



南開大學
Nankai University

网络空间安全学院

恶意代码分析与防治技术课程实验报告

实验八：使用 WinDBG 调试内核

姓名：周钰宸

学号：2111408

专业：信息安全

2024 年 1 月 14 日

1 实验目的

1. 复习教材和课件内第 10 章的内容。
2. 完成 Lab10 的实验内容，对 Lab10 的三个样本进行依次分析，编写 Yara 规则，并尝试使用 IDAPython 的自动化分析辅助完成。

2 实验原理

2.1 IDAPro

IDA 是一款广泛使用的交互式反汇编工具，它支持多种平台和文件格式。IDA 的主要功能是将机器代码转换为人类可读的汇编代码，从而帮助研究人员理解和分析程序的功能和行为。

2.2 IDAPython

IDAPython 是一个 IDA 插件，允许用户使用 Python 语言来脚本化 IDA 的功能。这为自动化任务和复杂的分析提供了巨大的灵活性。可以调用 IDA 的所有 API 函数，使用 Python 的功能模块编写强大的自动分析脚本。

2.3 OllyDBG

OllyDbg 是一个 32 位的汇编级别的调试器，主要用于 Microsoft Windows。它是反向工程和软件分析中的一个流行工具。OllyDbg 的特点是其用户友好的界面、多窗口模式、直接修改代码、以及强大的插件支持。

2.4 WinDBG

WinDBG 是一款功能强大的调试工具，最初由微软开发用于 **Windows 操作系统的内核和应用程序调试**。

2.4.1 特点

1. **强大的调试功能**: WinDBG 提供了丰富的调试功能，允许分析程序的内部状态、执行流程、变量值等。它支持源代码级和汇编级的调试，能够帮助分析人员深入了解病毒或恶意软件的运行机制。
2. **内核级调试**: 用于调试 Windows 操作系统的内核，这对于分析与操作系统交互的恶意软件非常有用。它还支持内核模块的加载和卸载、驱动程序的调试等操作。
3. **插件支持**: 用户可以编写自定义插件来扩展其功能，以满足特定的需求。这意味着分析人员可以根据自己的需求定制工具。
4. **支持多种符号文件**: 能够识别和使用符号文件，这有助于在调试过程中还原函数名、变量名等信息，使分析更容易理解。

2.4.2 病毒分析工具的优势

WinDBG 正如课本中第十章所讲，其作为病毒分析工具具有 IDA Pro 和 OllyDBG 所不具有的多种显著优势，具体而言：

1. **内核级调试**：WinDBG 是一款强大的内核调试工具，允许分析人员深入研究操作系统内部的运行，这对于分析需要操作系统交互的恶意软件非常重要。**IDA 和 OllyDBG 主要用于用户态应用程序的逆向工程，而 WinDBG 能够涵盖内核级的分析。**
2. **脚本自动化**：虽然 IDA 和 OllyDBG 也支持脚本编写，但 WinDBG 的脚本自动化能力相对较强，允许分析人员编写复杂的自动化任务，以便更快地处理大量的样本和分析数据。
3. **跨平台支持**：WinDBG 能够支持 .NET 程序的调试，这对于分析跨平台恶意软件非常有用。虽然 IDA 也支持 .NET 程序的反汇编，但 WinDBG 提供了更直接的调试接口。
4. **符号文件支持**：WinDBG 具有出色的符号文件支持，可与微软的符号服务器和其他符号文件进行交互。这使得在分析过程中还原函数名、变量名等信息变得更加容易，帮助分析人员更好地理解代码。

总的来说，WinDBG 作为一款强大的调试工具，具有丰富的功能和广泛的应用领域，使其成为病毒分析和恶意软件研究的重要工具之一。它能够帮助分析人员深入了解恶意软件的内部工作原理，有助于有效地识别、分析和应对安全威胁。**本次实验中也会重点使用 WinDBG 对 Lab10 的三个病毒进行全面的分析。**

2.5 Yara

3 实验过程

3.1 实验环境及工具

虚拟机软件	VMware Workstation 17 Pro
宿主机	Windows 11 家庭中文版
虚拟机操作系统	Windows XP Professional
实验工具 1	主机 WinDBG(X86).Ink
实验工具 2	OllyDBG 2.01
实验工具 3	IDAPro 6.6.14.1224
实验工具 5	虚拟机 Windbg_x86_6.12.2.633
配套工具	Python 2.7.2

表 1: 本次实验环境及工具

3.2 实验环境搭建

本次实验由于需要使用主机的 WinDBG 通过 VMware 搭建的管道 pipe 对虚拟机进行内核级别的调试。而与用户态调试不同，内核调试需要一些初始化设置。首先需要设置虚拟操作系统并开启内核调试，然后配置 VMware 使虚拟机与宿主系统之间有一条虚拟化的串口，同时还应该配置宿主操作系统中的 WinDbg。

因此为了在后面用到后方便使用，这里提前将**本次实验所需要的环境和配置搭建完成。**

3.2.1 虚拟机配置

我这里使用的虚拟机是 VMware 和操作系统 Windows XP，因此针对于二者进行配置。

1. 修改 boot.ini:

首先通过 WindowsXP 中的搜索功能找到通常来说隐藏的文件 boot.ini, 用于控制一些操作系统的启动时的操作:



图 3.1: boot.ini

图3.1可以看到成功找到文件后打开, 显示了内容 boot loader 和 operating systems, 这学期跟宫晓利宫老师的操作系统课上正好学到过, 这是配置内核启动的一些东西。

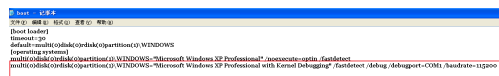


图 3.2: 配置 boot.ini

图3.2可以看到在原先的启动方式上添加了新的开机系统选项 Microsoft Windows XP Professional with Kernel Debugging。具体而言:

- /debug: 代表开启内核调试;
- /debugport=COM1: 告诉系统使用的端口来连接调试与被调试系统;
- /baudrate=115200: 代表串口数据的传输速率。

接下来重新启动虚拟机:

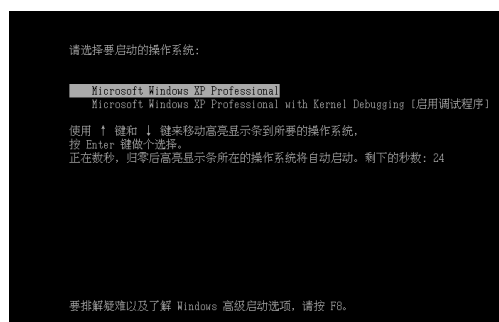


图 3.3: 重启 XP

图3.3看到此时显示了两个可以开机选择的操作系统，其中第一个是我们刚刚配置的**内核调试级别**的操作系统。

2. 创建串行端口：

接下来因为之前设置 boot.init 时标明了一个使用的端口，而端口我们还没有进行创建。因此关闭虚拟机创建新的串行端口。需要关机是由于这种硬件操作通常需要虚拟机处于关闭状态：



图 3.4: 创建串行端口

图3.4可以看到一些值得注意的地方：

- 删除打印机，然后创建新的硬件选择串行端口：这是书中教材中没有提到的注意事项，我也是通过摸索发现的。一般情况下，对于虚拟机来说，虚拟打印机通常会映射到虚拟串行端口，例如串行端口 1 (COM1)。这是为了模拟实际硬件环境，因为在实际计算机中，打印机通常会通过串行端口连接到计算机。因此如果不删除打印机，创建的新端口会是串行端口 2 即 COM2 和前面的不匹配。
- 命名管道：\\.\pipe\com_1。
- 设置该端是服务器，另一端是应用程序 (WinDBG)。
- I/O 模式轮询时主动放弃；
- 启动时连接；

到此就完成了全部的虚拟机操作系统的配置了，接下来回到主机配置 WinDBG。

3.2.2 WinDBG 配置

主机上由于要调试的目标操作系统是 32 位的 Windows XP，因此应该使用 32 位对应 WinDBG X86，无需再次下载。而虚拟机的 Windows XP 似乎不自带 WinDBG，需要额外下载。

1. 虚拟机中 WinDBG 下载

虚拟机通过网上论坛找到了一个版本为 Windbg_x86_6.12.2.633，下载到 Program Files 中：

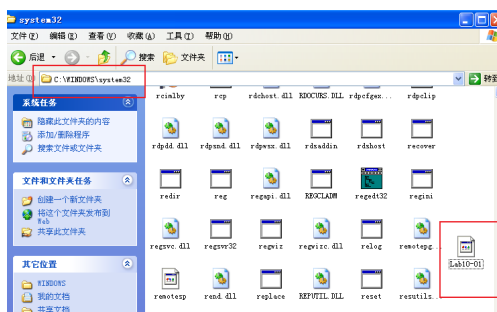


图 3.5: 虚拟机 WinDBG

2. 配置符号表路径:

接下来打开 WinDBG 根据需要自动从 Microsoft 下载符号,具体而言从路径:`c:\Symbols*http://msdl.microsoft.`

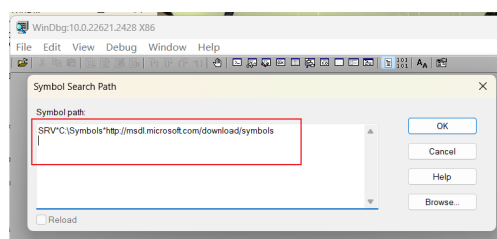


图 3.6: 设置符号表自动下载路径

然后保存工作环境为”Malware Analysis Lab10 “

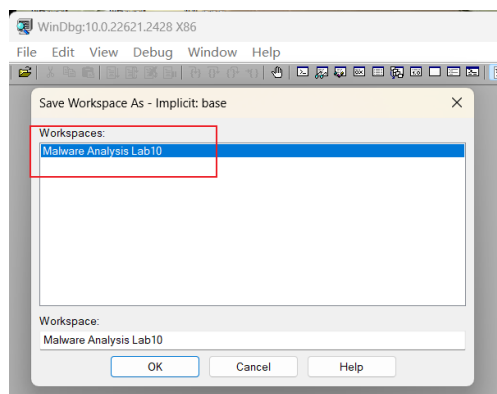


图 3.7: 保存工作环境

3. 配置连接虚拟机

接下来需要使用之前设置好的端口`\\.\pipe\com_1`来连接虚拟机,直接来到 WinDBG 的属性位置,输入`-b -k com:pipe,port=...,resets=0`其中-k 表示指定调试器在内核模式下调试。

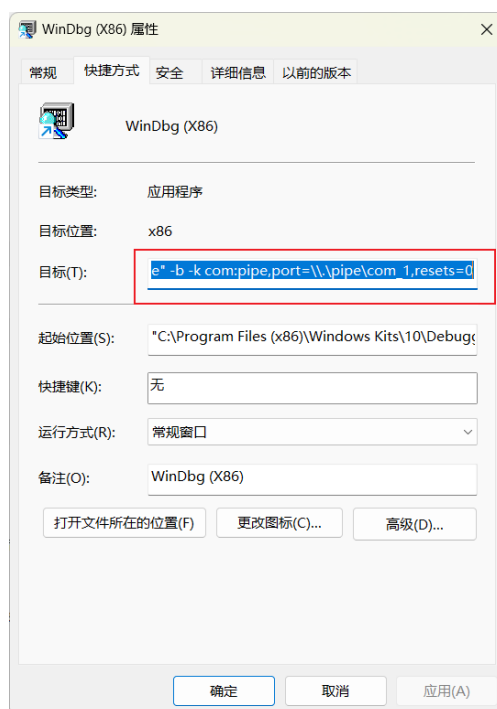


图 3.8: 配置连接虚拟机

然后为了能够每次正确进入内核，将程序设置为永久的管理员模式启动。



图 3.9: 管理员模式

3.2.3 连接尝试

最后虚拟机在开机 30 秒选择之前先开本机的 WinDBG。同时虚拟机选择内核调试模式的操作系统开启：

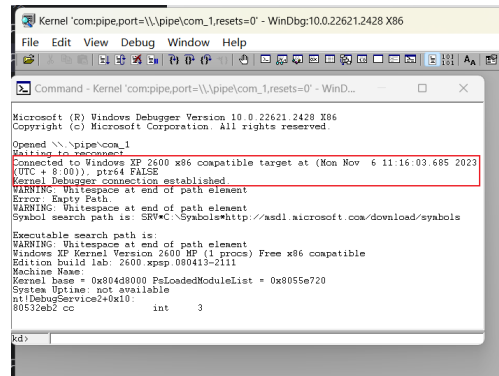


图 3.10: 连接结果

图3.10可以看到显示了连接已经成功建立，Windows XP 是兼容的目标。即成功连接。除此之外也能看到一些关于符号搜索的路径，机器名字，内核基址 0x804d8000 等。另外也有个错误信息提示空路径，这因为我还没加载可执行文件，问题不大的。接下来尝试使用 g 命令让虚拟机继续开启：

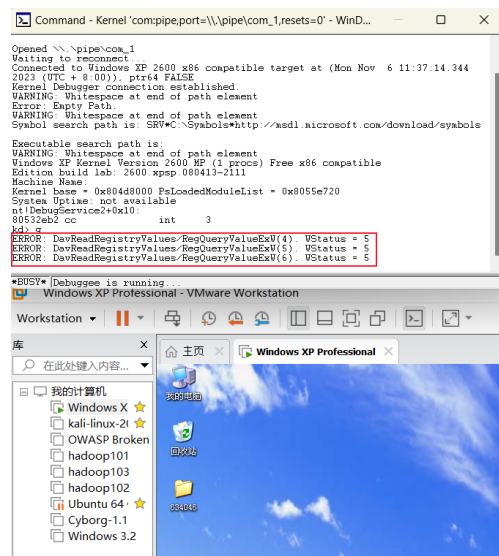


图 3.11: g 调试

图3.11可以看到虚拟机能够正常开启并来到桌面了。虽然如果一直回车仍有一个错误提示 DavReadRegistryValues/RegQueryValueExW 的调用以及 WStatus = 5 的错误状态。这些错误通常涉及到注册表操作以及 Windows 文件系统（WebDAV）相关的问题。

但是暂时不会涉及因此忽略，到此环境配置成功，开始在虚拟机中进行病毒的分析实验。

3.3 Lab010-01

本实验包括一个驱动程序和一个可执行文件。你可以从任意位置运行可执行文件，但为了使程序能够正常运行，必须将驱动程序放到 C:\Windows\System32 目录下，这个目录在受害者计算机中已经存在。可执行文件是 Lab10-1.exe，驱动程序是 Lab10-01.sys。

3.3.1 静态分析

1. Lab10-01.exe

首先对病毒进行简单的静态分析，使用 PEiD 查看其加壳情况和导入表：

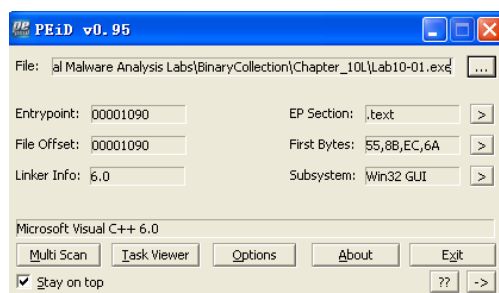


图 3.12: PEiD 加壳

图3.12可以看到没有加壳，然后查看其导入表（值得一提的是它还有资源节，但是没有什么提示：）

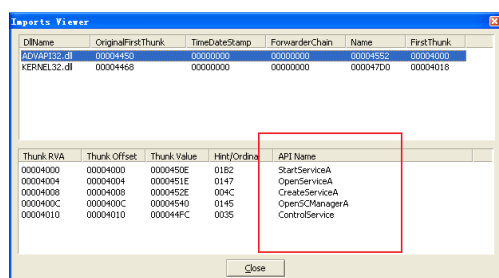


图 3.13: PEiD 导入表

图3.13可以看到一些有意思的导入函数：其中 **OpenSCManager** 是打开服务管理器，**Open**、**Control**、**Start**、**Create** 等和服务有关。因此推断病毒可能是创建了一个服务，然后启动并不断操作它。

除此之外的 Kernel32.dll 中几乎没有发现与系统进行交互的 Windows API。接下来使用 PEView 查看其字符串：

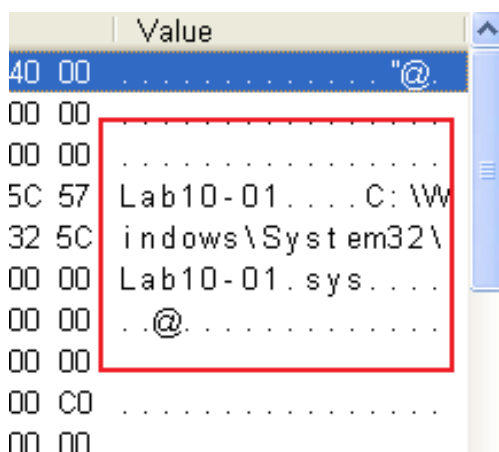


图 3.14: 字符串

图3.14看到了最有意思的字符串就是 **C:\Windows\System32\Lab10-01.sys**。这个是 **Lab10-01** 的另一个系统文件的名字。但是奇怪的是现在这个文件并不在这个路径下，根据提示，我们将其移动到 C:\Windows\System32 下：

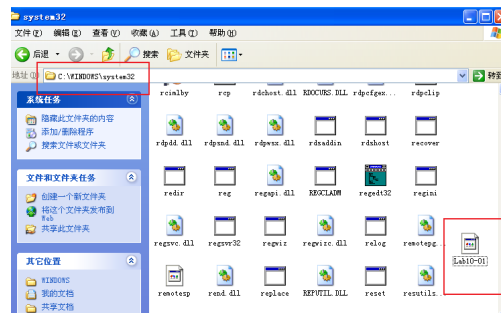


图 3.15: 移动 Lab10-01.sys

图3.15l 可以看到 Lab10-01.sys 已经被移动到了正确的路径下，接下来对其进行分析。

2. Lab10-01.sys

现在查看这个驱动文件，先使用 PEiD 查看其加壳情况和导入表：

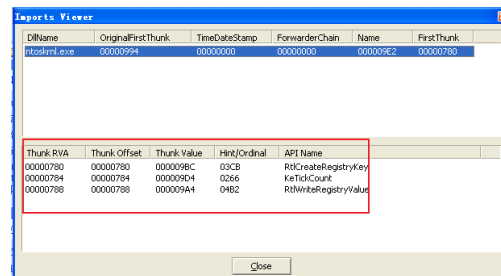


图 3.16: 导入表

图3.16可以看到一些有意思的导入函数，而且仅有这三个：

- **KeTickCount:** 在操作系统内核中使用，与计算系统启动时间或系统运行时间有关，在所有驱动中都会有。
- **RtlCreateRegistryKey:** 创建或打开注册表中的一个键。
- **RtlWriteRegistryKet:** 向注册表中的一个键写入数据。

这告诉我们这个驱动程序 Lab10-01.sys 很有可能存在对注册表进行操作的行为。当然，它没有进行任何加壳。然后使用 PEView 查看其字符串：

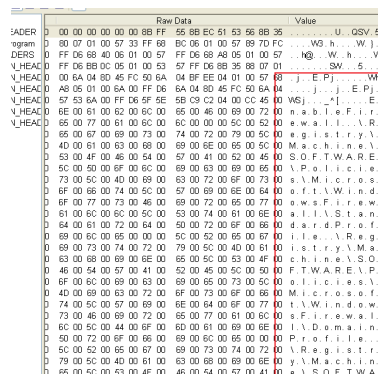


图 3.17: 查看驱动字符串

图3.17虽然看起来比较费劲和不连贯，但是如果把它们来连起来看：

- EnableFirewall: 这看起来是一个列表项，可能表示一个配置选项或开关。它可能与防火墙的启用或禁用相关。**如果设置为 0，则相当于禁用防火墙。**
- \Registry\Machine\SOFTWARE\Policies\Microsoft\WindowsFirewall: 指向系统注册表中的一个位置。用于存储 Windows 防火墙策略的地方。Windows 系统中的策略通常存储在注册表中，以控制不同功能和设置的行为。
- ...\\StandardProfile: 标准配置文件。
- ...\\DomainProfile: 域配置文件

后两者这些配置文件可以具有不同的规则和策略，以适应不同的网络环境。另外值得注意的是开头的前缀\Registry\Machine 不是之前分析的\HKEY_LOCAL_MACHINE，前者代表内核态访问注册表，后者是用户态访问注册表。

上述这些分析都告诉我们这个驱动程序和可执行文件都会对注册表进行操作。

基本静态分析结束，后面会主要使用 Procmon，IDA 和最重要的 WinDBG 进行静态及动态的结合分析。

3.3.2 静态与动态分析结合

从现在开始主要进行动态分析，使用 WinDBG 和其它工具辅助静态分析。

1. IDA Pro

首先直接使用 IDA 进行更深层次的静态分析，直接查看其 main 函数：

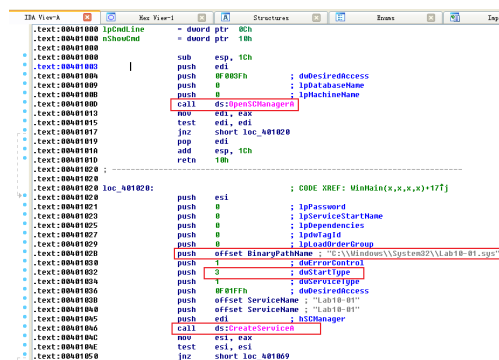


图 3.18: Lab09-01main

图3.18看到了一些重要的东西：

- call OpenSCManager: 打开服务管理器，这个我们之前分析导入函数的时候就发现了；
- push offset BinaryPathName: 其中路径是 System32 下的 Lab10-01.sys。并且作为创建的服务参数之一，这告诉我们服务使用了 Lab10-01.sys 的代码。
- push 3(dwStartType): 查阅资料可知，服务开始类型为 3 对应着 SERVICE_KERNEL_DRIVER，及这个文件将在内核中使用。
- push offset ServiceName(Lab10-01): 作为参数之一，告诉我们服务的名字是 Lab10-01。

即病毒实现了使用 sys 驱动程序的代码创建一个会被加载到内核的服务名为 Lab10-01。然后继续分析后面内容：

```
.text:00401052      push     0F01FFh          ; dwDesiredAccess
.text:00401057      push     ntfs!NtOpenFile ->+00000001
.text:0040105C      push     edi              ; hSchtManager
.text:0040105D      call     ds:OpenServiceA
.text:00401063      mov      esi, eax
.text:00401065      test     esi, esi
.text:00401067      jz       short loc_401066
.text:00401069
.text:00401069 loc_401069:
.text:00401069      push     0                ; CODE XREF: WinMain(x,x,x,x)+50Fj
.text:00401069      push     0                ; lpServiceGuidectors
.text:0040106A      push     0                ; dwNameServiceArgs
.text:0040106B      push     esi              ; hService
.text:0040106C      call     ds:StartServiceA
.text:00401074      test     esi, esi
.text:00401076      jz       short loc_401086
.text:00401078      lea      eax, [esp+24h+ServiceStatus]
.text:0040107C      push     eax              ; lpServiceStatus
.text:0040107D      push     1                ; dwControl
.text:0040107F      push     esi              ; hService
.text:00401080      call     ds:ControlService
.text:00401086
```

图 3.19: 查看更多代码

图3.19能看到更多值得观察的内容，具体而言：

- OpenServiceA：前面会创建服务 Lab10-01，但是如果创建失败，会直接打开同名的服务。这样做保证了如果病毒重复运行只有一次产生目的的服务。
- StartServiceA：打开服务；
- push 1(dwControl): 作为 ControlService 的参数,其查询资料后含义为 **SERVICE_CONTROL_STOP** 即请求服务停止运行，执行必要的清理工作并停止服务。
- ControlServiceA：结合之前的参数我们知道控制服务的调用会导致服务的停止，然后卸载掉相关的驱动。

接下来再使用 IDA 查看下 Lab10-01.sys 的代码，直接进入到的驱动的开始位置 DriverEntry 函数：

```
INIT:00010959 ; ----- SUBROUTINE -----
INIT:00010959 ; Attributes: bp-based frame
INIT:00010959 ; NtStatus _stdcall DriverEntry(PDRIVER_OBJECT DriverObject, PUNICODE_STRING RegistryPath)
INIT:00010959 ; public DriverEntry
INIT:00010959 ; proc near
INIT:00010959 ; DriverObject = dword ptr 8
INIT:00010959 ; RegistryPath = dword ptr 0Ch
INIT:00010959 ; mov     edi, edi
INIT:00010959 ; push    ebp
INIT:00010959 ; mov     ebp, esp
INIT:00010959 ; call    sub_10970h
INIT:00010963 ; pop     ebp
INIT:00010964 ; jmp     sub_10986h
INIT:0001096A ; DriverEntry
INIT:0001096A ; endp
```

图 3.20: DriverEntry

图3.20可以看到在此处并没有做任何实际有用的加载驱动等的行为，直接对一些寄存器进行了操作，因此推测其跳转到的 sub_10906 才是真正的 Driver_Entry 主体。进入其中直接查看其反编译探究其行为：

```
IDA View-A | Pseudocode-A
1 int __stdcall sub_10906(int a1, int a2)
2 {
3     *(_DWORD *){a1 + 52} = sub_10486;
4     return 0;
5 }
```

图 3.21: sub_10906

图3.21看到其只是将某个东西加上偏移量然后存入到一个新的函数位置 sub_10486。到此查看：

```

00401080 sub_10486 (int a1)
00401080 {
00401080     int ValueData; // [sp+5] [bp-30]
00401080
00401080     ValueData = 0;
00401080     RtlCreateRegistryKey(0, L"\\Registry\\Machine\\Software\\Policies\\Microsoft\\Firewall");
00401080     RtlCreateRegistryKey(0, L"\\Registry\\Machine\\Software\\Policies\\Microsoft\\WindowsFirewall\\DomainProfile");
00401080     RtlCreateRegistryKey(0, L"\\Registry\\Machine\\Software\\Policies\\Microsoft\\WindowsFirewall\\StandardProfile");
00401080     RtlWriteRegistryValue(0, L"\\Registry\\Machine\\Software\\Policies\\Microsoft\\WindowsFirewall\\DomainProfile",
00401080         ValueName,
00401080         ValueData,
00401080         REG_DWORD);
00401080     RtlWriteRegistryValue(0, L"\\Registry\\Machine\\Software\\Policies\\Microsoft\\WindowsFirewall\\StandardProfile",
00401080         ValueName,
00401080         ValueData,
00401080         REG_DWORD);
00401080 }

```

图 3.22: sub_10486

图3.31看到了许多我们之前提到的注册表函数了，具体而言：

- **NTSTATUS**：函数返回值，即操作成功或失败的状态；
- **RtlCreateRegistryKey**：创建了 Regisrty 的 MWindowsFirewall 目录下的 Domain-Profile 和 StandardProfile。这和我们静态分析结果相同。
- **RtlWriteRegistryValue**：写入注册表值。它在 DomainProfile 和 StandardProfile 的注册表路径下，写入了名为 ValueName 的注册表值，值的类型为 4 (DWORD)，值的数据为 ValueData，而 ValueData 在函数一开始被初始化为 0。

综合而言，这段代码的主要目的是在注册表中创建一些特定路径下的键和写入与 Windows 防火墙策略相关的注册表值。这可能是病毒在尝试某种配置或策略设置的一部分，用于控制 Windows 防火墙在特定情况下的行为。

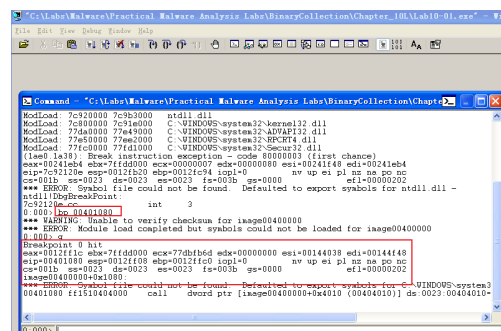
由于 ValueName 仍未知，我们需要进一步探究其内容，但推测就是 Lab10-01 的名字。

此时仔细思考，如果我们直接运行 exe，它会加载驱动但立即结束和卸载，无法观察；而如果运行之前使用内核调试器，但此时 sys 还没被加载到内存中，cpu 无法对其执行，而如果这个阶段内核仍然不可控，那么我们无论如何也无法分析到 sys 文件，因此接下来要使用 WinDBG 了。

2. WinDBG

将结合使用 XP 中的 WinDBG 和主机的 WinDBG 进行实验，分析这个 Lab10-01.exe 和 Lab10-01.sys。

根据前面最后的分析，为了能够正确捕捉到驱动加载的行为，我们需要在驱动被加载后和卸载前之间加入断点，因此根据之前的分析，ControlService 地址位置为 0x00401080，因此通过命令 bp 00401080 打上断点。



```

Command: .!C:\Program Files\Microsoft Windows\WinSxS\x-wwp-00000000-00000000-00000000-00000000\Lab10-01.exe
ModLoad: 7c901200 7c902000 ntldr.dll
ModLoad: 7c902000 7c903000 C:\WINDOWS\system32\kernel32.dll
ModLoad: 77d41000 77d42000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77d42000 77d43000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77d43000 77d44000 C:\WINDOWS\system32\Secur32.dll
ModLoad: 77d44000 77d45000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77d45000 77d46000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77d46000 77d47000 C:\WINDOWS\system32\SHELL32.dll
ModLoad: 77d47000 77d48000 C:\WINDOWS\system32\OLE32.dll
ModLoad: 77d48000 77d49000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77d49000 77d4a000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77d4a000 77d4b000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77d4b000 77d4c000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77d4c000 77d4d000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77d4d000 77d4e000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77d4e000 77d4f000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77d4f000 77d50000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77d50000 77d51000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77d51000 77d52000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77d52000 77d53000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77d53000 77d54000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77d54000 77d55000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77d55000 77d56000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77d56000 77d57000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77d57000 77d58000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77d58000 77d59000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77d59000 77d5a000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77d5a000 77d5b000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77d5b000 77d5c000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77d5c000 77d5d000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77d5d000 77d5e000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77d5e000 77d5f000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77d5f000 77d60000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77d60000 77d61000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77d61000 77d62000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77d62000 77d63000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77d63000 77d64000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77d64000 77d65000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77d65000 77d66000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77d66000 77d67000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77d67000 77d68000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77d68000 77d69000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77d69000 77d6a000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77d6a000 77d6b000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77d6b000 77d6c000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77d6c000 77d6d000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77d6d000 77d6e000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77d6e000 77d6f000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77d6f000 77d70000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77d70000 77d71000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77d71000 77d72000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77d72000 77d73000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77d73000 77d74000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77d74000 77d75000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77d75000 77d76000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77d76000 77d77000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77d77000 77d78000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77d78000 77d79000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77d79000 77d7a000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77d7a000 77d7b000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77d7b000 77d7c000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77d7c000 77d7d000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77d7d000 77d7e000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77d7e000 77d7f000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77d7f000 77d80000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77d80000 77d81000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77d81000 77d82000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77d82000 77d83000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77d83000 77d84000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77d84000 77d85000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77d85000 77d86000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77d86000 77d87000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77d87000 77d88000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77d88000 77d89000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77d89000 77d8a000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77d8a000 77d8b000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77d8b000 77d8c000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77d8c000 77d8d000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77d8d000 77d8e000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77d8e000 77d8f000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77d8f000 77d90000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77d90000 77d91000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77d91000 77d92000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77d92000 77d93000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77d93000 77d94000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77d94000 77d95000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77d95000 77d96000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77d96000 77d97000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77d97000 77d98000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77d98000 77d99000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77d99000 77d9a000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77d9a000 77d9b000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77d9b000 77d9c000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77d9c000 77d9d000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77d9d000 77d9e000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77d9e000 77d9f000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77d9f000 77da0000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77da0000 77da1000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77da1000 77da2000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77da2000 77da3000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77da3000 77da4000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77da4000 77da5000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77da5000 77da6000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77da6000 77da7000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77da7000 77da8000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77da8000 77da9000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77da9000 77daa000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77daa000 77dab000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77dab000 77dac000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77dac000 77dad000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77dad000 77dae000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77dae000 77daf000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77daf000 77db0000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77db0000 77db1000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77db1000 77db2000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77db2000 77db3000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77db3000 77db4000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77db4000 77db5000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77db5000 77db6000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77db6000 77db7000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77db7000 77db8000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77db8000 77db9000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77db9000 77dba000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77dba000 77dbb000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77dbb000 77dbc000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77dbc000 77dbd000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77dbd000 77dbe000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77dbe000 77dbf000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77dbf000 77dc0000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77dc0000 77dc1000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77dc1000 77dc2000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77dc2000 77dc3000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77dc3000 77dc4000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77dc4000 77dc5000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77dc5000 77dc6000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77dc6000 77dc7000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77dc7000 77dc8000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77dc8000 77dc9000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77dc9000 77dca000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77dca000 77dcb000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77dcb000 77dcc000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77dcc000 77dcd000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77dcd000 77dce000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77dce000 77dcf000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77dcf000 77dd0000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77dd0000 77dd1000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77dd1000 77dd2000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77dd2000 77dd3000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77dd3000 77dd4000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77dd4000 77dd5000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77dd5000 77dd6000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77dd6000 77dd7000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77dd7000 77dd8000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77dd8000 77dd9000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77dd9000 77dda000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77dda000 77ddb000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77ddb000 77ddc000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77ddc000 77ddd000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77ddd000 77dde000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77dde000 77ddf000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77ddf000 77de0000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77de0000 77de1000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77de1000 77de2000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77de2000 77de3000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77de3000 77de4000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77de4000 77de5000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77de5000 77de6000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77de6000 77de7000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77de7000 77de8000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77de8000 77de9000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77de9000 77dea000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77dea000 77deb000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77deb000 77dec000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77dec000 77ded000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77ded000 77dee000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77dee000 77def000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77def000 77df0000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77df0000 77df1000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77df1000 77df2000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77df2000 77df3000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77df3000 77df4000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77df4000 77df5000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77df5000 77df6000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77df6000 77df7000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77df7000 77df8000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77df8000 77df9000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77df9000 77dfa000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77dfa000 77dfb000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77dfb000 77dfc000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77dfc000 77dfd000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77dfd000 77dfe000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77dfe000 77dff000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77dff000 77e00000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77e00000 77e01000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77e01000 77e02000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e02000 77e03000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77e03000 77e04000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77e04000 77e05000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77e05000 77e06000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e06000 77e07000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77e07000 77e08000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77e08000 77e09000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77e09000 77e0a000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e0a000 77e0b000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77e0b000 77e0c000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77e0c000 77e0d000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77e0d000 77e0e000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e0e000 77e0f000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77e0f000 77e10000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77e10000 77e11000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77e11000 77e12000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e12000 77e13000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77e13000 77e14000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77e14000 77e15000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77e15000 77e16000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e16000 77e17000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77e17000 77e18000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77e18000 77e19000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77e19000 77e1a000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e1a000 77e1b000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77e1b000 77e1c000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77e1c000 77e1d000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77e1d000 77e1e000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e1e000 77e1f000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77e1f000 77e20000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77e20000 77e21000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77e21000 77e22000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e22000 77e23000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77e23000 77e24000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77e24000 77e25000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77e25000 77e26000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e26000 77e27000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77e27000 77e28000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77e28000 77e29000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77e29000 77e2a000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e2a000 77e2b000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77e2b000 77e2c000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77e2c000 77e2d000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77e2d000 77e2e000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e2e000 77e2f000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77e2f000 77e30000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77e30000 77e31000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77e31000 77e32000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e32000 77e33000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77e33000 77e34000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77e34000 77e35000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77e35000 77e36000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e36000 77e37000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77e37000 77e38000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77e38000 77e39000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77e39000 77e3a000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e3a000 77e3b000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77e3b000 77e3c000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77e3c000 77e3d000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77e3d000 77e3e000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e3e000 77e3f000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77e3f000 77e40000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77e40000 77e41000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77e41000 77e42000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e42000 77e43000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77e43000 77e44000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77e44000 77e45000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77e45000 77e46000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e46000 77e47000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77e47000 77e48000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77e48000 77e49000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77e49000 77e4a000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e4a000 77e4b000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77e4b000 77e4c000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77e4c000 77e4d000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77e4d000 77e4e000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e4e000 77e4f000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77e4f000 77e50000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77e50000 77e51000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77e51000 77e52000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e52000 77e53000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77e53000 77e54000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77e54000 77e55000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77e55000 77e56000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e56000 77e57000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77e57000 77e58000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77e58000 77e59000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77e59000 77e5a000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e5a000 77e5b000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77e5b000 77e5c000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77e5c000 77e5d000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77e5d000 77e5e000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e5e000 77e5f000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77e5f000 77e60000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77e60000 77e61000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77e61000 77e62000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e62000 77e63000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77e63000 77e64000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77e64000 77e65000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77e65000 77e66000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e66000 77e67000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77e67000 77e68000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77e68000 77e69000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77e69000 77e6a000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e6a000 77e6b000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77e6b000 77e6c000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77e6c000 77e6d000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77e6d000 77e6e000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e6e000 77e6f000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77e6f000 77e70000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77e70000 77e71000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77e71000 77e72000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e72000 77e73000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77e73000 77e74000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77e74000 77e75000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77e75000 77e76000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e76000 77e77000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77e77000 77e78000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77e78000 77e79000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77e79000 77e7a000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e7a000 77e7b000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77e7b000 77e7c000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77e7c000 77e7d000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77e7d000 77e7e000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e7e000 77e7f000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77e7f000 77e80000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77e80000 77e81000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77e81000 77e82000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e82000 77e83000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77e83000 77e84000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77e84000 77e85000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77e85000 77e86000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e86000 77e87000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77e87000 77e88000 C:\WINDOWS\system32\USER32.dll
ModLoad: 77e88000 77e89000 C:\WINDOWS\system32\RPCRT4.dll
ModLoad: 77e89000 77e8a000 C:\WINDOWS\system32\ADVAPI32.dll
ModLoad: 77e8a000 77e8b000 C:\WINDOWS\system32\GDI32.dll
ModLoad: 77e8b000 77e
```

接下来回到之前已经启动的主机的 WinDBG 中(前面的过程都是在内核调试的虚拟机中进行,并且提前使用主机上的 WinDBG 进行了连接和 go),现在 break 暂停内核,然后通过命令!drvobj 试图找到已经被加载的驱动对象 lab10-01:

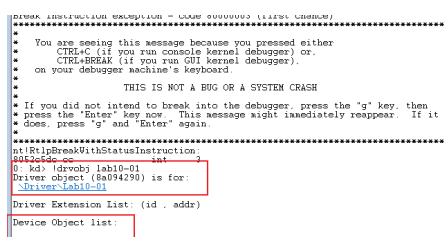


图 3.24: 查看驱动对象

图3.24看到了有一些重要信息:

- Driver object:8a094290，这个就是驱动对象加载的地址。并且名字是 Lab10-01。
- Device Object list：为空，证明这个驱动程序不会供用户空间的应用程序访问。

b 在知道了驱动对象的地址，通过命令 `dt __DRIVER_OBJECT 8a094290` 查看其对应的信息：

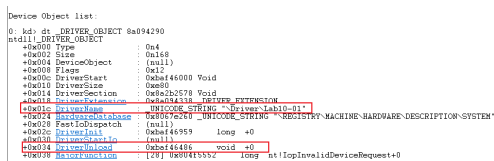


图 3.25: 驱动信息

图3.25看到了一些重要信息:

- 驱动名：Lab10-01，再次验证了之前的猜测；
- DriverUnload: +0x34，意味着在偏移 0x34 位置处是卸载的调用函数，对其地址进行断点调试。在 0xbaf46486 处。

于是在卸载函数运行之前设置一个断点在此位置 bp 0xbaf46486:

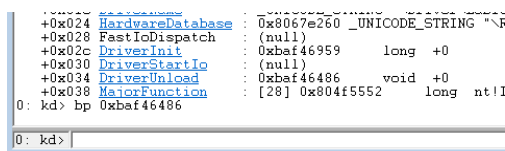


图 3.26: 断掉卸载函数位置

在主机上设置完了断点之后，命令 `g` 让虚拟机内核继续执行，然后在虚拟机的 WinDBG 中，也 `g` 恢复其运行。由于接下来就是我们刚刚设置的断点，因此内核调试器立即命中了断点，在卸载之前虚拟机卡死。

然后我们回到主机的 WinDBG 中，单步调试代码，发现了一些有意思的东西：

```

~: ~~~~~
nt!RtlpGetRegistryHandle+0x76:
805e7d56 7473      je         nt!RtlpGetRegistryHandle+0xeb (805e7ddb)
0: kd> t
nt!RtlpGetRegistryHandle+0xeb:
805e7ddb 51        push     ecx
0: kd> t
nt!RtlpGetRegistryHandle+0xec:
805e7ddc 8d85e8dfff lea      eax,[ebp-218h]
0: kd> t

```

图 3.27: RtlpGetRegistryHandle

图3.27可以看到一个 **RtlpGetRegistryHandle**，用于获取注册表的句柄，可以用于访问和操作注册表中的数据。证明了病毒此时要进行注册表相关的操作了

```

~: ~~~~~
nt!RtlCreateRegistryKey+0xd6:
805e883a 85c0      test     eax,eax
1: kd> p
nt!RtlCreateRegistryKey+0x18:
805e883c 7c10      jnl      nt!RtlCreateRegistryKey+0x2a (805e884e)
1: kd> p
nt!RtlCreateRegistryKey+0x1a:
805e883e f6450b40 test     byte ptr [ebp+0Bh],40h
1: kd> p
nt!RtlCreateRegistryKey+0x1e:
805e8842 7508      jne      nt!RtlCreateRegistryKey+0x28 (805e884c)
1: kd> p
nt!RtlCreateRegistryKey+0x20:
805e8844 ff750c    push    dword ptr [ebp+0Ch]
1: kd> p
nt!RtlCreateRegistryKey+0x23:
805e8847 e8d486f1ff call    nt!ZwClose (80500f20)
1: kd> p
nt!RtlCreateRegistryKey+0x28:
805e884c 33c0      xor     eax,eax
1: kd> p
nt!RtlCreateRegistryKey+0x2a:
805e884e 5d        pop     ebp

```

图 3.28: RtlCreateRegistryKey

图??可以看到两次调用了 **RtlCreateRegistryKey**，用于创建注册表中的一个新的键（key），可以用于在注册表中创建新的数据结构。实际上一共出现了三次，这里只截取了两次。

```

~: ~~~~~
nt!IopLoadUnloadDriver+0x19:
8058245b 33ff      xor     edi,edi
1: kd> p
nt!IopLoadUnloadDriver+0x1b:
8058245d eb49      jmp     nt!IopLoadUnloadDriver+0x66 (805824a8)
1: kd> p
nt!IopLoadUnloadDriver+0x66:
805824a8 53        push    ebx
1: kd> p
nt!IopLoadUnloadDriver+0x67:
805824a9 897e28    mov     dword ptr [esi+28h],edi
1: kd> p
nt!IopLoadUnloadDriver+0x6a:
805824ac 53        push    ebx

```

图 3.29: IopLoadUnloadDriver

图3.29在最后的位置我们还看到了 **IopLoadUnloadDriver**，于加载或卸载设备驱动程序，可以用于控制系统中设备驱动程序的加载和卸载操作。证明要进行驱动程序的卸载了。到此分析就接近尾声了。

除此之外，过程中还注意到了实际上程序还有两次调用 **RtlWriteRegistryValue** 和两次将 **EnableFirewall** 设置为 0。值得一提的是后者是在内核态中进行的操作，在核心中禁用防火墙，有一些病毒查杀的 app 会以为是系统所为，而难以被检测到。

最后，如果想要在具体地分析卸载函数，最好在 IDA Pro 中进行。由于 WinDBG 中的地址显示和 IDA 不同，但是地址相对于基址的偏移是相同的，因此可以先在 WinDBG 中计算出来地址偏移。使用 **lm** 查看函数从 WinDBG 加载文件开始时的偏移量：

```

badee000 badef400 vmsbmouse (deferred)
badfe000 badff100 dump_VHLLIB (deferred)
bae9b000 bae9bc00 audstub (deferred)
bae9f000 bae9fd00 dxqthk (deferred)
baf46000 baf46e80 Lab10_01 (no symbols)
bafcb000 bafcb800 Null (deferred)
bf800000 bf9c2800 win32k (deferred)
bf9c3000 bf9d4600 dxq (deferred)
bf9d5000 bfba0a00 vmx_fb (deferred)

Unloaded modules:
b0224000 b024f000 kmixer.sys
b1a83000 b1aae000 kmixer.sys
bae83000 bae84000 drmkau.sys
baaa8000 baab5000 DMusic.sys
b1aae000 b1ad1000 aac.sys
1: kd> ||

```

图 3.30: 偏移量

图3.30看到显示了 Lab10-01 的文件加载位置是 baf46000，而卸载函数在 baf46486，相互减得到偏移量 0x486。而 IDA 中加载的基地址是 0x00100000，因此理论上卸载函数是在 0x00100486 处。

到 IDA 中查看：

```

00101005 sub_101005 (int a1)
00101005 {
00101005     int ValueData; // [sp+4] [bp-30]
00101005
00101005     ValueData = 0;
00101005     RtlCreateRegistryKey(0, "\\Registry\\Machine\\SOFTWARE\\Policies\\Microsoft\\Firewall");
00101005     RtlCreateRegistryValue(0, "\\Registry\\Machine\\SOFTWARE\\Policies\\Microsoft\\Firewall\\DomainProfile");
00101005     RtlCreateRegistryValue(0, "\\Registry\\Machine\\SOFTWARE\\Policies\\Microsoft\\Firewall\\StandardProfile");
00101005     RtlWriteRegistryValue(
00101005         0,
00101005         "\\Registry\\Machine\\SOFTWARE\\Policies\\Microsoft\\Firewall\\DomainProfile",
00101005         "No",
00101005         REG_DWORD,
00101005         0);
00101005     return RtlWriteRegistryValue(
00101005         0,
00101005         "\\Registry\\Machine\\SOFTWARE\\Policies\\Microsoft\\Firewall\\StandardProfile",
00101005         "No",
00101005         REG_DWORD,
00101005         0);
00101005 }
    
```

图 3.31: sub_10486

图3.31中我们其实已经在这部分开头的 IDA 中分析过了, RtlCreateRegistryKey 和 RtlWriteRegistryValue 的操作均验证了我们使用 WinDBG 的分析。Lab10-01 分析到此结束。

3.3.3 实验问题

分析完病毒 Lab10-01 的可执行程序 and 驱动程序后，对问题进行回答。

1. Q1: 这个程序是否直接修改了注册表 (使用 procmon 来检查)?

回答：之前没使用 Procmon，现在进行动态检查：

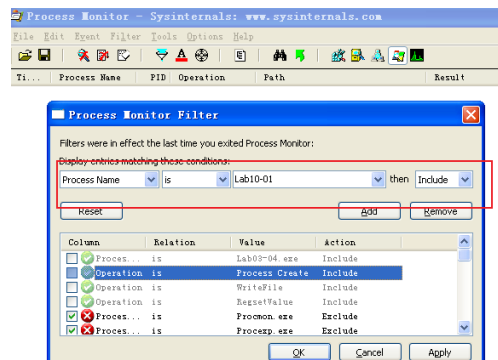


图 3.32: Procmon

图3.32设置对应的过滤器（包括进程名为各种服务名的过滤器）然后以管理员身份运行该程序后发现了其行为包括：

- RegSetValue: 写键值 HKLM\SOFTWARE\Microsoft\Cryptography\RNG\Seed。这与静态分析结果相同。
- createServiceA: 对注册表进行间接修改。

值得一提的是病毒也在内核态对注册表进行了修改，不过这些修改无法被运行在用户态的 Procmon 检测到。

2. Q2: 用户态的程序调用了 ControlService 函数，你是否能够使用 WinDbg 设置一个断点，以此来观察由于 Controlservice 的调用导致内核执行了怎样的操作？

回答：在前面使用 WinDBG 动态分析的过程中，我们通过在虚拟机中也打开一个 WinDBG 加载可执行程序与主机的 WinDBG 配合使用。当虚拟机中的可执行程序被 bp 00401080 处的断点停止了后，通过!drvobj 命令找到了驱动程序地址。并在这个位置设定断点，重启可执行文件后触发。

3. Q3: 这个程序做了些什么？

回答：综合前面静态和动态分析，我们可以得出结论——

- 这个程序创建一个服务来加载驱动 Lab10-01.sys；
- 驱动代码会创建注册表键\Registry\Machine\SOFTWARE\Policies\Microsoft WindowsFire-wall 下的 DomainFile 和 StandardFile。
- 以上操作会在内核态禁用防火墙，难以察觉。

3.4 Lab10-02

该实验的文件为 Lab10-02.exe。在进行分析前使用快照方便后期的恢复。

3.4.1 静态分析

首先使用 PEiD 检查其加壳和导入函数情况：

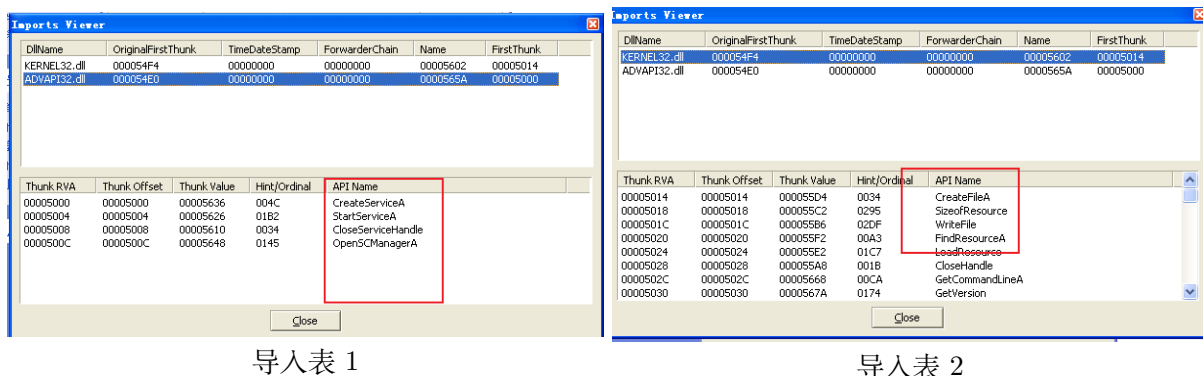


图 3.33: 查看导入表

仍然没有被加壳，另外在图3.33中看到一些有意思的导入函数：

- OpenSCManager, CreateServiceA 等一系列服务函数：推测病毒再次创建和启用了服务类似的功能。
- CreateFile 和 WriteFile：告诉我们病毒可能在某处进行了文件的创建和写入。
- SizeOfResource 和 LoadResource：程序带有资源节，并且调用了；

由于调用了资源节并进行了某些处理，因此有必要进一步查看，通过 Resource Hacker 查看结果如下：

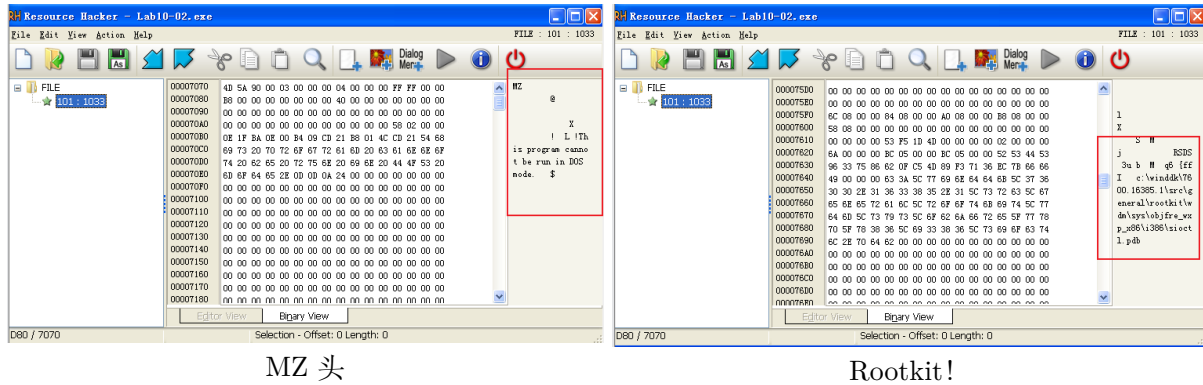


图 3.34: Resource Hacker

图3.34看到资源节中一些有趣的东西：

- MZ 头：包含了另一个 PE 头部，这可能代表病毒 Lab10-02 会使用别的恶意文件。
- C:\...\rootkit：我们明确看到了 rootkit 的字样，即一种恶意软件，可以在计算机系统中植入并在用户不知情的情况下进行恶意操作，例如窃取个人信息、监视用户活动等。

综上初步推测病毒会创建服务和充当 Rootkit。

接下来使用 PVIEW 查看其字符串：

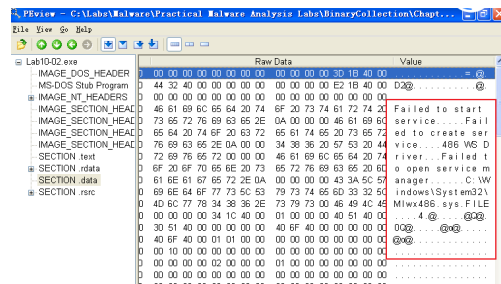


图 3.35: PVIEW

图3.35可以看到很多有趣的字符串：

- Failed to create/start service(open service manager)：这都与我们静态分析导入表结果相同，病毒会尝试创建服务，并且推测如果中间出现问题会提示错误。
- 486 WS Driver：看起来像是一个驱动名字；
- C:\Windows\System32\Mwx486sys：这明显是一个恶意的系统文件，并且结合之前导入表分析的函数，猜测这个就是新创建并且写入的文件。并且根据它的名字 sys，推测其可能包含操作系统加载的内核代码。

静态分析到此结束，后面进行一定的动态分析。

3.4.2 静态分析和动态分析结合

在此之后，主要使用 IDA 和 PeStudio 还有一些其它工具进行静态和动态的综合分析。这里没必要使用 WinDBG 的原因会稍后解释。

首先再次使用 Procmon 查看运行的效果：

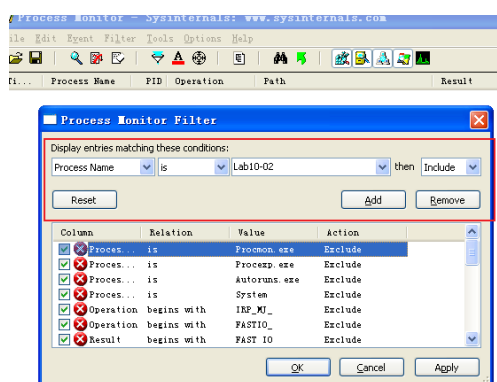


图 3.36: Procmon 监控

图3.36再使用类似的监控过滤器(后面也尝试了一些服务名),最后观察到确实病毒在 C:\Windows\System32 下创建了文件,并且这个文件又被充当了一个服务。

但是然后我们去这个目录下却没有发现这个文件,而我们又没在导入表分析中发现删除文件的函数,因此推测其进行了某种隐藏。这与“用户不知情的情况下进行恶意操作”相吻合,加上还和内核有关的驱动,进一步验证了病毒可能是个 Rootkit。

基于此,我们尝试寻找这个 Rootkit,通过命令行中 sc 查到了这个服务 486 WS Driver (字符串分析结果)：

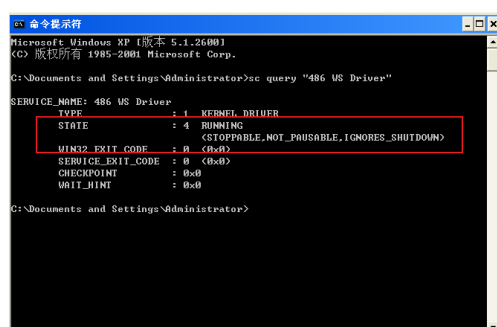


图 3.37: sc

图3.37看到服务仍在运行,但是鉴于硬盘上没有发现,推测其内核代码在内存中。然后使用 WinDBG 查看验证：

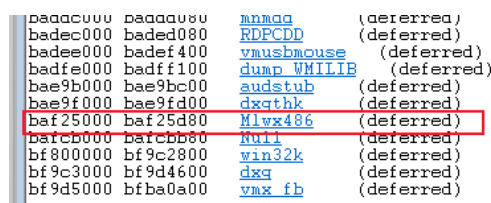


图 3.38: lm

图3.38是使用主机的 WinDBG 连上虚拟机进行内核调试,使用 lm 命令查看被加载的驱动,确实发现了和字符串分析结果相同的名为 Mlws486 位于 baf25000 位置。即这个驱动已经被加载到内存,但不会在硬盘到的文件系统中显式,也符合 Rootkit 的特征。

接下来由于知道了这个恶意 RootKit 会加载恶意驱动，但如果它想要让它的驱动运行，进而控制系统行为，那么它需要将 SSDT（即系统服务描述符表）中的一个函数指针修改为指向其自己的代码，这样当应用程序尝试调用这个系统服务时，实际上执行的是 Rootkit 的代码。

因此我们通过命令 `dd dwo(KeServiceDescriptorTable) L100` 查找从 KeServiceDescriptorTable 开始的前 100 个双字的内存内容：

ReadVirtual:	80505650	not properly sign extended
80505650	805cc188	805ee740 805ee4f6 80617744
80505660	8064614e	805ca4ae 805f8ae4 805f472c
80505670	805f4918	805b93da 8060f7ba 80577ed6
80505680	806170e4	806170e4 80540bb6 806113ac
80505690	8061200c	baf25486 805bf60e 8057b1b2
805056a0	8060f882	8057802a 80616880 8057ba1e
805056b0	805d6c1a	805a63c8 805cdf50 805ccb7e
805056c0	805ee820	806117aa 806185f4 80579be2
805056d0	b021a300	806238d4 80617ed2 805c6288
805056e0	80623f80	80618682 8057c800 805b959c
805056f0	805c1056	806158b4 805c4c3c 80617100
80505700	806170ba	8061208c 8061384c 806177fc
80505710	806138de	b021a350 805b9c2a 8057ccea
80505720	805d2232	80545eb4 80615526 8057d48a
80505730	8057d9f4	805a6e50 805b528a 805d3754
80505740	8061800a	806159e4 80579eda 80644028

图 3.39: KeServiceDescriptorTable

图3.39可以明显看到恶意驱动 `Mlws486` 的地址 `baf25486` 位于 `baf25000` 到 `baf25d80` 之间！证明有某个系统服务被替代了，且在 `ntoskrnl` 模块中。

然后将快照恢复，我们可以发现被替代的函数是 `NtQueryDirectoryFile`，这是一个用于提取文件和目录信息的通用函数。当应用程序（如 `FindFirstFile`、`FindNextFile` 或 `Windows 资源管理器`）调用这个函数时，实际上执行的是 Rootkit 中的代码，而不是原本的系统服务。这样，Rootkit 就可以控制这个函数的行为，例如隐藏某些文件，这也解释了为什么我们找不到 `Mlwx486.svs` 文件。

接下来我们需要重点分析其所拥有的恶意驱动文件的目的，还有病毒可能存在的其它行为。比如去探索恶意驱动代码代替 `NtQueryDirectoryFile` 的挂钩函数 `baf25d80` 的函数作用。而分析函数目的使用 IDA 要比 WinDBG 动态分析方便的多，因此后面将重点使用 IDA 分析功能。

首先打开 IDA 查看其 `main` 函数的反编译：

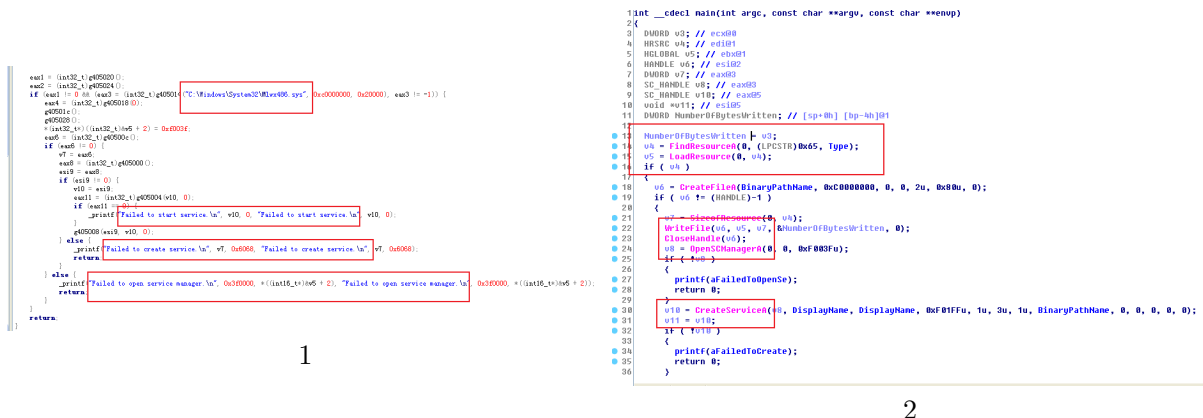


图 3.40: main 反编译

然后我们可以看到图3.40一些有意思的地方：

- `FindResourceA` 和 `LoadResourceA`：我们发现其加载了其内部的资源，然后结合另外一个图可知加载的资源后面被写入新创建的 `Mlwx486.sys`。这就解释了为什么 `Lab10-02` 没有自己的 `sys`，而它的 `sys` 就在它的资源节！
- `System32` 下的 `Mlwx486.sys`：这和我们之前分析的一样，会放在 `System32` 即正常的系统文件中，这样可以隐藏起来避免怀疑。

- Service: 开启了服务管理器, 创建 486 WS Driver 等各种行为, 然后将 **Mlwx486.sys** 设置为服务的可执行文件。
- 在开启服务过程中, 出现问题会报错, 这也在静态分析字符串发现了。

这些都和我们之前分析的相同, 然后我们回到刚才的目的, 我们要找到替代原先 SSDT 中的挂钩函数的目的, 挂钩函数显然在驱动 **Mlwx486** 中, 而经过刚才的分析 **Mlwx486** 又在资源节中! 因此我们直接试图将资源节提取出来, 使用工具 **ResourceHacker**:

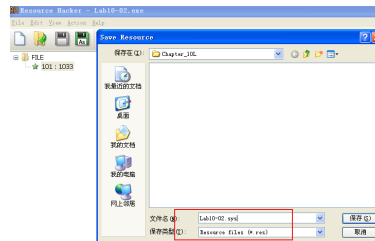


图 3.41: RH 提取 Lab10-02 资源节

图3.41提取到全部的资源节直接命名为 **Lab10-02.sys** 也就是后来写入的 **Mlwx486.sys**。接下来使用 IDA 对这个文件进行分析:



图 3.42: sys 的 DriverEntry

图3.42的入口函数 **DriverEntry** 直接调用了 **sub_10772**, 返回 **sub_10706**。其中值得注意的是 **sub_10706** 传入的参数分别是驱动程序的对象和注册路径, 推测其为主函数。接下来查看这两个:

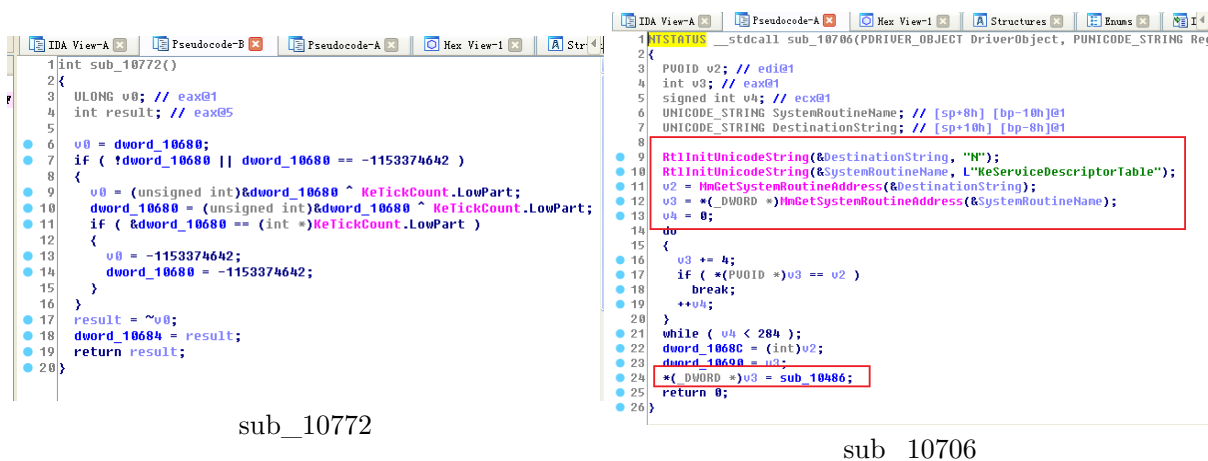


图 3.43: 两个函数

图3.43看到:

- **sub_10772**: 实现中使用了 **KeTickCount.LowPart**, 一个系统计时器的计数器值, 每个时钟周期递增一次。这是在进行一些驱动程序都需要用到的初始化。

- **sub_10706:**同样也是在初始化。不过其将 SystemRoutineName 设置为 KeServiceDescriptorTable。调用 MmGetSystemRoutineAddress 函数获取”KeServiceDescriptorTable” 对应的系统例程地址。然后去遍历找某个指针。最后替换为 sub_10486 函数的地址。这样一来,当系统调用”N”时,实际上会执行 sub_10486 函数的代码。

通过上面的分析我们发现,sub_10486 或许就是我们再 WinDBG 分析的要找的那个内核挂钩函数。而实际上其实也可以换个角度思考,加载的位置 10486 正好是 IDA 载入的 10000 偏移 0x486 的位置,这个和动态分析结果相同 (baf25486-baf25000=0x486),所以这也是为什么恶意程序叫 Mlwx486! 因为偏移位置是 486,我太聪明了嘿嘿,这都被我发现了!

然后去查看 sub_10486 位置的代码:

1 **NTSTATUS** __stdcall sub_10486(HANDLE FileHandle, HANDLE Event, PIO_APC_R...

2 {

3 **PUOID** v11; // esi@1

4 **NTSTATUS** v12; // eax@1

5 **PUOID** v13; // edi@1

6 **char** v14; // bl@4

7 **NTSTATUS** RestartScana; // [sp+38h] [bp+30h]@1

8

9 v11 = FileInformation;

10 v12 = NtQueryDirectoryFile(

11 FileHandle,

12 Event,

13 ApcRoutine,

14 ApcContext,

15 IoStatusBlock,

16 FileInformation,

17 FileInformationLength,

18 FileInformationClass,

19 ReturnSingleEntry,

20 FileName,

21 RestartScan);

22 v13 = 0;

23 RestartScana = v12;

24 if (FileInformationClass == 3 && v12 >= 0 && !ReturnSingleEntry)

25 {

26 while (1)

21 RestartScan);

22 v13 = 0;

23 RestartScana = v12;

24 if (FileInformationClass == 3 && v12 >= 0 && !ReturnSingleEntry)

25 {

26 while (1)

27 {

28 v14 = 0;

29 if (RtlCompareMemory((char *)v11 + 94, &word_1051A, 8u) == 8)

30 {

31 v14 = 1;

32 if (v13)

33 {

34 if (*((_DWORD *)v11))

35 *((_DWORD *)v13) += *((_DWORD *)v11);

36 else

37 *((_DWORD *)v13) = 0;

38 }

39 if (!*((_DWORD *)v11))

40 break;

41 if (!v14)

42 v13 = v11;

43 v11 = (char *)v11 + *((_DWORD *)v11);

44 }

45 }

46 return RestartScana;

47

48 }

图 3.44: sub_10486

图3.44看到了一些关于这个挂钩函数重要的内容:

- 1 中的 NtQueryDirectoryFile: 这点和之前的动态分析结果相同,这个函数在内核中被替代了。不过这个恶意的 sys 竟然也调用了标准的 NtQueryDirectoryFile 查询,根据传入的参数进行相应的操作。
- 2 中的 RtlCompareMemory: 函数比较文件信息中的某个偏移处的内容与 word_1051A 的内容,如果相同则执行一些操作。
- word_1051A: 经过确认,就是 Mlwx, 即那个恶意的驱动文件的名字。
- 隐藏文件的操作

因此该函数会在系统试图对目录文件进行遍历查询时候调用。它会装作正常的 NtQueryDirectory-File 执行,不过,当它查找到某个文件名字正好是 Mlwx486 时候,它会通过指针来进行一些操作。

而这个操作就与隐藏文件有关。我们重新回顾整个代码:

- 遍历链表:使用循环遍历链表,通过不断更新指针 v11 来移动到下一个 FILE_BOTH_DIR_INFORMATION 结构。
- 查找以”Mlwx”开头的文件:使用 RtlCompareMemory 函数来比较当前 FILE_BOTH_DIR_INFORMATION 结构中特定偏移处的内容是否以”Mlwx”开头。如果相同,表示找到了目标文件。

- 修改链表中前一个结构的第一个域：如果找到了目标文件，则设置标志 v14 为 1，然后检查 v13 是否为空。如果 v13 不为空，则更新前一个结构的第一个域，使其指向下一个结构，从而跳过当前结构。
- 跳出循环：在循环中，如果当前结构的第一个域为 0，则跳出循环。否则，如果未找到目标文件，则将 v13 设置为当前结构的地址，以备后续可能需要更新前一个结构的第一个域。

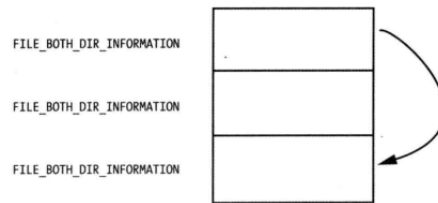


图 3.45: sub_10486 指针操作

图3.45准确地描述了会进行的操作。因为 NtQueryDirectoryFile 函数返回的是一个包含多个 FILE_BOTH_DIR_INFORMATION 结构的链表，其中每个结构对应一个文件的信息。这些结构之间通过第一个域来链接，即每个结构的第一个域指向下一个结构的偏移量。因此 sys 恶意驱动通过类似于指针跳转的方法实现了对文件的隐藏，这就解答了我们动态分析留下的疑问。

3.4.3 实验问题

到此就基本完成了所有对 Lab10-02 的分析了，开始回答问题：

1. Q1: 这个程序创建文件了吗？它创建了什么文件？

回答：通过之前的静态分析导入函数 CreateFile 或者是 Procmon 的监视我们都发现了文件被创建。根据我们 IDA 和 WinDBG 的分析，其创建的文件是 Mlwx486.sys，写入的内容就是 Lab10-02 的资源节，这个恶意驱动通过某种方式进行了隐藏，并且躲在了 System32 下装作正常的程序。

2. Q2: 这个程序有内核组件吗？

回答：根据我们 WinDBG 的分析结果，Lab10-02 所具有的内核组件就是那个恶意程序 Mlwx486 的内容，它被存储在这个文件的资源节中。最后将 Mlwx486 写入硬盘并作为一个服务加载到内核。

3. Q3: 这个程序做了些什么？

回答：根据我们上面的分析可知，这个程序在 System32 下创建了文件 Mlwx486.sys。然后把它装作是一个正常的 NtQueryDirectoryFile 函数加载在内核中，它使用 SSDT 钩来覆盖 NtQueryDirectoryFile 的入口，并且实现将任何以 Mlwx 开头的文件全部隐藏。

一言以蔽之，它就是一个隐藏文件的 Rootkit。

3.5 Lab10-03

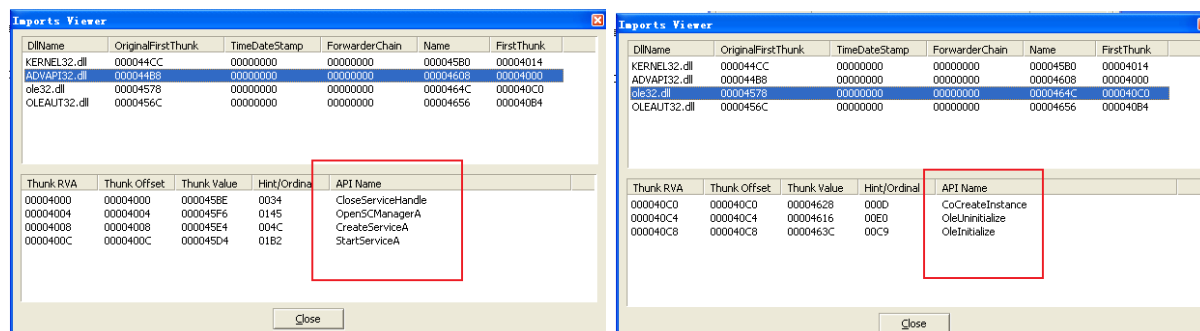
本实验包括一个驱动程序和一个可执行文件。你可以从任意位置运行可执行文件，但为了程序能够正常运行，必须将驱动程序放到 C:\Windows\System32 目录下，这个目录在受害者计算机中已经存在。可执行文件是 Lab10-3.exe，驱动程序是 Lab10-03.sys。

3.5.1 静态分析

首先对 exe 和 sys 进行一些简单的静态分析：

1. Lab10-03.exe

首先使用 PEiD 查看其加壳情况和导入表：



导入表 1

导入表 2

图 3.46: 查看导入表

同样没有进行加壳，然后查看其导入表，图3.46又可以看到一些有意思的东西：

- ADVAPI32.DLL：其中包含 **OpenSCManagerA**，**CreateServiceA** 和 **StartServiceA** 等，再次与服务有关。
- ole.dll： **CoCreateInstance** 和 **OleInitialize** 都是 COM 库中的函数，COM 对象的创建和初始化过程中起着关键的作用。具体而言前者创建 COM 对象，后者在当前单元初始化 COM 库。

由此可知，病毒很有可能再次存在服务相关的操作，并且还会使用 COM 库。接下来查看其字符串：

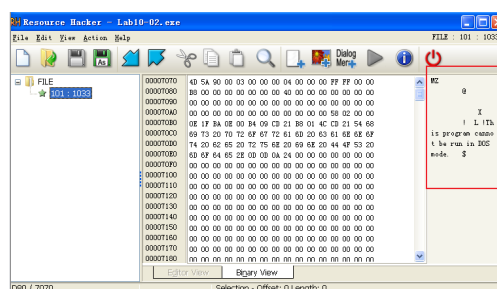


图 3.47: exe 字符串

图3.47能看到一些有意思的字符串：

- <https://www.malwareanalysisbook.com/ad.html>：经典的广告彩蛋网址。
- \Process Helper：帮助用户管理和优化工作流程。推测可能和进程操作有关。
- C:\Windows\System32\Lab10-03.sys。即另一个 sys 的系统文件在 system32 下。

由此可知，病毒还有网络和进程相关的行文，并且还需要 sys 在 System32 路径下才能正常运行。于是将 sys 移动到这个路径下后继续分析。

2. Lab10-03.sys

首先使用 PEiD 查看其加壳情况和导入表：

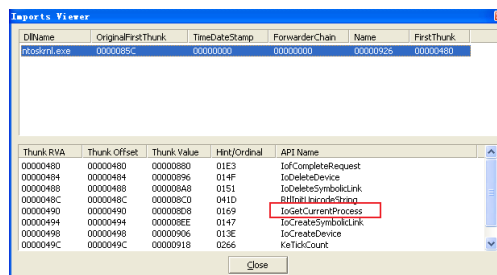


图 3.48: sys 导入表

图3.48显示仍然没进行过加壳，然后导入表有一些值得注意的：

- IoGetCurrentProcess：获取当前进程，目的可能是在当前进程的信息，驱动对其进行 xui
- IoCreate(Delete)SymbolicLink: 驱动创建或者删除符号表；
- IoCreate(Delete)Service: 驱动创建或者删除设备；

然后使用 PEView 查看其字符串：

```

.....
.....jg.O.....
r.....RSDS
.3u..b..M..q6.{ff
=...c:\winddk\76
00.16385.1\src\g
eneral\rootkitpr
ochide\wdm\sys\o
bjfre_wxp_x86\i3
86\siocli.pdb...
.....
    
```

图 3.49: sys 字符串

图3.49看到有意思的字符：rootkit 再次出现，提示我们可能这次面临的和之前的 Lab10-02 一样，都是 rootkit。

3.5.2 静态和动态结合分析

在进行分析前，先按照之前所述的将 Lab10-03.sys 移动到 System32 下，拍好虚拟机快照。

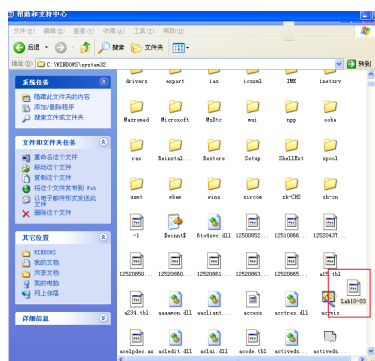


图 3.50: 将 Lab10-03.sys 移动到 System32 下

接下来将重点使用 IDA 进行深度的静态分析和 WinDBG 进行内核级别的动态分析调试。

1. IDA 加载可执行文件：

进入后直接来到 Winmain 函数的位置查看其反编译代码：



图 3.51: Winmain

图3.51看到 WinMain 函数中一些值得注意的地方：

- OpenSCManager：打开服务管理器
- CreateServiceA：创建服务，传入参数为 Displayname 为 Process Helper，二进制路径 binaryPath 为 C:\Windows\System32\Lab10-03.sys。即另一个 Lab10-03 的系统文件。
- 然后使用 StartService 试图打开服务；
- CreateFileA：打开 Filename 即\\.\ProcHelper 的句柄。
- DeviceIoControl：通过查阅资料，其中第三个参数代表 lpInBuffer 和第五个参数 lpOut-Buffer 都为 0。打开的设备即创建服务对应的驱动不会发送和接收任何信息到驱动中。除此之外还有控制码 dwIoControlCode 即 0xABCDEF01。
- Ole 操作：Ole 初始化后 CoCreateInstance 创建了 COMM 对象实例，之后创建了一个 VARIANT 变量传递参数，然后 SysAllocString 分配内存与字符串，其中 psz 代表了静态分析发现的经典的彩蛋网址。
- 进入一个无限循环 while(1)：并且每次进行一个函数指针的调用，实际上是会访问到刚才的彩蛋网站，然后 sleep30 秒后周期性的再次执行。由此实现广告骚扰。

综合所示，恶意代码创建了一个名为 Process Helper 的服务，其加载了刚刚被我们拖进 System32 下的 Lab10-03.sys。然后调用句柄 ProcHelper 句柄，打开了由其驱动的内核设备。最终目的就是反复地打开一个网址骚扰你。。。

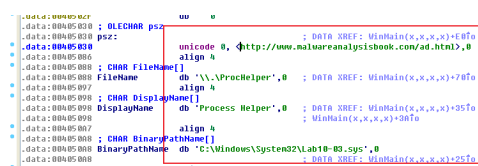


图 3.52: 一些字符串

2. IDA 加载驱动文件：

直接进入其驱动入口函数 DriverEntry:



图 3.53: DriverEntry

图3.53看到了其直接调用了 sub_10794 和 sub_10706。查看这两个函数:

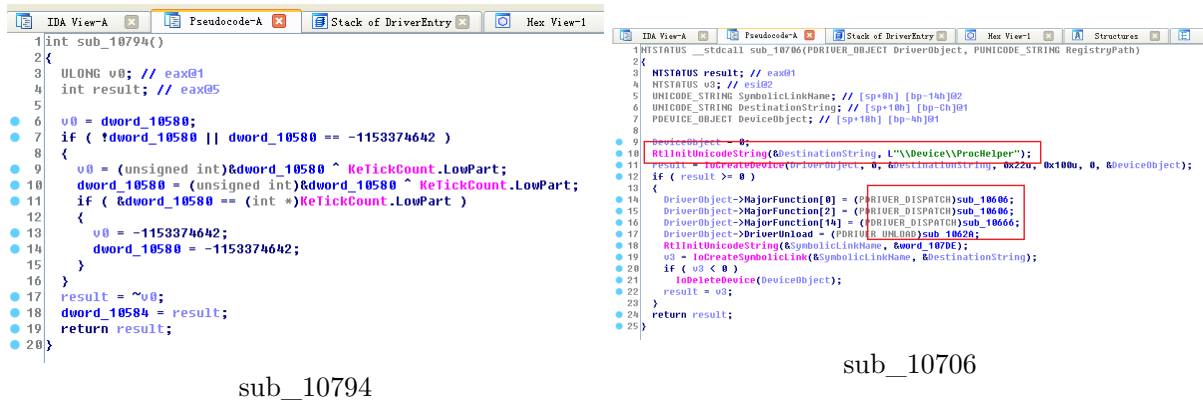


图 3.54: 调用函数

图3.54中可以发现 sub_10794 和之前分析 Lab10-02 的驱动程序一样, 做一些基础的初始化操作, 但是对于 sub_10706 也很多值得注意的地方:

- RtlInitUnicodeString 函数来初始化 DestinationString 的 UNICODE_STRING 类型的变量。其中代表设备名称\\Device\\ProcHelper。
- IoCreateDevice: 用于创建一个用于 IO 通信的设备对象, 便于驱动程序和外界交互使用;
- DriverObject 的功能设定:
调用的 MajorFuntion 的 0 即 IRP_MJ_CREATE 和 2 即 IRP_MJ_CLOSE 位置对应的都是 sub_10606。它们都是 IRP (I/O Request Packet) 主要功能码, 用于处理文件和设备的创建和关闭操作。在驱动程序开发中, 可以通过这两个位置来处理文件和设备的创建和关闭请求。即 sub_10606 负责创建和关闭请求。
调用的 MajorFuntion 的 14 即 IRP_MJ_DEVICE_CONTROL 用于设备控制的功能码, 交给 sub_10666 处理。
- DriverUnload: 即将驱动程序的卸载请求等工作交给 sub_1062A。
- IoCreateSymbolLink: 其中 word_107DE 代表的是 DosDevices\ProcHelper。即一个用户模式的 MS-DOS 可以访问的符号链接。
- 如果 IoCreateSymbolLink 返回值为 0 即失败, 直接 IoDeleteDevice 删除设备。

于是接下来查看三个函数:sub_10606, sub_10666 和 sub_1062A

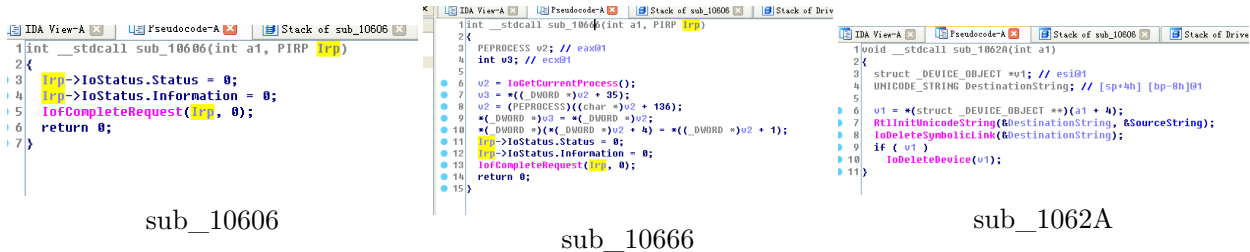


图 3.55: 调用函数

- sub_10606: IoCompleteRequest。即开始和结束 IRP 处理请求时候通知系统。
- sub_10666: 调用 IoGetCurrentProcess 获取当前进程的 EPROCESS 结构。然后，它修改了该结构中的某些字段。最后 IoCompleteRequest 来完成 IRP，并返回 0。由于这个函数对应的是 IPR_MJ_DEVICE_CONTROL，因此推测驱动的主要工作就是这些，获取进程的 EPROCESS 结构并进行处理。
- sub_1062A: IoDeleteSymbolicLink 来删除一个符号链接，IoDeleteDevice 来删除该设备对象。即用于清理在驱动程序卸载或停止时创建的资源。

到此为止，我们能分析的都差不多了，为了进行更深层次的分析，选择使用 WinDBG 进行内核的动态调试。

3. WinDBG 调试内核:

由于已经做好了断点，我们直接开始双击运行，确实看到了广告网址不停地被打开，但由于它翻不了墙，一直在失败（乐）。

然后回到主机 WinDBG 直接 break 开始操作：

首先既然我们已经知道了设备对象的名字就是 ProcHelper 在 \Device。只有找到 ProcHelper 驱动的设备对象，才能找到它其中执行的函数。因此直接通过!deobj 命令查找：

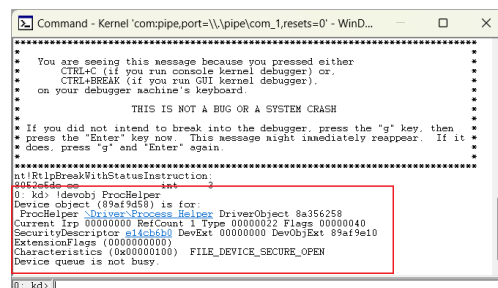


图 3.56: 查看 ProcHelper 设备对象

图3.56看到 Process Helper 中存储的设备对象的 DriveObject，地址为 8a356258。

DriveObject 中包含了我们之间使用 IDA 分析的各个 MAJOR Funtion 即各个函数的指针。而为了查看其中的主函数，我们必须到存储了 DriverObject 的 DRIVER_OBJECT 数据结构中，通过 dt 命令查看标注的驱动对象：

```

Device object (89af9d58) is for:
ProcHelper .Driver.Process.Helper DriverObject 8a356258
Current Irp 00000000 RefCount 1 Type 00000022 Flags 00000040
SecurityDescriptor 01c3b3b0 DevExt 00000000 DevObjExt 89af9e10
ExtensionFlags (00000000) FILE_DEVICE_SECURE_OPEN
Characteristics (0x00000100)
Device queue is not busy.
0: kd> dt nt!_DRIVER_OBJECT 8a356258
+0x000 Type           : User
+0x002 Size           : 0x168
+0x004 DeviceObject   : 0x89af9d58 _DEVICE_OBJECT
+0x008 Flags          : 0x12
+0x00c DriverStart    : 0xbaef4000 Void
+0x010 DriverSize     : 0xe00
+0x014 DriverSection  : 0x89b05490 Void
+0x018 DriverExtension : 0x8a356300 _DRIVER_EXTENSION
+0x01c DriverName     : UNICODE_STRING "\Driver.Process.Helper"
+0x024 HardwareDatabase : 0x8067e260 _UNICODE_STRING "\REGISTRY\MACHINE\HARDWARE\DESCRIPTION\SYSTEM"
+0x028 FastIoDispatch : (null)
+0x02c DriverInit     : 0xbaef47cd long +0
+0x030 DriverUnload    : (null)
+0x034 DriverUnload    : 0xbaef462a void +0
+0x038 MajorFunction   : [28] 0xbaef4606 long +0

```

图 3.57: 查看 sys 的驱动对象

图3.57中看到:

- DriverInit 和 DriverEntry: 在 IDA 中分析过;
- DriverUnload: 之前在 IDA 中也分析过, 在 sub_1062A 中, 删除符号链表和设备对象。
- MajorFuntion: 即那些主函数的函数指针, 也是我们要重点分析的, 位于地址 0xbaef4606 处。包含了最重要的一些驱动程序代码。

于是接下来到这个位置查看主函数代码, 通过 dd 命令和 0x1C 的附加选项, 即标出来想要查看的主函数代码个数:

```

+0x002 Size           : 0x168
+0x004 DeviceObject   : 0x89af9d58 _DEVICE_OBJECT
+0x008 Flags          : 0x12
+0x00c DriverStart    : 0xbaef4000 Void
+0x010 DriverSize     : 0xe00
+0x014 DriverSection  : 0x89b05490 Void
+0x018 DriverExtension : 0x8a356300 _DRIVER_EXTENSION
+0x01c DriverName     : UNICODE_STRING "\Driver.Process.Helper"
+0x024 HardwareDatabase : 0x8067e260 _UNICODE_STRING "\REGISTRY\MACHINE\HARDWARE\DESCRIPTION\SYSTEM"
+0x028 FastIoDispatch : (null)
+0x02c DriverInit     : 0xbaef47cd long +0
+0x030 DriverUnload    : (null)
+0x034 DriverUnload    : 0xbaef462a void +0
+0x038 MajorFunction   : [28] 0xbaef4606 long +0
0: kd> dd 8a356258+0x38 L1C
ReadVirtual. 8a356290 not properly sign extended
8a356290 baef4606 804f5552 baef4606 804f5552
8a3562a0 804f5552 804f5552 804f5552 804f5552
8a3562b0 804f5552 804f5552 804f5552 804f5552
8a3562c0 804f5552 804f5552 baef4666 804f5552
8a3562d0 804f5552 804f5552 804f5552 804f5552
8a3562e0 804f5552 804f5552 804f5552 804f5552
8a3562f0 804f5552 804f5552 804f5552 804f5552

```

图 3.58: Major Funtions

图3.58看到其中的每一项都对对应着一个驱动可以处理的不同类型请求。注意到大部分函数为 0x804f5552。重复的出现代表着驱动处理这个请求时候出现了问题, 因此我们好奇地使用 ln 命令查看这函数:

```

0: kd> dd 8a356258+0x38 L1C
ReadVirtual. 8a356290 not properly sign extended
8a356290 baef4606 804f5552 baef4606 804f5552
8a3562a0 804f5552 804f5552 804f5552 804f5552
8a3562b0 804f5552 804f5552 804f5552 804f5552
8a3562c0 804f5552 804f5552 baef4666 804f5552
8a3562d0 804f5552 804f5552 804f5552 804f5552
8a3562e0 804f5552 804f5552 804f5552 804f5552
8a3562f0 804f5552 804f5552 804f5552 804f5552
0: kd> ln ffffffff804f5552
Browse module
Set bp breakpoint
(804f5552) nt!IoInvalidDeviceRequest | (804f5588) nt!
IoGetDeviceAttachmentBase
Exact matches:
nt!IoInvalidDeviceRequest (_IoInvalidDeviceRequest#8)
0: kd>

```

图 3.59: 0x804f5552

图3.59看到该函数被命名为 IoInvalidDeviceRequest。这代表着这个函数专门用来处理一些不有效的非法的设备访问请求, 这就解释了为什么它会多次出现了。

除此之外, 在图3.58中出现的另一个函数是 0xbaef4606。而它实际上就是之前 IDA 分析的 sub_10666 函数。我们回忆一下它的功能: 正如之前分析的, 它调用 IoGetCurrentProcess 获取当前进程的 EPROCESS 结构, 然后它接着偏移 0x88 访问, 然后再偏移 0x8C 访问下一个 DWORD 大小的数据。

为了知道这个偏移量的内容究竟做了什么，我们可以直接通过 dt 命令查看 EPROCESS 结构体这个偏移对应的内容：

```
0: kd> dt nt!_EPROCESS
+0x000 Pcb                : _KPROCESS
+0x06c ProcessLock        : _EX_PUSH_LOCK
+0x070 CreateTime         : _LARGE_INTEGER
+0x078 ExitTime           : _LARGE_INTEGER
+0x080 RundownProtect     : _EX_RUNDOWN_REF
+0x084 UniqueProcessId    : Ptr32_Void
+0x088 ActiveProcessLinks : LIST_ENTRY
+0x090 QuotaUsage         : [3]_UInt4B
+0x09c QuotaPeak          : [3]_UInt4B
+0x0a8 CommitCharge       : _UInt4B
+0x0ac PeakVirtualSize   : _UInt4B
+0x0b0 VirtualSize        : _UInt4B
```

图 3.60: EPROCESS 偏移

图3.60看到偏移 0x88 和 0x8C 的位置都是 **LSIT_ENTRY** 字段。

通过查阅资料我们知道这个 **LSIT_ENTRY** 结构实际上就是一个带有 **BLINK** 和 **FLINK** 的双向链表。我们想到之前 **sub_10666** 中在获取进程信息后正好有一系列细微的地址操作。回到 IDA 中查看驱动程序的对应位置：

```
PAGE:00010666 ; int __stdcall sub_10666(int, PIRP Irp)
PAGE:00010666 sub_10666 proc near ; DATA XREF: sub_10706+5
PAGE:00010666 Irp = dword ptr 0Ch
PAGE:00010666
PAGE:00010666 mov edi, edi
PAGE:00010666 push ebp
PAGE:00010666 mov ebp, esp
PAGE:00010666 call ds:IoGetCurrentProcess
PAGE:00010671 mov ecx, [eax+0Ch]
PAGE:00010677 add eax, 88h
PAGE:0001067C mov edx, [eax]
PAGE:0001067E mov [ecx], edx
PAGE:00010680 mov ecx, [eax]
PAGE:00010682 mov [ecx+4], eax
PAGE:00010685 mov [ecx+4], eax
PAGE:00010688 mov ecx, [ebp+Irp] ; Irp
PAGE:00010688 and dword ptr [ecx+18h], 0
PAGE:0001068F and dword ptr [ecx+1Ch], 0
PAGE:00010693 xor dl, dl ; PriorityBoost
PAGE:00010695 call ds:IoCompleteRequest
PAGE:00010698 xor eax, eax
PAGE:00010699 pop ebp
PAGE:0001069E ret 8
PAGE:0001069E sub_10666 endp
```

图 3.61: 双向链表操作

图3.61看到了 **sub_10666** 中的一些关键操作，总的来说实现的是通过不断地获取和调整 **BLINK** 和 **FLINK**，将前一项的 **FLINK** 指针覆盖来跳过当前项。这样的话实现了这个进程会在操作系统遍历进程链表时被直接忽略无法发现，这样隐蔽操作的内核行为也和我们静态分析的字符串提示相吻合，这个病毒和驱动程序又是可恶的 **RootKit**！

最后附上一张运行了很久后的结果：

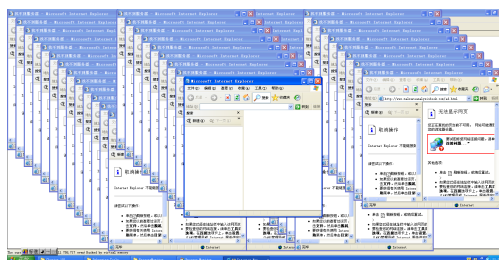


图 3.62: 结果

真的很恐怖,,，远离病毒，人人有责。

3.5.3 实验问题

现在完成了全部分析后来统一回答实验中的问题。

1. Q1: 这个程序做了些什么?

回答: 这个程序也是个 Rootkit。根据我们分析的, 在用户态运行可执行程序后, 它会将 System32 下的驱动程序在用户态加载进内核, 当成一个服务。然后每隔 30 秒就弹出一个彩蛋网址的广告。并且这个驱动会获取当前进程信息, 然后通过从系统链表中摘除进程环境块 PEB 的方式来隐藏进程。

2. Q2: 一旦程序运行, 你怎样停止它?

回答: 根据3.62的可怕结果可以发现, 一旦程序运行, 除了重启以外, 没有任何一种办法可以轻易停止它。

3. Q3: 它的内核组件做了什么操作?

回答: 我们发现了驱动程序会在内核中负责响应来隐藏进程。具体而言, 它通过操作双向链表的指针, 摘除了 PEB 的 DeviceIoControl 请求。

3.6 Yara 检测

3.6.1 Sample 提取

利用课程中老师提供的 Scan.py 程序, 将电脑中所有的 PE 格式文件全部扫描, 提取后打开 sample 文件夹查看相关信息:

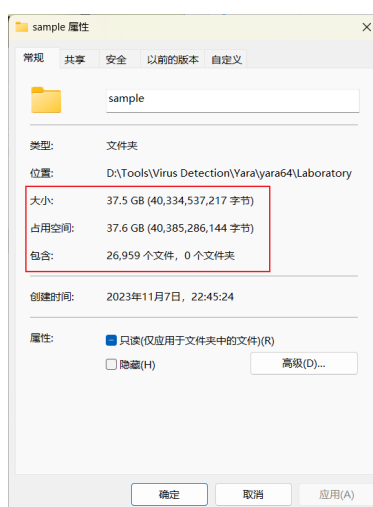


图 3.63: Sample 信息

图3.63可以看到从电脑中提取了所有 PE 格式文件后的文件及 sample 大小为 37.5GB。然后由于本次实验的 Lab10-01.sys 和 Lab10-03.sys 不属于 PE 文件, 因此手动将其放入 sample 中, sample 包含一共 26959 个文件。Yara 编写的规则将目标从 sample 中识别成功检测出本次 Lab10 的全部五个恶意代码包括 exe 和 sys。

3.6.2 Yara 规则编写

本次 Yara 规则的编写基于上述的病毒分析和实验问题, 主要是基于静态分析的 Strings 字符串和 IDA 分析结果。为了能够更好地进行 Yara 规则的编写, 首先对之前分析内容进行回归。分别对 Lab010-01, 02, 03.exe 和 Lab10-01.sys 和 Lab10-03.sys 可以利用的病毒特征进行分条总结如下:

1. Lab10-01.exe:

- **Lab10-01:** 创建服务的名字。
- **C:\Windows\System32\Lab10-01.sys:** 代表着寻找 System32 下的 sys, 这个将会是绝杀。

2. Lab10-01.sys:

- **\Registry\Machine\SOFTWARE\Policies\Microsoft\WindowsFirewall:** 其下还有域名 DomainProfile 和 StandardProfile。是驱动病毒试图寻找的有关防火墙的注册表项。
- **EnableFirewall:** 是将注册表设置为禁用, 在内核态难以察觉。

3. Lab10-02.exe:

- **Failed to start service:** 没有成功开启服务的错误提示, 同样的还有 create 和 open service manager;
- **486 WS Driver:** 也就是加载的驱动名字;
- **C:\Windows\System32\Mlwx486.sys:** 这个很有可能就是绝杀, 就是其在 System32 下创建的隐藏驱动文件的名字。

4. Lab10-03.exe:

- **<https://www.malwareanalysisbook.com/ad.html>:** 经典的广告彩蛋网址。
- **\Process Helper:** 帮助用户管理和优化工作流程。和进程操作有关。
- **C:\Windows\System32\Lab10-03.sys.** 即另一个 sys 的系统文件在 system32 下。

5. Lab10-03.sys:

- **IoGetCurrentProcess:** 即获取当前进程信息的导入函数
- **c:\winddk\7600.16385.1\src\general\rootkitprochide\wdm\sysobjfre_wxp_x86\i386\siocctl.pdb:** 即 rootkit 提示符。

因此加上必要的一些修饰符如 wide、ascii 以及 nocase 后, 编写如下 Yara 规则:

```
1 rule Lab10_01_exe
2 {
3 meta:
4     description = "Lab10_01_exe:Yara Rules"
5     date = "2023/11/8"
6     author = "ErwinZhou"
7 strings:
8     $clue1 = "Lab10-01" wide ascii
9     $clue2 = "C:\\Windows\\System32\\Lab10-01.sys" wide ascii
10 condition:
11     all of them //Lab10-01.exe
12 }
```



```
13 rule Lab10_01_sys
14 {
15 meta:
16     description = "Lab10_01_sys:Yara Rules"
17     date = "2023/11/8"
18     author = "ErwinZhou"
19 strings:
20     $clue1 = "\\Registry\\Machine\\SOFTWARE\\Policies\\Microsoft\\WindowsFirewall"
21             wide ascii
22     $clue2 = "EnableFirewall" wide ascii nocase
23 condition:
24     all of them //Lab10-01.sys
25 rule Lab10_02
26 {
27 meta:
28     description = "Lab10_02:Yara Rules"
29     date = "2023/11/8"
30     author = "ErwinZhou"
31 strings:
32     $clue1 = "Failed to start service" wide ascii
33     $clue2 = "486 WS Driver" wide ascii
34     $clue3 = "C:\\Windows\\System32\\Mlwx486.sys" wide ascii
35 condition:
36     all of them //Lab10-02.exe
37 }
38 rule Lab10_03_exe
39 {
40 meta:
41     description = "Lab10_03_exe:Yara Rules"
42     date = "2023/11/8"
43     author = "ErwinZhou"
44 strings:
45     $clue1 = "http://www.malwareanalysisbook.com/ad.html" wide ascii
46     $clue2 = "Process Helper" wide ascii nocase
47     $clue3 = "C:\\Windows\\System32\\Lab10-03.sys" wide ascii
48 condition:
49     all of them //Lab10-03.exe
50 }
51 rule Lab10_03_sys
52 {
53 meta:
```

```
54     description = "Lab10_03_sys:Yara Rules"
55     date = "2023/11/8"
56     author = "ErwinZhou"
57 strings:
58     $clue1 = "IoGetCurrentProcess" wide ascii
59     $clue2 = "c:\\winddk\\...\\siocctl.pdb" wide ascii
60 condition:
61     all of them //Lab10-03.sys
62 }
```

上面 yara 的 Lab10-03.sys 处没有展示全部的字符串，但实际上测试是用的全长的串的。然后使用如下 Python 代码进行对 sample 的扫描：

```
1 import os
2 import yara
3 import time
4 # 加载YARA规则
5 rules = yara.compile('D:\Tools\Virus Detection\Yara\yara64\Lab10.yar')
6 # 初始化计数器
7 total_files_scanned = 0
8 total_files_matched = 0
9 def scan_folder(folder_path):
10     global total_files_scanned
11     global total_files_matched
12     # 检查文件夹是否存在
13     if os.path.exists(folder_path) and os.path.isdir(folder_path):
14         # 遍历文件夹内的文件和子文件夹
15         for root, dirs, files in os.walk(folder_path):
16             for filename in files:
17                 total_files_scanned += 1
18                 file_path = os.path.join(root, filename)
19                 with open(file_path, 'rb') as file:
20                     data = file.read()
21                     # 扫描数据
22                     matches = rules.match(data=data)
23                     # 处理匹配结果
24                     if matches:
25                         total_files_matched += 1
26                         print(f"File '{filename}' in path '{root}' matched YARA rule(s):")
27                         for match in matches:
28                             print(f"Rule: {match.rule}")
29     else:
```

```
30     print(f'The folder at {folder_path} does not exist or is not a folder.')
31 # 文件夹路径
32 folder_path = 'D:\Tools\Virus Detection\Yara\yara64\Laboratory\sample'
33 # 记录开始时间
34 start_time = time.time()
35 # 递归地扫描文件夹
36 scan_folder(folder_path)
37 # 记录结束时间
38 end_time = time.time()
39 # 计算运行时间
40 runtime = end_time - start_time
41 print(f"Program runtime: {runtime} seconds.")
42 print(f"Total files scanned: {total_files_scanned}")
43 print(f"Total files matched: {total_files_matched}")
```

3.6.3 Yara 规则执行效率测试

扫描结果如下图所示：

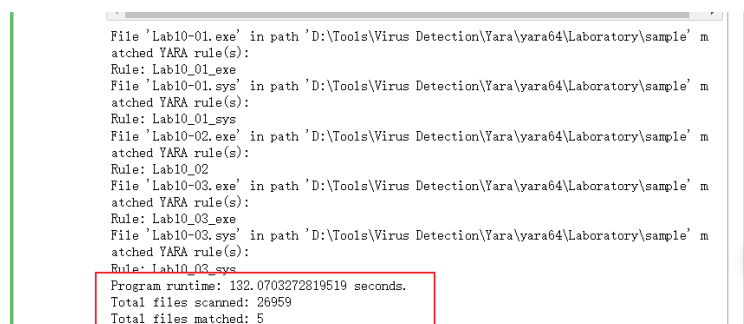


图 3.64: Yara 检测结果

图3.64可以看到能够成功地从 26959 个文件中唯一地识别检测到五个病毒文件包括两个 Lab10-01 和 Lab10-03 的 sys。

并且仅用时 132.07 秒，时间性能较快。总的来说，Yara 规则编写和检测较为成功。

3.7 IDAPython 辅助样本分析

根据要求，分别选择病毒分析过程中一些系统的过程编写如下的 Python 脚本：

3.7.1 查看所有函数调用

这段代码的主要目的是遍历 IDA Pro 中的所有函数，并输出每个函数的起始地址（以十六进制表示）和函数的名称。这对于太长的程序定位函数调用很有用。

```
1 for func in idutils.Functions(): print hex(func), idc.get_func_name(func)
```

3.7.2 查看函数的开始和结束地址

获取当前光标位置所在的函数范围，并将该函数的起始地址和结束地址打印到输出窗口。在本次实验中有利于例如 Lab10-01 中方便在 WinDBG 和 IDA 中进行相同偏移量的地址转换（载入地址不同）。

```
1 # 获取当前光标位置的地址
2 ea = here()
3 # 获取地址所在函数的范围
4 func = idaapi.get_func(ea)
5 # 打印函数的起始地址和结束地址
6 print "Start: 0x%x, End 0x%x" % (func.startEA, func.endEA)
```

3.7.3 函数反汇编

遍历并打印函数内的所有指令的地址和反汇编代码。这对于分析和理解函数内部的指令和操作非常有用在本次实验中有助于我们快速定位所有函数，比如 Lab10-02 中的 sub_10486，然后精准分析。

```
1 # 获取当前光标位置的地址
2 ea = here()
3 # 使用 IDA Pro 函数属性获取函数的起始地址和结束地址
4 start = idc.get_func_attr(ea, FUNCATTR_START)
5 end = idc.get_func_attr(ea, FUNCATTR_END)
6 # 初始化当前地址为函数的起始地址
7 cur_addr = start
8 # 循环遍历函数的指令
9 while cur_addr <= end:
10     # 打印当前地址的十六进制表示和反汇编代码
11     print hex(cur_addr), idc.generate_disasm_line(cur_addr, 0)
12     # 获取下一个指令的地址
13     cur_addr = idc.next_head(cur_addr, end)
```

3.7.4 库函数查看

这段代码的主要目的是遍历 IDA Pro 中的所有函数，检查它们是否被标识为库函数，并将这些库函数的地址、类型和名称打印出来。对于本次实验中可以快速地将 Lab10-01 的注册表操作的各种导入函数识别出来。

```
1 # 遍历所有函数列表
2 for func in idutils.Functions():
3     # 获取函数的属性标志
4     flags = idc.get_func_attr(func, FUNCATTR_FLAGS)
5     # 检查函数是否是库函数
```

```
6     if flags & FUNC_LIB:
7         # 如果是库函数, 打印函数的地址、函数类型 (FUNC_LIB), 以及函数名称
8         print hex(func), "FUNC_LIB", idc.get_func_name(func)
```

3.7.5 获取函数的全面信息

遍历已知函数的指令, 检查每个指令是否是 call 或 jmp 指令, 并且该指令的操作数类型是寄存器。对于满足条件的指令, 它将打印出指令的地址和反汇编行。我们重点关注函数是否是 FUNC_LIB 或者 FUNC_FLAGS, 然后获取其全面的指令信息包含函数名等。这有助于我们详细通过函数了解病毒目的。

```
1 # 遍历已知函数列表
2 for func in idutils.Functions():
3     # 获取函数的属性标志
4     flags = idc.get_func_attr(func, FUNCATTR_FLAGS)
5     # 检查函数是否是库函数或者是函数跳转 (thunk)
6     if flags & FUNC_LIB or flags & FUNC_THUNK:
7         continue
8     # 获取函数内的指令地址列表
9     dism_addr = list(idutils.FuncItems(func))
10    # 遍历函数内的指令地址
11    for line in dism_addr:
12        # 获取指令的助记符 (mnemonic)
13        m = idc.print_insn_mnem(line)
14        # 检查指令是否是 'call' 或 'jmp'
15        if m == 'call' or m == 'jmp':
16            # 获取指令操作数的类型
17            op = idc.get_operand_type(line, 0)
18            # 如果操作数类型是寄存器 (register), 则打印指令和地址
19            if op == o_reg:
20                print "0x%x %s" % (line, idc.generate_disasm_line(line, 0))
```

使用上述 Python 脚本在 IDAPro 中便可以辅助进行分析。经过测试全部与使用 IDA 动态分析的结果相同, 编写较为成功。

4 实验结论及心得体会

4.1 实验结论

本次实验, 通过结合使用静态分析工具和动态分析方法, 对 Lab10 的三个恶意代码进行了全面的分析, 并依次回答了书中的问题。并在其中重点使用了 WinDBG 进行内核级别的调试并且应用到了很多上课学到的 Windows 内核级别的知识。然后结合之前的分析和 IDA 的 String 模块编写了 Yara 规则, 对病毒样本进行了成功检测并且时间性能较强。

最后为本次实验中编写了对应可以辅助分析的 IDAPython 脚本，并且成功实现了辅助功能。
总的来说，实验非常成功。

4.2 心得体会

本次实验，我收获颇丰，这不仅是课堂中的知识，更是我解决许多问题的能力，具体来说：

1. 首先我进一步熟练掌握到了将课堂中学习的病毒分析工具 IDAPro，用于更全面的动态分析；
2. 其次我还在这个过程中发现了许多 Windows 内核级别的事情，分清楚了内核态和用户态的区别，**有些病毒在内核态的行为也可以说是十分狡猾，难以发现，可恶的病毒 (" > 目 <)**；
3. 在这个过程中还涉及了配置 VMware 端口和其它配置等内容，最后第一次使用了 WinDBG 进行内核级别的调试，掌握了其使用的一些基本技巧，能够熟练使用其进行简单的病毒分析了；
4. 除此之外我还精进了我编写更为高效的 Yara 规则的能力，更好地帮助我识别和检测病毒；
5. 我还精进了使用 IDAPython 脚本对病毒进行辅助分析。

总的来说，本次通过亲自实验让我感受到了很多包括 IDA 和 IDAPython，也加深了我对病毒分析综合使用静态和动态分析的能力，最重要的是学习到了很多 Windows 内核的知识和使用 WinDBG 进行内核态调试的方法。培养了我对病毒分析安全领域的兴趣。我会努力学习更多的知识，辅助我进行更好的病毒分析。

感谢助教学姐审阅:)