**恶意代码分析与防治技术实验报告**

**Lab7**

**学号：2013018 姓名：许健 专业：信息安全**

1. **实验内容**

完成课本Lab7的实验内容，编写Yara规则，并尝试IDA Python的自动化分析。

1. **问题解答**
2. Lab7-1
3. 当计算机重启时，这个程序如何确保它继续运行（达到持久化驻留）？

**使用IDAPro查看导入表**

**CreateService**(创建一个可以在启动时刻运行的服务。恶意代码使用该函数来持久化、隐藏，或者启动内核驱动)

**OpenSCManager**(打开一个到服务控制管理器的句柄。任何想要安装、修改或是控制一个服务的程序，都必须要调用这个函数，才能使用其他服务操纵函数)

**StartServiceCtrlDispatcher**(由服务使用来连接到服务管理控制器进程的主线程。任何以服务方式运行的进程必须在启动后30s内调用这个函数，在恶意代码中找到这个函数，可以告诉你它的功能应该以服务方式运行)

**InternetOpenUrl**(使用FTP、HTTP或HTTPS协议连接来打开一个特定的URL，如果URL固定，则可以作为基于网络的特征码)

**InternetOpen**(初始化WinINet中的一些高层次互联网访问函数，比如InternetOpenUrl和InternetReadFile。搜索InternetOpen函数是找到互联网访问功能初始位置的一个好方法。InternetOpen函数的一个参数是User-Agent，有时也可以作为基于网络的特征码)

**OpenMutex**(打开一个双向互斥对象的句柄，可以被恶意代码用来确保在任意给定时间内系统只能有一个运行实例。恶意代码通常使用固定名字来为互斥对象命名，因此可以用作很好的主机特征)

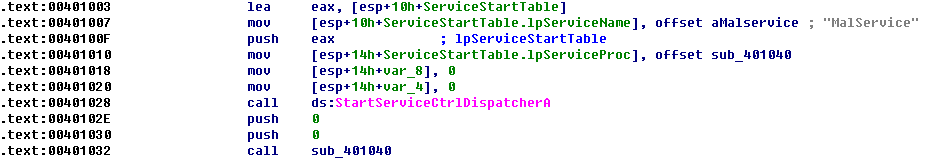
**CreateMutex**(创建一个互斥对象，可以被恶意代码用来确保在给定时刻只有一个实例在系统上运行)

**GetModuleFilename**(返回目前进程装载某个模块的文件名。恶意代码可以使用这个函数，在目前运行进程中修改修改或赋值文件)

结论：恶意代码可能创建一个服务，来保证它会在系统被重启后运行；程序可能连接到一个Url并下载内容；恶意代码创建并使用互斥量确保只有一个实例在系统中运行

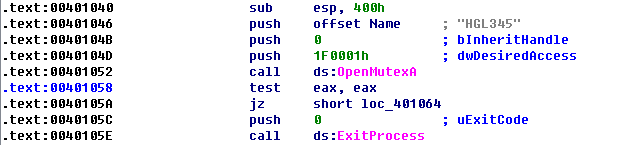
**分析main函数**

主函数调用**StartServiceCtrlDispatcher**，说明程序期望作为一个服务被运行。

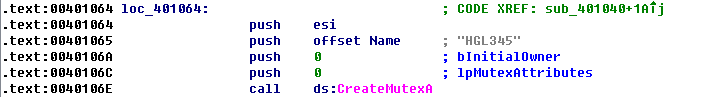


**检查sub\_401040函数**

函数调用**OpenMutex**，试图获取一个名为HGL345的互斥量句柄。如果这个调用成功，程序就会退出。



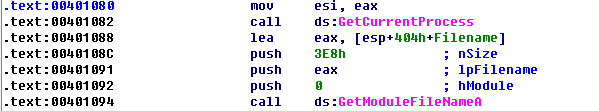
查看”**HGL345**”的交叉引用，在另一处我们看到创建互斥量的**CreateMutex**函数



接下来函数调用了**OpenSCManager**，它打开一个服务控制管理器的句柄，以便这个程序可以添加或修改服务。



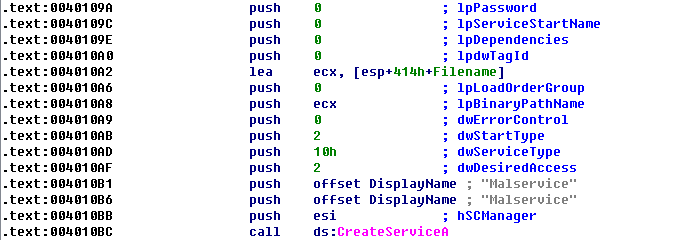
**GetModuleFilename**函数返回当前正在运行的可执行程序或一个被加载DLL的全路径名。



这个全路径名被**CreateService**用来创建一个服务。

**BinaryPathName**与用**GetModuleFilename**调用得到的当前正在运行可执行程序路径是一样的。GetModuleFilename调用是需要的，可以动态获取当前运行可执行程序路径。

安装的服务名称为”**Malservice**”



使用MSDN查看**dwServiceType**参数的值，0x10表示在其自己的进程中运行的服务。

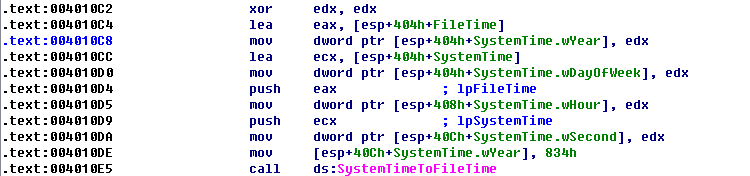


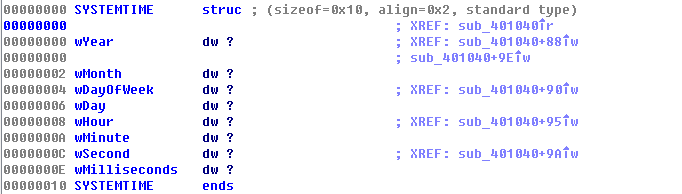
使用MSDN查看**dwStartType**参数的值，0x02暗示这个服务在系统启动时自动运行。



**SystemTimeToFileTime**在不同时间格式之间转换

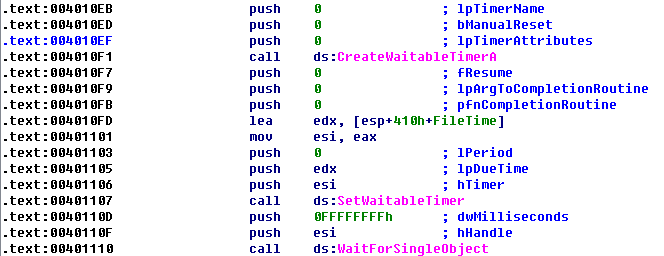
SYSTEMTIME是Windows多个时间结构体之一，SystemTime.wYear被设置为834h(2100年)



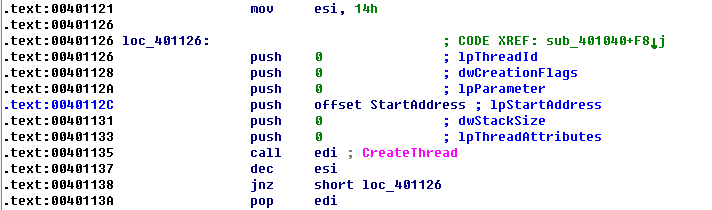


接下来程序调用**CreateWaitableTimer**、**SetWaitableTimer**、**WaitForSingleObject**

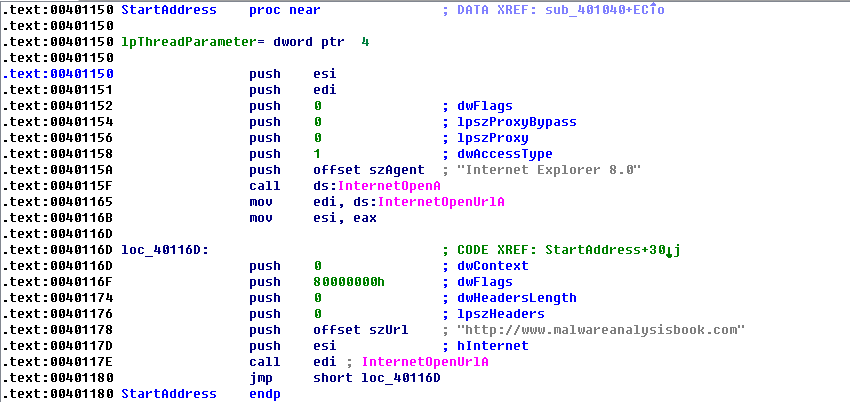
传给SetWaitableTimerlp的**DueTime**参数是SystemTimeToFileTime返回的FileTime，随后使用**WaitForSingleObject**进入等待，直到2100年1月1日。



代码接着循环20次，创建20个线程，传给CreateThread的参数**StartAddress**函数是线程的起始地址。



双击查看**StartAddress**函数，该函数调用InternetOpen来初始化一个到网络的连接，接着在一个循环中调用InternetOpenUrl，循环结尾jmp是一个无条件跳转，这个代码会一直执行下去，一直下载www.malwareanalysisbook.com的主页。可以知道恶意代码发动的是一个DDos攻击。



1. 为什么这个程序会使用一个互斥量？

保证在同一时间这个程序只有一份实例在运行。

1. 可以用来检测这个程序的基于主机特征是什么？

一个名为HGL345的互斥量、Malservice服务

1. 检测这个恶意代码的基于网络的特征是什么？

用户代理“Internet Explorer 8.0”、和http://www.malwareanalysisbook.com通信

1. 这个程序的目的是什么？

在2100年1月1日的半夜对http://www.malwareanalysisbook.com发起DDos攻击

1. 这个程序什么时候完成执行？

这个程序永远不会完成，它在一个定时器上等待直到2100年，到时候创建20个线程，每一个无限循环。

1. Lab7-2
2. 这个程序如何完成持久化驻留？

基础静态分析只能看到有限的导入函数，都是COM相关的

**OleInitialize**(用来初始化COM库。使用COM对象的程序必须在调用任何COM功能之前，调用这个函数)

**CoCreateInstance**(创建一个COM对象。COM对象提供了非常多样化的功能。类标识CLSID会告知你哪个文件包含着实现COM对象的代码)

一个Unicode字符串http://www.malwareanalysisbook.com/ad.html



尝试动态分析，恶意程序打开Internet Explorer并弹出一个广告。并不存在任何这个程序修改系统或安装它自己以便在计算机重启后运行的证据。

这个程序没有完成持久化驻留，它运行一次然后退出。

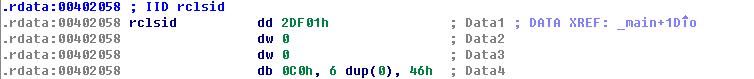
1. 这个程序的目的是什么？

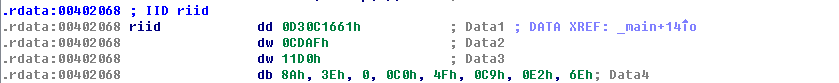
使用IDA Pro分析代码，定位到main函数

恶意代码首先初始化COM，并调用**OleInitialize**和**CoCreateInstance**获得一个COM对象。返回的COM对象被保存在栈上的一个变量ppv。为了判断程序在使用什么COM功能，我们需要检查接口标识符(IID)以及类标识符(CLSID)



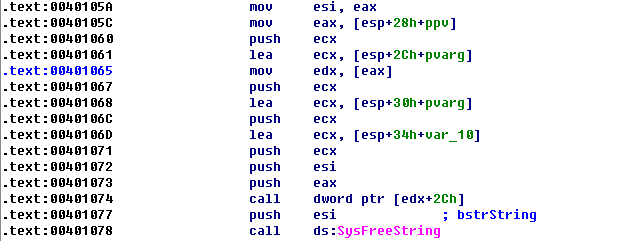
IID对应IWebBrowser2，CLSID对应Internet Explorer





COM对象会被一些函数访问，[edx+2Ch]指向的是IWebBrowser2的Navigate函数，该函数被调用，Internet Explorer将导航网址http://www.malwareanalysisbook.com/ad.html，显示一个一次性的广告。

调用完Navigate函数之后会执行一些清理函数，然后程序终止。



1. 这个程序什么时候完成执行？

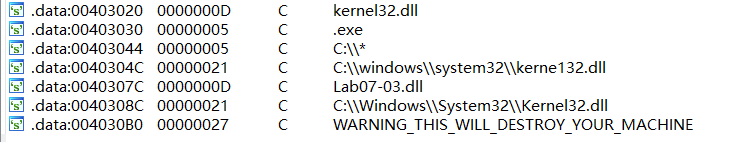
这个程序在显示这个广告后完成执行。

1. Lab7-3

**静态分析：**

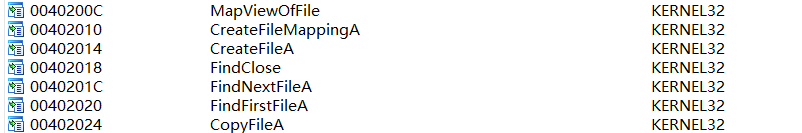
**Exe**

存在可疑字符串kerne132.dll



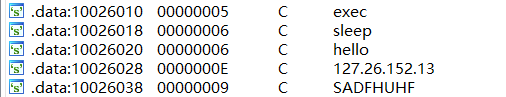
与文件相关的导入函数

CreateFileA 、 CreateFileMappingA以及MapViewOfFile表明该程序可能打开一个文件，并将其映射到内存中。这个 FindFirstFileA 和FindNextFileA函数组合，可能会搜索文件，并使用CopyFileA 对找到的文件进行复制操作。该程序没有导入Lab 07-03.dll（DLL中的函数）、LoadLibrary或者GetProcAddress，暗示它可能没有在运行时加载那个DLL。

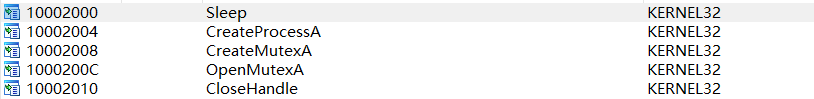


**Dll**

ip127.26.152.13



与创建进程、创建互斥量、网络通信有关的导入函数





**分析exe文件**

定位main函数，程序开始判断是否有两个参数，否就结束，是就继续

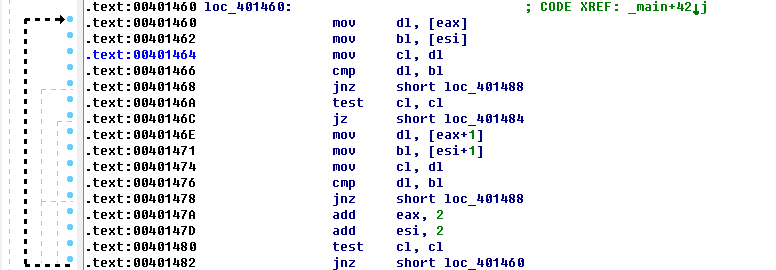


赋值操作,将第二个参数传给eax，将那一串的字符传给esi



接下来是一段循环结构，在不断比较eax和esi的每个字节是否相等，只有全等程序才能继续运行，否则结束，这里我们可以知道程序要运行需要一个参数

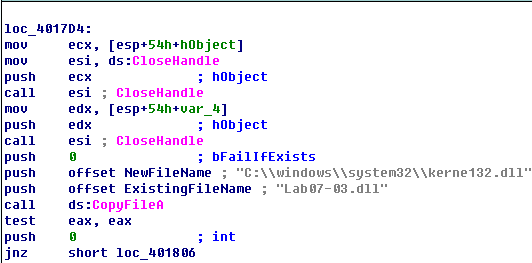
“WARNING\_THIS\_WILL\_DESTROY\_YOUR\_MACHINE”



从loc\_40148D到loc\_401538之间的代码分别打开lab04-03.dll和kernel32.dll文件并且把他们都映射到当前进程地址空间，以能够对其数据随意操作。ebp存lab07-03.dll的基地址，esi存的是kernel32.dll被映射到进程的基地址。

loc\_401538之后的一系列代码，主要是对esi的一系列操作，还有reps，大部分是mov等指令。我们暂时推测这段代码应该是对kernel32的重写操作。

关闭了之前两个映射的句柄，然后复制Lab07-03.dll到kerne132.dll



再往下调用4011E0函数，查看流程图主要是查找文件，搜索C：驱动器查找.exe文件，在.exe文件中找到kernel.dll的位置，用kerne1.dll替代，这意味着kerne132.dll会替代kernel32.dll被修改过的可执行文件加载。

使用动态分析可以看到该程序打开文件系统上的每一个exe文件

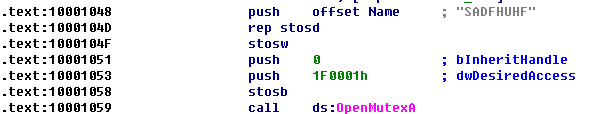
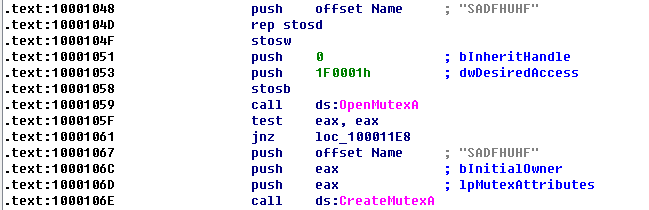
检查这个程序如何修改kernel32.dll以及Lab07-03.dll。使用md5检查发现运行Lab07-03.exe程序后Lab07-03.dll名称修改为kerne132.dll，并且有导出函数，该导出函数是kernel32.dll中的导出函数，并且作了重定向，所有功能还是kernel32.dll中的功能。

**分析DLL**

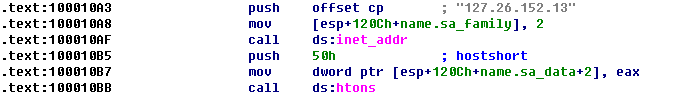
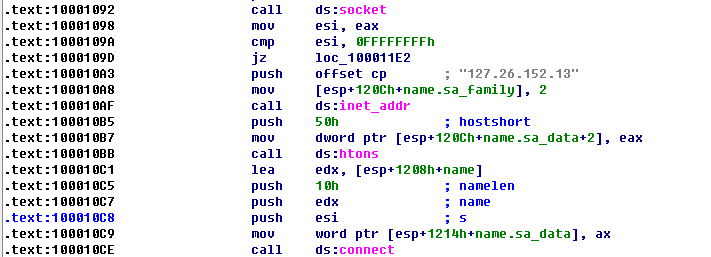
\_alloca\_probe函数分配栈



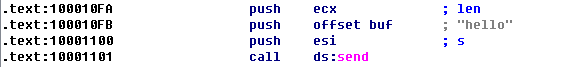
创建互斥量"SADFHUHF"，可以作为主机特征



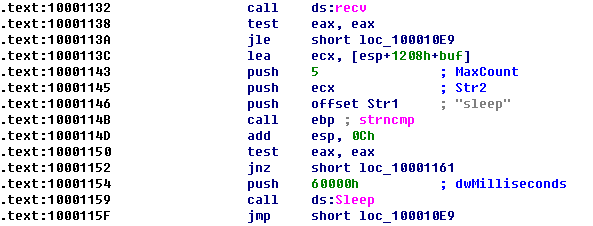
连接ip地址为127.26.152.13的主机



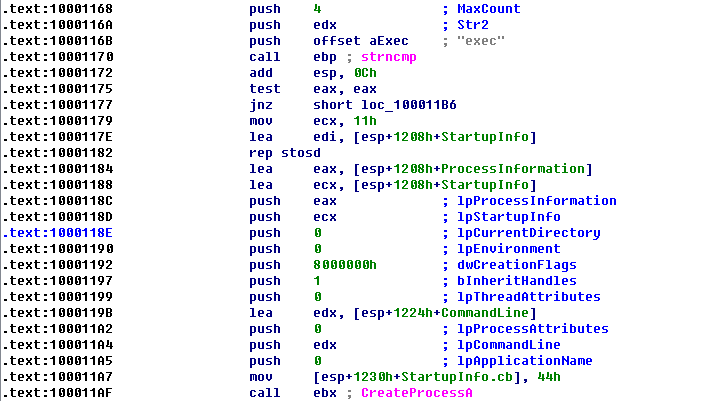
程序发送了hello，接着等待接受对面的信息



判断接收到的信息是不是sleep命令。是就睡觉



不是就看看发过来的信息是不是exec命令。是就执行创建进程操作，然后再发送hello，等待不断执行这个操作，直到收到q才结束程序。



Commandline是buf第五个字节位置，是远程发送的绝对地址，exec path执行本地一个exe程序



1. 这个程序如何完成持久化驻留，来确保在计算机被重启后它能继续运行？

通过写一个DLL到C:\windows\system32\，并修改系统上每一个导入它的.exe文件，达到持久化驻留。

1. 这个恶意代码的两个明显的基于主机特征是什么

通过硬编码来使用文件名kerne132.dll和使用一个硬编码来命名互斥量SADFHUHF

1. 这个程序的目的是什么？

创建后门程序来接远程主机，且难以被删除。两个命令分别用于执行和休眠

1. 一旦这个恶意代码被安装，你如何移除它？

难以删除，因为感染了系统上每一个使用kerne132.dll的exe文件。

最好的办法是从一个备份恢复系统，或者删除kerne132.dll中恶意的内容，也可以写一个程序来取消所有对PE文件的修改。

1. **编写Yara规则**

include ".\includes\IsPE.yar"

include ".\includes\SizeLimit.yar"

rule Lab7\_1

{

strings:

$string1 = "Malservice" wide ascii

$string2 = "HGL345" wide ascii

$string3 = "malwareanalysisbook.com" wide ascii

$string4 = "Explorer 8.0" wide ascii

condition:

IsPE and SizeLimit and (all of ($string\*))

}

rule Lab7\_2

{

strings:

$string1 = "malwareanalysisbook.com/ad.html" wide ascii

condition:

IsPE and SizeLimit and $string1

}

rule Lab7\_3\_dll

{

strings:

$string1 = "SADFHUHF" wide ascii

$string2 = "exec" wide ascii

$string3 = "hello" wide ascii

$string4 = "127.26.152.13" wide ascii

condition:

IsPE and SizeLimit and (all of ($string\*))

}

rule Lab7\_3

{

strings:

$string1 = "kerne1" wide ascii

$string2 = ".exe" wide ascii

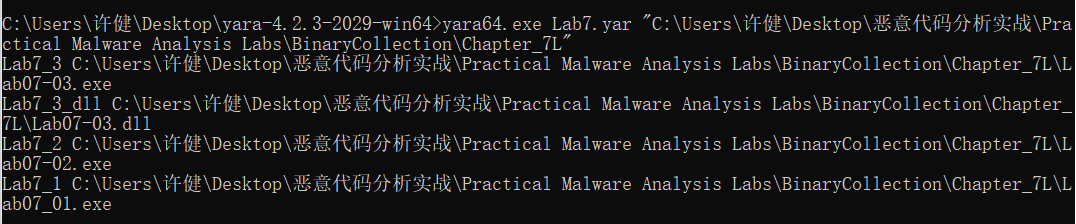
$string3 = "C:\\\*" wide ascii

$string4 = "WARNING\_THIS\_WILL\_DESTROY\_YOUR\_MACHINE" wide ascii

condition:

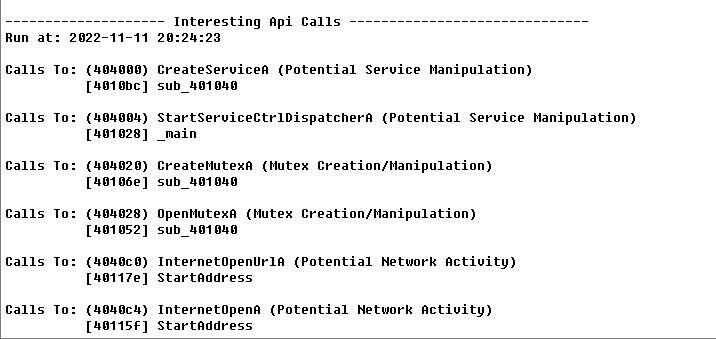
IsPE and SizeLimit and (all of ($string\*))

}



1. **编写IDAPython脚本**

由于我们分析恶意代码的时候主要定位可疑API，抓住程序的重点。因此编写一个帮助查找恶意代码经常使用API的脚本很有必要，主要功能是返回API的交叉引用地址。功能如下：



# Interesting\_Api\_Calls.py

from idautils import \*

from time import strftime

INTERESTING\_SUBSTRINGS = [\

("CreateFile", "File Creation/Drop"),\

("CopyFile", "Filesystem Manipulation"),\

("DeleteFile", "Filesystem Manipulation"),\

("CFile::Write", "File Creation/Drop"),\

("CFile::Open", "File Creation/Drop"),\

("CFileFind::", "Filesystem Crawling"),\

("CFile::Read", "File Access"),\

("Http", "Potential Network Activity"),\

("Socket", "Potential Network Activity"),\

("Internet", "Potential Network Activity"),\

("Inet", "Potential Network Activity"),\

("CreateProcess", "Spawns a new process"),\

("Mutex", "Mutex Creation/Manipulation"),\

("ProcessMemory", "Potential Process injection"),\

("Service", "Potential Service Manipulation"),\

("Debugger", "Anti-Analytics"),\

]

if \_\_name\_\_ == '\_\_main\_\_':

print("\n-------------------- Interesting Api Calls ------------------------------")

print("Run at: %s" % (strftime("%Y-%m-%d %H:%M:%S")))

for call\_ea, name in sorted(Names()):

demangled\_name = Demangle(name, INF\_SHORT\_DN)

if demangled\_name != None:

name = demangled\_name

for keyword, note in sorted(INTERESTING\_SUBSTRINGS):

if keyword.lower() in name.lower():

xrefs = sorted([ x for x in CodeRefsTo(call\_ea, 0) if SegName(call\_ea) != ".data" ])

if len(xrefs) > 0:

print("\nCalls To: (%x) %s (%s)" % (call\_ea, name, note))

for xref in xrefs:

print(" [%x] %s" % (xref, GetFunctionName(xref)))