Windows interrupt context kernel overflow exploits

FlashSky@xfocus.org
FangXing@venustech.com.cn

- 特别感谢

  KeJi(KeJi@venustech.com.cn)在WINDOWS内核方面的讨论和参与

- 感谢

  感谢ALERT7和焦点所有的成员

  感谢启明星辰积极防御实验室的所有成员

仅以此文献给所有 不甘于平凡、超越极限、追求完美的人员

驱动内核溢出利用初步

- 讨论的范围
  - 内核驱动溢出，而非int 2e等触发的内核溢出。
  - 可本地／远程触发的内核驱动溢出的利用
  - 发生在中断上下文的内核独占模式的驱动溢出，而非进程上下文的，通过DeviceIoControl的内核驱动溢出。

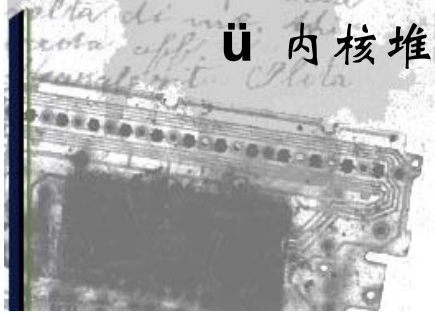
- 涉及的技术
  - 更通用的非挂接中断方式的,驱动调用安全返回／非安全返回的利用技术
  - WINDOWS内核中一些非公开的结构和信息
  - 一些特殊的驱动,如注入在0号进程驱动的溢出利用的特性

驱动内核溢出利用初步

- 进程上下文与中断上下文内核的比较
  - 进入的方式比较、本地/远程,int 2e或deviceiocontrol/DPC中断
  - 影响比较.非独占/独占,触发进程相关/0,8号进程相关
  - 技术点比较:安全返回和调度调度/返回地址搜索技术

- 当前内核溢出研究状况
  - LINUX下:中断上下文考虑到安全返回.
  - WINDOWS:主要是在进程上下文的内核溢出研究上

- **WINDOWS 驱动内核的一些特性**
  - ü 运行于8号／0号进程空间内，这些进程空间不影射用户空间，0号进程的KPEB具备特殊性
  - ü 8号／0号进程同时用于处理线程调度和DPC调度
  - ü 运行在DPC级，处于独占模式，处于关中断，IRQL为2或2以上的级别上，因此不会响应缺页映射处理
  - ü 内核ESP堆栈外数据的随机性
    - ü可用的数据空间
    - ü一些返回值不可利用
    - üSHELLCODE里的ESP ADD的操作指令
  - ü 内核ESP堆栈外数据的随机性
  - ü 内核堆栈的长度



- **WINDOWS 驱动内核利用的一些特性**
  - ü 功能的利用代码是需要在用户态模式下
    - ü 核心态完成SHELLCODE的功能代码需要更多的指令代码，
    - ü 需要处理更多的一些细节，如页面映射等。
    - ü 内部的内核函数我们无法定位和使用
    - ü 需要我们自己处理相关的线程和DPC调度问题，否则无法完成我们需要的操作。
  - ü 溢出发生在独占的DPC中断级的，需要使得用户态SHELLCODE能够被执行且不引起因DPC和线程调度问题导致的死锁或崩溃

内核安全返回利用方法思路

- 内核安全返回的原理
  - 堆栈溢出以后，堆栈剩余的内容也并非是完全覆盖了，在未被覆盖的区域，我们能够找到足够的信息使得我们的程序来安全返回。

  - 在函数调用1里发生的溢出,溢出以后覆盖了函数调用1,2的内容,但是堆栈里函数调用3和以后的内容还是保持着,因此在我们SHELLCODE处理以后,我们可以恢复到函数调用 3 处执行返回.

函数调用1积存器保留信息
函数调用1返回地址
函数调用1参数
函数调用2积存器保留信息
函数调用2返回地址
函数调用2参数
函数调用3积存器保留信息
函数调用3返回地址
函数调用3参数

溢出以前的堆栈

SSSSSSS
JMP ESP 地址
覆盖内容和SHELLCODE
函数调用2参数
函数调用3积存器保留信息
函数调用3返回地址
函数调用3参数

溢出以后的堆栈

- **LINUX下常见的内核驱动溢出利用方法和要求**
  - ü 在2002年焦点峰会上,XFOCUS的成员alert7做了一个关于LINUX系统下,内核溢出利用技术的讲解,其主要过程如下
    - ü 在中断上下文中修改系统服务,使得对应的一个服务/中断指向自己特定的一定代码,并保留以前的处理入口地址
    - ü 溢出程序安全的返回。
    - ü 当发现被替换的系统服务的调用时,在进程的上下文里进入到我们的的HOOK代码里,把用户层SHELLCODE拷贝到进入这个服务的进程的空间,修改进程内核堆栈里的用户身份标志为特权,恢复以前的处理入口地址
    - ü 返回系统调用到用户态并使得代码地址运行到我们拷贝的SHELLCODE的地址上并执行。

- 我们的WINDOWS驱动内核溢出安全返回利用构思
  - Linux利用方式移植到WINDOWS上存在的问题
    - WINDOWS下的系统服务调用表相关的一些结构和指针信息是未公开的
    - WINDOWS下的线程/进程权限的结构是未公开的。
    - 整个机制需要二次内核操作
    - 需要额外的保存系统服务的HOOK代码（不能直接存放于堆栈上）
  - 我们的内核溢出利用设想的技术要点
    - 在内核中一次的,运用多道环境进程的虚拟空间映射机制,直接将我们的SHELLCODE拷贝到我们可以选择的用户进程的虚拟空间上.
    - 主动地修改和控制我们选择的用户进程的执行我们的SHELLCODE

- 安全返回利用需要考虑的问题
  - ü 要能获得所有进程和线程的内核环境块(KPEB/KTEB)
  - ü 能够切换到指定进程的虚拟空间上
  - ü SHELLCODE数据拷贝页面的状态和页面的属性
  - ü 内核中控制指定进程／线程的EIP

- 安全返回利用的限制
  - ü 被溢出的内核必须要能安全返回,但是很多时候并非能安全的返回

内核安全返回的SHELLCODE

- 内核SHELLCODE
  - 需要解决的问题
  - 解决的具体实现
- 用户SHELLCODE
  - 需要解决的问题
  - 解决的具体实现

内核非安全返回的利用方法一

- 为什么不能安全返回
  - ü 堆栈空间有限,无法保存足够安全返回的信息
  - ü 返回时里需要填充需要某些结构里的正确的返回值
  - ü 各个平台的不通用性
  - ü 安全返回饶过的代码使得加锁后未经过解锁操作导致系统的死锁
  - ü 寄存器无法恢复

- 非安全返回利用一的构思
  - ü 通过iret指令,使得内核执行的系统进程直接返回到用户态下面去执行是否可行呢?
- 非安全返回利用一需要考虑的问题
  - ü 系统处于完全独占的状态下，无法调度和建立其他进程，事件与信号操作也很容易导致死锁
  - ü 执行的用户层代码不能产生其他的DPC调用的操作,如对文件普通的写操作是会产生磁盘缓冲写入DPC调用的导致死锁
  - ü 系统进程的用户态的DLL未映射，用户层代码无法使用通常的利用DLL的API这样形式的代码
  - ü 0号进程的线程的KTEB的一些特殊字段值的问题使得用户代码调用系统服务进入内核导致崩溃

- 我们的方案
  - ü 通过使用INT 2E调用的参数直接使用系统API调用,绕过难以加载系统DLL的问题
  - ü 选择非网络的，不导致DPC调度的API完成建立一个处于系统启动目录下的BAT文件的操作
  - ü 使得系统快速重新启动，获得下次我们BAT文件的正常执行
  - ü BAT文件通过TFTP从网络上下载我们的EXE执行

- 非安全返回利用一的限制和问题
  - ü 系统是独占的,必须通过重起获得执行
  - ü 可选择的API（必须是系统服务层的API和能不引起DPC调度的API）和实现的SHELLCODE功能极其有限，因此重起必须等待用户登陆才能获得利用，而且很难隐蔽,而且启动目录路径是硬代码
  - ü 因为无法直接使用加载DLLAPI代码，代码长度比较大

内核非安全返回方法一的SHELLCODE

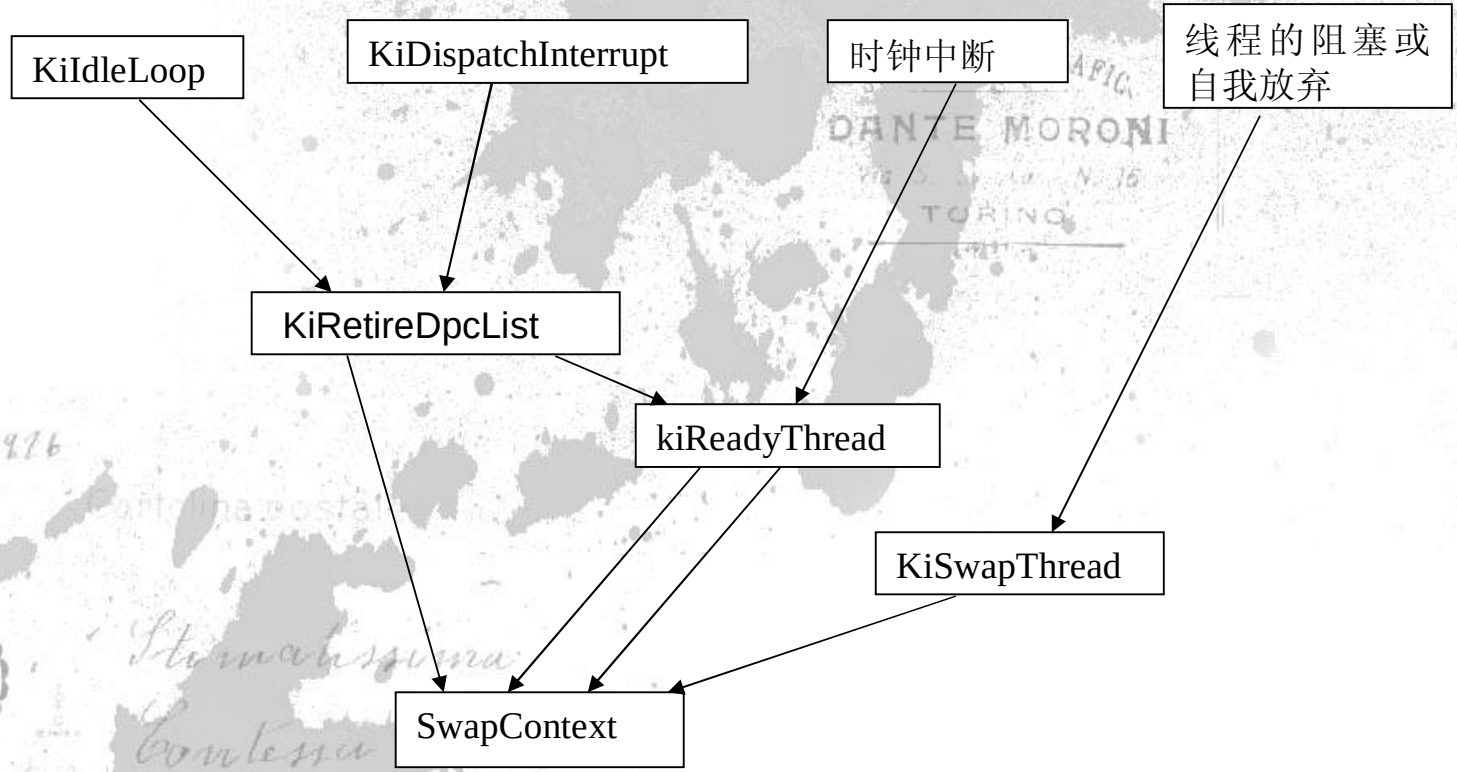
- 内核SHELLCODE
  - 需要解决的问题
  - 解决的具体实现
- 用户SHELLCODE
  - 需要解决的问题
  - 解决的具体实现

内核非安全返回的利用方法二

- 焦点：线程和DPC的调度恢复
  - ü 利用方法一限制的条件太多,无法写出强大的SHELLCODE代码,就连注册表我们也无法进行写操作处理
  - ü 利用方法一实现烦琐
  - ü 需要我们对WINDOWS内核的DPC调度与线程的调度有更多的了解

- 线程和DPC的调度过程
  - ü 硬件的高级中断和APC
  - ü DPC和时钟中断
  - ü 线程调度
    - ü 线程阻塞和自动放弃CPU
    - ü 时间片抢占
    - ü IDLE进程调度

- 线程和DPC的调度关键函数过程
  - ü KiIdleLoop函数
  - ü KiDispatchInterrupt函数
  - ü KiRetireDpcList函数
  - ü KeReadythread函数
  - ü KiSwapthread函数
  - ü SwapContext函数
  - ü 时钟中断


KiIdleLoop
KiDispatchInterrupt
时钟中断
线程的阻塞或
自我放弃
KiRetireDpcList
kiReadyThread
KiSwapThread
SwapContext

调用大致序列

- 非安全返回利用二的构思
  - ü 我们的溢出时处于KiRetireDpcList调用处理里
  - ü 利用线程的核心堆栈上还保留着进入KiRetireDpcList之前的一些信息,利用系统调度的机制，使得其自动恢复到调用KiRetireDpcList之前的状态开始执行，恢复线程与DPC的调度。

- 非安全返回利用二需要考虑的问题
  - ü 判断和选择一个合适的用户态的进程的线程进行调度切换
  - ü 利用系统调度切换的新的进程的内核处理和返回
  - ü 缺页处理
  - ü 再次进入DPC和线程调度的时候,的上次中断了的调度的恢复

- 非安全返回利用二的限制和问题
  - ü 调度恢复的数据也在当前内核堆栈里，如果大量的数据覆盖也可能导致被破坏而不能利用，而非安全返回利用方法一则相对这要宽松一些
  - ü 避免了死锁，但是可能对应的驱动不可再用

内核非安全返回方法二的SHELLCODE

- 内核SHELLCODE
  - 需要解决的问题
  - 解决的具体实现
- 用户SHELLCODE
  - 需要解决的问题
  - 解决的具体实现

- 与非安全返回利用方法一的区别
  - ü 可以加载我们需要的库,也解决了缺页问题,无需要使用INT 2E这样方式的,SHELLCODE可以功能更强大,编写更简单
  - ü 解决了DPC调度问题,不会因为调用有DPC调度的API产生死锁或崩溃,如注册表写的操作、
  - ü 也解决了线程调度问题,其实是可以写出与安全返回方法一样的SHELLCODE代码来执行的.只是需要考虑是否因网络驱动的溢出导致了网络不可再用的问题、

更深入的讨论和总结

- 运行于内核实模式的SHELLCODE
  - ü 实模式映射与虚模式映射的问题
  - ü 通过页面去处理文件的SHELLCODE,需要对NTFS文件系统的结构很了解、
  - ü 在WINDOWS下,还不能直接调用DOS的中断

- 以上讨论的技术的推荐适用的范围
  - ü 如果是可以安全返回的话,建议使用安全返回的技术,功能最为强大
  - ü 如果不能安全返回,但是内核线程的核心堆栈的与调度相关的结构未被破坏,建议使用非安全返回的利用技术二
  - ü 如果不能安全返回,而且内核线程的核心堆栈的最后关于调度的结构也被覆盖了的话,我们只能使用非安全返回的利用技术一

演示

- SYMANTEC DNS内核溢出漏洞介绍
  - ü 漏洞简介
  - ü 该内核驱动的特殊性
  - ü 漏洞的原理和利用的一些事项
    - ü 漏洞原理简介
    - ü 漏洞利用限制
    - ü 一些其他可以被利用的特性

- XP上安全返回利用演示
- 2000上安全返回利用演示
- 2000上非安全返回利用一演示
- 2000上非安全返回利用二演示

Q/A

Thanks !