《软件加密技术内幕》

1、PE中使用了相当多的结构数组， 用结构保存信息 然后结构项用了指针/或者rva地址

PE 磁盘和内存是一致的，可能会在运行中动态修改 window加载器 loader 和壳加载器

映射到内存 偏移 模块 模块句柄，基地址

区块section 输入输出表 资源 基址重定位表 rdata text代码 rsrc资源 reloc idata 可以自己创建区块

两个对齐值

相对虚拟地址rva 相对于pe装入地址（前面装入的是系统） 代码区块的rva 1000

获得可执行模块装入地址GetMoudlleHandle

数据目录表 一共16个结构 指向 输入 输出表 资源 异常处理表 重定位 TLS线程局部存储初始化 dll 输入地址表的开始iat 延迟装入

输入 加载器会定位所有 dll函数 隐含连接和显式连接LoadLibrary GetProcAddress

加载器会连接kernel32.dll ntdll.dll

输入地址表IAT指向每个dll 以及其中的函数 ，包含函数的地址 优化修饰call prt []

文件结构有32位和64位区别 dos头 nt头 nt头包含文件头 选项头（重要）

Optional头有很多重要信息 区块大小 包括装入地址 执行入口的rva(指向启动函数不是main) 代码块起始rva 数据rva 区块对齐大小 装入内存后的总尺寸 校验和（）

子系统枚举 堆栈 和堆的默认大小 最后一项：数据目录表的地址

区块表 是个数组 区块大小需要对齐 4k 8k

块名 区块磁盘大小，内存大小 块rva 磁盘文件偏移 块重定位（dll） 块属性

块属性:包含代码 可执行 包含已初始化数据 包含未初始化数据 可丢弃（装入后不需要）

不能页交换（完整装入） 共享（dll吧） 可读 可写

常见块 text代码 数据读写data 全局变量 只读 输入表idata一般被合并 relaease会分出来 edata输出表 被合并 rsrc只读 crt运行时 .tls 。reloc基址重定位（dll才需要）

输出表 符号（输出函数或变量） :创建时间 版本号默认0 dll名字的rva 条目数 函数的rva 函数名字的rva

输出地址表 输出序数表 输出名称表 系统从表中获取rva，然后跳转地址加载函数

输出转向 转到另一个dll

数据可能是值，也可能是个地址rva

输入表 iid 输入名称表 绑定 dll名

输入地址表iat api的内存rva 序数

Pe检查输入表，把dll加入到内存 输入函数的真实地址 绑定输入

延迟装入

资源 数据目录中的 资源目录 包含rva 和大小

资源指针

基址重定位 .reloc 数据目录 basereloc条目

调试目录 debug信息

.net头

TLS初始化 tls变量 数据目录 tls目录

程序异常数据

2、PE分析工具的编写 内存映射文件

格式检查 读取文件头 选项头 数据目录表16项 区块表 输出表 输入表

3、调试API 可以实现调试器的大部分功能 加载附加 修改 线程运行环境

恢复挂起的线程 ContinueDebugEvent 捆绑进程DebugActiveProcess 卸载进程DebugActiveProcessStop

产生断点异常DebugBreak 相当于int3 强制退出FatalExit 退出前先Int3

刷新指令高速缓存 获取线程执行环境GetThreadContext 获取一个Context结构

获取线程描述符表入口GetThreadSelectorEntry 是否处于调试环境isDebbugerPresent()

字符串传递给调试器 读取进程内存 ReadProcessMemory 设置线程环境 等待发生调试事件 写内存

调试事件 进程创建 进程异常 退出最后一个线程 线程退出 装载dll 发送调送消息字符串 卸载dll 每次接受到事件时 填入到debug\_context结构返回 事件类型事件进程 事件线程Id

U结构 创建 退出 创建线程 加载dll 异常 输出 rip

1. 创建调试进程 CreateProcess 在标志中加入调试
2. 调试循环体 监控调试事件是一个循环 waitfordebugevent 等待调试挂起线程 恢复线程continuedebutevent
3. 处理调试事件 根据debug\_context中的u结构

异常结构 u中的 exceptoion\_debug\_info 中的异常记录 exception\_record

异常类型码 可继续异常 不可继续异常 发生地址 异常信息参数的数目

最常见的异常 断点异常和单步异常 处理异常:系统默认函数 恢复运行

1. 线程环境 线程挂起时保存到运行环境 context 包括寄存器 堆栈 描述符表等状态

Contextflags floating\_save\_area get 用set函数可以在挂起时修改线程的寄存器

甚至是eip 跳转

用户环境

内核环境 恢复线程 resumeThread 挂起suspendThread

1. 如何注入代码

需要开辟或者找到一段空间来存入代码 VitualAllow 还可以把某段代码先保存 再替换 执行新代码后恢复 （利用堆或者栈）

过程WaitForDebugEvent 调试循环体 挂起线程 VitualProtectEx 改写读写权限

ReadProcessMemory 保存线程环境GetThreadContext WriteProcessMemory

代码执行结束处下断点 保存Context拷贝 修改context eip跳转到插入代码处 恢复线程 等到断点处再次挂起 恢复原代码 恢复原context

注意堆栈平衡 相对地址转为线性虚拟地址GetThreadSelectorEntry

6、编写脱壳机 SEH产生非正常跳转，干扰调试 SEH编号 发生地址 类型

这个地址的值是固定的吗？ 输入表一般在解密时破坏，阻止 然后dump程序

脱壳 需要设置断点 读取原始代码并保存（dump）写回原始代码

清除断点 控制 eip 自校验int3 另一种断点 设置线程context dr值 硬件断点

加壳程序内存一般本来就可读写

1. 文件补丁和内存补丁 增加代码

文件补丁:直接修改可执行文件 要破校验

内存补丁 loader（壳，加载器）运行中修改 createProcess

跨进程存取内存 debugapi 调试寄存器dr0~dr7 标志寄存器

通知机制 exception\_debug\_event 异常消息 信息

单步模式 单步标志寄存器 sf 地址需要字节对齐

Int3中断 会将异常记录.异常码 设为断点 getThreadContext获得eip

Debug api 断点 复原

反调试壳

调试寄存器 drx 只有4个能用 0~3 6状态7控制 drx设为地址 7设为控制

此时 收到异常消息 single\_step

Win2000后不能对入口函数设断 必须用ntdll.ntcontinue函数做跳板 不能用setThreadContext 而要借esp地址为跳板

四、windows异常处理 c++的异常处理try except SEH VEH

Assert宏 if esle try catch 非法操作 应用程序错误 遇到问题需要关闭 异常原因 堆栈溢出 非法指令 读写保护内存

可恢复异常 不可恢复异常

结构化异常处理 挂起程序——exception\_debug\_info，包含异常记录，第一次改变

发送异常消息exception\_debug\_event

如果异常没处理或者没调试 调用seh 线程相关异常处理例程 处理程序可以有一个或多个，一个不处理可以交给另一个 如果仍然没处理 转到进程相关异常处理

SetunhandledexceptionFilter 如果进程相关也没处理 调用系统默认处理（对话框）关闭和调试 如果不调试，则退出进程

终结之前还会再次调用所有异常处理过程

监视线程某代码是否异常 ，监视所有线程是否异常 final 最后调用的异常处理或者叫顶层异常处理

数据结构 TIB或teb 线程信息块 总是由[fs:0]指向 第一项就是异常列表 异常链表指向 异常注册 包含异常处理过程的地址

异常指针 包括记录和上下文

异常记录 类型码 异常标志 嵌套的异常指针 异常地址 附加消息 读冲突/写冲突

线程上下文context 控制寄存器 通用寄存器 段 浮点寄存器 调试

1、API 设置错误模式 设置顶层异常回调函数 引发异常 获取/设置线程环境 堆栈展开

设置顶层异常回调函数 SetUnhandledExceptionFilter 一个进程只能注册一个，每次调用都会修改 并返回上次设置函数的地址

根据返回值进行下步行动 已处理，结束 已处理，继续运行 未处理，转发

检测ExceptionFlags 不可修复异常交给系统。。。

2、线程异常

注测线程回调函数 tib-exception\_registoration结构有注册的值 该结构有函数地址和prev指针 形成异常处理链

返回值 继续 继续查找 注册多个异常处理，嵌套-prev链表 ，每个回调函数对异常类型进行检测if

4、堆栈展开 exceptionflags 1、不可修复异常 2、展开操作用于清理，恢复

RtUnwind

9x系统可以用seh进入ring0 回调中修改cs ss 2、seh实现单步自跟踪 eflags tf位为1 执行指令——单步异常——异常处理再次置1

1. c++封装seh机制
2. VEH 向量异常化处理

AddVectoredExceptionHandler 注册veh回调 也形成一个链表，需要输入链表头的函数，每次注册都加到尾部

异常发生先执行调试器 ，然后是veh veh的好处是可以随意修改函数链表的顺序

四、防破解技术

利用SEH技术 防止在函数多处设置断点 seh链表加入自己的节点 int3触发异常 转入自己的异常处理函数 实现一种自跟踪机制 修改context 把regFlags tf设为1

contextFlags改为CONText\_full 这样每次执行指令将产生单步异常 再次异常处理中把tf设为1 就可以每次指令处理一次 每次执行指令会自动清除TF位

追踪自己的函数 不追踪api 判断下一个指令是 否int3

硬件断点 利用seh技术 故意产生一个异常进入异常处理 然后处理中设四个硬件断点进行 每次硬件断点时触发step异常处理 用计数器统计step异常执行次数

反加载技术

Isdebugger 父进程 seh反跟踪 监测调试寄存器

TEB检测 fs:0 从TEB获得PEB偏移 30h fs:[30h ] peb中有 beingdebugger标志 2h

Isdebbuger（） 插件可以修改标志

父进程 一般是Explorer cmd.exe 或者service.exe

获取所有进程快照 查找进程 遍历进程 获取PID然后对比是否自己的进程

父进程名 检查名字

3、反监视

监视工具 regmon filemon

搜索窗口标题 findwindow getwindowtext 类名，标题

捕获句柄 （如果程序调用了驱动的话）

4、反静态分析

无关代码 跳转 比较 循环

1、堆栈实现跳转 代替jmp 利用 push 地址 和ret配合 ret将返回到堆栈保存的地址

2、寄存器跳转 jmp eax 或者call [eax] eax的值可以来自堆栈 或者mov 也可以故意弄个玄虚 push 去 隔一会pop回

3、花指令 插入垃圾干扰判断 导致错误定位指令起始位置

4、SMC代码 动态加密/解密代码 对付静态分析

5、人机对话的信息隐藏 比如字符串跟踪 先加密 执行时再解密 也可以放在外部文件中

反dump lordPE 都是用Module32NEXT获取进程信息 有个moudleentry32结构

进程基址 映像大小 句柄 模块名 路径 pid 由大小和基址获取进程位置 然后readprocessmemory 检验dos头 nt头是否完整 不完整直接获取exe文件的头

反dump应该错误大小和基址 修改正在执行文件的文件头 40000正常 30000无法装载

50000非法 改成dl 无法执行 文件头很多地方可以改

所以干脆直接清空，用原文件就行

完整性校验 hash 计算文件散列值 md5 crc sha 对字符串生成4字节散列

读取全部文件（或者部分）字符串转为crc 和原crc对比

校验和 MapFileAndCheckSum

内存校验 1、对代码段校验（数据区可读写） 代码只读 pe中获取 rva 内存大小 基地址 计算某段内存的crc 比较和之前的crc 可以防止int3中断，防不了硬件dr

加壳后读取的是外壳的代码区rva和大小

代码和数据结合的保护方案

注意页面保护 非法操作

五、外壳编写基础 先于原始文件加载 把加密的原始文件还原 压缩壳 加密壳

虚拟机壳

资源 压缩 输入表 区块融合 额外数据

1. 判断是否exe 校验文件头 dos和pe mz 最后校验是否dll
2. 读入文件 文件偏移法getFileSize 申请内存 Readfile WriteFile 缓冲区 或者复制到内存

或者用CreateFileMapping MapViewOfFile映射到内存

注意pe文件中读出的地址都是 rva 或者va 这种方法必须转化为偏移才能找到数据

方法2 SizeOfImage 节表中的虚拟地址 和虚拟大小 读入各区块

1. 额外数据 需要单独读入
2. Exe去除重定位信息 重定位区清除 5、按区块压缩 6、资源区块 不可压缩 图标 版本 资源目录不可压缩 7、 各区块融合 对齐 不足处用00填充 修改区块表数据 新rva 8、破坏输入表 加密时破坏输入表 换种形式存储- 在执行时写回输入表 但是写入的是壳中的函数地址rva 中转

壳反跟踪 反脱壳 反dump 反trace 反监控 反静态

还原 初始化输入表 dll加壳需要重定位

外壳加在原始程序上 开辟新区块 原始程序的数据保存在外壳

新pe头（入口rva） 新节 自建的输入表

9、Dll头 保存入口参数 多次进入 重定位项 输入表

10、反破解 进程枚举 花指令 进程快照 xxx32first函数 xx32next函数

如进程Process32next moudle32next

七、防查壳工具

1、 对抗工具 特征码 2、被加密程序判断自己是否被脱壳（是否有壳） crc 文件大小 2、检查标记 壳执行时在某处做标记，程序检查是否有壳标记 同步对象event 信号量semaphore 互斥量 create 3、原子检查标记 内核原子表 局部原子 全局

4、存储 文件映射是可以进程共享的 程序和壳共享的文件 5、优线级检查

6、外部文件检查 keyfile ini 壳创建文件 程序检查 7、注测表 8、壳中使用定时器

9、外部检测 DLL查是否 有壳 10、HOOKapi

11 SDK加密

附录

内联汇编 （目前只支持32）

\_\_asm{

代码

}

支持所有指令 masm表达式 可以使用c++的数据 变量 标号 函数名 常量

内联写函数，调用c函数

也可以用独立汇编