https://github.com/monoxgas/sRDI

Use Cases / Examples

Before use, I recommend you become familiar with Reflective DLL Injection and it's purpose.

Convert DLL to shellcode using python

```
from ShellcodeRDI import *

dll = open("TestDLL_x86.dll", 'rb').read()
shellcode = ConvertToShellcode(dll)
```

Load DLL into memory using C# loader

```
DotNetLoader.exe TestDLL_x64.dll
```

Convert DLL with python script and load with Native EXE

```
python ConvertToShellcode.py TestDLL_x64.dll
NativeLoader.exe TestDLL_x64.bin
```

Convert DLL with powershell and load with Invoke-Shellcode

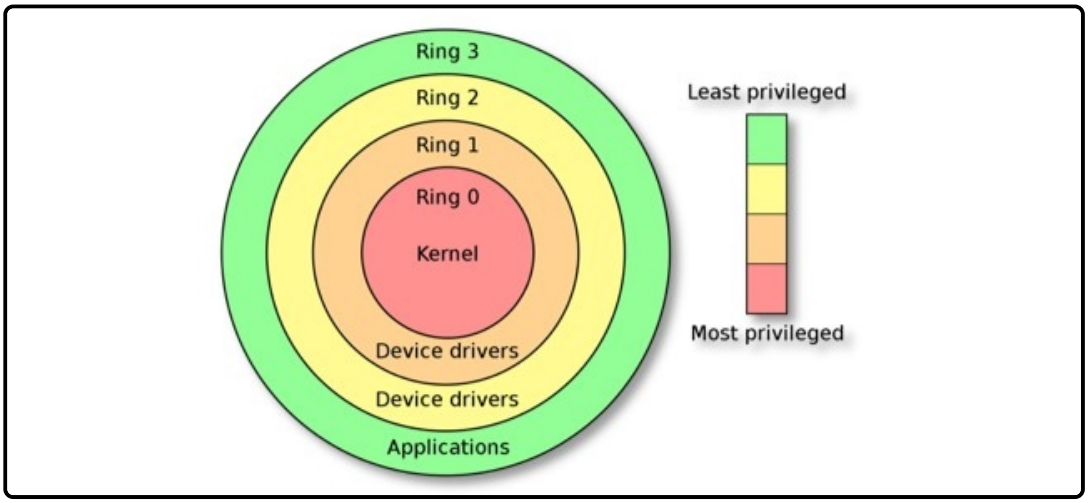
```
Import-Module .\Invoke-Shellcode.ps1
Import-Module .\ConvertTo-Shellcode.ps1
Invoke-Shellcode -Shellcode (ConvertTo-Shellcode -File TestDLL_x64.dll)
```

系统直接调用

Windows操作系统中实际只使用了两个特权级别：

一个是Ring3层，平时我们所见到的应用程序运行在这一层，所以叫它用户层，也叫User-Mode。所以下次听到别人讲（Ring3、用户层、User-Mode）时，其实是在讲同一个概念。

一个是Ring0层，像操作系统内核（Kernel）这样重要的系统组件，以及设备驱动都是运行在Ring0，内核层，也叫



通过这些保护层来隔离普通的用户程序，不能直接访问内存区域，以及运行在内核模式下的系统资源。

当一个用户层程序需要执行一个特权系统操作，或者访问内核资源时。处理器首先需要切换到Ring0模式下才能执行后面的操作。

切换Ring0的代码，也就是直接系统调用所在的地方。

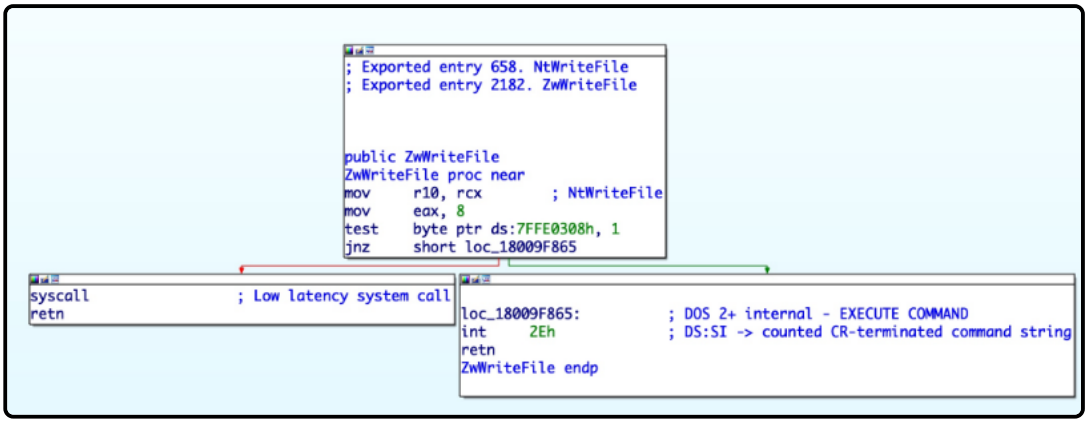
我们通过监控Notepad.exe进程保存一个.txt文件，来演示一个应用层程序如何切换到内核模式执行的：

我们可以看到 notepad调用了kernel32模块中的WriteFile 函数，然后该函数内部又调用了ntdll中的NtWriteFile来到了Ring3与Ring0的临界点。

因为程序保存文件到磁盘上，所以操作系统需要访问相关的文件系统和设备驱动。应用层程序自己是不允许直接访问这些需要特权资源的。

应用程序直接访问设备驱动会引起一些意外的后果（当然操作系统不会出事，最多就是应用程序的执行流程出错导致崩溃）。所以，在进入内核层之前，调用的最后一个用户层API就是负责切换到内核模式的。

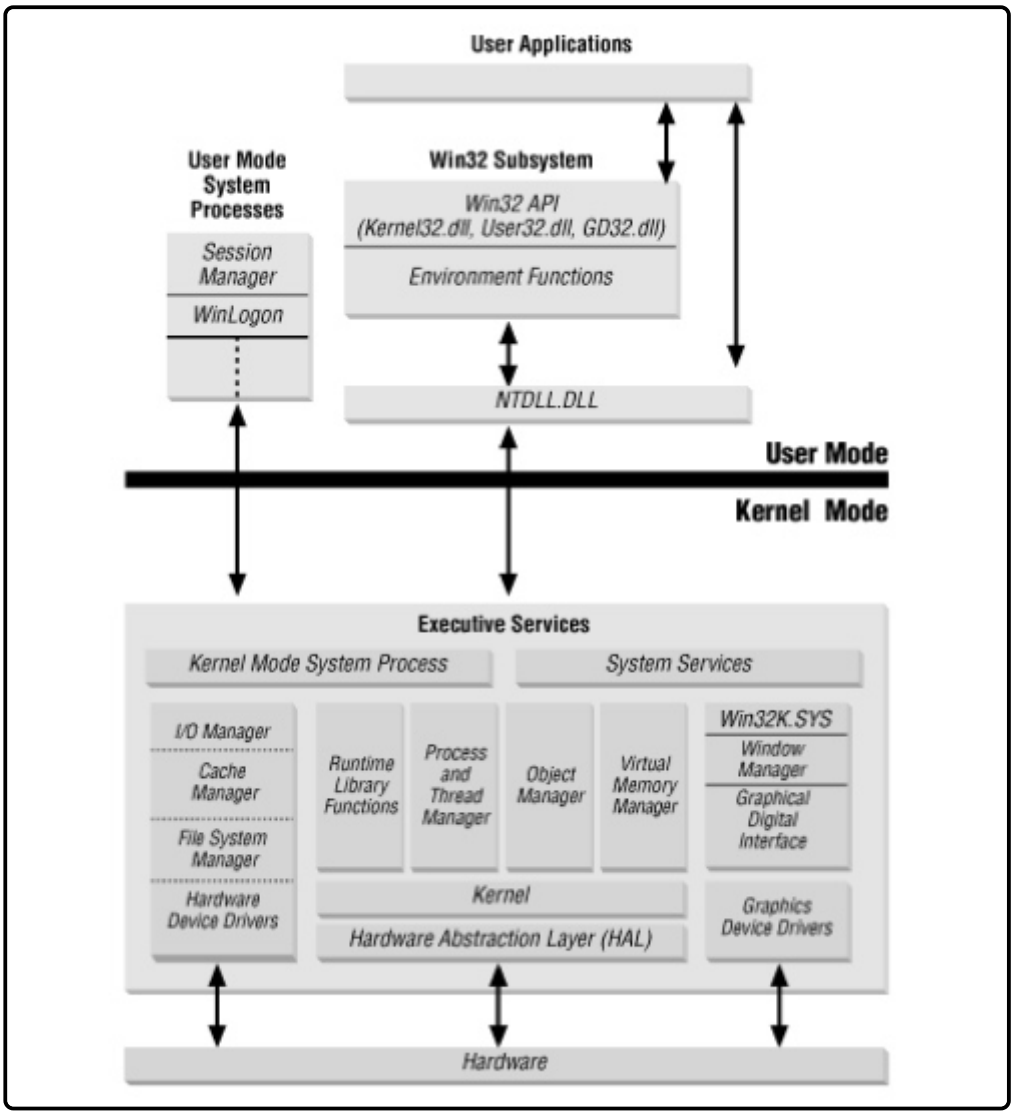
CPU中通过执行syscall指令，来进入内核模式，至少x64架构是这样的。



把被调用函数相关的参数PUSH到栈上以后，ntdll中的NtWriteFile函数的职责就是，设置EAX为对应的"系统调用号"，最后执行syscall指令，CPU就来到了内核模式（Ring0）下执行。

进入内核模式后，内核通过diapatch table（SSDT），来找到和系统调用号对应的Kernel API，然后将用户层栈上的参数，拷贝到内核层的栈中，最后调用内核版本的ZwWriteFile函数。

当内核函数执行完成时，使用几乎相同的方法回到用户层，并返回内核API函数的返回值（指向接收数据的指针或文件句柄）。


User Applications
User Mode System Processes
Session Manager
WinLogon
Win32 Subsystem
Win32 API (Kernel32.dll, User32.dll, GD32.dll)
Environment Functions
NTDLL.DLL
User Mode
Kernel Mode
Executive Services
Kernel Mode System Process
System Services
I/O Manager
Cache Manager
File System Manager
Hardware Device Drivers
Runtime Library Functions
Process and Thread Manager
Object Manager
Virtual Memory Manager
Win32K.SYS
Window Manager
Graphical Digital Interface
Kernel
Hardware Abstraction Layer (HAL)
Graphics Device Drivers
Hardware