2021/01/21_32位汇编_第9课_利用异常对代码加密(反调试)、结构化异常处理(SEH)

笔记本:	32位汇编
创建时间:	2021/1/21 星期四 10:45
作者:	ileemi

程序被调试状态，程序中的异常会被调试器接收，软件接收不到异常处理。

反调试

产生异常的代码下面的代码可以正常运行，当接受到异常时，有程序自己处理异常，可以修改寄存器eip的值（跳过异常代码），继续执行程序（程序可以运行，但是不能进行正常调试）。

> 解决上面不能正常调试的问题，可以将出现异常的汇编代码进行nop处理，后面的代码就可以正常调试。

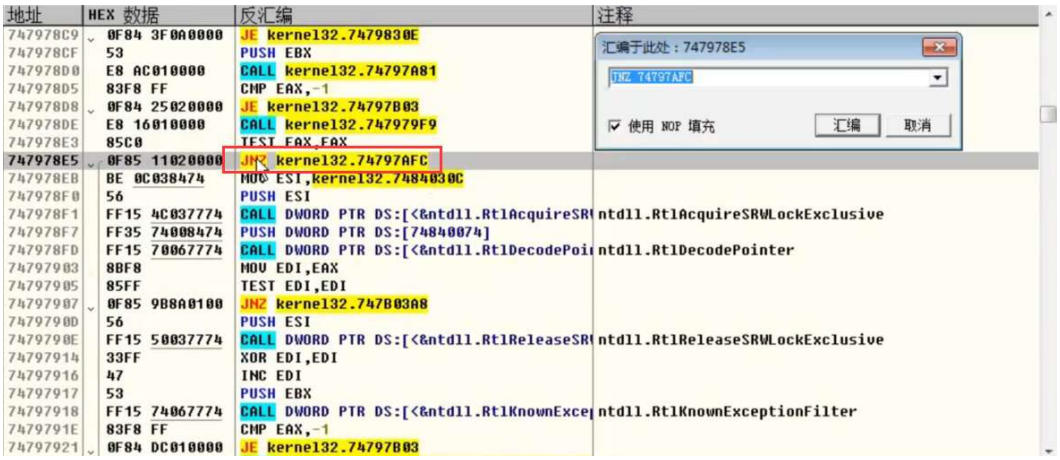
代码加密

对产生异常后，程序后面需要执行的代码进行加密（逐字节加密，注意修改内存保护属性（修改代码区的数据）），当异常来的时候，程序自己处理异常时，对异常后面加密的代码进行解密。防止程序被调试时，异常代码被 "nop" 掉后（加密后，即便是 "nop"，后面的反汇编代码也是加密后的）。

> 这里的代码加密如果是在触发异常后进行的加密，加密后的代码不会写入到可执行文件中，可以自己对可执行文件进行修改或者在触发异常前对代码进行加密。

对于使用 ollydbg和Windbg 来说，调试程序时，异常不会来（接受不到），程序的代码就不会被解密。当使用 x64dbg 调用异常回调函数后（调试器会接受到筛选器异常 ），加密代码会自行解密，

通过技术手段也可以对加密的程序进行调试，程序的异常让调试器一定调到：
通过调试程序，查看程序是否进行了筛选器异常注册，如果注册了，就可以到 "kernerl32.dll" 模块中对 "UnhandledExceptionFilter" API的内部进行分析（修改条件调转），这样就会是异常回调函数一定运行。



调试器的单步原理

> 也是产生一个异常

int3 异常代码为 0x80000003，而调试器下断点也是 0x80000003，也就是说下一个断点时，调试器会优先接收。

单步步入调试器也会接收这个异常，在程序中用代码模拟一个单步异常，当调试器接收这个异常后，异常程序接收不到，程序加密的代码就不会进行解密。

单步会修改标志寄存器（TF），在异常回调函数中修改标志寄存器（TF）的数值以及寄存器eip的值，让调试器接收单步异常（0x80000004），这样即便是 "处理异常继续执行" 程序也不能正常运行，加密的代码没有被解密。

> 逆向对抗中常用的 "手法"，学习异常不但可以处理程序中的错误同时也可以用来加密程序中的代码，进行程序反调试，懂原理的话也可以做到反反调试。

筛选器异常的缺点

使用汇编语言编写的程序，出现异常后，异常地址可以进行快速的定位，但是当程序是高级语言编写的话，出现异常后，对异常的地址不发进行判断（不确定是哪个函数）。保存的日志其中有些内容有些开发者不了解。

快速定位异常位置

- 异常处理可以以函数为单位，因为异常的出现不是预知的，假设可以为程序中每个函数做一个异常的回调函数，到有异常出现的时候就可以快速定位到出问题的函数代码位置（筛选器异常在程序中只能注册一次，它是全局的）。
- 当有多个线程调用同一个函数时，当有一个线程内部出现异常的时候，此处出现异常的位置就不确定，可以使用**线程函数为单位，每个线程对应一个回调函数。**

结构化异常处理（SEH）

> Windows提供了这种结构化异常处理的功能（**线程函数为单位**），可以为指定线程的函数注册异常。作用域为局部，用来检测进程中某特定线程函数内部是否发生异常。

在C语言中可以使用 "**__try{} __except{}**"。微软提供了使用方法，并没有公开其实现的原理。使用 "**__try{} __except{}**" 关键字操作系统就会对其进行注册，将地址保存到一个数据结构中（方便查询）。

搞清楚结构化异常实现的原理的好处：

- 可以自己做异常处理
- 可以提高逆向代码时，代码的阅读
- 可以用来做反调试
- 可以用来做代码加密

线程函数为单位，每个线程就需要给一个地址（使用异常的前提），微软使用段寄存器FS来管理地址问题，例如：

- A进程，地址通过FS表示为：lea eax, fs:[偏移] ; 00401000
- B进程，地址通过FS表示为：lea eax, fs:[100] ; 00402100

不同的线程（以线程为单位）执行同样一段代码，其地址不一样（内存空间不一样），做到每个线程一个表，由操作系统和CPU硬件进行配合。

在Win32汇编中编译器默认把段寄存器 FS 假设为一个结构体，在使用段寄存器 FS 时，就需要将其设置为通用寄存器（假设一下，例如：assume fs:nothing）

相关结构体

线程环境块 _TEB

> FS:[0] 指向线程环境块（_TEB）的首地址（指向一个结构体，每个结构体都和线程相关，每个线程拿到的东西都属于自己）。

_TEB 结构设计可以通过 Windbg 分析。通过 Windbg 打开一个程序，输入 **dt _teb** 进行查看。"dt" 命令可以符号信息来查看结构体的成员和偏移。

_TEB在 "ntdll32" 模块中

```
struct _TEB {   // FS段指向_TEB
    struct _NT_TIB {// +0
        // ExceptionList -- 异常处理链表
        // StackBase
        // ...
    }
    // ...
};
```

线程信息块 _NT_TIB

> 结构体:

```
typedef struct _NT_TIB {
    // 保存异常回调函数的链表
    struct _EXCEPTION_REGISTRATION_RECORD *ExceptionList; // +0
    PVOID StackBase;
    PVOID StackLimit;
    PVOID SubSystemTib;
union {
    PVOID FiberData;
    DWORD Version;
};
    PVOID ArbitraryUserPointer;
    struct _NT_TIB *Self;
} NT_TIB;
```

> FS:[0] 总是指向TEB，即总是指向当前线程的TIB，其中0偏移的指向线程的异常链表，ExceptionList 是指向保存异常回调函数链表（_EXCEPTION_REGISTRATION_RECORD结构）的一个指针。

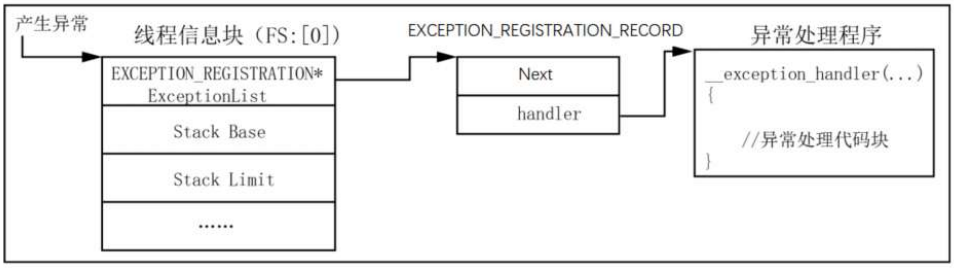
_EXCEPTION_REGISTRATION_RECORD

> 结构体:

```
typedef struct _EXCEPTION_REGISTRATION_RECORD {
    //指向前一个 EXCEPTION_REGISTRATION 的指针
    struct _EXCEPTION_REGISTRATION_RECORD *NEXT;
    PEXCEPTION_ROUTINE Handler; // 保存当前异常处理回调函数的地址
} EXCEPTION_REGISTRATION_RECORD;
```

> 每次有新的 "结构化异常处理" 就往链表头部插入异常回调函数的地址

异常回调函数的调用过程



注册 SEH

晚注册的异常回调函数的地址应该往链表头部插入，当异常时，优先从链表头部往尾部遍历，根据异常回调函数的返回值进行判断这个异常由哪个回调函数进行处理，没有回调函数处理，异常交给操作系统进行处理。

可以尝试将链表结构体放到堆栈中，当一个函数执行完毕后，方便将遍历头改为下一个遍历项。

注销 SEH

> 每个函数处理自己的异常，可以在函数结束时，修改保存异常回调函数的链表头，将注册前的链表头地址替换成注册后的链表头地址（将注册后的链表头项删除）。异常来的时候，这个函数不会处理与自己无关的异常。注意：**每个注册 SEH 异常的线程函数在函数结束时都应该注销异常。**

代码示例：

```
MY_FUN2  proc
    push  ebp
    mov  ebp,  esp

    ;  注册 SEH
    push  offset  FUN2_except_handler
    mov  eax,  fs:[0]
    push  eax
    mov  fs:[0],  esp

    ;  触发异常
    mov  eax,  0
    mov  dword  ptr[eax],  1
```

```
    ;  卸载 SEH
    mov  eax,  dword  ptr  [ebp  -  8]
    mov  fs:[0],  eax

    leave
    ret
MY_FUN2  endp
```

异常处理回调函数

> 结构体：可通过 "EXCPT.h" 查看

```
EXCEPTION_DISPOSITION __cdecl _except_handler  (
    // 指向包含异常信息的 EXCEPTION_RECORD 结构
    struct _EXCEPTION_RECORD* ExceptionRecord,
    // 指向保存异常回调函数链表的
    void* EstablisherFrame,
    // 指向线程环境CONTEXT结构的指针线程异常处理的注册和卸载异常回调函数的调用过程
    struct _CONTEXT* ContextRecord,
    void* DispatcherContext
)
// 4种返回值及含义
// 1. 回调函数处理了异常，可以从异常发生的指令处重新执行。
ExceptionContinueExecution
// 2. 回调函数不能处理该异常，需要SEH链中的其他回调函数处理。
ExceptionContinueSearch
// 3. 回调函数在执行中又发生了新的异常，即发生了嵌套异常
ExceptionNestedException
// 4. 发生了嵌套的展开操作
ExceptionCollidedUnwind
```

异常展开

异常码为：C0000027（异常展开）表示由操作系统将保存异常回调函数的链表项全部删除，卸载前会通知回调函数（将没有释放的内存空间自己进行释放），也就是回调函数会调用两次（第一次问处理不处理异常，不处理会第二次通知，将其（异常回调函数）从链表中删除）

> 清理资源

SEH 结构化异常处理的应用

> 在程序开始时注册一个SEH，然后故意触发异常，将程序的核心代码放入到异常的回调函数中，防止程序被调试。

"inc2l.exe" 程序就是这样的做法，观察其入口代码就可以发现其在程序入口代码处注册了结构化异常，并触发了异常，使程序不能正常被调试。

- 通过分析可以找到异常回调函数的地址，通过调试器在异常回调函数首地址下断点
- 重新调试程序，Shift + F9 忽略程序异常，断点就可以被断下。