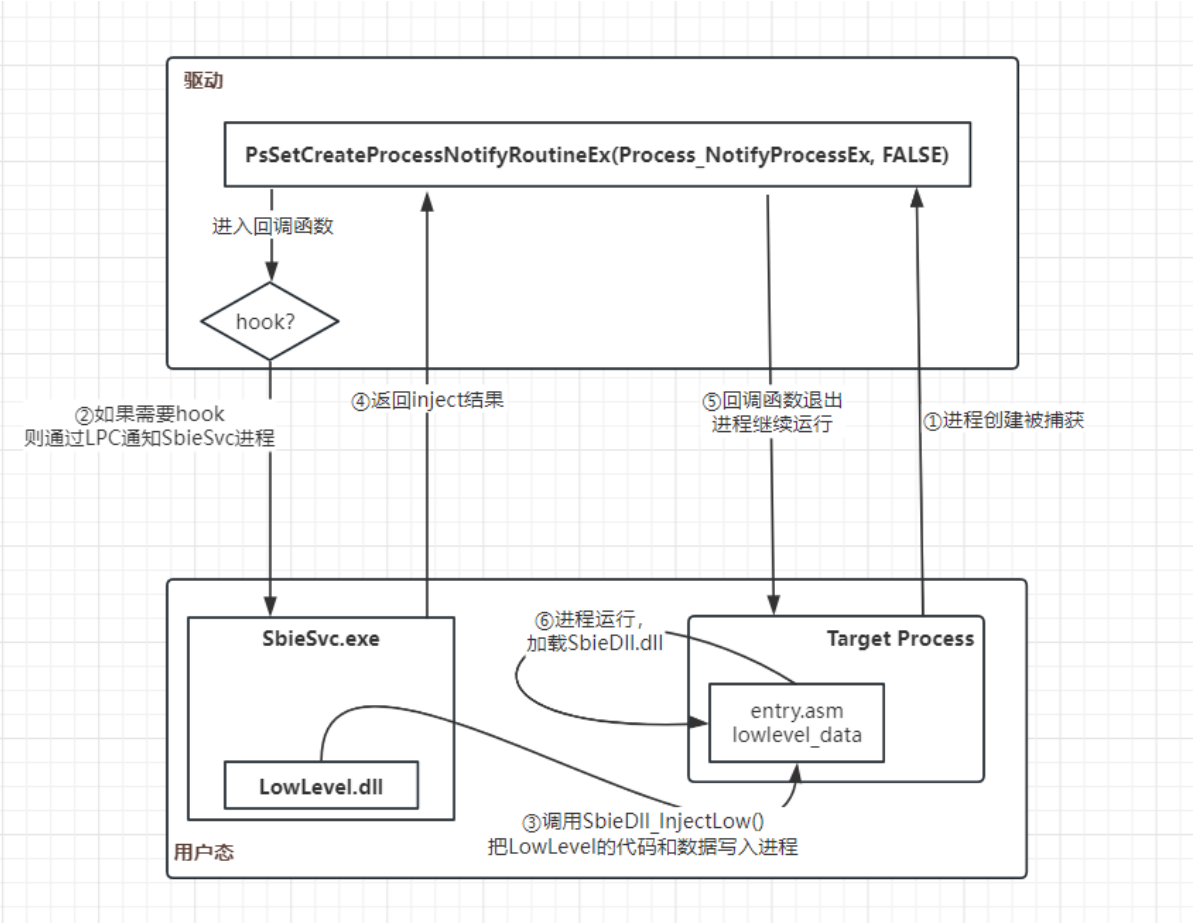
sandboxie进程注入实现分析



内核

sandbox内核也是通过 `PsSetCreateProcessNotifyRoutineEx` api来监控进程启动， 在这个API的回调函数中做如下操作：

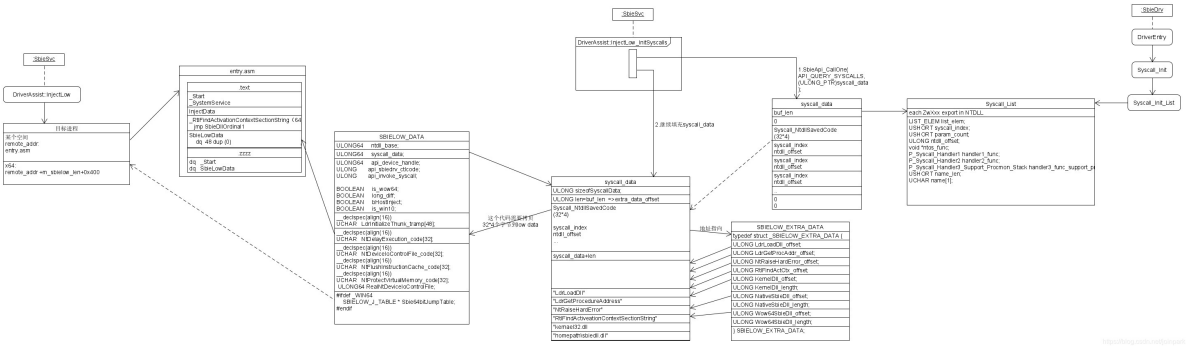
1. 内核获得进程信息后会判断是否需要hook。需要hook就通过LPC 通知用户态的SbieSvc进程。
2. 此时驱动循环等待一段时间， 用以接收用户态返回的注入结果。
3. 等待超时或收到用户态的结果后，才会退出回调函数， 进程继续运行。

用户态

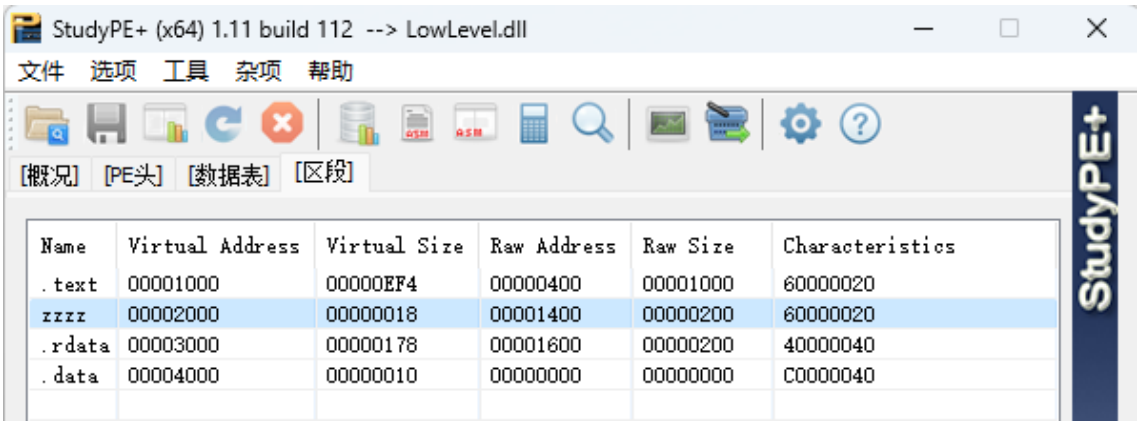
step1

用户态进程SbieSvc收到通知， 开始执行`LowLevel.dll`库中的一个函数 `SbieDll_InjectLow()`，通过 `NtAllocateVirtualMemory` 和 `WriteProcessMemory` 远程给目标进程写入一些信息。

写入内容大致如下：



- 注入`LowLevel.dll`的 .text 和 zzzz 区段，*`LowLevel.dll`是由 c 和 汇编 实现的一个动态库工程，zzzz区段中应该是 .text区段中 一些关键代码和数据的入口信息*



Name	Virtual Address	Virtual Size	Raw Address	Raw Size	Characteristics
.text	00001000	00000EF4	00000400	00001000	60000020
zzzz	00002000	00000018	00001400	00000200	60000020
.rdata	00003000	00000178	00001600	00000200	40000040
.data	00004000	00000010	00000000	00000000	C0000040

- hook了`ntdll!LdrInitializeThunk`函数， 让这个函数跳转到注入的代码段中。
- 由驱动上报的一些数据 sys_data 也写入了目标进程中

远程注入实现demo：

```
//
// libMessageBoxDll.dll 导出两个函数， myMessageBox 和 myMessageBox2
// 通过这种方式可以hook一个进程中的myMessageBox到myMessageBox2， （也可以是进程中任意的可执行代码的位置）
//
#include "stdio.h"
#include "Sbie.h"


int main() {
    MyMessageBox(nullptr, "test", "test", MB_OK);
    Inject_LowLevel(29236);
    return 0;
}

bool Inject_LowLevel(long ProcessId) {

    const ULONG _DesiredAccess =
            PROCESS_DUP_HANDLE | PROCESS_TERMINATE | PROCESS_SUSPEND_RESUME
            | PROCESS_SET_INFORMATION | PROCESS_QUERY_INFORMATION
            | PROCESS_VM_OPERATION | PROCESS_VM_READ | PROCESS_VM_WRITE;

    HANDLE hProcess = NULL;
```

```
    hProcess = OpenProcess(_DesiredAccess, FALSE, ProcessId);
    if (hProcess == NULL){
        printf("open process failed");
        return false;
    }

    // inject code to process
    char* myCode = "";
    void* remoteCode =  InjectCode(hProcess, strlen(myCode), myCode);

    // Hook myMessageBox
    HMODULE myMsgDll = GetModuleHandleA("libMessageBoxDll.dll");
    if (myMsgDll == NULL) {
        printf("not find dll, %x", GetLastError());
        return FALSE;
    }
    LONG_PTR myMessageBoxFunc = GetFuncPtr(myMsgDll, "MyMessageBox");
    if (!myMessageBoxFunc) {
        printf("not find MyMessageBox in dll");
        return false;
    }
    LONG_PTR myMessageBox2Func = GetFuncPtr(myMsgDll, "MyMessageBox2");
    if (!myMessageBox2Func) {
        printf("not find MyMessageBox2 in dll");
        return false;
    }



    if (!InjectWriteJump(hProcess, (UCHAR *) myMessageBoxFunc, (UCHAR
*)myMessageBox2Func)) {
        printf("injectWriteJump failed");
    }

    return true;
}

LONG_PTR GetFuncPtr(HMODULE dllModule, char* funcName) {
    ULONG_PTR funcPtr = (ULONG_PTR) GetProcAddress(dllModule, funcName);
    if (!funcPtr) {
        return 0;
    }
    unsigned char* code = (unsigned char *)funcPtr;
    if (*(ULONG *)&code[0] == 0x24048b48 && code[0xa] == 0x48) {
        funcPtr += 0xa;
    }
    printf("%s %p\n", funcName, funcPtr);
    return funcPtr;
}


void* InjectCode(HANDLE hProcess, SIZE_T size, const void* pCode) {

    SIZE_T outSize;

    void* remoteAddr = InjectAllocVirMem(hProcess, size, true);
```

```
    bool bRet = WriteProcessMemory(hProcess, remoteAddr, pCode, size, &outSize);

    if (bRet) {
        return remoteAddr;
    } else {
        return NULL;
    }

}


void* InjectAllocVirMem(HANDLE hProcess, SIZE_T size, BOOLEAN executable) {

    void* remote_addr = NULL;

    //NtAllocateVirtualMemory(hProcess, &remote_addr, i, &region_size, MEM_COMMIT
| MEM_RESERVE, executable ? PAGE_EXECUTE_READWRITE : PAGE_READWRITE);

    remote_addr = VirtualAllocEx(hProcess, NULL, size, MEM_COMMIT | MEM_RESERVE,
executable ? PAGE_EXECUTE_READWRITE : PAGE_READWRITE);

    return remote_addr;
}



bool InjectWriteJump(HANDLE hProcess, void* targetPtr, void* injectPtr) {

    ULONG myProtect;
    UCHAR jumpCode[20];

    // >= win10 可用
    jumpCode[0] = 0X48;
    jumpCode[1] = 0xe9;
    *(ULONG*)(jumpCode + 2) = (ULONG)(ULONG_PTR)injectPtr - (ULONG)
(ULONG_PTR)targetPtr - 6;

    if (!VirtualProtectEx(hProcess, targetPtr, 6, PAGE_READWRITE, &myProtect)) {
        printf("change failed \n");
        return false;
    }
    size_t outLen;
    if (!WriteProcessMemory(hProcess, targetPtr, jumpCode, 6, &outLen)){
        printf("write failed \n");
        return false;
    }
    if (!VirtualProtectEx(hProcess, targetPtr, 6, myProtect, &myProtect)) {
        printf("change2 failed \n");
        return false;
    }

    return true;
}
```

step2

通知驱动注入结果， 驱动退出`PsSetCreateProcessNotifyRoutineEx`的回调函数， 进程开始执行注入代码。注入的代码会判断进程的类型， 最终将SbieDll.dll加载入进程

这部分代码 汇编是代码的入口， 调用c程序的`EntrypointC()`方法， 有点看不懂。