

Windows安全原理——自启动项的查看和分析 研究与开发报告

1.开发环境

OS: Microsoft Windows 10 (10.0.18362.836)

IDE: Microsoft Visual Studio 2017(.NET Framework 4.6.1)

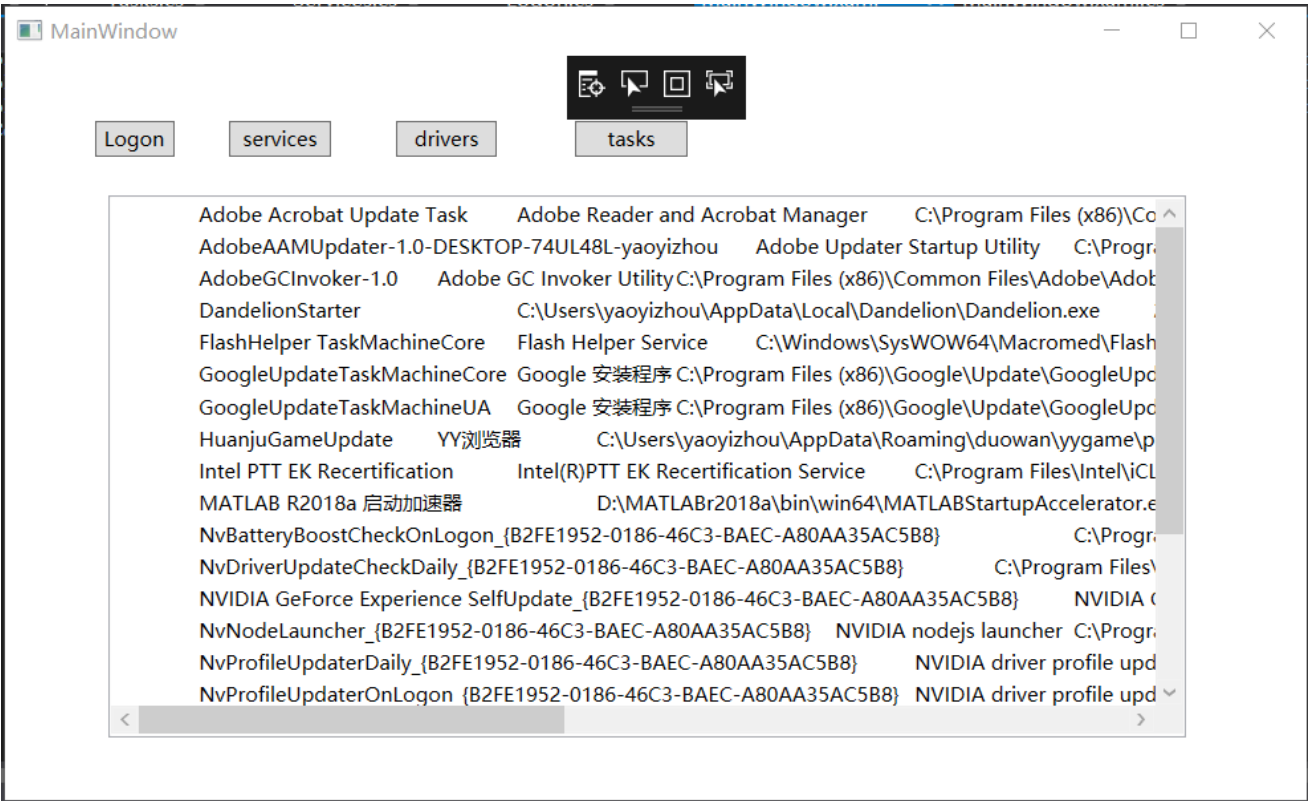
2.界面设计

由于编程语言限定在 C/C++/C#，并且需要开发友好的图形化操作界面，然而我先前并没有这几门语言开发用户界面的经验，在开发初期技术路线的选择上陷入了迷茫。在和同学之间的交流中，了解到由 Microsoft 公司设计的 C# 语言基于 .NET Framework，对于 Windows 系统编程具有良好的支持；此外，由微软推出的同样基于 .NET Framework 的 WPF 框架，能够完美契合我用户界面的设计需求，且用户友好程度明显要由于 QT。微软的 Visual Studio 编译器对 WPF 框架有着良好的支持，用户可以通过拖动控件的方式来开发图形化界面。最终我决定采用VS 创建WPF项目来完成本次大作业的开发部分。

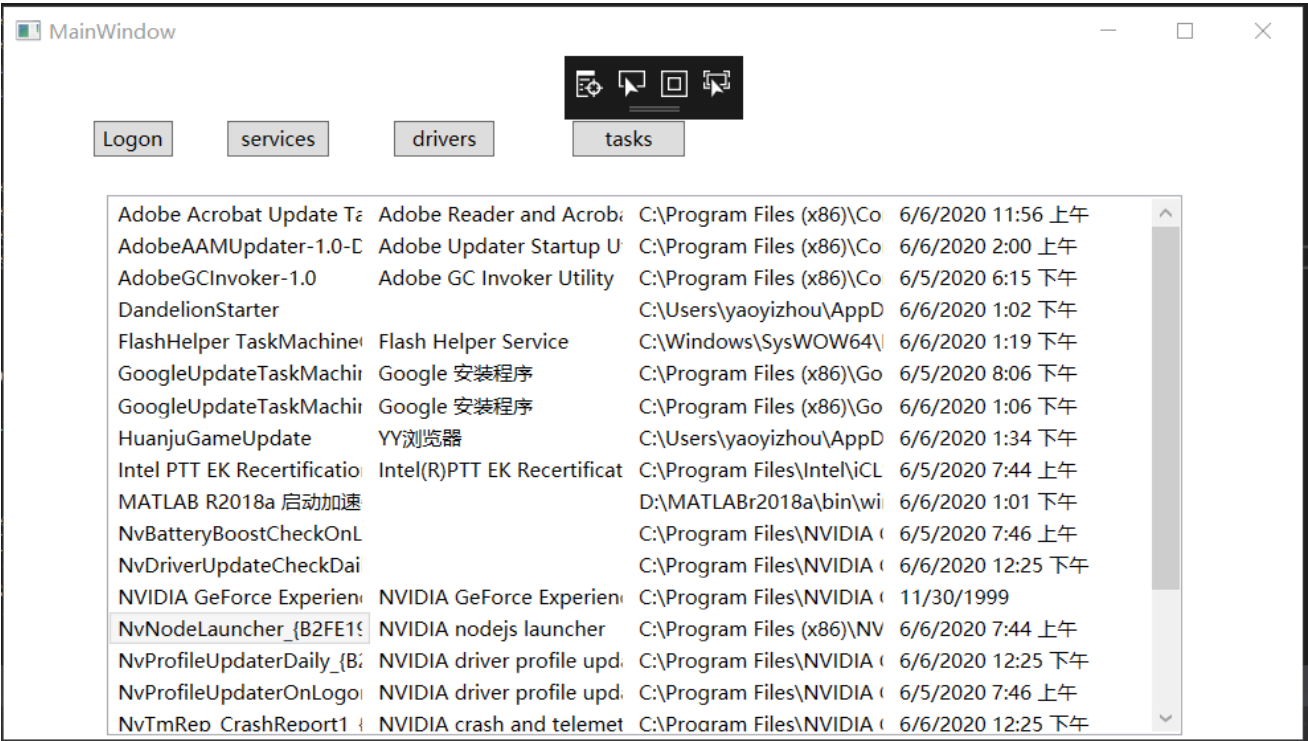
C# 的语法和 C++ 基本一致，我很快就熟悉了 C# 的编程，并且花了一定的时间来熟悉它的 System 命名空间。在查看自启动项的过程中，我主要使用到了 System 下的通用类型 Object、string，System.Collections 下的集合类型 ArrayList，以及打开注册表所必须的 Microsoft.Win32 下的键类型 RegistryKey 等。

界面的设计我采用 button + listbox 控件的方式，并参考 Autoruns 显示5类信息，分别是：自启动项项目名称、项目描述、发布者、程序路径以及上一次修改时间。

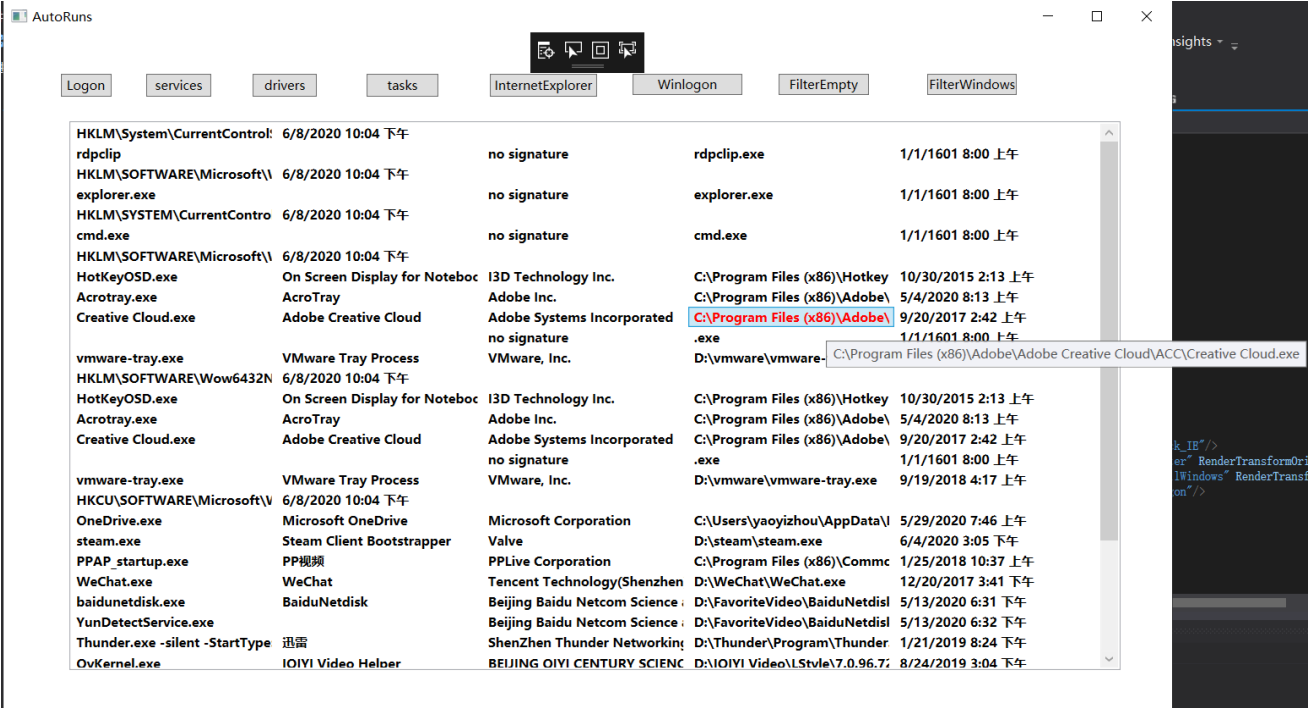
我遇到的第一个问题是如何分列显示每一条记录的内容，下图是我刚完成 Logon 的查询时的界面效果，只是简单地将信息拼接为一个字符串显示，可以看到由于每个字符串长度不同，给人的感觉非常混乱。



WPF 实现了将页面开发人员和程序员的分离，在简单地了解了 XAML 格式之后，我成功实现了分列显示，效果图如下，给人的感觉就清爽了很多。但是又出现了新的问题，字符串的长度超过了既定显示的范围，就看不到完整的查询内容了。



通过查阅资料，我又了解到可以通过添加 tooltip 控件，在鼠标位于某一项内容上时，提示这一项的完整内容。但是，我又面临了后台如何判断鼠标位于哪一项的问题，我花了很多时间，试了很多网上的方法，效果都不好，最后通过在后台生成 tooltip 控件，并通过查阅 MSDN 上的文档设置其 IsEnabled 和 Content 成员达到了我的目的。此外，我又添加了点击某一项，使其变成红色的逻辑。最终的效果如下：



我还试图尝试类似 Autoruns 的方式将键名那一行的背景设置为别的颜色，以便和查询结果进行区分，但是囿于对 XAML 并不熟悉，在尝试一些方法无果后，最终我放弃了这个尝试。

3.自启动项查询

3.1 Logon

将 Autoruns 的 Hide Empty Locations 和 Hide Windows Entries 选项关闭后，我看到了 Autoruns 所查询的所有注册表键名以及目录。Logon 下显示了3个目录，分别是 C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Startup、%USERPROFILE%\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup 和 C:\Users\yaoyizhou\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup，前两者和课上所说的一致，而后者则是 %USERPROFILE% 重定向后的结果。

通过 Autoruns 的查询我发现我所使用的主机上的自启动项集中在

HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run、
HKLM\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Run 和
HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run 这几个键下，和课上所介绍的一致。

注册表键的打开和查询可以通过 RegistryKey 类下的 OpenSubKey、GetSubKeyNames、GetValue 等成员函数实现。

典型的 Logon 下的键格式如下

\\HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run

	名称	类型	数据
> miniconfig	(默认)	REG_SZ	(数值未设置)
> Mobility	BaiduYunDetect	REG_SZ	"D:\FavoriteVideo\BaiduNetdisk\YunDetectSe...
> Notifications	BaiduYunGuanjia	REG_SZ	"D:\FavoriteVideo\BaiduNetdisk\baidunetdisk....
> PenWorkspace	EpicGamesLau...	REG_SZ	"D:\Epic Games\Launcher\Portal\Binaries\Win...
> Policies	HCDNClient	REG_SZ	"D:\IQIYI Video\LStyle\7.0.96.7232\QyKernel.e...
> PrecisionTouchPad	OneDrive	REG_SZ	"C:\Users\yaoyizhou\AppData\Local\Microsof...
> Prelaunch	PPLiveAP	REG_SZ	"C:\Program Files (x86)\Common Files\PPLive...
> Privacy	QyClient	REG_SZ	"D:\IQIYI Video\LStyle\7.0.96.7232\QyClient.e...
> PushNotifications	Steam	REG_SZ	"D:\steam\steam.exe" -silent
> RADAR	TGP	REG_SZ	D:\Program Files (x86)\WeGame\tdgp_daemon....
> Run	Thunder	REG_SZ	D:\Thunder\Program\Thunder.exe -silent -Star...
> RunOnce	Wechat	REG_SZ	D:\WeChat\WeChat.exe
> Screensavers			
> Search			

可以看到键的value存放的便是自启动项的名称，而value的数据部分则是路径。还剩下发布者、项目描述以及最后一次修改的时间无法得到。查阅资料可以知道，System.Diagnostics 下的 FileVersionInfo 类提供了根据可执行文件的路径获取描述信息的方法，而value的数据部分并不完全都是路径，有些还带有引号，甚至是参数，所以，读取到value的数据部分后还需要将其解析成标准的路径格式。System.IO 下的 FileInfo 类提供了根据可执行文件的路径获取上一次修改时间的方法。System.Security.Cryptography.X509Certificates 类提供了根据可执行文件路径获取证书内容的方法，而证书的格式通常以 CN=company_name 开头，只需要提取证书中CN字段后的内容就可以了。至此，已经能够实现 Logon 注册表内容的读取。

下面是我在实现过程遇到的问题与困难记录。

(1)无法打开HKLM下的键

一开始，考虑到注册表的重定向问题，设置所有的键打开方式都是可写，这时即使程序在运行时拥有管理员权限，也无法打开 HKLM 下的键。因此只能将键的打开方式设置为只读。不过，经过测试比对发现，我的查询结果和 Autoruns 的查询结果一致，所以我之后都是以只读方式打开键了。

(2)添加过滤功能

没有过滤功能的话，直接查询各个键得到的结果非常多，并且含有大量的空键。为了用户友好性，必须要提供过滤功能。我采用的方法是在存放查询结果的类中指定一个标志 IsEntry 来区分键和查询结果，再指定一个标志 IsEmpty 来指示键查询结果是否为空；在应用程序后台，设置一个 FilterFlag 变量指示是否过滤空键，一个 FilWinFlag 变量指示是否过滤掉发布者为 Microsoft window 的查询记录，这两个变量由用户决定，默认是都生

效。

(3)验证签名

还没有解决

3.2 Services

在完成 Logon 查询的基础上，进行 Services 的查询就容易多了，只需要查询 HKLM\System\CurrentControlSet\Services 这一键下的内容就可以了。在通过注册表查看该键下的内容时，我发现了一些不同的情况。

有一些键没有子键，这些键的信息可以通过两个个value，ImagePath DisplayName 中分别找到路径和描述

有一些键有子键，并含有 Parameter 子键，而 Parameter 子键下无有用信息，这些键的 ImagePath 的内容通常是 .sys 驱动程序。

有一些键有子键，并含有 Parameter 子键，而 Parameter 子键下 ServiceDll 的内容是路径，而该键 ImagePath 的内容通常是 svchost.exe 接参数的形式，这时就不能使用 ImagePath 的内容了。下面是一个典型的例子

计算机\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\AJRouter			
ahcache	名称	类型	数据
AJRouter	(默认)	REG_SZ	(数值未设置)
Parameters	Description	REG_SZ	@%SystemRoot%\system32\AJRouter.dll,-1
Security	DisplayName	REG_SZ	@%SystemRoot%\system32\AJRouter.dll,-2
TriggerInfo	ErrorControl	REG_DWORD	0x00000001 (1)
ALG	FailureActions	REG_BINARY	80 51 01 00 00 00 00 00 00 00 03 00 00 ...
amdgpio2	ImagePath	REG_EXPAND_SZ	%SystemRoot%\system32\svchost.exe -k Loc...
amdi2c	ObjectName	REG_SZ	NT AUTHORITY\LocalService
AmdK8	ServiceSidType	REG_DWORD	0x00000001 (1)
AmdPPM	Start	REG_DWORD	0x00000003 (3)
amdsata	Type	REG_DWORD	0x00000020 (32)
amdsbs			

计算机\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\AJRouter\Parameters			
ahcache	名称	类型	数据
AJRouter	(默认)	REG_SZ	(数值未设置)
Parameters	ServiceDll	REG_EXPAND_SZ	%SystemRoot%\System32\AJRouter.dll
Security	ServiceDllUnlo...	REG_DWORD	0x00000001 (1)
TriggerInfo			

另外还发现了一些键，在这些键下看上去没有任何有用信息，因此不再考虑。

根据以上的情况分类，并过滤掉路径中以 .sys 结尾的(视为系统驱动程序)，最终得到了和采用 Autoruns 查询基本一致的结果。

下面是我在实现过程遇到的问题与困难记录。

(1)解析环境变量

DisplayName 中的内容是 dll 路径，而不是描述，并且还带有环境 SystemRoot。这个困扰了我很久，直到后来参考同学的方法才得以解决，具体来说是将 advapi.dll 中的函数封装到 C# 内，而这个函数能自动解析环境变量，并找到指定的文件，从而提取中该服务 dll 的描述。

(2) 过滤问题

查到的服务信息同样需要过滤，并且我发现使用 Logon 中的方法并不能得到绝大部分服务的发布者信息，而当我将这些不能得到发布者信息的服务全部过滤掉时，我发现查询结果和 Autoruns 中的查询结果一致，这说明这些查不到发布者信息的服务其实是由 Microsoft Windows 所签名的。和前面的问题一样，由于我未能实现验证签名，并且已经能得到不错的效果，所以我就没有再深入了。

3.3 Drivers

系统驱动程序和系统服务的查询在同一个键下，因此可以采用相同的查询方式，并通过路径后缀名是否为 .sys 来判断。但是当我完全套用 Services 的代码时，我发现得到的查询结果远少于 Autoruns 的查询结果，仔细分析注册表中驱动服务的 ImagePath 内容，我发现大致可以分为如下几类：

\\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\ADP80XX

ADP80XX

Parameters

StartOverride

adsis

Cache

Options

名称	类型	数据
(默认)	REG_SZ	(数值未设置)
ErrorControl	REG_DWORD	0x00000001 (1)
Group	REG_SZ	SCSI Miniport
ImagePath	REG_EXPAND_SZ	System32\drivers\ADP80XX.SYS

\\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\acpipagr

Wdf

acpipagr

Enum

Parameters

Wdf

ArniPmi

名称	类型	数据
(默认)	REG_SZ	(数值未设置)
DisplayName	REG_SZ	@acpipagr.inf,%SvcDesc%;ACPI Processor Ag...
ErrorControl	REG_DWORD	0x00000001 (1)
ImagePath	REG_EXPAND_SZ	\SystemRoot\System32\drivers\acpipagr.sys

还有一种是 ImagePath 就是标准路径，这种不需要解析。由于系统驱动全部都位于 C:\windows\System32 下，所以解析路径非常方便。

下面是我在实现过程遇到的问题与困难记录。

(1)直接查services，根据sys结尾判断

比autoruns查到的少很多，驱动程序能查到描述的也比较少，并且用 Service 和 Logon 中查描述和查发布者的方法都失败了。几乎所有的记录都没有正确查到发布者。

3.4 Scheduled Tasks

计划任务以 XML 文件格式保存在 C:\Windows\System32\Tasks 文件夹中，而打开这个文件夹需要提供管理员权限。 Autoruns 查询计划任务似乎不是通过查询注册表的。网上找了一些资料，发现可以在VS中通过添加 .COM 程序集中的 TaskScheduler 引用，来实现计划任务的查询。具体来说，通过调用 TaskScheduler 命名空间下的 TaskSchedulerClass 类及其成员函数就可以获得所有的计划任务。在读取路径的过程中发现不能简单地通过 IRegisteredTaskCollection 类的 Path 成员变量来获得路径，而是需要解析其 Xml 成员变量。最终我查询到的结果和 Autoruns 一致。

3.5 Internet Explorer

在做完4个基本功能的查询之后，我打算再看看别的自启动方式的查询。基于 Internet Explorer 的自启动主要是通过操作 Browser Helper Objects(BHO) 来实现的。通过 Autoruns 提供的注册表键名，我首先查看了该键下的内容，如下图所示

\\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects\{004B0726-A010-4ABF-8556-FCDB7F1FCA1E}

Browser Helper Objects

{004B0726-A010-4ABF-8556-FCDB7F1FCA1E}

{31D09BA0-12F5-4CCE-BE8A-2923E76605DA}

{AE7CD045-E861-484f-8273-0445EE161910}

{D0498E0A-45B7-42AE-A9AA-ABA463DBD3BF}

{F4971EE7-DAA0-4053-9964-665D8EE6A077}

名称	类型	数据
(默认)	REG_SZ	XunleiBHO

和之前的查询不同的是，这些键下乍一眼看上去并没有任何有价值的信息。

通过查阅网上的资料得知这些子键名是16字节的CLSID字符串，当IE加载BHO时，会读取16字节的CLSID，而这个CLSID所包含的信息其实在另一个注册表键 HKLM\SOFTWARE\Classes\CLSID\ 下，

\\HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{0000002F-0000-0000-C000-000000000046}\InprocServer32			
CLSID	名称	类型	数据
{0000002F-0000-0000-C000-000000000046}	(默认)	REG_SZ	C:\Windows\System32\oleaut32.dll
InprocServer32	ThreadingModel	REG_SZ	Both
{00000300-0000-0000-C000-000000000046}			

发现 HKLM\SOFTWARE\Classes\CLSID***\InprocServer32\ 下的默认值的内容便是IE会加载的 dll 路径。至此，查看基于 Internet Explorer 的自启动的思路便明晰了，并且可以通过 Logon 中用到的方法由路径得到描述、发布者、最近一次修改时间，而自启动项名可以采用BHO下的默认值中的内容。最终得到的查询结果和 Autoruns 一致。

另外由于重定向问题，每个 dll 出现重复的。由于 Autoruns 也是这样的，所以没有改。

值得一提的是，在查阅相关资料的时候得知，一般的安全软件会读取整个CLSID进行 dll 匹配，而IE在工作时只会读取16字节的CLSID(如果长于16字节，则读前16字节)，这就使恶意代码有了可乘之机，所以我在编写代码的时候，对CLSID的长度进行了检查，如果长于16字节，就会提示警告信息。幸运的是，我的主机上没有查到恶意的CLSID。

3.6 Winlogon

Autoruns 查基于 Winlogon 的自启动项，正常情况下是没有内容的，将 Hide Empty Locations 和 Hide Windows Entries 选项关闭后，可以得到需要查询的注册表键，查看其中的某个键，结果如下

\\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\Credential Providers\{1b283861-754f-4022-ad47-a5eaaa618894}				
<div>> Credential Provider Filters</div> <div>▼ Credential Providers</div> <div><div>{01A30791-40AE-4653-AB2E-FD210019AE88}</div><div><div>{1b283861-754f-4022-ad47-a5eaaa618894}</div><div>{1ee7337f-85ac-45e2-a23c-37c753209769}</div><div>{2135f72a-90b5-4ed3-a7f1-8bb705ac276a}</div><div>{25CB8996-92ED-457e-B28C-4774084BD562}</div><div>{27F8DB57-B613-4AF2-9D7E-4FA7A66C21AD}</div><div>{2D8B3101-E025-480D-917C-835522C7F628}</div></div></div>		名称	类型	数据
		 (默认)	REG_SZ	Smartcard Reader Selection Provider

发现其格式和IE的BHO下的键差不多，尝试通过直接照搬查询BHO的方法，发现取得了不错的效果，其中大部分的键都获取了正确的结果。然而有一个键 HKLM\SOFTWARE\Microsoft\Windows

NT\CurrentVersion\Winlogon\GpExtensions 下的内容没有被正确解析，查看该键下的内容，结果如下

\\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\GPExtensions\{0ACDD40C-75AC-47ab-BAA0-BF6DE7E7FE63}			
<div><div>AlternateShells</div><div>AutoLogonChecked</div><div>GPExtensions</div><div><div>{0ACDD40C-75AC-47ab-BAA0-BF6DE7E7FE63}</div><div>{16be69fa-4209-4250-88cb-716cf41954e0}</div><div>{25537BA6-77A8-11D2-9B6C-0000F8080861}</div><div>{35378EAC-683F-11D2-A89A-00C04FBBCFA2}</div><div>{3610eda5-77ef-11d2-8dc5-00c04fa31a66}</div><div>{426031c0-0b47-4852-b0ca-ac3d37bfc39}</div></div></div>	名称	类型	数据
	(默认)	REG_SZ	Wireless Group Policy
	DisplayName	REG_EXPAND_SZ	@wlgpclnt.dll,-100
	DllName	REG_EXPAND_SZ	wlgpclnt.dll
	GenerateGrou...	REG_SZ	GenerateWLANPolicy
	NoGPOListCha...	REG_DWORD	0x00000001 (1)
	NoUserPolicy	REG_DWORD	0x00000001 (1)
	ProcessGroup...	REG_SZ	ProcessWLANPolicyEx

这就不需要再从 HKLM\SOFTWARE\Classes\CLSID***\InprocServer32\ 中查找了，直接读取子键的 DllName 的内容即可，并且根据 Autoruns 的查询可知，这些 dll 都在 C:\Windows\System32\ 下，所以尽管大部分 DllName 只给出了 dll 的名字而没有路径，对查询也没有太大的影响。最终我得到的查询结果和 Autoruns 一致。

AutoRuns					
<div> <div>Logon</div> <div>services</div> <div>drivers</div> <div>tasks</div> <div>Internet Explorer</div> <div>Winlogon</div> <div>Filter Empty</div> <div>Filter Windows</div> </div>					
{1b283861-754f-4022-ad47-a5}	6/9/2020 7:19 下午				
Smartcard Reader Selection Pr	Windows 智能卡凭据提供程序	no signature	C:\WINDOWS\system32\Smari	3/19/2019 12:45 下午	
Smartcard WinRT Provider	Windows 智能卡凭据提供程序	no signature	C:\WINDOWS\system32\Smari	3/19/2019 12:45 下午	
PicturePasswordLogonProvide	旧版凭据提供程序	no signature	C:\WINDOWS\system32\credp	3/19/2019 12:45 下午	
GenericProvider	凭据提供程序	no signature	C:\WINDOWS\system32\credp	3/19/2019 12:45 下午	
TrustedSignal Credential Provi	TrustedSignal 凭据提供程序	no signature	C:\WINDOWS\system32\Trust	3/19/2019 12:45 下午	
NPPProvider	凭据提供程序	no signature	C:\WINDOWS\system32\credp	3/19/2019 12:45 下午	
Second Authentication Factor	Microsoft 随行验证器凭据提供程序	no signature	C:\Windows\SysWOW64\devic	3/19/2019 12:45 下午	
CngCredUICredentialProvider	Microsoft CNG CredUI 提供程序	no signature	C:\WINDOWS\system32\cngcr	3/19/2019 12:45 下午	
PasswordProvider	凭据提供程序	no signature	C:\WINDOWS\system32\credp	3/19/2019 12:45 下午	
Smartcard Credential Provider	Windows 智能卡凭据提供程序	no signature	C:\WINDOWS\system32\Smari	3/19/2019 12:45 下午	
Smartcard Pin Provider	Windows 智能卡凭据提供程序	no signature	C:\WINDOWS\system32\Smari	3/19/2019 12:45 下午	
WinBio Credential Provider	WinBio 凭据提供程序	no signature	C:\WINDOWS\System32\BioCr	3/19/2019 12:45 下午	
PINLogonProvider	旧版凭据提供程序	no signature	C:\WINDOWS\system32\credp	3/19/2019 12:45 下午	
NGC Credential Provider	Microsoft Passport 凭据提供程序	no signature	C:\Windows\SysWOW64\ngcci	9/14/2019 9:35 上午	
CCertProvider	证书凭据提供程序	no signature	C:\WINDOWS\system32\certC	3/19/2019 12:45 下午	
WLIDCredentialProvider	Microsoft® Account Credentia	no signature	C:\WINDOWS\system32\wlidc	3/19/2019 12:45 下午	
FIDO Credential Provider	FIDO 凭据提供程序	no signature	C:\WINDOWS\system32\fidoc	3/19/2019 12:45 下午	
{5537E283-B1E7-4EF8-9C6E-7A}	6/9/2020 7:19 下午				
CRasProvider	RAS PLAP 凭据提供程序	no signature	C:\WINDOWS\system32\raspl	3/19/2019 12:45 下午	
HKLM\SOFTWARE\Microsoft\	6/9/2020 7:19 下午				
{0ACDD40C-75AC-47ab-BAA0-}	802.11 组策略客户端	no signature	C:\Windows\System32\wlgrpcli	3/19/2019 12:45 下午	
{25537BA6-77A8-11D2-9B6C-}	文件夹重定向组策略扩展	no signature	C:\Windows\System32\fdexplor	3/19/2019 12:45 下午	
{3610eda5-77ef-11d2-8dc5-00}	Windows Shell 磁盘配额支持 DL	no signature	C:\WINDOWS\System32\diskq	3/19/2019 12:45 下午	
{426031c0-0b47-4852-b0ca-ac}	GPTExt	no signature	C:\Windows\System32\gptext	3/19/2019 12:45 下午	
{4CFB60C1-FAA6-47f1-89AA-0}	IEAK branding	no signature	C:\Windows\SysWOW64\iedk	2/13/2020 3:54 下午	
{7909AD9E-09EE-4247-BAB9-7}	Enroll Engine DLL	no signature	C:\Windows\System32\dmenn	10/23/2019 2:19 下午	

4. 自启动技术的研究

4.1 Logon

一个简单的方式是直接的两个自启动目录 `C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Startup`、`C:\Users\yaoyizhou\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup` 中放入想要自启动的程序的快捷方式即可。前者是系统的启动文件夹路径，在任何用户的每次登录时，应用程序都将启动；而后者是用户的启动文件夹路径，只对安装该应用程序的用户可用。在实践中，发现前者放入快捷方式需要提供管理员权限，而删除其中的快捷方式没有提示我提供管理员权限；后者则都不需要提供管理员权限。

在注册表自启动键下添加记录也可以实现用户登录时自启动，常见的几个键是：

```
"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run",
"HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run",
"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\Explorer\Run",
"HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\Explorer\Run",
```

`HKCU` 下的键不需要提供管理员权限就可以写入，`HKLM` 下的键则需要提供管理员权限才能写入。那么如何在不提供管理权限的情况下写入 `HKLM` 下的键呢？我查阅了一些资料，大概有如下几种方式：禁用UAC，这样可以获得整个系统的权限，然而这对于恶意代码来说比直接写入 `HKLM` 更困难，并且大部分人都不会蠢到主动禁用UAC；使用后台服务写入 `HKLM`，后台服务具有对 `HKLM` 键的写访问权限，因此这是一个可行的方案，恶意代码可以伪装成正常的后台服务，以骗取管理员权限；还有一种方式是管理员编辑注册表相关表项上的权限来允许写访问。

基于 `Logon` 的自启动在注册表和自启动目录都能直接找到，故隐蔽性较差。

参考资料：<https://social.msdn.microsoft.com/Forums/vstudio/en-US/46b8d4a8-a99c-4796-9b94-4d493aac6674/how-we-can-get-write-access-in-hklm-hkeylocalmachine-without-admin-account?forum=vcgeneral>

4.2 Services & Drivers

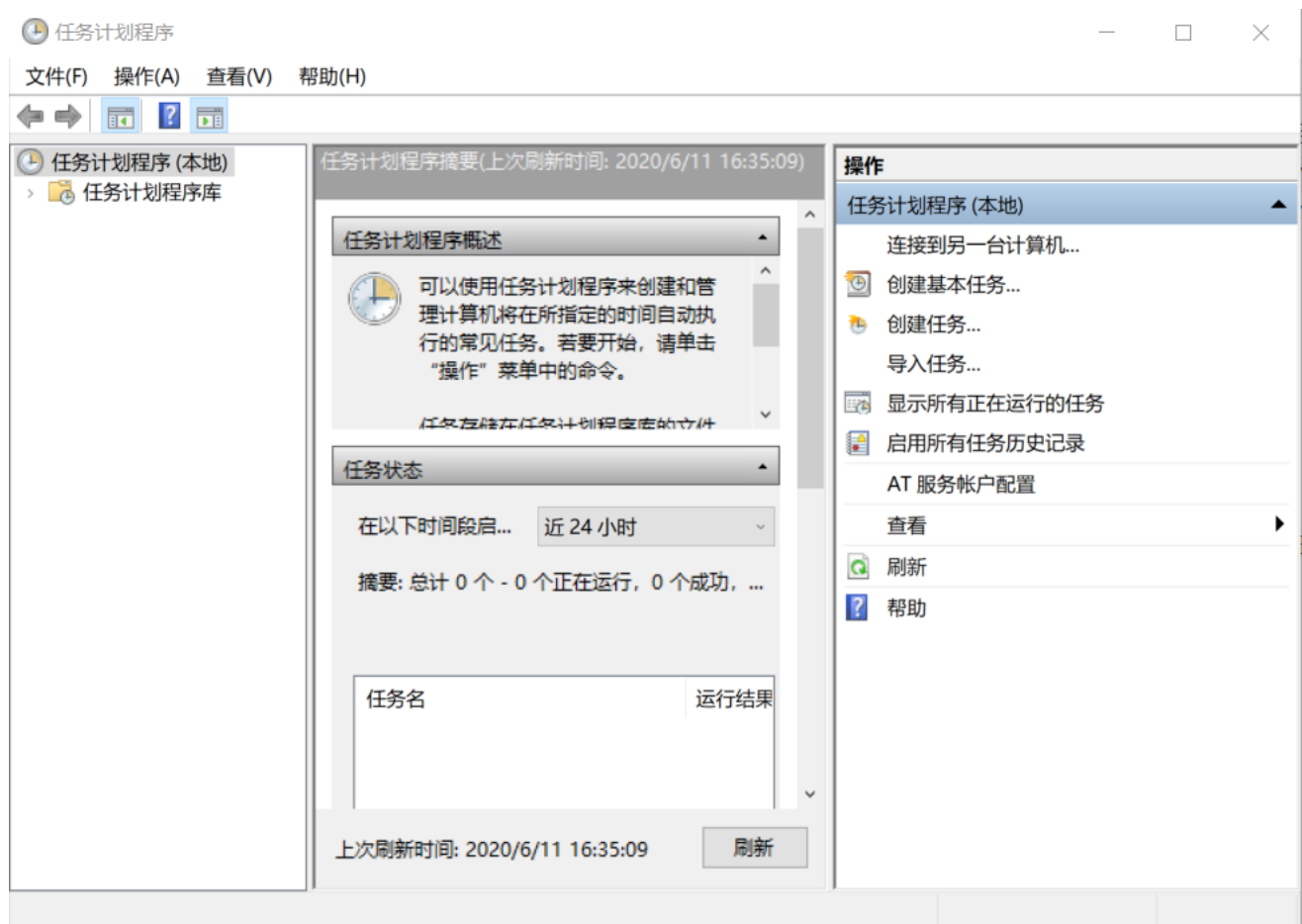
服务程序是后台运行的进程，常用来执行特定的任务，不需要和用户进行交互。服务程序的加载有两种方式，通过 `svchost.exe` 启动和 Windows 直接启动，两者在不同时刻加载服务。这两种方式也恰好分别对应了查询时的查 `ImagePath` 或 `ServiceDll` 两种情况。而驱动程序可以理解为是始终运行在内核态的服务程序。恶意代码可以通过上述 §4.1 中提到的方法或利用系统中的一些漏洞提升 UAC，以写入 `HKLM\System\CurrentControlSet\Services` 这个键下，并设置 `Start` 的内容来决定在什么时刻自启动。

`Services` 和 `Drivers` 数目较多，并且是基于 `dll` 和 `sys` 加载的，普通用户难以辨别并且手动删除服务也比较危险，所以其隐蔽性要好于计划任务和 `Logon`。

参考资料: <https://www.bleepingcomputer.com/tutorials/how-malware-hides-as-a-service/>

4.3 Scheduled Tasks

使用计划任务可以在用户登录时完成应用的自启动，Windows 10 提供了相应的应用任务计划程序，用户可以创建任务，添加想要实现自启动的应用。虽然添加计划任务并不需要提供管理员权限，但是使计划任务以管理员权限运行是需要管理员权限的。那么该如何绕过管理员权限？在网上找了许多资料，并没有找到相关的方法或实现思路。由于计划任务都位于同一目录下，其隐蔽性也不是很好。



4.4 Internet Explorer

Browser Helper Object 是 DLL 模块，旨在作为 IE 浏览器的附加组件来提供附加功能。当系统上安装 BHO 后，每次启动 IE 浏览器时，这些 DLL 都会被自动加载。而攻击者可以通过安装恶意的 BHO 来利用此功能以窃取用户的密码或进行其他的恶意活动。当 BHO 注册到系统时，会在

`HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects` 等注册表位置注册一个子键，这个子键的 key 是 `CLSID`，其默认长度是 16 字节，IE 浏览器启动时根据它来查找需要加载的 DLL 模块。而恶意的 BHO 可以设置长于 16 字节的 `CLSID`，浏览器在加载时，只会读取前 16 字节，这个“部分”的 `CLSID` 可以指向

一个恶意的 DLL，而使用 IE 浏览器的管理加载项或 Autoruns 查看 BHO 时，这两个工具都不会显示已安装的恶意 BHO，因为它们会读取整个 CLSID 字符串，这个 CLSID 可以指向一个非恶意的 DLL。由于能躲过 Autoruns 的检查，这种方法的隐蔽性较好，因此，我实现的查看基于 BHO 的自启动，改进了 Autoruns 的缺点。

参考资料: <https://www.greyhathacker.net/?p=106>

4.5 Winlogon

当用户登录时，攻击者可能会滥用 Winlogon 的功能来加载并执行恶意 DLL 或恶意的可执行文件。我在 <https://attack.mitre.org/beta/techniques/T1547/004/> 中找到了一些通过修改 Winlogon 下的子键来完成自启动的恶意代码的例子，但是奇怪的是在我主机的注册表相应位置下并没有对应的子键。不过根据 Autoruns 查询的方法和我自己实验的结果，可以推断出基于 Winlogon 的自启动和基于浏览器 BHO 的自启动是类似的，Winlogon 在启动时会加载一些提供附加功能的 DLL，而这些 DLL 是通过 CLSID 来指示的，通过构造特殊的 CLSID 可以让 Winlogon 加载恶意的 DLL 而不会被类似 Autoruns 的软件发现，因此拥有较好的隐蔽性。同样的，我实现的查看基于 Winlogon 的自启动程序会检查 CLSID 的长度，并做出相应的警告。

5.总结与体会

本次自启动项的查看大作业查阅了很多网上的资料和微软的官方文档，并使用了 C# 和 WPF 完成了一个简易的自启动项查看软件，在实现的过程中遇到了很多困难，其中大部分通过网络和同学的交流得以解决，但是还有一些问题并没有得到很好的解决。总而言之，从本次作业中收获到了很多，提升了 coding 能力和查阅资料的能力。