病毒分析要掌握的技能

虽然这里面的技能都比较久远了，但是常识还是要了解的

1._declspec(naked) 告诉编译器不要优化代码

对于 jmp 类型的 hook， 如果自己的过程没有使用_declspec(naked)，那么系统会自动给添加一些额外的代码，控制堆栈平衡，但是这些额外的代码会破坏被 hook 函数的堆栈。

对于 call 类型的 hook，如果使用_declspec(naked)修饰的话，要注意自己恢复堆栈平衡。

```
#define NAKED __declspec(naked)
void NAKED code(void)
{
     __asm
    {
        ret
    }
}
```

使用__declspec(naked)关键字定义函数:

1）使用 naked 关键字必须自己构建 EBP 指针 (如果用到了的话);

2）必须自己使用 RET 或 RET n 指令返回 (除非你不返回);

_delcspec(naked)用在驱动编写，C 语言内嵌汇编完成一些特定功能。

2.虚拟机检测

这里可以看<软件调试>第十八章的内和调试引擎，详细讲述虚拟机调试

3.PE 结构

必须弄清楚的，包括可能的一些加壳知识

4.注入方式

很多病毒的一些行为了，注入方式挺多的

5.反调试技术 反虚拟机检测 花指令解决办法 IDC 脚本 网络数据分析 调试方法

6.一定的汇编知识

1）mov edi，edi

就是两个字节的 NOP，与程序中 NOP 意义相同

那为什么用 mov edi，edi 不用两个 NOP 呢

因为 NOP 的 CPU 时钟周期要比用 MOV EDI ,EDI 指令要长 ,为了提高效率，就采用 MOV EDI，EDI

防止线程中断导致现场被破坏

对齐字节的，凑字节，但是 nop 效率低，就用 mov edi，edi

现在编译出来的都是 call xxxxxxxxxx 然后 jmp [MessageBoxA]

1.减少 PE 模块中重定位的数量

2.尽量将模块里的重定位移出代码主代码区，以便进程之间的代码共享

2）test eax，eax

test 指令操作是目的操作数和源操作数按位逻辑“与“运算，结果不送回目的操作数然后根据结果设置 SF、ZF、和 PF 标志位，并将 CF 和 OF 标志位清零。而 JE 是当 ZF=1 时跳转。

即，当 eax 的值等于 0 时跳转。因此说，这里的 test 就是检测 eax 的值是不是 0

test eax,eax 将 eax 作与操作，和 and 一样，只是不改变 eax 的值

test eax,eax cmp eax,2

如果 eax 为 0 ZF = 1 作 je 跳转

如果 eax 为 0 ZF = 1 作 jz 跳转， 不为 0 则 jnz 跳转

3）Cmp eax, 2

如果 eax-2=0 即 eax = 2 就设置零标志 ZF = 1 JZ 跳转

4）mov dl,0dh 回车控制符 odh

mov dl,0ah 换行控制符 oah

5）or al,al 判断是否为 0

jz done

or al,20h 使小写字母转化为大写字母

6）mov ax,5

lea ax,[ax+6]

此时 ax=11.象这种情况,lea 基本上可以看成相加，但要比 add 速度快

更多的需要自己去积累了

7）SSDT Hook

一些函数的 Hook，不过现在应该没用了，内核保护的很好的，也很容易被检测

8）调用约定

C 的调用方式_cdecl(参数从右向左依次入栈) 清理方式：调用者清理

对于参数可变化的:printf("XXXXX");

```
void demo_cdecl(int w,int x, int y, int z)
asm:
push z
push y
push x
push w
call demo(进入了函数，开始执行了。。。。。。)
add esp, 16 4*sizeof(int)
```

标准调用_stdcall

```
void demo_stdcall(int w,int x, int y, int z)
asm:
push z
push y
push x
push w
call demo(进入了函数，开始执行了。。。。。。)
ret 16
```

快速调用_fastcall 前两个参数将被分配给 ECX ,EDX 其它按着_stdcall 调用方式 ret 8

C++调用约定 使用 this 指针

VC 提供了 thiscall 调用， 将 this 传递给 ecx

gc++中被当做静态变量，存放在栈顶

题目 1：写出 DLL 劫持原理，并写出哪些 DLL 不能被劫持

Windows 操作系统在加载 PE 文件的时候通过输入表会优先加载程序目录下的 DLL，而之后才再系统目录中寻找。伪造的 dll 制作好后，放到程序当前目录下，这样当原程序调用原函数时就调用了伪造的 dll 的同名函数，进入劫持 DLL 的代码，处理完毕后，再调用原 DLL 此函数。

此种方法只对除 kernel32.dll、ntdll.dll 等核心系统库以外的 DLL 有效，如网络应用程序的 ws2_32.dll、游戏程序中的 d3d8.dll，还有大部分应用程序都调用的 lpk.dll、sxs.dll，这些 DLL 都可被劫持。

题目 2：内核模式下，允许用什么工具进行调试

SoftICE WinDBG(本质是 KD) 还有国产的 Syserdebugger

题目 3：写出实模式下寻址方式

立即数寻址方式

寄存器寻址方式

存储器寻址方式

题目 4：概括讲解下游戏木马与下载器的特征。

题目 5：GDT 和 LDT 分别表示什么

Global Descriptor Table 全局描述符表

Local Descriptor Table 本地描述符列表

IDT：Interrupt Descriptor Table 中断描述符表

题目 6：详细讲解 SSDT 与 hook SSDT 的区别

通过 ring3 与 ring0 直接的 win API 联系起来的一个函数服务描述表

例：调用 Createthread -> 最后是调用 ntCreatethread 来达到创建线程的目的

那么 hook ssdt 就是通过 ring0 下系统钩子的形式在 SSDT 函数服务描述表建立拦截函数调用以及创建的一个监视过程

进 ring0 的方法太多，比如常规的中断门，陷阱门，调用门等！

这个里面还牵涉内存映射，全局变量共享等概念

简单的来说，SSDT 是 正常行为

HOOK SSDT 是 WS 行为。

SetwindowshookEx -> CallNextHookEx

题目 7：HOOK API 与 API HOOK 跟什么有关系？

题目 8：特征码分为几种，特征跟病毒是什么关系？

特征码分为两种，内存特征，文件特征

1：内存特征：表示在内存中存在病毒的宿主程序，称代码式注入或 DLL 注入

2：文件特征：通过读取某个文件 PE 信息，或者文件偏移量,取出长达 26 个十六进制字符存入病毒库

现在杀软用的都是 Debug 机制，简称：虚拟机查杀

特征就是证明这个是不是病毒，如果是病毒该如何处理。

题目 9 : Hook OpenProcess 会导致什么 ,冰刃下 SSDT 红色部分表示什么？

题目 10：主动防御的包括哪些？

HOOK HOOK 再 HOOK 各种病毒行为。改注册表拉，PE 感染拉，SYS 感染拉 ShellCode 识别。

怎么对一个样本进行详细分析，是否进行过详细分析，怎么提取的特征码，分析过哪个病毒。

7.什么是 shellcode？原理是什么？

Shellcode 实际是一段代码（也可以是填充数据），是用来发送到服务器利用特定漏洞的代码，一般可以获取权限。另 外，Shellcode 一般是作为数据发送给受攻击服务的。

8.jmp 的机器码是多少？

答：jmp 的机器码是 EB、E9、EA、FF。

9.pe 文件结构大体是什么？

10.怎样判断一个文件是 exe 还是 dll？

IMAGE_FILE_HEADER 中的文件属性字段中 普通的 EXE 文件这个字段值一般是 010fh，DLL 文件这个字段的值是 0210h

11.什么是壳？

12.木马分为哪几类？

13.call A

A：pop eax

指令是什么意思 调用 pop eax 语句

1）_STDCALL 的参数压栈方式，堆栈平衡方式

这个比较基础，是从右到左依次压入，CALL 内平衡

2）C 语言里 wsprintf 参数的压栈方式，为什么

这个也比较基础，就是从右到左依次压入，CALL 外平衡。为什么，，是因为 wsprintf 的参数是可变的。

3）PE 里面物理和文件地址的转换

这个也比较基址，我自觉得对 PE 还算了解，还算熟悉，所以关于 va,rva 等这个没有问题。

4）常见的注入方式

Windows Hook、远程线程注入 DLL、远程线程注入代码

5）如何分析一个数据的方法：

我答：通过输出字符串，导出函数或者 CE 工具，定位内存地址，通过 IDA 静态分析结合 OD 动态调试，很快就可以找到想要的了。

病毒分析师需要的技能如下：

1、至少要了解病毒的行为一般都有哪些

2、病毒行为的详细过程要知道

3、一些简单的工具的开发至少要会

4、分析工具的使用至少要会

5、至少要能写一些辅助病毒分析的工具

作为一个病毒分析师需要的是了解病毒的种类，以及他们的行为特征，还需要了解他们变异后的特点，还要了解病毒常用的 API，以及各类病毒专用的 API，比如纯 Ring3 的病毒和加载驱动的病毒，它们的不同之处是加载驱动的病毒里面有着特定的行为，使用了一些专门的 API，以及注册表操作等等，对于一个病毒分析师，我们需要精确的定位是不是病毒，或者是不是盗号木马，不能单靠 API 来判定是否为病毒，比如一个程序里面是正常使用 OpenFile,但是你一看见使用了这个 API，就把它定义为病毒就错误了，对于这种，我们需要写一些工具来判

定它的行为特征，比如 HOOK 病毒调用的 API，我们创建一个挂起进程的病毒，我们这时创建远程线程 HOOK 掉相关的 API ,把病毒使用的相对危险的 API 的参数全部记录下来，这样我们就能快速判断一般的普通病毒了，对于那些猛一点的病毒，或者破坏系统的病毒，这时我们就需要在安全的环境下分析病毒了，以免病毒破坏我们的病毒分析环境，比如你在真实机器上分析一些盗号木马，或者盗取银行账号的木马，那就比较危险了，一般我认为的病毒分析的环境是，如果是感染的病毒，我们需要开着还原系统，还原各个磁盘，一般分析比较危险的病毒这样应该足够了，对于那些破坏还原的病毒，我相信一般的病毒分析是不能分析的！

我们还需要了解和使用常用的病毒分析工具，比如反汇编工具，和动态调试器，各种监视器，以及反 Rootkit 工具，一般的分析流程总结如下：

1、保护自身机器的安全，也就是还原保护

2、监视器开着，

3、反汇编工具，动态调试器

4、自己的病毒分析辅助工具

5、总结病毒的类型

6、总结病毒的行为和危害

就是需要掌握的技能了，下面的还有一些基本的技能。

我们需要写一些辅助工具来帮助我们辅助分析病毒，下面的是 Ring3 的说明，当然你也可以写 Ring0 的工具。

1、注册表监视器，一些病毒经常注册表来对抗一些杀软或者保护自身，比如映像劫持，还有，我前面发过的注册表控制 360 的开关。

2、文件监视器，一些病毒比如键盘记录这种储存量比较大的东西，就要写入文件了，以及创建文件等等，

3、线程监视器，我们需要监视远程线程的创建，何时创建，何时停止，还是不停止

4、进程监视器，一些病毒经常创建进程来加载一些 DLL，以及启动人家的进程。

5、网络监视器，病毒必备的工具，病毒一般得到自己需要的东西后，就发送到自己的机器上，这时我们可以写一些监视工具来辅助抓包分析，抓到的数据包分析等等，以及得到目标的 IP 地址，好报案等等。

常用的 Win API 函数（在这里给出做木马病毒常用到的 API）

1、限制程序功能函数

EnableMenuItem 允许、禁止或变灰指定的菜单条目

EnableWindow 允许或禁止鼠标和键盘控制指定窗口和条目(禁止时菜单变灰）

2、对话框函数

CreateDialog 从资源模板建立一非模态对话窗

CreateDialogParam 从资源模板建立一非模态对话窗

CreateDialogIndirect 从内存模板建立一非模态对话窗

CreateDialogIndirectParam 从内存模板建立一非模态对话窗

DialogBox 从资源模板建立一模态对话窗

DialogBoxParam 从资源模板建立一模态对话窗

DialogBoxIndirect 从内存模板建立一模态对话窗

DialogBoxIndirectParam 从内存模板建立一模态对话窗

EndDialog 结束一模态对话窗

MessageBox 显示一信息对话框

MessageBoxEx 显示一信息对话框

MessageBoxIndirect 显示一定制信息对话框

GetDlgItemInt 得指定输入框整数值

GetDlgItemText 得指定输入框输入字符串

GetDlgItemTextA 得指定输入框输入字符串

Hmemcpy 内存复制 （非应用程序直接调用）

3、磁盘处理函数

GetDiskFreeSpaceA 获取与一个磁盘的组织有关的信息 ,以及了解剩余空间的容量

GetDiskFreeSpaceExA 获取与一个磁盘的组织以及剩余空间容量有关的信息

GetDriveTypeA 判断一个磁盘驱动器的类型

GetLogicalDrives 判断系统中存在哪些逻辑驱动器字母

GetFullPathNameA 获取指定文件的详细路径

GetVolumeInformationA 获取与一个磁盘卷有关的信息

GetWindowsDirectoryA 获取 Windows 目录的完整路径名

GetSystemDirectoryA 取得 Windows 系统目录(即 System 目录)的完整路径名

4、文件处理函数

CreateFileA 打开和创建文件、管道、邮槽、通信服务、设备以及控制台

OpenFile 这个函数能执行大量不同的文件操作

ReadFile 从文件中读出数据

ReadFileEx 与 ReadFile 相似，只是它只能用于异步读操作，并包含了一个完整的回调

WriteFile 将数据写入一个文件

WriteFileEx 与 WriteFile 类似，只是它只能用于异步写操作，并包括了一个完整的回调

SetFilePointer 在一个文件中设置当前的读写位置

SetEndOfFile 针对一个打开的文件，将当前文件位置设为文件末尾

CloseHandle 关闭一个内核对象。其中包括文件、文件映射、进程、线程、安全和同步对象等

_lcreat 创建一个文件

_lopen 以二进制模式打开指定的文件

_lread 将文件中的数据读入内存缓冲区

_lwrite 将数据从内存缓冲区写入一个文件

_llseek 设置文件中进行读写的当前位置

_lclose 关闭指定的文件

_hread 将文件中的数据读入内存缓冲区

_hwrite 将数据从内存缓冲区写入一个文件

OpenFileMappingA 打开一个现成的文件映射对象

CreateFileMappingA 创建一个新的文件映射对象

MapViewOfFile 将一个文件映射对象映射到当前应用程序的地址空间

MapViewOfFileEx （内容同上）

CreateDirectoryA 创建一个新目录

CreateDirectoryExA 创建一个新目录

RemoveDirectoryA 删除指定目录

SetCurrentDirectoryA 设置当前目录

MoveFileA 移动文件

DeleteFileA 删除指定文件

CopyFileA 复制文件

CompareFileTime 对比两个文件的时间

SetFileAttributesA 设置文件属性

SetFileTime 设置文件的创建、访问及上次修改时间

FindFirstFileA 根据文件名查找文件

FindNextFileA 根据调用 FindFirstFile 函数时指定的一个文件名查找下一个文件

FindClose 关闭由 FindFirstFile 函数创建的一个搜索句柄

SearchPathA 查找指定文件

GetBinaryTypeA 判断文件是否可以执行

GetFileAttributesA 判断指定文件的属性

GetFileSize 判断文件长度

GetFileTime 取得指定文件的时间信息

GetFileType 在给出文件句柄的前提下，判断文件类型

5、注册表处理函数

RegOpenKeyA 打开一个现有的注册表项

RegOpenKeyExA 打开一个现有的注册表项

RegCreateKeyA 在指定的项下创建或打开一个项

RegCreateKeyExA 在指定项下创建新项的更复杂的方式

RegDeleteKeyA 删除现有项下方一个指定的子项

RegDeleteValueA 删除指定项下方的一个值

RegQueryValueA 获取一个项的设置值

RegQueryValueExA 获取一个项的设置值

RegSetValueA 设置指定项或子项的值

RegSetValueExA 设置指定项的值

RegCloseKey 关闭系统注册表中的一个项（或键）

6、时间处理函数

CompareFileTime 比较两文件时间

GetFileTime 得文件建立，最后访问，修改时间

GetLocalTime 得当前本地时间

GetSystemTime 得当前系统时间

GetTickCount 得 windows 启动至现时毫秒

SetFileTime 设置文件时间

SetLocalTime 设置本地时间

SetSystemTime 设置系统时间

7、进程函数

CreateProcessA 创建一个新进程

ExitProcess 以干净的方式关闭一个进程

FindExecutableA 查找与一个指定文件关联在一起的程序的文件名

FreeLibray 释放指定的动态链库

GetCurrentProcess 获取当前进程的一个伪句柄

GetCurrentProcessId 获取当前进程一个唯一的标识符

GetCurrentThread 获取当前线程的一个伪句柄

GetExitCodeProces 获取一个已结束进程的退出代码

GetExitCodeThread 获取一个已结束线程的退出代码

GetModuleHandleA 获取一个应用程序或动态链接库的模块句柄

GetPriorityClassA 获取特定进程的优先级别

LoadLibraryA 载入指定的动态链接库，并将它映射到当前进程使用的地址空间

LoadLibraryExA 装载指定的动态链接库，并为当前进程把它映射到地址空间

LoadModule 载入一个 windows 应用程序，并在指定的环境中运行

TerminateProcess 结束一个进程

1.通过 FindWindow 读取窗体的句柄

2.通过 GetWindowThreadProcessId 读取查找窗体句柄进程的 PID 值

```
var
nProcId:DWord;
nProcId:=GetWindowThreadProcessId(hFound, @nProcId);
```

3. 用 OpenProcess(PROCESS_QUERY_INFORMATION Or PROCESS_VM_OPERATION Or PROCESS_VM_READ Or PROCESS_VM_WRITE, 0, ProcessId)打开查到 PID 值的进程. 此打开具备读取,写入,查询的权限

4.ReadProcessMemory 读出指定的内存地址数据

```
BOOL ReadProcessMemory(
HANDLE hProcess, // 被读取进程的句柄；
LPCVOID lpBaseAddress, // 读的起始地址；
LPVOID lpBuffer, // 存放读取数据缓冲区；
DWORD nSize, // 一次读取的字节数；
LPDWORD lpNumberOfBytesRead // 实际读取的字节数；
);
```