单例防并发 关于TLS 线程局部存储

进程快照，可以读取各种表，来获取进程在内存的地址，大小pid,句柄等各种信息

进程间通信 文件映射 共享内存 匿名管道 命名管道 邮件槽 剪贴板

动态数据交换dde dll 远程过程调用 sockets netibos wm\_copydata

直接把内存中的指令加入到高速缓存FlushInstructionCache，然后执行，是否可以绕过hook？

内存页是4k

什么是windows服务 ，服务项目

Windows很多函数返回值是没有具体错误信息的，需要调用getlasterror获取错误代码

Winform clr .net framework 只是对win32api的封装吗? Api封装的是内核函数

Windows用到的特殊数据结构 字符数组 结构体 指针 引用通常在调用时传参，为的是取出函数的值 宏 字符串指针

系统定义的钩子拦截函数 或者自己改写api调用代码实现钩子

APIHOOK 基址 ce模块基址+偏移 双击地址，数据结构可能保存在一起

HOOK拦截消息 线程钩子 系统钩子 安装钩子 卸载钩子 拦截对API函数的调用

键盘钩子 鼠标钩子

全局钩子

钩子类型 发送到窗口过程的消息

钩子回调函数 下一个钩子函数 钩子类型 函数地址 所在实例句柄

这些函数写成dll 然后主程序注入代码 ，调用dll函数 一般是hook到原系统dll函数入口 需要保存原地址 然后 jmp 用读写内存来改写代码

一、字节序:针对大小超出一字节的数据 不包括数组等 windows上是小端

字符集 Unicode

2、Win32api 子系统到核心层 使用dll 显式调用 隐式连接（在创建的时候lib连接）

子系统 kernel user gdi

常用 获取窗口文字 获取对话框句柄 getdlgitem getdlgitemtext获取对话框文字 获取对话框数值 显示消息框

句柄：标识各种东西 如进程，文件，设备，菜单 进程句柄表

注册表函数 gdi函数 文件系统函数 窗口，进程的遍历

消息机制 应用消息队列 系统消息队列 先到系统 然后传给应用

父消息到子消息，子消息到父消息，句柄的传递，窗口句柄，进程句柄，

发送消息 wm\_command 一般是选择控件时发送 \_destory \_quit \_gettext拷贝到缓冲

保护模式 16位 32eax 64位rax 段寄存器 fs gs 实模式 dos 保护模式windows

虚拟内存 ds:[] 段选择器放的是段选择子 不寻址 只是全局描述符表 和本地描述符表GDT LDT的指针

平坦模式 4G虚拟空间 用的是映射方式 操作系统全程映射 应用代码只在自己的时间映射? 数据 dll会被映射到进程的空间内

应用权限 ring0 ring3

3、wow64

二、动态分析 调试器 常用断点 插件 run trace hit trace

X64dbg mdebug windbg

内存 线程 CPU（寄存器 ） 堆栈 数据（内存直接数据）反汇编（内存解释出代码） 信息

单步 跳过call 跳到ret 返回程序领空

断点 in3 0xcc 硬件dr0-dr3寄存器 内存 消息 条件 条件记录

保存修改的程序 称为补丁或者爆破

Run trace 记录执行过的指令和状态 方便回溯 hit trace判断已执行过的代码

三、静态分析 文件类型分析

2、反汇编引擎 beanEngine Udis86 CAPSTONE ASMJit

KeyStone

静态反汇编 IDA

十六进制工具

四、逆向分析技术 汇编代码分析(部分代码会被编译器优化)

**什么情况下需要从内存读取到寄存器 寄存器是一切操作的中转 包括计算**

代码分析 函数参数 寄存器传参 引用传参

堆栈局部变量 寄存器局部变量

全局 .data 直接读写 只读区 常量

数组

虚函数

Jmp短转移 长转移

设置条件 SET

字符串 c dos pascal delphi

运算符 加减 用add sub 优化 lea

指令修改技巧 字节替换 1 个 两个 inc push pop jmp

寄存器清0

测试寄存器是否0

转移

五、演示版保护 序列号 注册机 警告窗口 时间限制 菜单功能限制 keyFile

网络验证 光盘检测 禁止多开 断点设置技巧

找加密函数 找验证函数

跟踪api 获取文本getwindowtext getdlgitemtext getdlgitemint 显示对话框messagebox dlgbox showwindow createwindow

读取注册码 注册表getqueryvalue ini文件 getprivateprofilestring getprofilestring getprivateprofileint 普通文件createfile \_lopen c语言文件操作会用到系统api吗？或者nativeapi

2、数据约束性 一般 输入离输出不远 临时计算一般用局部变量保存

搜索查找到的值

3、

消息断点

提示信息

4、字符串如何比较 存入寄存器比较 cmp eax ebx (2)函数比较 传参 call 返回值 eax () (3)函数比较 （4）串比较

5、明码比较 （2）非明码 f1(明文)=f2(密文) 验证明文密文是否正确，不需要存储密文

6、计时器 setTimer 消息wm\_timer 多媒体计时器 timeSetEvent gettickcount()获取运行时间 timegettime 时间限制需要保存时间，存在文件或者注册表 需要获取当前时间，或者总运行时间 getsystemtime getlocaltime getfiletime

7、菜单功能限制 不可用 enablemenuitem()禁止菜单

8、keyfile 文件读写createfile getfilesize getfileattributes()文件属性 setfilepointer()设置文件指针 findfirstfile查找文件 readfile

可以伪造文件 可以用filemon监视文件操作

9、网络验证 send recv 抓包分析 发送 的数据包 3、写个服务端模拟数据，转发 或者修改客户端

10、cd 驱动器类型 getdrivetype() 符号 getlogicdrives() 根路径getlogicdrivestrings()

11、只运行一个exe 查找窗口 findwindow getwindowtext 互斥对象createmutex 使用共享区快 所有实例共享，通过计数器来判断

12 断点设置技巧 字符串 注册表 文件访问 ini 时间 对话框 光驱

六、加密算法 散列md5 sha sm3 对称加密 rc4 tea idea blowfish aes sm4

公开密钥算法 ras dsa 椭圆曲线 sm2

其他算法 ctc32 base64

常见加密库

七、系统 内核基础

权限级别 内存空间布局 windows启动过程 电脑的完整启动过程

R3与r0通信

内核函数

驱动模块

内核的重要数据结构

内核对象

Ssdt teb peb

内核调试 winbdg双机调试 查看ssdt shadow ssdt

八、异常处理 异常列表 处理过程

Seh 数据结构 处理程序注册和卸载 异常分发的过程 线程异常处理 栈展开 顶层

X64 异常分发 wow64

Seh进行验证输入 seh加密解密应用 veh api hook

异常列表 硬件异常 软件异常 中断类型号， 软件异常用RaiseException强制触发

1. 过程 挂起线程 填写结构体 exception\_debug\_info{ exceptionrecord 异常记录

dwfirstchance} 使dwfirstchance=1 发送exception\_debug\_event消息

异常处理程序:线程处理链（不能处理就挂起，然后dwFristchance归0 ），进程（顶层），系统默认（对话框并退出） ，结束之前还会最后调用一次

线程相关 进程相关

1. TEB结构 线程环境块 位于fs:0 第一项是 异常注册结构 exception\_registeration
2. exception\_registeration 包含处理函数地址 和prev指针，形成一条链
3. 关于访问违规， 忽略程序异常 当写入或者读取 0~FFFh 空间会异常
4. 当异常发生 ，线程堆栈压入三个结构 excption\_pointers{ exception\_record,context } 异常记录，线程上下文。

可以从栈顶读取

6、异常记录 : 异常代码 异常标志 指向另一个异常记录 异常发生地址 异常信息exception\_information

代码 禁止读写页面 断点异常 单步异常(tf) 无效句柄 读写冲突 堆栈溢出 除零错误 无效指令 不存在的页面 读写冲突

1. 线程环境 contextflags有效的域

寄存器 调试 dr0~dr7 浮点寄存器 段寄存 edi esi eax ebx cx dx

Bp sp ip cs ss

1. 异常处理回调函数 参数 异常记录,异常注册，线程环境

返回值 设置context并返回 拒绝处理，发往下一个函数 回调中再次触发异常 嵌套展开

1. 改变线程环境 可以修改寄存器 实现跳转执行地址，数据修改

Context.eip

九、调试api 函数 事件 创建进程并跟踪 循环体 调试事件的处理程序 线程环境 代码注入

十、VT技术 硬件虚拟化 结构 汇编指令 EPT 检测VT支持 VMCS配置 开启

内存隐藏 VT调试

十一、PE格式 基地址 载入的地址 虚拟地址 内存地址 相对虚拟 pe中写的地址 文件偏移 dll需要重定位

入口点 执行的第一行代码 （一般指启动函数，调试器一般会断在main函数）

段选择子 cs:va

DOS头 PE头 字段 结构

Dos头 mz头 dos stub 从dos头到pe头 基址加rva pe头：image\_nt\_header结构

字段 signature 004500h ”PE00” fileheader 文件头 包含信息 平台区块数 创建日期 符号表指针 符号个数 dll还是exe image optional header 结构大小

Optionalheader 可选映像头 是文件头的补充 是否可执行文件 连接器版本号 代码区块总大小 数据区块总大小 入口rva 代码区rva 数据区rva 装入基地址 内存区块对齐值 文件区块对齐值 操作系统应用和子系统版本号 映像总尺寸 dos头 pe头区块表总大小

校验和 初始化堆栈大小 数据目录表条数 （16） 数据目录表指针

数据目录表: 导出表 导入表 资源表 附加表 安全表 基址重定位表 调试 tls iat

Nt头之后是区块表 是一个结构数组 每项都是 image\_section\_header区块头

块名 尺寸（没对齐，文件大小） 区块rva 对齐后尺寸（虚拟大小） 文件中的偏移 区块属性（可读可写、代码还是数据，共享块） .text .rdata .data 行号表用于调试

块 .text .data .rdata .crt .rsrc .tls .reloc

对齐 :磁盘中的 内存中的 区块偏移开始位置都是对齐值的倍数，不足地方用0填充，由于页是4K 1000h 64是8k 所以都是1000h的倍数

文件偏移换成rva,对齐值相同 ，则偏移=rva 否则不同

输入表 loadlibrary getprocaddress获取函数指针地址 隐式加载时是由系统加载器完成的（如系统 dll）,此时可以直接调用函数名，不用设法获取dll函数名

输入地址表 指向函数指针 序号 和被修改的函数名 函数地址就在iat中

输入表在可选头， 里面是 iid数组 image\_import\_descriptor 每个iid 代表一个dll

输入名称表的rva timedatestamp 时间 第一个函数的索引0 dll名字的字符串指针

输入地址表rva firsttrunk

Iat 是数组 每项都是image\_trunk\_data结构

image\_trunk\_data:以序号或者函数名输入 函数地址 序数 函数名

绑定输入 遍历iat image\_trunk\_data[] 数组 用真实地址替换

输出表 属于数据目录表 指向 image\_export\_directory结构 文件生成时间 版本号 模块真实名称 函数索引的基数0 函数个数 函数名个数 函数地址数组 函数名数组 序列号数组

基址重新定位 .reloc 重定位表 系统决定dll映射基址

用数据目录表的 image\_directory\_entry\_basereloc 多个重定位块 由多个image\_base\_relocation结构组成 重定位数据的rva 长度 /TypeOffset 重定位类型，偏移

资源 rva 大小 每项

TLS初始化

基于表的异常处理

十二、注入技术

1. DLL 注入

远线程 APC

提升权限 1、权限令牌 2、RtlAdjustPrivilege 目标进程注入dll open pid

创建线程 创建新线程 创建远线程 RtlCreateUserThread 申请内存load dll

使用线程环境寄存器，实现跳转 执行dll函数之后返回

向 用户模式APC队列插入APC对象 执行其中注册的函数

改注册表

挂钩消息 setwindowhook

手动加载dll 先读ntdll kernel.dll 获取NtFlushInstructionCache LoadLibraryA、GetProcAddress、VirtualAlloc的地址 然后自行实现loaddll 获取函数地址 GetProcAdress

十三、Hook技术 （包括系统提供的hook和自己修改代码跳转的hook）

Hook和dll注入可以结合使用

Hook 代码要么在dll，要么在内存现写

Hook windows api函数

可以修改windowsapi 或者直接修改代码跳转到自己 的函数， 在自己的函数中调用winapi

1、windows的原生消息钩子

监视事件消息，键盘钩子 外壳钩子 线程钩子 ，系统钩子，拦截各种消息

Setwindowhook(消息，钩子回调函数，全局?，局部?)

2、Api hook(inline hook) getmoudlehandle 获取dll模块基地址 getprocaddress 获取函数地址

修改地址保护属性 virtualProtect 修改跳转 ret回来， 记得要把api恢复

IAT HOOk inline HOOk

分类 address hook inlinehook

基于异常处理的hook(seh hook) 自行触发异常，用异常处理来检测或者跳转，如果不处理异常，就会导致出错

Hook位置选择

Hook 实施过程 address hook inline 异常

Ssdthook 内核hook

二次hook

Detour

检测恢复对抗

十四、漏洞分析技术

漏洞原理 溢出 UAF ShellCode

漏洞利用

十五、壳 各种壳介绍 原程序处理-壳本体-连接 壳(loader)本体可能用汇编或者c写

Upx aspack asprotect 压缩加密反跟踪 crc 花 加密算法 api钩子和程序通信 Armadillo execryptor themida

虚拟机 vm 只需要保护部分代码 vmprotect 可以保护代码 函数 加密

十六、脱壳

1、 加载过程 保存入口参数，外壳执行，恢复程序，跳转到原程序 pushhad popad pushfd popfd

壳自己需要的API， getProcAddress Getmoudlehandle loadlibrary,外壳调用其他函数就是用这些方法 可能自己实现

解密各区块, iat初始化填写 （壳自己的iat要改），把引入函数地址重新获取

重定位处理

填充输入表的时候可以用hook过的api，实现自己的功能 （）

跳转到oep

脱壳机 手动脱壳

1、寻找OEP 跨段指令 内存访问断点 栈平衡 编译语言特点

1. DUMP 原理 反DUMP 快照，模块遍历 moudle32next，利用moudleentry32结构{结构大小，模块id,进程pid,进程基址，进程大小，进程句柄，进程路径}

反dump 修改文件头 修改内存可读写

1. 重建输入表 IAT（输入地址表）地址 和大小 import REC 输入表加密

反重建填写的:iat地址是hook的api，在其中检测追踪，再跳转到真正的api

Call 或者 call jmp

输入表加密 : 保留原输入表文件在内存中，保留输入表数据，但是加密了， 直接破坏了原表，存入了输入函数字符串手动获取地址

1. DLL 脱壳 OEP dump 输入表 重定位表 附加
2. 数据
3. PE文件优化 输入表存放位置 资源重建 区块调整 修正文件头 入口点，数据基址 代码基址 映像大小
4. 需要对loader原理进行分析，加载过程
5. 静态和动态脱壳
6. 实例

十七、防范

1、 防算法求逆 游击战术，分成多段 ，分成多个文件，多处验证 防止某一处被爆破

虚实 ，假函数 ，假地址，假文件

把内存中的内容反复搬家 拷贝（内存监视断点） 注册表和文件

抵御静态分析 花指令 汇编器的算法 线性扫描 递归行进 花指令在代码中填入数据 db 可能会认错 或者转向无效跳转

smc 在代码执行前先修改，加密代码，动态解密，可以用单步破解，需要反单步反断点

多层smc，一层层解密，

信息隐藏 隐藏人机对话的字符串等

多态 变形 把代码变形，等价替换，把\*变成+ 干扰

2、文件完整性校验 把crc值存起来 然后比较，可以存在文件头 或者其他地方

crc 校验和

内存校验 整个代码段或者片段 防止Int3

3、代码和数据结合

十八、 反跟踪

Beingdebugged 读取peb的beingdebuged peb存在teb中 fs:0 是结构体 teb+30h是peb xp后peb地址随机 getthreadselectorentry 获取线程段选择子线性地址

NtqueryinformationProcess 获取peb

ntglobalflag peb中 在beingdebug之前的 会被设置标志 0X70 清除掉

heap magic 堆检测 ntglobalflag留下的痕迹 堆flags

源头阻止: 在load\_dll\_debug\_event时就把beingdebugged设为false,第二次load\_dll\_debug\_event 设为true

Native

查询进程信息 线程信息 对象信息 系统信息

18.2.1 CheckRemoteDebuggerPresent 检测调试器 如何实现的?

使用了NtQueryInfomationProcess() Zw 根据不同的ProcessInformationClass查询进程对象的信息，信息有编号 查询了7号信息ProcessDebugPort,调试器通信端口句柄，无法设置  
18.2.2 ProcessDebugPort   
18.2.3 ThreadHideFromDebugger

ZwsetiInformationThread 可以设置线程相关信息 ThreadInformationClass列表

有ThreadHideFromDebugger 禁止某线程调试 原理是把Thread对象的HideFromDebugger设为True

18.2.4 DebugObject 682 用户态调试的资料 native调试的谜底

创建进程设置模式 debug\_process 或者debugactiveprocess

一层层跟进windows的封装

会创建一个debug对象 ZwCreateDebugObject 存储句柄 NtcurrentTeb()->dbgssreserved[1]

获取debug对象 dbguigetthreaddebugobject() set

查询对象，ZwQueryOBject() 如果DebugObject不为0，有调试器 获取到ObjectInformationClass{} 结构 包括对象基本信息 名称信息 对象类型信息 所有类型

18.2.5 SystemKernelDebuggerInformation 686

ZwQuerySystemInformation() 查询系统信息

18.2.6 Native API 688 子系统到kernel.dll然后到ntdll.dll进入内核层，ntoskrnel.exe 到硬件抽象hal.dll

ntdll提供接口（函数），封装成子系统

createFile-> createFileA /W->NtCreateFile->内核层真正的函数 系统函数调用可以用int 2e或者是ntdll.dll调用 几乎所有Nt函数都是被Stub函数（sysenter指令）转入ntoskrnl的真正函数

所有Nt ZW 函数只是包装了一个索引 ，定位内核真正函数罢了 ，

调用了KiFastSystemCall进入系统内核层

Sysenter指令 sysexit rdmsr wrmsr指令 转入msr system\_eip\_msr寄存器地址

2、SDT 服务描述表SDT(或者ssdt)-> 系统服务表SST

Sdt{ntoskrnl,win32k}

Sst: servicetable函数地址表

3、Zw NT区别

在用户模式和内核模式都有nt zw函数 ，函数名相同，内核nt程序负责和设备交互，先检查参数，再发送irp io请求包

NtCurrentTEB()

4、用户模式到内核模式的切换 系统调用systemcall ioctl上下文传递 进程通信alpc

18.2.7 Hook和AntiHook 693 反调试手段用Native API

可以对Native API动手脚 找到Zw代码 ，修改跳转到自己函数的地址 返回 甚至直接返回

防止hook 代码片段检测 ，检测跳转地址是否正确

把dll原本的api代码抽离出来，然后 放在别处，跳转加载，最后删除，类似于inline hook===恢复api调用

自己实现loadlibrary 可以用int 2e调用系统函数 //要破解int2e会调用KiSystemService 需要修改ssdt 或者inlinehook 内核

2、反调试小技巧 句柄检测 驱动程序等文件句柄 检查工具

检测服务 内核设备驱动 服务名

UnhandledExceptionFilter

Int 2d 内核ntroskrn.exe运行DebugServices用的 ring3 使用会异常，如果有调试器不会异常 int2d可用于检测和代码混淆

OD 查找特征码 从不同位置提取的字节 :所有进程枚举 进程的目标地址有特征码

2、检查模块dll debughelp.dll库 快照，moudle32next 改名就行了

3、窗口 findwindow enumwindow(可以指定回调) getwindowtext getforegroundwindow获取前台窗口

4、查找进程 枚举

5、setDebugPrivilege 权限继承了调试器的 使用openProcess(csrss.exe)能打开说明启用了权限（限管理员）

6、异常处理 setUnhandledExceptionFilter 最后会传到顶层 处理 设置然后抛出异常，如果被调试，异常会被调试器接受

设置一个高层的exception Filter 抛出违规内存访问，被调试的话 就会收到两个异常

不调试就被exception Filter处理:修改 context.eip跳转 继续执行

有的程序是手工设置exception Filter的 用汇编，防止下断api

7、锁住前台的窗口 enablewindow() ，之后会恢复

8、blockinput 锁住键盘 c a d强制解除

攻击调试器漏洞

1、溢出漏洞

2、dr漏洞 捕获到异常会清0dr drx Seh中设置dr 必须要有dr才能继续执行

调试器检测

防止附加

附加:debugactiveProcess ntdll.dll zwContinue DbgBreakPoint(int3)

把这两个函数给hook掉

父进程检测 teb.clientid或者 getcurrentprocessid pid 快照函数 获取父进程pid,是否explorer.exe的pid

时间差 rdtsc获取开机到现在运行的周期 eax edx 64位 或者GetTickCount()

trap flag陷阱标志tf tf=1时 执行完当前语句后就触发单步异常，seh中处理 跳转地址，如果没跳转，就说明被调试

调试会阻断异常处理?？？

双进程保护

调试器机制 createprocess或debugactiveprocess h获取进程id 基址 -waitfordebugevent等待调试事件-,

十九、外壳编写 原程序读取处理 加 外壳用汇编写?或者用c++ 加载过程 添加到原程序

加壳程序 ，外壳部分

加壳：pe读入内存，区块压缩加密 输入表，重定位变形 外壳和原pe连接

外壳:类似于pe loader 处理输入表和重定位表，解密，跳转执行

1、判断文件类型 2、读入pe：直接读文件 或者利用文件头按照rva分块读入并对齐

用文件映射也行 直接读文件的话不方便用rva找到各处数据

3、读取附加数据 不属于任何区段，不会进入内存，而要用文件读取函数读exe

4、输入表 一破坏原输入表 把其中的函数名，地址提取出来，换方式存储

还原程序，填写输入表的时候，把函数地址指向自己的函数，在自己的函数中调用系统函数

5、重定位表 找到重定位数据区块，清零 把原重定位表换个形式 由外壳实现重定位

6、压缩

7、区块融合

外壳部分 在原程序之前执行

Pe加载器：由iat获取api地址，填写到iat中， 重定位重定位项 winmain/dllmain

二十、虚拟机设计

字节码指令系统 把机器指令变成字节码， 然后vm解释成机器指令

1、x86指令分类 指令 操作数

虚拟机 基于栈 基于注册?

调度器 压入 虚拟环境 vmcontext 包括寄存器 平衡堆栈

Handler

指令拆解

异常处理

二十一、VMP逆向和还原

二十二、补丁技术 pediy

文件补丁 直接修改exe中的 代码

内存补丁 （loader类似壳） 技术:跨进程内存存取（create） readprocessmemory vitualprotect 问题是修改内存的时机（不能已经读取过了代码，需要创建进程后就挂起，修改代码后再恢复）

debugapi (通知机制) 调试API dll注入 单步运行机制

调试异常事件 异常记录 把事件发送给调试进程 解析异常记录中的信息 处理

1. 方法一、 createprocess 使用单步模式 ，每执行一次都会发送exception\_debug\_event tf模式
2. 使用int3

使用调试寄存器 DrX

如何停在入口点

单步

用ntdll.ntcontinue作为跳板

Dll劫持 伪造同名dll，同样的输出表

Hook vt技术

Smc补丁 单层

补丁工具

二十三、代码二次开发 进行大量修改

数据对齐 4字节 2字节

增加空间 间隙 手工构造区块 工具

调用函数 输入函数 调用dll

重定位 重定位表修复 自定位

输出函数

消息循环的修改 wndproc

菜单扩展

Dll扩展 接口 消息循环

二十四、其他语言平台 易 delphi vb c# java jit