

恶意代码复习资料

前言：本资料仅仅针对2023级老师画的重点 以及课件中出现的选择题

第一章：

恶意代码数量的变化趋势？

- ☒ A 不断增多
- ☐ B 逐渐减少
- ☐ C 保持基本稳定
- ☐ D 趋于消失

以下那一场战争运用了网络攻防对抗技术？

- ☒ A 海湾战争
- ☒ B 科索沃战争
- ☐ C 第二次世界大战
- ☒ D 俄乌战争

下面哪些系统或设备可能被计算机病毒感染？

- ☒ A 计算机、智能手机
- ☒ B 打印机、网络路由器
- ☒ C 摄像头、智能家居设备
- ☒ D 智能汽车、智能电网、智慧城市

Q:如何对抗每天增多的恶意软件？

A:启发式检测 动态行为分析 机器学习等

数据科学家是否会取代计算机病毒分析工程师？

- ☐ A 数据科学家会取代计算机病毒分析工程师；
- ☒ B 数据科学家不能解决计算机病毒问题；
- ☐ C 网络安全法，震慑了计算机病毒作者，没有人写计算机病毒了；
- ☐ D 网络安全教育的普及，使计算机病毒威胁越来越小，不需要病毒防治了

恶意代码分析的目标

- 1.检测发生的行为
- 2.定位出受感染的主机和文件
- 3.解刨出可疑文件
- 4.找出可用于检测的特征码
- 5.建立机器学习监测系统
- 6.如何检测并控制损害

以下不是恶意代码分析目标的是（）

- ☐ A 对可疑程序进行深入分析，确定该程序是否有恶意行为
- ☐ B 定位被感染的机器或者文件
- ☒ C 恶意代码的优化和改进
- ☐ D 衡量并消除恶意代码对系统造成的破坏

基本静态分析技术

不执行可执行文件 不查看具体指令

优点：快速 简单

缺点：难以分析更高级的恶意软件 并且可能会忽视重要的表现

工具：VirusTotal strings

基本动态分析技术

运行恶意代码 并观察恶意代码在系统上的行为

注意使用虚拟机 并拍摄快照

优点：运行简单

缺点：需要安全的测试环境 并且并不是对所有恶意软件都有效

工具：Regshot Process Monitor Process Explorer 沙箱（自动运行并生成报告 但是运行环境单一）

高级静态分析技术

将可执行文件装载到反汇编器中，查看程序指令，来发现恶意代码到底做什么的逆向工程

优点：分析结果更为准确

缺点：分析过程复杂 需要分析人员理解汇编码 分析汇编结构和操作系统 并且不是对所有恶意代码都是有效的

工具：IDA Pro等

高级动态分析技术

使用调试器 检查恶意代码的运行时刻的内部状态

优点：准确 易于观察恶意代码的行为

缺点：调试复杂

工具：Windbg ollydbg等

恶意代码分析技术包括（）

A

基本静态分析，例如virustotal、strings

B

基本动态分析，例如沙箱等

C

高级静态分析，例如IDA Pro等

D

高级动态分析，例如OllyDbg、WinDbg等

恶意代码的类型

- 后门 允许攻击者访问受攻击的设备
- 僵尸网络 所有被一个僵尸网络感染的设备 会从一台命令服务器接受相同的命令 允许攻击者访问系统
- 下载器 下载恶意代码
- 间谍软件 手机信息并发送给攻击者 嗅探器 键盘记录器
- 启动器 启动恶意代码 确保隐蔽性或者是提权
- 内核套件 (rootkit) 隐藏恶意代码的恶意代码
- 病毒或蠕虫 复制自身并感染其他设备

Which type of malware conceals the existence of other code?

- A. Backdoor
- B. Botnet
- C. Downloader
- D. Keylogger
- E. Rootkit

- ☐ A Mass恶意代码会尽可能多的感染各种计算机
- ☐ B APT恶意代码只针对特定的目标进行感染
- ☒ C Mass恶意代码比APT有更大的威胁，杀毒软件更难检测到
- ☐ D APT恶意代码可能会“潜伏”很多年不被杀毒软件查杀

MASS 大量的 即大众性恶意代码 APT即为针对性恶意代码 顾名思义

以下哪些方法是恶意代码分析过程中不建议使用的（）

- ☐ A 在进入细节分析之前对恶意代码要有一个概要性的理解
- ☐ B 尝试多从不同角度，使用不同工具和方法来分析恶意代码
- ☒ C 对全部反汇编指令直接进行逐行分析
- ☐ D 先使用基本的动态和静态分析工具，定位可疑的静态和动态特征。

第二章：基本静态防治技术

使用基本静态分析我们能够获得哪些恶意代码的特征？

- ☒ A URLs
- ☒ B File Names
- ☒ C Registry Keys
- ☒ D API functions

strings扫描得到的结果 做过实验的都很清楚

Q:杀毒软件的名称

A:360 金山毒霸 卡巴斯基 诺顿 Mcfree Trend

常用的病毒识别网站：VirusTotal

躲避杀毒软件检测的方法：

混淆 多态性：语法混淆 变形：语义混淆

不同的杀毒软件使用了不同的病毒特征库和启发式规则，因此多杀毒软件交叉检测可以提高病毒的逃逸检测难度

Q:为什么文件的哈希值可以作为杀毒软件的特征？有什么优点和缺点？

A：相同的文件只有唯一的哈希值并且哈希值通常不会产生碰撞（密码学的内容hhhh）

优点：准确 速度快（因为只扫描哈希 不扫描具体文件）

缺点：需要大量的hash 很容易被躲避（文件发生细微的变化 哈希值也会随之改变）

hash算法

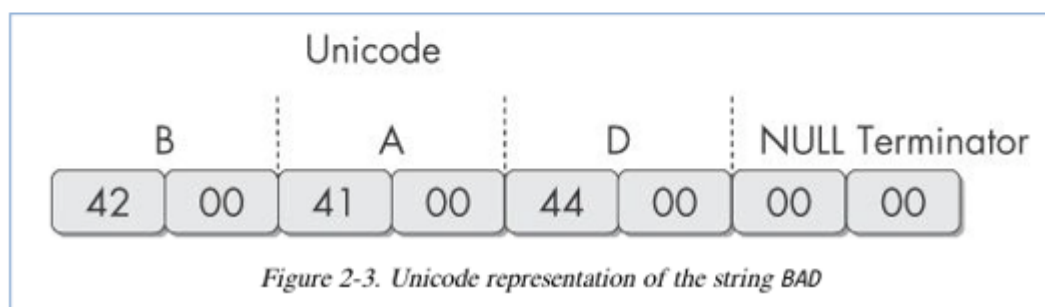
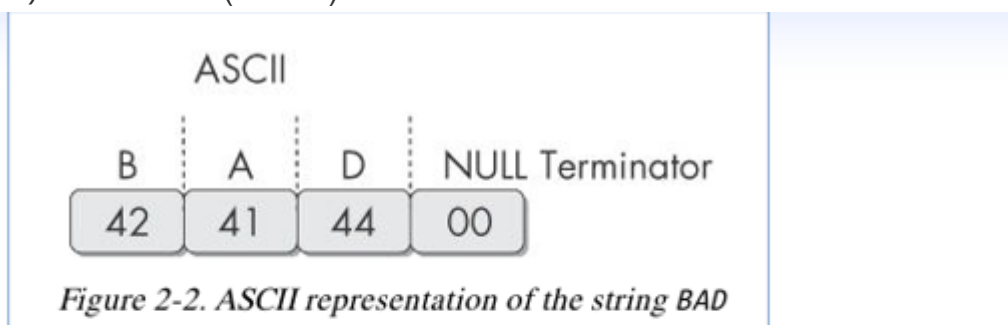
MD5 -消息摘要算法

SHA1 -安全哈希算法（其实不安全 只是比MD5安全而已）

计算工具 HashCalc

字符串常用的编码形式：

ASCII（8 bits） UNICODE（16 bits）



字符串隐藏的方法：加密 加壳 压缩 混淆

常用的加壳手段：UPX 检测加壳软件：PEiD（有安全隐患）

加壳技术和混淆技术有哪些作用？

- ☒ A Compress file size
- ☒ B Hide URL and IPs
- ☒ C Conceal significant strings
- ☐ D Change code behaviors

加壳不改变恶意代码实际功能 只是压缩了体积和隐藏重要信息

Q:文件头中有哪些信息可以作为恶意代码的特征？

PE文件：

在Windows上执行的每一个文件都是PE文件格式的 PE文件包含加载二进制可执行文件所需要的全部信息

PE文件头包含：

- 有关代码的信息
- 应用类型
- 必需的库函数
- 空间要求

在PE文件头中可以提取到哪些信息？

- ☒ A Type of application
- ☒ B Required library functions
- ☒ C Space requirements
- ☒ D Code entry point

导入表和导出表的函数都能在PE文件头中看到 根据这些函数我们可以推测出恶意代码的功能 如果看到导入函数和导出函数过少 我们就需要结合其他分析方法

库文件有哪些装载的方式?

- ☒ A Static
- ☒ B Runtime
- ☒ C Dynamic
- ☐ D Obfuacated

下面哪些函数可以被加壳代码用来动态加载其它的API函数?

- ☒ A LoadLibrary
- ☒ B GetProcAddress
- ☐ C FindFirstFile
- ☐ D ShowWindow

PE文件结构 (重点)

组成部分:

- .text:包含CPU执行指令 一般来说是唯一包含代码的节也是唯一可以执行的节
- .rdata imports & exports
- .data 包含程序的全局数据

- .rsrc 资源节 包含字符串 图像 菜单选项等
 - 时间戳 显示文件的编译时间 但是可以被伪造
- Resource Hacker:破解在资源节中隐藏的图像 或者是字符串 可执行代码等

第三章：Yara引擎

Yara介绍：

识别并分类恶意代码 跨平台

没有特征库 需要自行编写Yara规则

以下对Yara引擎的描述哪些是正确的？

A

可以跨平台

B

可以用来识别和分类恶意代码

C

Yara引擎本身不提供杀毒功能

D

Yara规则由1组字符串和1个布尔表达式构成

Yara语言的规则和C++类似

rule silent_banker : banker tag字段

{ 规则名

meta: 描述信息

description = "This is just an example"

strings: 规则字段

\$a = {6A 40 68 00 30 00 00 6A 14 8D 91}

\$b = {8D 4D B0 2B C1 83 C0 27 99 6A 4E 59 F7 F9}

\$c = "UVODFRYSIHLNWPEJXQZAKCBGMT"

condition:

\$a or \$b or \$c 条件判断字段

}



南大

下面哪一个是无效的规则名称?

A

00_banker

B

Trojan_0x234

C

My_first_rule

D

silent_banker:banker

Yara可以匹配16进制字符串 文本字符串 正则表达式字符串 可以使用?作为通配符

```
rule JumpExample
```

```
{
```

```
strings:
```

```
//使用 '[' 作为跳转，与任何长度为0-2字节的内容匹配
```

```
$hex_string1 = { 00 11 [2] 44 55 }
```

```
$hex_string2 = { 00 11 [0-2] 44 55 }
```

```
//该写法与string1作用完全相同
```

```
$hex_string3 = { 00 11 ?? ?? 44 55 }
```

```
condition:
```

```
$hex_string1 and $hex_string2
```

```
}
```



南开
Nankai U

```
$hex_string = { 00 11 ( 33 44 | 55 | 66 ?? 88 ) 99 }
```

该vara规则可以匹配以下哪几个十六进制串

A

00 11 33 44

B

00 11 55 99

C

00 11 66 77 88

D

00 11 66 56 88 99

nocase 不区分大小写 wide 匹配宽字符串 wide ascii 表示两者连用 xor匹配异或后的字符串 base64匹配
base 64编码的字符串

\$wide_string = “nankai” fullword, 以下哪些字符串会匹配 \$wide_string

A

www.nankai.edu.cn

B

ilovenankai

C

i-nankai

D

nankai university

\$a = “text1”, \$b = “text2”, \$c = “text3” , \$d = "text4"
condition:

(\$a or \$b) and (\$c or \$d)

下面哪个选项可以被规则匹配上？

A

text1 text2

B

text1 text3

C

text1 text4

D

text3 text4

strings:

```
$a = "dummy1"
```

```
$b = "dummy2"
```

condition:

//a字符串出现6次，b字符串大于10次

```
#a == 6 and #b > 10
```



at 指定字符串匹配的位置

- at 匹配字符串在文件或内存中的偏移

strings:

```
$a = "dummy1"
```

```
$b = "dummy2"
```

condition: //a和b字符串出现在文件或内存的100和200偏移处

```
$a at 100 and $b at 200
```



in 指定字符串匹配的范围

- **in** 在文件或内存的某个地址范围内匹配字符串

strings:

```
$a = "dummy1"
```

```
$b = "dummy2"
```

condition:

```
$a in (0..100) and $b in (100..filesize)
```

filesize 文件大小匹配

- **filesize** 匹配文件大小

condition:

//filesize只在文件时才有用，对进程无效

//KB MB后缀只能与十进制大小一起使用

```
filesize > 200KB
```

entrypoint 入口点匹配

- 匹配PE或ELF文件入口点(高版本使用PE模块的 pe.entry_point 代替)

strings:

\$a = { E8 00 00 00 00 }

condition:

\$a at entrypoint



清华大学

判断文件是否是PE文件:

```
private global rule FileSizeAndIsPE {  
  condition:  
    uint16(0)== 0x5A4D and  // MZ 头  
    uint32(uint32(0x3C))== 0x00004550  // PE 头  
}
```

of 匹配部分字符串

- 匹配多个字符串中的某几个

strings:

\$a = "dummy1"

\$b = "dummy2"

\$c = "dummy3"

condition: //3个字符串只需匹配任意2个

2 of (\$a, \$b, \$c)



南開大

for 多字符串匹配

- `for AAA of BBB : (CCC)`
- 在BBB字符串集合中，至少有AAA个字符串，满足了CCC的条件表达式，才算匹配成功。

`for 1 of ($a, $b, $c) : (# > 3)`

//至少1个字符串在文件或内存中出现的次数大于3

any、all、them 多字符串匹配

- 在条件表达式中，可以使用\$依次代替字符串集合中的每一个字符串，#表示字符串的出现次数
- `for 1 of ($a, $b, $c) : ($ at entrypoint)`
- `for any of ($a, $b, $c) : ($ at entrypoint)`
- `for all of them : (# > 3)`

`$a = "55 8B EC"`

`for all of ($a*) : (@[2] < 0x400000)`，写出该表达式什么情况下匹配成功？

所有以\$a开头的字符串，在文件或内存中第2次出现的位置必须小于0x400000

for-in 多字符串匹配

- for AAA BBB in (CCC) : (DDD)

- 作用与for of类似，增加了下标变量与下标范围

```
for all i in (1, 2, 3) : ( @a[i] + 10 == @b[i] )
```

\$a每次在文件或内存中出现位置，都必须小于100。

（使用for-in表达方式来描述）

```
for all i in (1..#a) : ( @a[i] < 100 )
```

第四章 虚拟机技术

在物理机上动态分析计算机病毒有哪些缺点？

☒ A 难以清除计算机病毒

☒ B 对计算机造成破坏

☐ C 可控性好

☒ D 无法连接网络

在虚拟机上进行计算机病毒动态分析有哪些优点

- ☒ A 与主机隔离
- ☒ B 可控性好
- ☒ C 可以快速恢复计算机的状态
- ☐ D 虚拟机逃逸

虚拟机的优点：隔离性 可恢复性 可控性

缺点：虚拟机逃逸 以及安全风险（虚拟机软件的安全漏洞）

计算机病毒动态分析为什么选择Windows XP作为虚拟机的操作系统？

- ☐ A 没有病毒运行在Windows XP上
- ☒ B 恶意代码的攻击目标
- ☒ C 兼容性更好
- ☒ D 体积小、安装快

XP作为恶意代码的攻击目标 兼容性好 体积小 安装快

VMware虚拟机的网络连接方式有哪些？

A

Host-Only

B

Bridge

C

NAT

D

WiFi

第五章 基本动态分析

为什么要进行动态分析：

- 1.静态分析可能会因为包装 混淆而变得无效
- 2.动态分析可以帮助我们更直接地看出恶意代码的行为和作用

以下哪项可以用来隐藏程序中的字符串信息？

A

UPX

B

PEiD

C

ELF

D

DLL

沙箱

优点：基本动态分析多合一 方便快捷

缺点：价格昂贵 缺少命令交互 运行环境单一 分析时间限制 反虚拟机技术

沙箱的优点？

- ☐ A No command-line options
- ☒ B All-in-one software for basic dynamic analysis
- ☐ C Not record all events
- ☐ D Fixed Environment

动态链接库程序

exe程序可以直接运行 但是dll文件不行 但可以使用 rundll32.exe 来运行,例如：
rundll32.exe rip.dll install 来安装程序 也可以修改PE头使DLL变为EXE

服务的安装与启动

安装服务 rundll32 ipr32.dll InstallService ServiceName

启动服务 net start ServiceName

服务与应用的区别

- 功能 在后台运行的长期服务
- 用户交互 不直接与用户进行交互
- 权限 以管理员权限运行
- 生命周期 在系统启动时开始 直到系统关闭或用户手动关闭
- 运行模式 后台进程 对用户不可见

如何启动服务？

- ☒ A net start ServiceName
- ☐ B rundll32.exe abc.dll #num arguments
- ☐ C rundll32.exe abc.dll function arguments
- ☐ D rundll32.exe abc.dll InstallService
ServiceName

进程监视器：监视进程的活动 包括注册表 文件系统 网络 线程活动等 可以执行过滤

Process Monitor可以动态监控哪些进程的操作行为？

- ☒ A Registry
- ☒ B File system
- ☒ C Network
- ☒ D Process
- ☒ E Thread

进程浏览器：显示各个进程的运行状况（用颜色来表示）以及相互之间的关系 还可以看出进程装载的DLL库 字符串 数据预处理状态（DEP）ASLR状态

regShot:监控注册表状态

windows注册表的功能：系统设置 应用程序设置 硬件设置 安全和权限控制 系统状态和日志

wireshark:开源网络嗅探器

捕获数据包 记录网络流量 提供可视化 数据包分析

以下哪些技术可以用于计算机病毒网络行为的动态分析？

- ☒ A Sniffing Traffic
- ☒ B Simulating Services
- ☒ C DNS Spoofing

Which tool reveals the services hosted by scvhost?

- ☐ A Procmon
- ☒ B Process Explorer
- ☐ C Dependency Walker
- ☐ D Regshot
- ☐ E InetSim

Which tool is specifically intended to aid analysis of the network traffic from malware?

- ☐ A Procmon
- ☐ B Process Explorer
- ☐ C Dependency Walker
- ☐ D Regshot
- ☒ E INetSim

第六章 IDA python

IDA Python可以读取以下哪些信息？

- ☒ A 当前内存地址
- ☒ B 节 (segment) 信息
- ☒ C 指令助记符
- ☒ D 指令操作数 (operand)

基本块的定义参考编译原理 只有一个入口和出口

在一个基本块的指令序列中，除了最后一条指令，对于中间的指令，以下哪些指令是不会出现？

A jmp

B je

C call

D ret

具体的IDA python编写请参考复习课件和实验代码

第七章 Windows 恶意代码（重点）

windows API -windows应用编程接口

匈牙利命名法 属性+类型+对象描述

属性: g_ 全局变量 m_ C++类成员变量 s_静态分析

类型:

- w 单字 16位无符号
- dw 双字 32位无符号
- h 句柄 指向对象
- lp 长指针 指向另一种类型的指针
- Callback 被API调用的函数

Windows API编程中，一个变量名为pszMyString，该变量的数据类型是

- ☒ A 字符串指针
- ☐ B 句柄
- ☐ C 注册表键值
- ☐ D 类

句柄是对象生成的 在程序的一次运行中对象的句柄是不变的 通过句柄可以调用windows API对对象的操作

What you can do with a handle?

- ☒ A store it
- ☒ B refer to an object
- ☐ C arithmetic operation
- ☐ D pointer operation

Windows APIs use DWORD for ()

- ☒ A unsigned 32-bit integer
- ☐ B unsigned 16-bit integer
- ☐ C pointer
- ☐ D char

Windows注册表 (重点)

Windows注册表存储了Windows系统和应用程序的设置信息

键 (Key) :类似文件夹 可以有子键

值 (Value) :类似具体文件

HKEY_LOCAL_MACHINE HKLM 保存对当前机器的全局设置

HKEY_CURRENT_USER HKCU 保存对当前特定用户的设置

“HKEY_LOCAL_MACHINE\Software\Microsoft\Windows”

Which is the ROOT key?

- ☒ A HKEY_LOCAL_MACHINE
- ☐ B Software
- ☐ C Microsoft
- ☐ D Windows

病毒使用注册表实现自启动(重点)

- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run
- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce
- HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run
- HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnce

Run RunOnce

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\CurrentVersion\Run

Which API suffix means the API could accept ASCII string as parameter

☒ A

☐ B

☐ C Ex

Ex 代表扩展函数 A代表可以接受ASCII字符的参数

W 代表可以接受Wide 字符的参数

网络API(重点)

ws2_32.dll socket编程的函数库

socket listen bind accept connect send recv

不会的翻计网PPT

A compromised PC is remotely controlled by attacker. Which side is the compromised PC in the network connection?

- ☐ A Server side
- ☒ B Client side

Which functions are used by network client side?

- ☒ A socket
- ☐ B bind
- ☐ C listen
- ☒ D connect

WinINetAPI --比Winsock更高级的API

使用库：Wininet.dll

- InternetOpen 连接网络
- InternetOpenURL 连接到URL
- InternetReadFile 从下载文件读取数据
- **恶意代码常用的WindowsAPI(重点)**
- CreateFile ReadFile WriteFile
读写文件三件套
- CreateFileMapping MapViewOfFile
文件从硬盘转移到内存
- RegOpenKeyEx RegSetValueEx RegGetValue
注册表操作三件套
- InternetOpen InternetOpenURL InternetReadFile
网络三件套（实际上还有更多）

- CreateProcess 创建线程
CreateThread 创建线程 虚拟保护 虚拟内存分配 创建线程
- CreateMutex OpenMutex ReleaseMutex WaitForSingleObject
互斥锁管理一系列API
- CreateService StartService OpenSCManager
与服务相关的一系列API

跟踪恶意代码的运行

控制流跳转 -jmp及相关指令 call指令

DLL -动态链接库 在多个应用程序之间共享代码

优势： 占用空间小 可移植性强

DLL的结构同EXE的结构很相似 标志指示位不同 DLL具有更多的导出和更少的导入

DLLMain 是主函数未导出 指定为PE头的入口点 通常在函数加载和卸载库时使用

进程与线程

每个应用程序都含有1个或多个进程 每个进程都有自己的资源 且具有一个或多个线程 每个进程都会消耗CPU 文件系统 内存等资源 操作系统为每个进程都分配了不同的内存 线程是操作系统分配CPU时间的基本单元

恶意代码编写者创建两个线程和正在运行的应用程序进行通信

输入： 监听进程的套接字或管道

输出： 从进程的套接字中读取

Which is the basic unit for CPU time?

- ☐ A Application
- ☐ B Process
- ☒ C Thread
- ☐ D Function

When the OS switches to another thread, it saves all CPU values in a structure called the **thread context**

Which item is shared between threads in one process?

- ☒ A Memory space
- ☐ B Stack
- ☐ C Register
- ☐ D Thread context

Malware uses **mutex** to make sure there is only one copy of Malware is running **svchost.exe(重点)**

恶意代码中最常用的服务类型 将服务代码储存在一个DLL中 将多个服务代码合并到一个svchost.exe的共享进程中

Which service type uses svchost.exe to run services?

- ☐ A KERNEL_DRIVER
- ☒ B WIN32_SHARE_PROCESS
- ☐ C WIN32_OWN_PROCESS

When an exception occurs, execution transfers to the SEH(异常处理结构)

内核模式 and 用户模式

无法直接从用户模式转换到内核模式 常通过sysenter syscall int 0x2E函数来查找

内核模式：所有线程共享资源和内存地址

更少地安全检查 如果内核代码执行无效指令 那么操作系统将蓝屏死机 防火墙和部分反病毒软件都是在内核模式下运行的 内核模式下的恶意代码 **rootkit**

Which privilege levels are used by Windows OS?

- ☒ A Ring 0
- ☐ B Ring 1
- ☐ C Ring 2
- ☒ D Ring 3

Which privilege level could access hardware?

- ☒ A Ring 0
- ☐ B Ring 1
- ☐ C Ring 2
- ☐ D Ring 3

Which is the kernel-mode malware?

- ☐ A Worm
- ☐ B Trojan
- ☐ C Backdoor
- ☒ D Rootkit

原生API Native API (重点)

特点：比windows更低级的交互界面 在恶意代码中使用较少 受恶意代码编写者的欢迎

NativeAPI存放在Ntdll.dll中

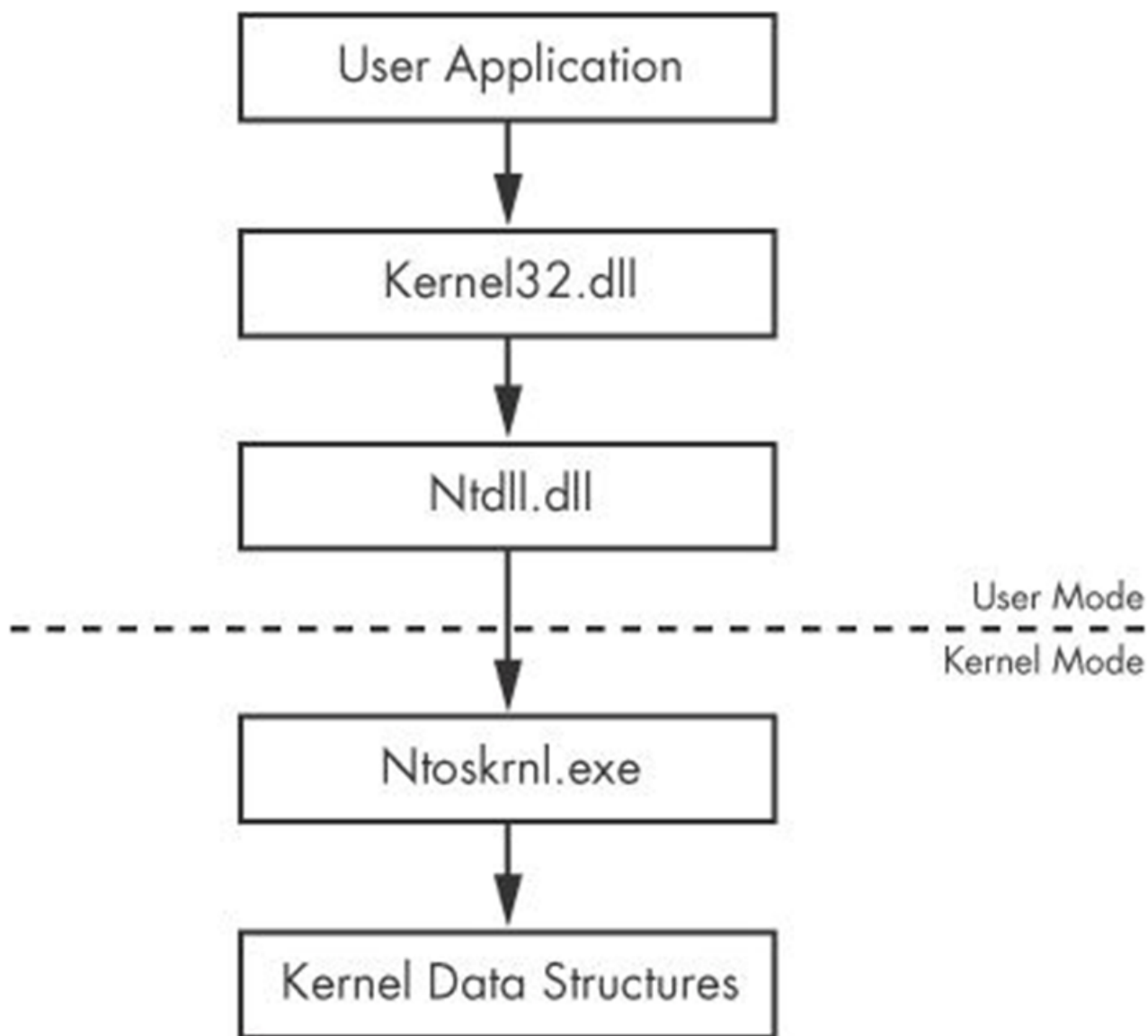


Figure 8-3. User mode and kernel mode

Native API没有官方文档 供Windows内部使用 可供程序使用 调用更强大更隐蔽
Windows API有官方文档 供全体开发者使用 也可以供程序使用 调用时迹象明显

- NTtQuerySystemInformation
- NTtQueryInformationProcess
- NTtQueryInformationThread
- NTtQueryInformationFile
- NTtQueryInformationKey

- Provide much more information than any available Win32 calls

Which dll is undocumented?

- ☐ A kernel32.dll
- ☒ B ntdll.dll
- ☐ C ws2_32.dll
- ☐ D wininet.dll

Which dll is rarely used by normal programs, but popular for malware?

- ☐ A kernel32.dll
- ☐ B user32.dll
- ☒ C ntdll.dll
- ☐ D wininet.dll

第八章 动态调试

调试器能够帮助我们以动态的视角理解程序的运行

两种调试器：

OillyDBG：用户模式下的动态调试器

WinDBG：支持内核模式下的调试

Which item below could view a sequence of CPU instructions execution ?

- ☐ A Source-level debugger
- ☒ B Assembly-level debugger
- ☐ C Disassembler
- ☐ D ProcessMonitor

汇编级调试器能看到具体的CPU指令

用户模式和内核模式的区别（重点）

- 特权级 内核模式的权限更高
- 执行指令 内核态能执行特权态指令
- 访问硬件 内核态能直接访问硬件和寄存器
- 任务数量 内核模式单任务 但用户模式可以多任务

Which item below must require two computers connected together?

- ☐ A Source-level debugger
- ☒ B Kernel-mode debugger
- ☐ C User-mode debugger
- ☐ D Assembly-level debugger

Which item below is almost never used by malware analysts?

- ☒ A Source-level debugger
- ☐ B Assembly-level debugger
- ☐ C User-mode debugger
- ☐ D Kernel-mode debugger

Which item below is suggested by Windows automatically after it crashes?

- ☐ A Source-level debugger
- ☐ B Assembly-level debugger
- ☐ C User-mode debugger
- ☒ D Kernel-mode debugger

Which item might miss important functionality?

- ☐ A single-step
- ☒ B step-over
- ☐ C step-into
- ☐ D step-out

Which item will stop debugger at the first instruction of the called function?

☒ A single-step

☒ B step-into

☐ C step-over

☐ D step-out

Which item will run after the function returns?

☐ A single-step

☐ B step-into

☒ C step-out

☐ D step-over

Which item will stop at the first function after call instruction?

- ☐ A single-step
- ☐ B step-into
- ☒ C step-over
- ☐ D step-out

What type of breakpoint uses Interrupt #3?

- ☒ A software breakpoint
- ☐ B hardware breakpoint
- ☒ C conditional breakpoint
- ☐ D single step

What type of breakpoint may make a program run slowly?

- ☐ A software breakpoint
- ☐ B hardware breakpoint
- ☒ C conditional breakpoint

What type of breakpoint changes the binary code?

- ☒ A software breakpoint
- ☐ B hardware breakpoint
- ☒ C conditional breakpoint

What type of breakpoint uses the DR registers?

- ☐ A software breakpoint
- ☒ B hardware breakpoint
- ☐ C conditional breakpoint

做多设置几个硬件中断?

- ☐ A 1
- ☐ B 2
- ☒ C 4
- ☐ D 8

Which item below handles exceptions during normal program execution and debugger attached?

- ☒ A First chance
- ☒ B Second chance
- ☒ C SEH
- ☐ D INT 3

Which item could cause an exception?

- ☒ A Access violation
- ☐ B Division by zero
- ☐ C Stack overflow
- ☐ D Breakpoint

Which item is used for single-stepping?

- ☐ A first chance
- ☐ B second chance
- ☐ C SEH
- ☐ D INT 3
- ☒ E trap flag

Which type of exception usually is ignored for malware analysis?

- ☒ A First chance
- ☐ B Second chance
- ☐ C SEH
- ☐ D INT 3
- ☐ E trap flag

A ring 3 process tries to access hardware directly. What exception will be thrown?

- ☐ A First chance
- ☐ B Second chance
- ☐ C /0
- ☐ D Invalid memory access
- ☒ E Privilege violation

Using binary modification, which operation we could do?

- ☒ A skip a function
- ☒ B test a function
- ☒ C software cracking
- ☒ D code reuse

How to force the malicious code to run different path without changing the settings on our system?

- ☒ A Modify instruction pointer
- ☒ B Change Windows API return value
- ☐ C Breakpoint
- ☐ D Set trap flag

第九章 WinDBG内核调试（重点）

驱动 本质上是一个软件组件 实现操作系统与设备间的通信 驱动程序一般由设备开发商开发
具体的访问过程：APP通过WIN API请求访问设备

Windows将访问请求转发给驱动程序 调用驱动程序提供的驱动函数 最后由驱动程序完成对设备的访问
一次设备的访问可能要经过多个驱动 这就涉及到驱动栈的概念

驱动栈包括过滤驱动（杀毒软件 防火墙等）和功能驱动（一个设备栈最多只有一个）

过滤驱动有两类：上层过滤驱动盒下层过滤驱动 分别位于功能驱动的前后

设备 与计算机连接的外部设备或虚拟设备

Windows用设备树来管理设备

每个设备节点是一个有序的设备对象 每个设备对象关联一个驱动

设备是物理设备的软件表示

下面那个选项可以被用户空间的应用程序直接访问？

- ☐ A 物理设备 physical hardware
- ☐ B 设备驱动 device driver
- ☒ C 设备对象 device object
- ☐ D Windows 内核

驱动必须被装进内核态中

驱动通过DriverEntry函数被装载

Which are supported by WinDbg?

- ☒ A user-mode debugging
- ☒ B kernel-mode debugging
- ☒ C rootkit debugging
- ☒ D application debugging

Which statements are true for **driver**?

- ☒ A creates and destroys device objects
- ☒ B loaded into kernel
- ☐ C can be accessed from user space
- ☒ D has a DriverEntry procedure

Which statements are true for DriverEntry?

- ☒ A a procedure in driver
- ☒ B registers callback functions
- ☒ C creates device
- ☒ D initializes driver object structure

Which items are usually manipulated by malicious drivers?

- ☐ A kernel32.dll
- ☐ B hardware
- ☒ C ntoskrnl.exe
- ☒ D hal.dll

WinDBG的使用

在WinDBG使用之前 必须配置好环境

在虚拟机中启用内核调试


在主机中配置WinDBG环境


在虚拟机和主机之间设置虚拟串行端口


常见命令


- `lm` 列举当前内存空间装载的所有模块
- `!dh` 查看指定模块的PE文件头信息
- `bp` 设置断点
- `g` 恢复程序的执行 断点处中断程序执行
- `u` 显示当前地址的反汇编码
- `d*` 读内存 `du` Unicode编码 `da` ascii编码 `db` 16进制和二进制编码
- `e*` 写内存 `ea` ascii编码写入 `eu` Unicode编码写入 `eb` 16进制字节写入
- `p` 执行下一条指令 `pc` 执行到下一条call指令 `pt` 执行到下一条ret指令 `pct` 执行到下一条call或ret指令
- `dp` 显示指针指向的内存信息 `dp@esp` 显示栈信息
- `dpa@esp` 以字符串形式展示 `dps@esp` L4以符号的形式展示 显示4行
- `|` 显示当前进程ID和进程名
- `~` 当前进程中所有线程的状态

How to read a Unicode string starting at 0x00402000

 A `da 0x00402000`

 B `du 0x00402000`

 C `ea 0x00402000`

 D `eu 0x00402000`

Which command could write a ASCII string into memory?

- ☐ A da
- ☒ B ea
- ☐ C du
- ☐ D lm

How to set a breakpoint at LoadLibrary?

- ☒ A bp LoadLibrary
- ☐ B ex LoadLibrary
- ☐ C lm LoadLibrary
- ☐ D da LoadLibray

Which command could continue execution after breakpoint?

- ☒ A g
- ☐ B bp
- ☐ C da
- ☐ D eu

Which module's name is nt in WinDbg?

- ☐ A kernel32.dll
- ☒ B ntoskrnl.exe
- ☐ C hal.dll
- ☐ D ntdll.dll

How to list closest symbol at a specified address?

- ☒ A ln
- ☐ B lm
- ☐ C du
- ☐ D dt

How to view a structure information at a specified memory address?

- ☒ A dt
- ☐ B da
- ☐ C du
- ☐ D dd

Which function is called first when a driver is loaded?

- ☒ A DriverInit
- ☐ B DriverEntry
- ☐ C DllMain
- ☐ D WinMain

Rootkit(重点)

Rootkit能修改操作系统内部的功能以达到隐藏自己的目的 大部分rootkit都是在内核态运行的

最常用的方法：系统服务描述表 (SSDT) 挂钩

SSDT 由微软内部使用 在内核中查找函数调用 通常不由第三方程序和驱动程序应用
三种从用户态跳转到内核态的方法：

- syscall
- sysenter
- int 0x2e

rootkit会更改SSDT中的某一项值为恶意代码的函数

常用的挂载函数 **NtCreateFile**

最简单检测SSDT hooking的方法：检查SSDT 观察是否有不合理的值

WinDBG 指令： **!m m nt**

中断描述表 (IDT)

允许硬件触发软件中断

驱动程序调用 **IOConnectInterrupt**来注册中断处理程序

指定中断服务例程 ISR

IDT中储存着ISR的信息

WinDBG指令： **!idt**

第十一章 恶意行为

下载器

下载另一份恶意代码

API: URLDownloadToFileA

在本地系统上执行 WinExec

启动器

准备另一份恶意代码的运行 立即或将来执行

包含在恶意代码中 比如存放在PE文件的资源节

以下哪类恶意代码从互联网上下载其它恶意代码?

- ☐ A Backdoor
- ☐ B Trojan
- ☒ C Downloader
- ☐ D Launcher

以下哪类恶意代码的文件中会包含其它恶意代码?

- ☐ A Backdoor
- ☐ B Downloader
- ☒ C Launcher
- ☐ D Trojan

后门

提供对受害机器的远程访问 不需要下载其他恶意代码

目前最常见的恶意代码

常用软件: netcat

Reverse Shell: 感染者呼叫攻击者要求执行命令

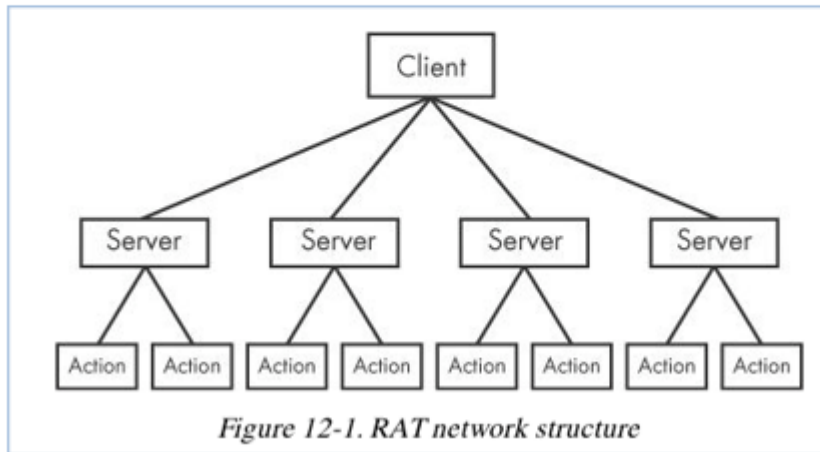
使用cmd.exe作为反向shell 有两种方法:

传统方法: CreateProcess

多线程版本: CreateThread CreatePipe

RATs

(Remote Administration Tools)



RAT和僵尸网络的区别

使用RAT只能控制少量主机 因为它要求攻击者和感染主机有更多的交互 针对性攻击
僵尸网络能控制大量主机 所有主机都在同一时刻被操控 进行大规模攻击

For netcat reverse shell, "nc -l -p 80" is running on
attacker machine or victim machine?

- ☒ A attacker machine
- ☐ B victim machine

以下哪种恶意代码类型可以发起大规模网络攻击?

- ☐ A Backdoor
- ☐ B RAT
- ☒ C Botnet
- ☐ D Launcher

凭证窃取的三种方法：

- 等待用户登录窃取
- 转存Windows中的存放信息的数据 如密码哈希值
- 击键记录器

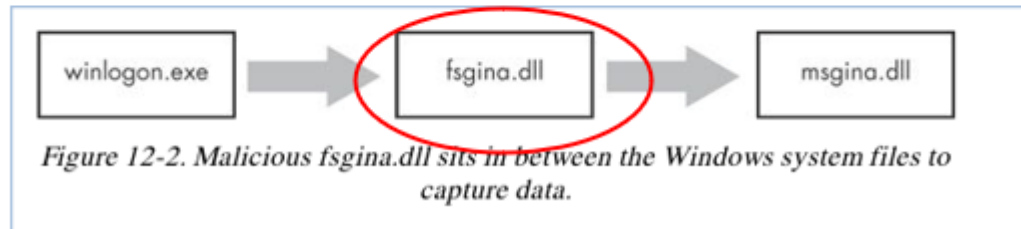
Gina是在msgina.dll中实现的

登录期间由Winlogon可执行文件加载 WinLogon会在winLogon和GINA中加载第三方选项

- HKLM\SOFTWARE\Microsoft\Windows

NT\CurrentVersion\Winlogon\GinaDLL

- Contains third-party DLLs to be loaded by WinLogon



msgina.dll中的函数大部分以wlx开头 多数导出函数都是msgina.dll中真实的函数 需要我们找到恶意函数

Windows的登录密码转存为LM或者是NTLM的哈希

LSASS本地安全认证子系统服务

pwdump利用DLL注入到lsass.exe中 调用GetHash获取每个用户未解密的口令哈希值

击键记录器

使用API: GetAsyncKeyState GetForegroundWindow

以下哪种技术被恶意代码用于凭证窃取？

- ☒ A Hash dumping
- ☐ B Keystroke logging
- ☒ C GINA interception
- ☐ D RATs

Which is the right logon credentials flow?
(fsgina.dll is a malicious file for credential stealing)

- ☐ A winlogon – msgina -- fsgina
- ☐ B fsgina – msgina -- winlogon
- ☒ C winlogon – fsgina -- msgina
- ☐ D msgina – fsgina -- winlogon

For hash dumping, Pwdump and PSH should inject a malicious dll into which process?

- ☒ A LSASS
- ☐ B svchost
- ☐ C SAM
- ☐ D Registry

Which ways are used for user-space keylogger?

- ☒ A polling
- ☐ B driver
- ☒ C hooking
- ☐ D hash dumping

持久性驻留机制

修改注册表 Run RunOnce RunServices RunServicesOnce ShellFolders

以下哪项注册表与持久化相关？

- ☒ A HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce
- ☐ B HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
- ☐ C HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explore\Shell Folders
- ☐ D HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\RunServices

以下哪些选项会阻断恶意代码的执行，需要用到持久化机制？

- ☒ A 系统重启
- ☒ B 登录用户的注销
- ☒ C 进程崩溃
- ☐ D 程序异常处理

Applnit_DLLs

Applnit_DLLs会在加载user32.dll时就被加载，可以插入DLL路径到该路径下就会获得加载的机会
注册表键值的最后一位为svchost

恶意DLL注册到Applnit DLLs，当进程装载以下哪个动态链接库时，恶意DLL会一同被装载到进程中？

- ☐ A kernel32.dll
- ☒ B user32.dll
- ☐ C ntdll.dll
- ☐ D Ws2_32.dll

WinLogon Notify

键值的末位为WinLogon

允许恶意代码在安全模式下被加载

当运行winlogon.exe产生一个事件 操作系统就会寻找Notify的注册表键来加载相应的DLL

以下哪项是基于Windows用户登录和注销事件，实现持久化机制？

- ☐ A 注册表的自启动项
- ☐ B 注册表中的Applnit DLLs
- ☒ C 注册表中的Winlogon Notification
- ☐ D 注册表中的启动目录 (Startup Folder)

以下哪种持久化机制在Windows系统的安全模式下仍可以使用？

- ☐ A 注册表的自启动项
- ☐ B 注册表的AppInit_DLLs
- ☒ C 注册表的Winlogon Notification
- ☐ D 注册表中的启动目录（Startup Folder）

以下注册表项与SvcHost持久化机制相关的是？

- ☒ A HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Svchost
- ☒ B HKEY_LOCAL_MACHINE\ System\ CurrentControlSet\Services
- ☐ C HKEY_LOCAL_MACHINE\ SOFTWARE\ Microsoft\ Windows
- ☐ D HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run

DLL默认搜索顺序

- 加载该程序的目录
- 当前目录
- 32位系统目录
- 16位子系统目录
- Windows目录
- PATH环境变量列举的目录

以下哪些技术可以被恶意代码用于持久化机制？

- ☒ A Winlogon Notify
- ☒ B AppInit DLLs
- ☒ C Trojanized system binaries
- ☒ D Dll load-order hijacking

权限提升

方式：DLL装载劫持 提权函数等

提权函数：

SeDebugPrivilege 函数允许管理员释放管理员权限

AdjustTokenPrivilege 提升权限

以下哪些技术可以被恶意代码用于权限提升？

- ☒ A DLL装载顺序劫持
- ☒ B 使用AdjustTokenPrivilege函数设置SeDebugPrivilege
- ☐ C SvcHost服务
- ☐ D 启动目录

用户态Rootkit

IAT 导入函数表

EAT 导出函数表

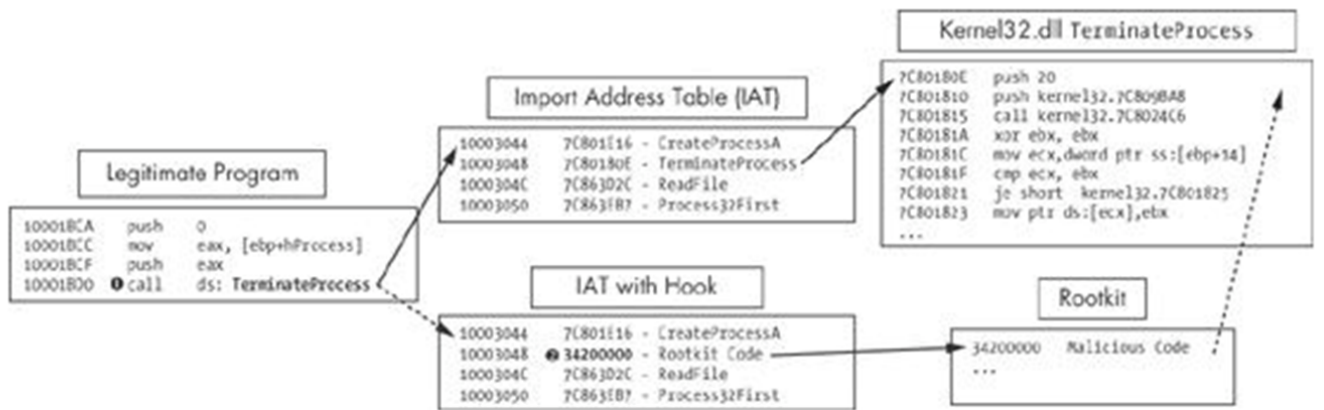


Figure 12-4. IAT hooking of TerminateProcess. The top path is the normal flow, and the bottom path is the flow with a rootkit.

本质上是替换IAT 使其执行恶意代码 简单但容易被发觉
内联Hook修改实际的函数代码 IAT Hook只修改函数指针

Which hook mechanism modifies function code?

- ☒ A Inline hook
- ☐ B IAT hook
- ☐ C Kernel rootkit

Which hook mechanism modifies pointer at user memory space?

- ☐ A Inline hook
- ☒ B IAT hook
- ☐ C Kernel rootkit

Which hook mechanism modifies pointers at kernel memory space?

- ☐ A Inline hook
- ☐ B IAT hook
- ☒ C Kernel rootkit

第十二章 隐蔽执行技术

启动器的作用：

设置自身或其他恶意软件

用于立即或将来执行

向用户隐藏自身行为

启动器的位置：通常位于资源节或PE覆盖层部分

启动器（Launcher）的功能包括哪些？

- ☒ A 解密
- ☒ B 隐蔽启动
- ☒ C 权限提升
- ☒ D 解压缩

进程注入（重点）

将恶意代码注入到正在运行的进程中

优点：隐藏自身恶意行为 绕过安全监测

常用的API函数：

- **VirtualAllocEx** 分配内存空间
- **WriteProcessMemory** 写内存

注入方式：

- DLL注入
- 直接注入

DLL注入：最常用的注入方式 把代码注入到调用LoadLibrary的远程线程中 加载时操作系统会自动调用包含恶意代码的DLLMain

注入的恶意代码拥有和被注入的代码相同的权限

DLL注入的步骤：

1.进程遍历：

- 寻找注入目标 **CreateToolhelp32Snapshot Process32First**
- 检索进程PID
- 获取句柄 **OpenProcess**

2.创建远程线程：

- **CreateRemoteThread**
- **VirtualAllocEx**
- **WriteProcessMemory**

分析步骤对应：

- 1.找到感染的进程名
- 2.找到恶意DLL名
- 3.识别注入模式

直接注入：不依赖于DLL 直接把恶意代码注入到远程线程中

常用的API：

WriteProcessMemory VirtualAllocEx

CreateRemoteThread LoadLibrary

GetProcAddress

进程注入的优点有哪些？

- ☒ A 获得被注入进程的权限
- ☒ B 伪装恶意行为
- ☒ C 可以有效穿透防火墙、躲避检测工具
- ☐ D 影响被注入进程的正常执行

进程注入需要用到的API函数有哪些？

- ☒ A VirtualAllocEx
- ☒ B WriteProcessMemory
- ☒ C CreateRemoteThread
- ☒ D OpenProcess

进程替换（重点）

将恶意代码替换到受感染进程的内核空间 通常替换svchost.exe

挂起状态下 进程将被加载到内存中 但是主线程被挂起

一般来说进程替换会使用 **ZwUnmapViewOfSection** 释放该节的内存

SetThreadContext 设置恶意代码线程的上下文

ResumeThread 运行恶意代码

以下哪项需要用到进程的Suspended状态？

- ☐ A Privilege escalation
- ☒ B Process replacement
- ☐ C DLL injection
- ☐ D Direct injection

以下哪项技术需要大量的定制化代码（customized code）？

- ☐ A Privilege escalation
- ☐ B Process replacement
- ☐ C DLL injection
- ☒ D Direct injection

以下哪项技术需要调用FindResource函数?

- ☐ A Privilege escalation
- ☐ B Process replacement
- ☐ C DLL injection
- ☐ D Direct injection
- ☒ E Resource extraction

以下哪项技术需要调用CreateRemoteThread 函数?

- ☐ A Privilege escalation
- ☐ B Process replacement
- ☒ C DLL injection
- ☒ D Direct injection
- ☐ E Resource extraction

SetWindowsEx

- lpfn
 - 指向挂钩过程的指针。
- hmod
 - **DLL** 的句柄，其中包含 lpfn 参数指向的挂钩过程。
- dwThreadId
 - 要与挂钩过程关联的线程的标识符
- 返回值 HHOOK
 - 如果函数成功，则返回值是挂钩过程的句柄。
 - 如果函数失败，则返回值为 **NULL**。
 - UnhookWindowsHookEx()

C++

```
HHOOK SetWindowsHookExA(  
    [in] int      idHook,  
    [in] HOOKPROC lpfn,  
    [in] HINSTANCE hmod,  
    [in] DWORD    dwThreadId  
);
```

C++

```
BOOL UnhookWindowsHookEx(  
    [in] HHOOK hhk  
);
```

南开大学
Nankai University

SetWindowsHookEx 常用的Hook API

本地钩子：处理钩子所在进程的消息

远程钩子：处理其他进程的消息

系统钩子（High-level remote hooks）系统开销大

线程钩子（Low-level remote hooks）要求被hook的进程必须安装hook

键盘记录器就是一种典型的hook

SetWindowsHookEx

- Parameters
 - idHook – type of hook to install
 - lpfn – points to hook procedure
 - hMod – handle to DLL, or local module, in which the lpfn procedure is defined
 - dwThreadId– thread to associate the hook with. **Zero = all threads**

注意SetWindowsHook的最后一个参数 如果为0 则hook所有线程

如果对同一个消息安装了多个钩子 则组成一个钩子链

挂钩过程必须调用 **CallNextHookEx** 才能够传递给下一个挂钩进程

同一个消息既有high-level 又有low-level 先执行low-level

Which technique manipulates messages from a process to itself?

- ☒ A Local hook
- ☐ B High-level remote hook
- ☐ C Low-level remote hook
- ☐ D Keylogger

以下哪种方法会挂钩所有的线程?

- ☐ A Local hook
- ☒ B High-level remote hook
- ☐ C Low-level remote hook

以下哪个描述是正确的？

- ☒ A 对于同一事件（如鼠标消息）既安装了线程钩子又安装了系统钩子，那么系统会自动先调用线程钩子，然后调用系统钩子。
- ☐ B 对同一事件消息可安装多个钩子处理过程，这些钩子处理过程形成了钩子链。
- ☐ C 当前钩子处理结束后应把钩子信息传递给下一个钩子函数。
- ☐ D 系统钩子会消耗消息处理时间，降低系统性能。只有在必要的时候才安装钩子，在使用完毕后要及时卸载。

以下哪个描述是正确的？

- ☒ A 钩子函数在完成对消息的处理后，要继续传递消息，必须调用 CallNextHookEx 来传递它，以执行钩子链表下一个钩子函数
- ☐ B 钩子使用 UnhookWindowsHookEx () 卸载
- ☐ C SetWindowsHookEx () 把钩子安装到钩子链表中
- ☐ D 钩子的卸载顺序一定要和安装顺序相反

Detours技术

Detours 是微软的早期库

轻松检测和扩展现有操作系统和应用程序功能

使用无条件转移指令来替换目标函数的最初几条指令，将控制流转移到截获函数。

应用实例：R77

下面哪种隐蔽方式是通过修改文件实现的？

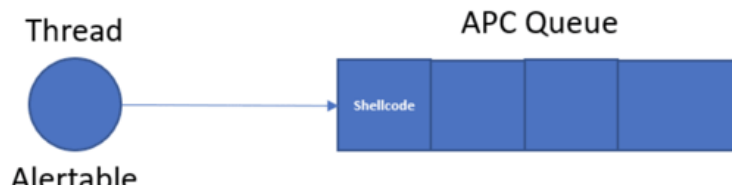
- ☒ A Detour
- ☐ B DLL Injection
- ☐ C Direct Injection
- ☐ D Hooks

APC注入

APC可以指示线程在常规执行之前执行其它代码

每个线程都会有一个APC队列 当线程处于 alterable state 时处理这些APC
用到的API:

- [SleepEx](#), [SignalObjectAndWait](#), [MsgWaitForMultipleObjectsEx](#), [WaitForMultipleObjectsEx](#), [WaitForSingleObjectEx](#)



只有当APC队列的函数执行结束后 线程才会沿着常规执行路径执行

内核模式APC(KAPC)

通常注入到svchost.exe中

用户模式APC(UAPC)

线程必须处于可更改状态 通常使用QueueUserAPC将函数排到远程线程

可更改状态的重要标志: **WaitForSingleObjectEx**

Which technique only works for threads in an alterable state?

- ☐ A Detours
- ☒ B APC
- ☐ C Hook
- ☐ D Injection

APC注入只能用于用户空间的程序，不能用在内核空间。

- ☐ A 正确
- ☒ B 错误

注意可更改状态是指线程被阻塞但是依旧可以接收外部事件的通知

第十三章 数据加密

为什么恶意代码需要加密

- 隐藏配置信息
- 隐藏可疑字符串
- 将自身伪装为合法的工具
- 将窃取的信息暂存

恶意代码通常会对哪些数据进行加密？

- ☒ A 恶意代码的配置信息
- ☒ B 偷取的数据、文件等
- ☒ C 恶意代码用到重要字符串
- ☒ D 伪装成正常软件所需要隐藏的信息

破解恶意代码的数据加密需要完成哪些工作？

- ☒ A 识别数据加密函数
- ☒ B 分析数据加密方法
- ☐ C 找到恶意代码的文件特征码
- ☒ D 解密被加密的数据

简单数据加密有哪些方法？为什么要使用简单加密方法？

凯撒密码 异或密码 ADD SUB base64编码 等

原因：体积小 开销低 不易被察觉

对字符串“HI”进行XOR数据加密，密钥是0x3c，“HI”的ASCII码是0x48 0x49，XOR加密后的数据

是?

0X74 和 0X75

Example: Encode HI with a key of **0x3c**

HI = 0x48 0x49 (ASCII encoding)

Data: 0100 1000 0100 1001

Key: 0011 1100 0011 1100

Result: **0111 0100 0111 0101**

对异或加密来说 一次异或是加密 两次异或就是解密了

缺点: 可能会泄露密钥 0x00 xor key

避免: 如果要加密的明文是0或密文本身则跳过

识别异或加密也很简单 使用IDA pro搜索XOR 然后逐个分析

base64编码 (重点)

所谓base64编码有多种形式 但他们的共同点是使用了26个英文字母的大小写以及10个数字 (0~9) 将6位数据转换为对应的字符 (如果不够6位则用'='填充)

寻找base64编码函数也很简单:

- 1.查找索引字符串 一般是64位
- 2.查找单独的填充字符 (通常是'=')

以下哪些指令可以被恶意代码用于数据加密和解密？

- ☒ A XOR
- ☒ B ADD、SUB
- ☒ C ROL、ROR
- ☐ D MOV

恶意代码常使用哪些简单的数据加解密方案？

- ☒ A 凯撒密码
- ☒ B XOR
- ☒ C Base64
- ☐ D MD5

有哪些常见的高强度加密算法？恶意代码中使用这些算法有哪些优点和缺点？

足够强大抵抗攻击（AES SSL）

优点：抗攻击性强 安全性好

缺点：易被察觉 可移植性差 需要导入函数库 对称加密需要隐藏密钥

检测工具：OpenSSL IDA Pro PEiD

熵：被加密的部分熵大

检测工具：IDA pro

使用库函数进行数据加密有哪些缺点？

- ☒ A 增加恶意代码体积
- ☒ B 降低移动性
- ☒ C 容易被发现
- ☒ D 对称加密需要隐藏密钥

有哪些检测恶意代码是否使用库函数加密数据的方法？

- ☒ A Entropy
- ☒ B 加密相关的常量
- ☒ C 加密相关函数的名字
- ☒ D 加密相关库的名字

Entropy值高的区域还是低的区域有可能是加密数据？

 Entropy值高的区域


 Entropy值低的区域


用户自定义加密 (Custom Encoding)

自定义的加密方法有哪些优点？


优点：体积更小 隐蔽性更高 更难被逆向分析 隐藏密钥

自定义的数据加密算法比使用标准库函数加密有哪些优点？

 增加逆向工程的难度

 结合简单加密方案，执行速度更快，体积小

 可以隐藏密钥

 不能直接找到解密函数

针对恶意代码使用的数据加密方法，逆向分析恶意代码时有哪些有效的解密方法？

两种方法：

1.编写解密函数 功能更强大 但编写困难

2.利用程序自解密 （在调试器中停止恶意软件并解码数据 隔离解密函数 并在其后设置断点 但有时候不知道何时停止）

针对恶意代码的加密数据，有哪些可行的解密方法？

- A** 等恶意代码自解密Self-decoding
- B** 编写解密函数
- C** 通过插装控制恶意代码来解密
- D** 暴力破解

第十四章 恶意代码网络行为分析

常见的网络防御对策：

- 防火墙 路由器
- DNS服务器 将恶意域名解析到内部主机 DNS沉洞技术
- 代理服务器 检测阻止对特定域的访问

DNS沉洞技术

某一个网络域名被判定为有害后 由安全厂商或运营商将其原本解析到的IP地址变更为无害IP地址

- 检测 阻拦有害流量 自动程序以及不需要的流量
- 监测已经失陷的主机数量和状态

代理服务器：代理用户取得网络信息

入侵检测系统（IDS）：

被动监听网络数据 旁路部署

检测可疑流量

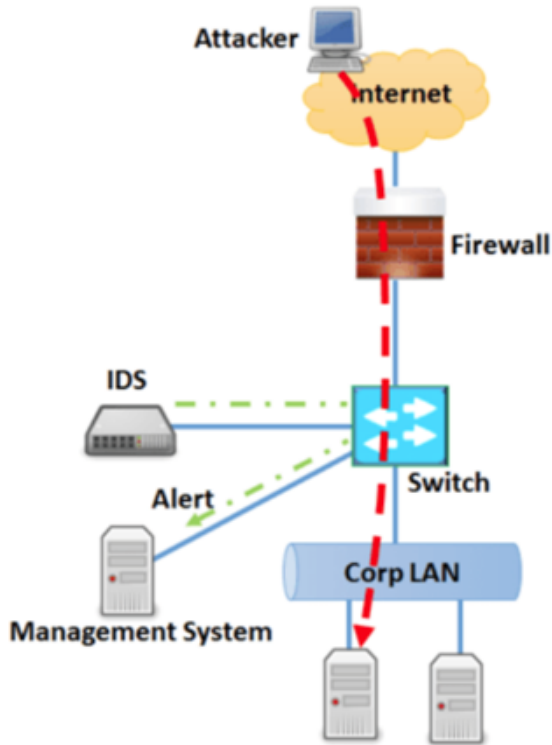
入侵防御系统（IPS）：

主动过滤网络数据 串联部署

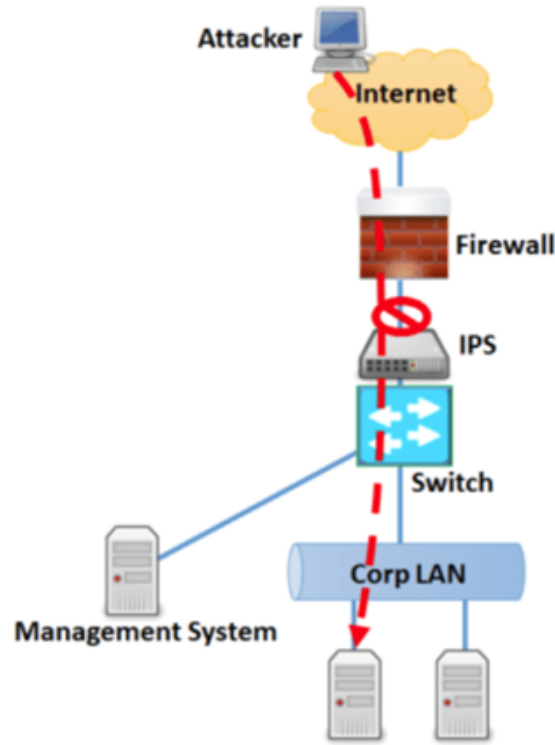
所有流量必须经过该设备才能继续达到目的地

检测到恶意流量后 IPS会中断连接并丢弃会话或流量

Intrusion Detection System



Intrusion Prevention System



代理服务器在网络防御的作用

- 代理服务器可以隐藏真实的IP地址
 - 过滤和阻止恶意流量
 - 访问控制和身份验证 确保只有通过认证的用户才能访问特定受限的资源
- OPSEC 运营安全 从敌手角度查看操作和项目

如何保证分析恶意代码的过程不被发现？

代理服务器 专用虚拟机 弹性云计算

查看恶意域名的网站：VirusTotal RobTex

网络流量中有哪些内容（content）可以用来检测恶意代码的网络行为？

IP地址和端口号 协议 域名 加密流量 TLS握手失败 网络连接持续时间

如何进一步提升恶意代码网络行为的特征质量？

- 多维度特征分析
- 行为模式分析
- 机器学习和人工智能技术
- 实时更新和持续监测
- 合作和信息共享
- 持续改进和优化