

# 实验三、恶意代码分析实验

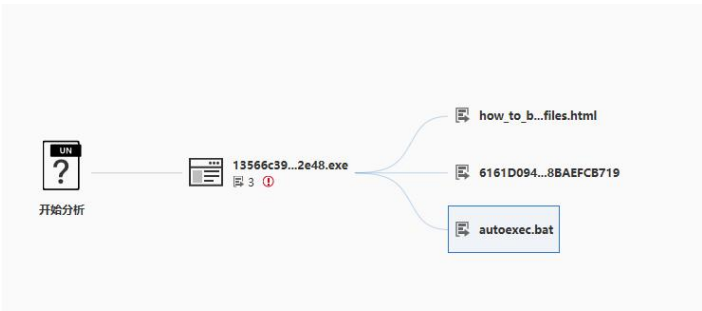
## 目录

实验三、恶意代码分析实验.....	1
1. 初步观察.....	1
2. 样本行为的详细分析.....	1
3. 样本危害性.....	9
4. 防护建议.....	10

### 1. 初步观察

选择微步云沙箱对该代码进行初步分析。该沙箱初步检测该文件为恶意。以下为检测内容：

- 1) 在行为签名中，高危行为有 3 个，分别为：
  - A) 注册表实现自启动
  - B) 将 282 个文件移动, 该操作表示有勒索软件正在执行加密操作
  - C) 修改 282 个文件为相同的扩展名或改变其内容，该操作表示有勒索软件正在执行加密操作
- 2) 可疑行为有 5 个，分别为：
  - A) 可执行文件只有一个节
  - B) 创建快捷方式
  - C) 窃取浏览器隐私信息
  - D) 创建新的文档类文件
  - E) 将自身拷贝到其他目录下
- 3) 低危行为有 3 个，分别为：
  - A) 读写 ini 文件
  - B) 尝试打开应用程序的配置文件
  - C) 在文件系统上创建可执行文件
- 4) 静态信息为：
  - 导入表为 KERNEL32. dll， ADVAPI32. dll， SHELL32. dll， SHLWAPI. dll， ntdll. dll
  - 执行流程为：



共释放 3 个文件

### 2. 样本行为的详细分析

- 1) 行为 1：释放文件至%AppData%处
- 在采用 PVIEW 进行静态分析时，可知该程序调用了 KERNEL32. dll，主要是针对文件进行的一系列操作。

Offset	Data	Description	Value
0000424	0000C030	HostName RUA	0487 SelfPointerEx
0000428	0000CCE4	HostName RUA	0952 CloseHandle
000042C	0000CF2	HostName RUA	054E InitWin
0000430	0000CFE	HostName RUA	008F CreateFileW
0000434	0000C0C	HostName RUA	02C0 HeapCreate
0000438	0000D1A	HostName RUA	01C3 GetCurrentProcess
000043C	0000C0E	HostName RUA	0119 ExitProcess
0000440	0000C3C	HostName RUA	08B5 CreateThread
0000444	0000C4C	HostName RUA	01C4 GetCurrentThread
0000448	0000C06	HostName RUA	0489 SetThreadPriority
000044C	0000D1A	HostName RUA	0487 HeapOfMemoryObjects
0000450	0000C0E	HostName RUA	0482 Sleep
0000454	0000C06	HostName RUA	0209 GetLocalTime
0000458	0000C0A	HostName RUA	0486 SelfPointer
000045C	0000C0C	HostName RUA	012E FindClose
0000460	0000C0C	HostName RUA	0544 WaitForSingleObject
0000464	0000C0D	HostName RUA	0545 WaitForSingleObject
0000468	0000C0C	HostName RUA	0547 WaitForSingleObject
000046C	0000C0C	HostName RUA	02C3 ReadFile
0000470	0000C0F	HostName RUA	053F WriteFile
0000474	0000C0F	HostName RUA	0214 GetModuleFileNameW
0000478	0000C0F	HostName RUA	0214 GetModuleFileNameW
000047C	0000C0F	HostName RUA	0214 GetModuleFileNameW
0000480	0000C0F	HostName RUA	0214 GetModuleFileNameW
0000484	0000C0F	HostName RUA	0214 GetModuleFileNameW
0000488	0000C0F	HostName RUA	0214 GetModuleFileNameW
000048C	0000C0F	HostName RUA	0214 GetModuleFileNameW
0000490	0000C0F	HostName RUA	0214 GetModuleFileNameW
0000494	0000C0F	HostName RUA	0214 GetModuleFileNameW
0000498	0000C0F	HostName RUA	0214 GetModuleFileNameW
000049C	0000C0F	HostName RUA	0214 GetModuleFileNameW
00004A0	0000C0F	HostName RUA	0214 GetModuleFileNameW
00004A4	0000C0F	HostName RUA	0214 GetModuleFileNameW
00004A8	0000C0F	HostName RUA	0214 GetModuleFileNameW
00004AC	0000C0F	HostName RUA	0214 GetModuleFileNameW
00004B0	0000C0F	HostName RUA	0214 GetModuleFileNameW
00004B4	0000C0F	HostName RUA	0214 GetModuleFileNameW
00004B8	0000C0F	HostName RUA	0214 GetModuleFileNameW
00004BC	0000C0F	HostName RUA	0214 GetModuleFileNameW
00004C0	0000C0F	HostName RUA	0214 GetModuleFileNameW
00004C4	0000C0F	HostName RUA	0214 GetModuleFileNameW
00004C8	0000C0F	HostName RUA	0214 GetModuleFileNameW
00004CC	0000C0F	HostName RUA	0214 GetModuleFileNameW

打开 IDA 与 Ollydbg 进行动态分析，进入 sub\_409C68。

```

; Attributes: noreturn
public start
start proc near
call sub_409C6B
push 0 ; uExitCode
call ExitProcess
start endp

```

程序进行一系列栈操作后，调用 GetEnvironmentVariableW 函数。

```

call sub_402779
mov ebx, GetEnvironmentVariableW
mov esi, eax
pop ecx
push ebp ; nSize
push esi ; lpBuffer
push offset aLocalappdata ; "LOCALAPPDATA"
call ebx ; GetEnvironmentVariableW
test eax, eax
jnz short loc_409D92

```

该函数参数与介绍如下，用于检索环境变量的值。

## Syntax

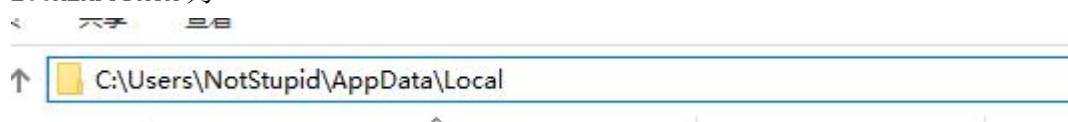
```

C++
Copy

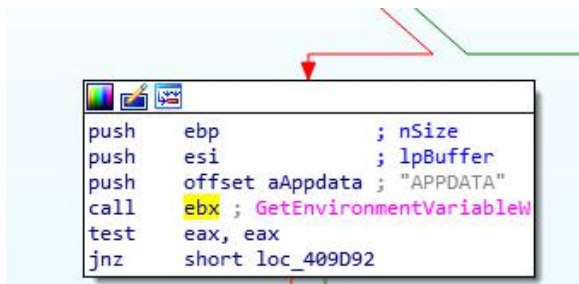
DWORD GetEnvironmentVariableW(
    [in, optional] LPCWSTR lpName,
    [out, optional] LPWSTR lpBuffer,
    [in]           DWORD   nSize
);

```

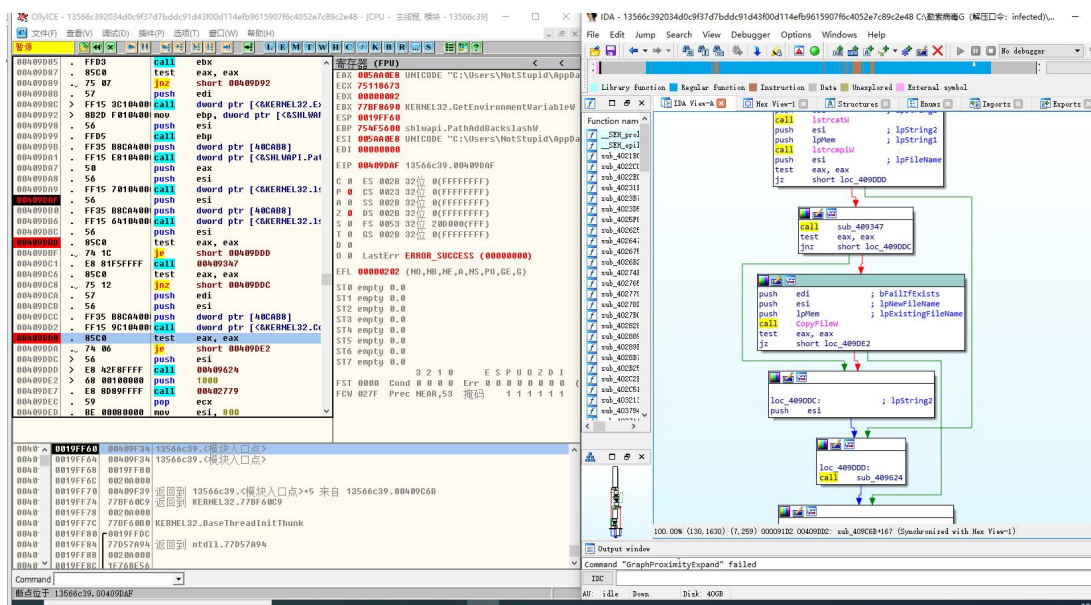
在本程序中，lpName=" LOCALAPPDATA"，即函数返回环境变量为 LOCALAPPDATA 的路径。虚拟机中 LOCALAPPDATA 为



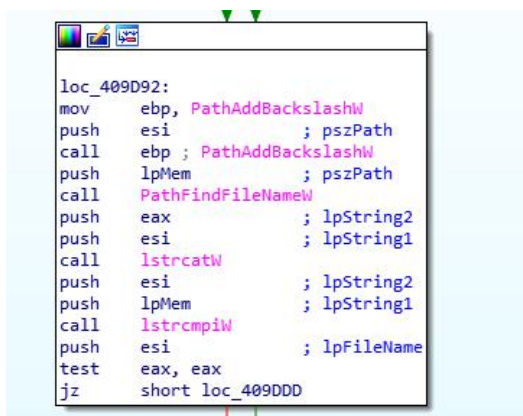
根据 jnz 语句可知该结构属于 if-else，如果成功没有获取 LOCALAPPDATA 的路径，再用同样方法查询 AppData（%AppData% 默认是用户根目录下的 AppData\Roaming 文件夹）。若仍未成功，跳转至函数末尾。



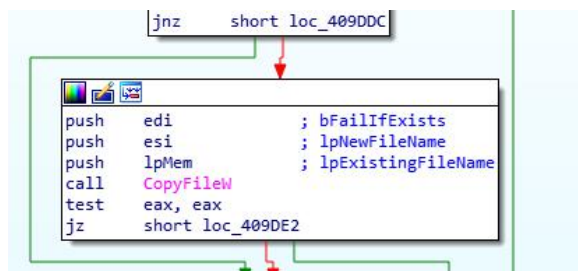
通过观察 o1lydbg 可以见到已经得到了 LOCALAPPDATA，直接跳转，不再进行 AppData 的判断



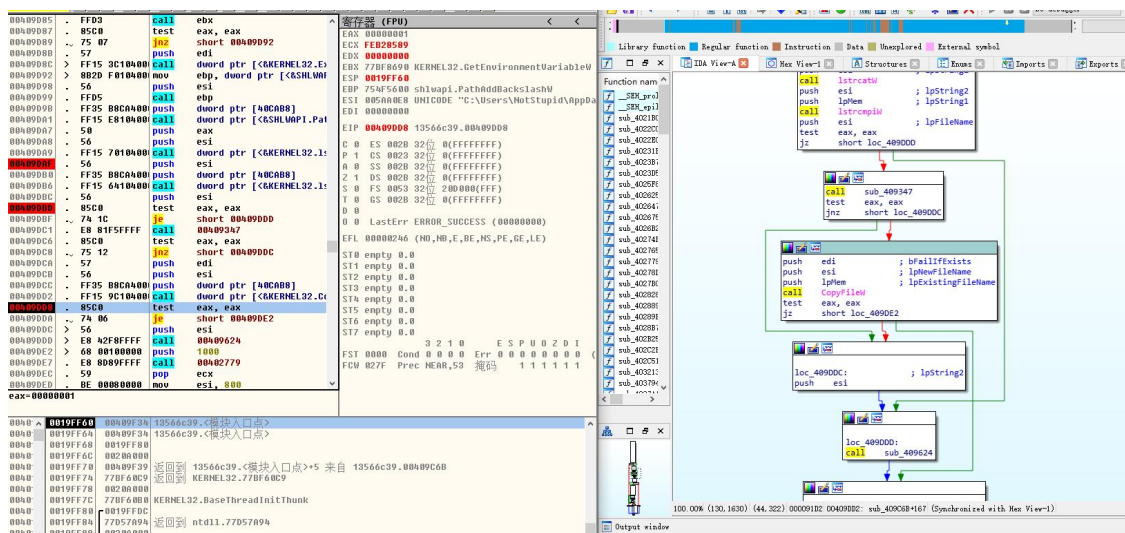
查找到路径后，调用 PathAddBackslashW 函数，在字符串后添加反斜线构成完整路径。调用 pathfindfilenameW 函数，根据文件路径取文件名，搜索 LocalAppData 目录下的所有文件。再调用 lstrcatW 函数，将搜索到的 LOCALAPPDATA 下的文件名接在 LOCALAPPDATA 的路径后，构成目标文件的完整路径。



调用 CopyFileW，将 lpMem 中的文件复制至 esi 所指的文件中



利用 011yDbg 在此处设置断点，打开 AppLocalData 所指文件夹，此时 appdata 目录下已经出现新文件（见底部）



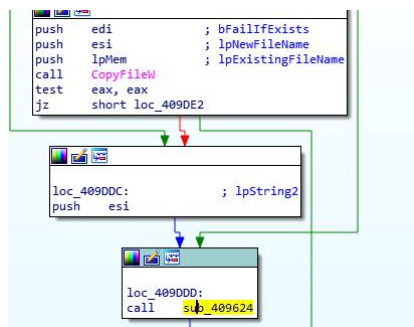
名称	修改日期	类型	大小
Comms	2021/11/6 15:52	文件夹	
ConnectedDevicesPlatform	2021/11/6 15:41	文件夹	
Microsoft	2021/11/9 18:50	文件夹	
MicrosoftEdge	2021/11/6 15:43	文件夹	
Packages	2021/11/9 18:56	文件夹	
PlaceholderTileLogoFolder	2021/11/6 16:19	文件夹	
Publishers	2021/11/6 15:42	文件夹	
Temp	2021/11/9 18:57	文件夹	
VirtualStore	2021/11/6 15:41	文件夹	
13566c392034d0c9f37d7bddc91d43f...	2020/1/25 21:58	文件	54 KB

## 2) 行为 2: 添加注册表键

在利用 PEVIEW 进行静态分析时，在函数导入表中可得该文件调用了 ADVAPI32.dll。该动态链接库与函数与对象的安全性，注册表的操控以及事件日志有关。

pFile	Data	Description	Value
00000400	0000CFB2	Hint/Name RVA	026E RegQueryValueExW
00000404	0000CFA2	Hint/Name RVA	0261 RegOpenKeyExW
00000408	0000CF90	Hint/Name RVA	0239 RegCreateKeyExW
0000040C	0000CF82	Hint/Name RVA	0230 RegCloseKey
00000410	0000CF70	Hint/Name RVA	00C1 CryptGenRandom
00000414	0000CF5A	Hint/Name RVA	00CB CryptReleaseContext
00000418	0000CF42	Hint/Name RVA	00B1 CryptAcquireContextW
0000041C	0000CF6C	Hint/Name RVA	027E RegSetValueExW
00000420	00000000	End of Imports	ADVAPI32.dll

再利用 IDA 与 011yDbg 进行动态分析。行为 1 结束后，进入 sub\_409624

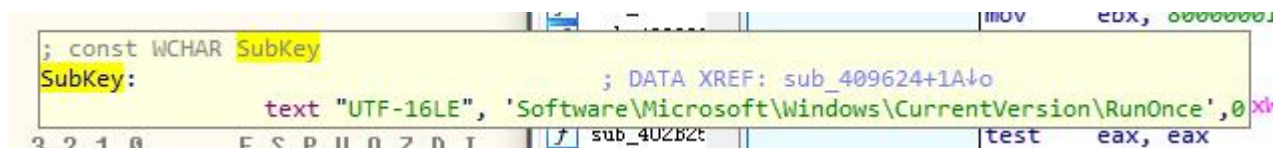


调用函数 RegOpenKeyExW，该函数用于打开一个指定的注册表键。

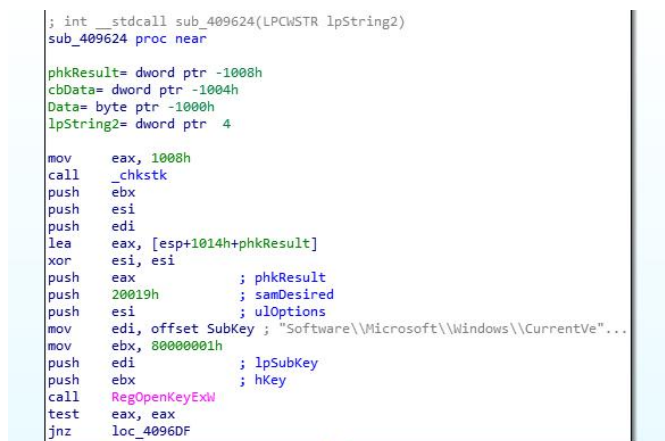
## Syntax

```
C++
LSTATUS RegOpenKeyExW(
    [in] HKEY hKey,
    [in, optional] LPCWSTR lpSubKey,
    [in] DWORD ulOptions,
    [in] REGSAM samDesired,
    [out] PHKEY phkResult
);
```

lpSubKey 的值如图

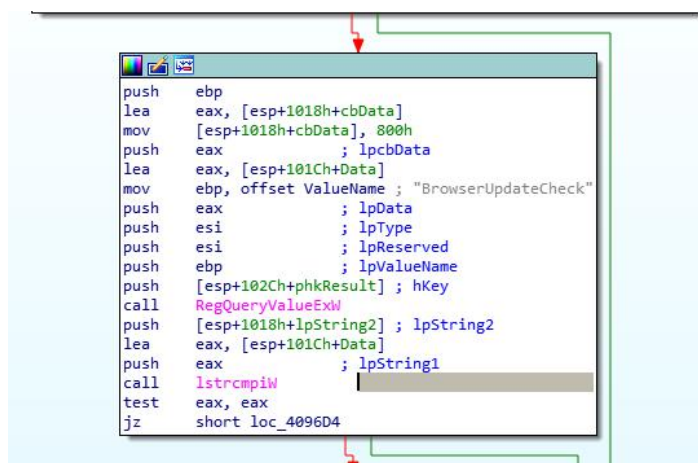


即需要打开键值为 software\microsoft\windows\currentversion\runonce 的注册表键。该键用于自动运行程序




调用函数 RegQueryValue 检索与打开的注册表项相关联的指定值名称的类型和数据，注册表值的名称是 BrowserUpdateCheck，打开注册表项的句柄 hkey 存在 phkresult 中。完成此操作后，成功设置自启动的注册表键值为 BrowserUpdateCheck。





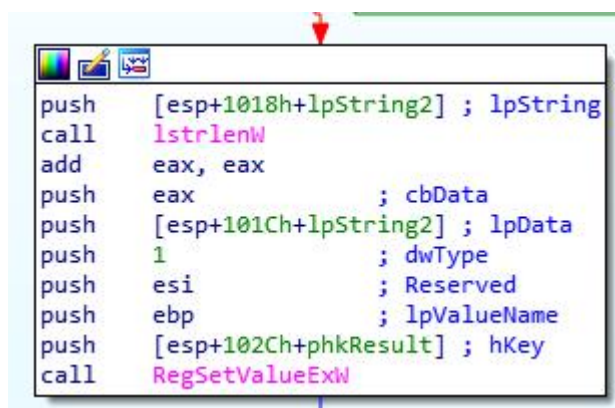
```
push    ebp
lea     eax, [esp+1018h+cbData]
mov     [esp+1018h+cbData], 800h
push    eax          ; lpcbData
lea     eax, [esp+101Ch+Data]
mov     ebp, offset ValueName ; "BrowserUpdateCheck"
push    eax          ; lpData
push    esi          ; lpType
push    esi          ; lpReserved
push    ebp          ; lpValueName
push    [esp+102Ch+phkResult] ; hKey
call    RegQueryValueExW
push    [esp+1018h+lpString2] ; lpString2
lea     eax, [esp+101Ch+Data]
push    eax          ; lpString1
call    lstrcpw
test    eax, eax
jz      short loc_4096D4
```

调用函数 RegCreateKeyExW，该函数用途为创建指定的注册表项，如果键已经存在，函数将打开它。已知键已存在，因此调用该函数的目的是打开新建的 BrowserUpdateCheck 键



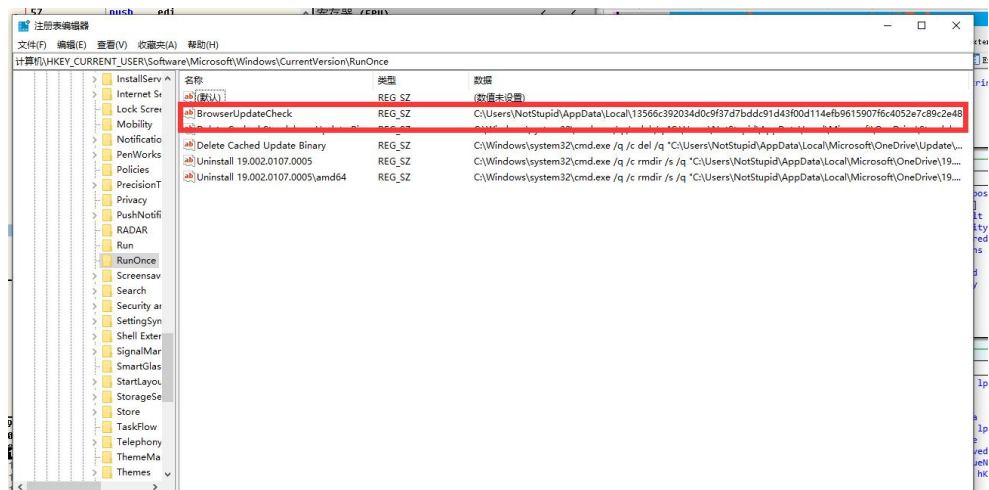
```
push    esi          ; lpdwDisposition
lea     eax, [esp+101Ch+phkResult]
push    eax          ; phkResult
push    esi          ; lpSecurityAttributes
push    20006h       ; samDesired
push    1            ; dwOptions
push    esi          ; lpClass
push    esi          ; Reserved
push    edi          ; lpSubKey
push    ebx          ; hKey
call    RegCreateKeyExW
test    eax, eax
jnz     short loc_4096D4
```

调用 RegSetValueExW，将新建键值设置为 ebp 所指字符串，即行为 1 中新建文件的完整路径名。



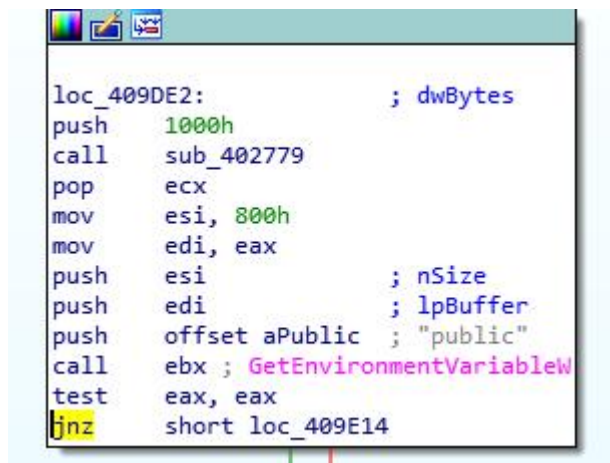
```
push    [esp+1018h+lpString2] ; lpString
call    strlenW
add     eax, eax
push    eax          ; cbData
push    [esp+101Ch+lpString2] ; lpData
push    1            ; dwType
push    esi          ; Reserved
push    ebp          ; lpValueName
push    [esp+102Ch+phkResult] ; hKey
call    RegSetValueExW
```

将断点设置为该段函数执行结尾处，打开注册表查看。病毒在注册表中添加了新的键值，名称为 BrowserUpdateCheck，数据为在行为 1 中在%localappdata%下新出现的文件。该操作能够达到病毒自启动的目的



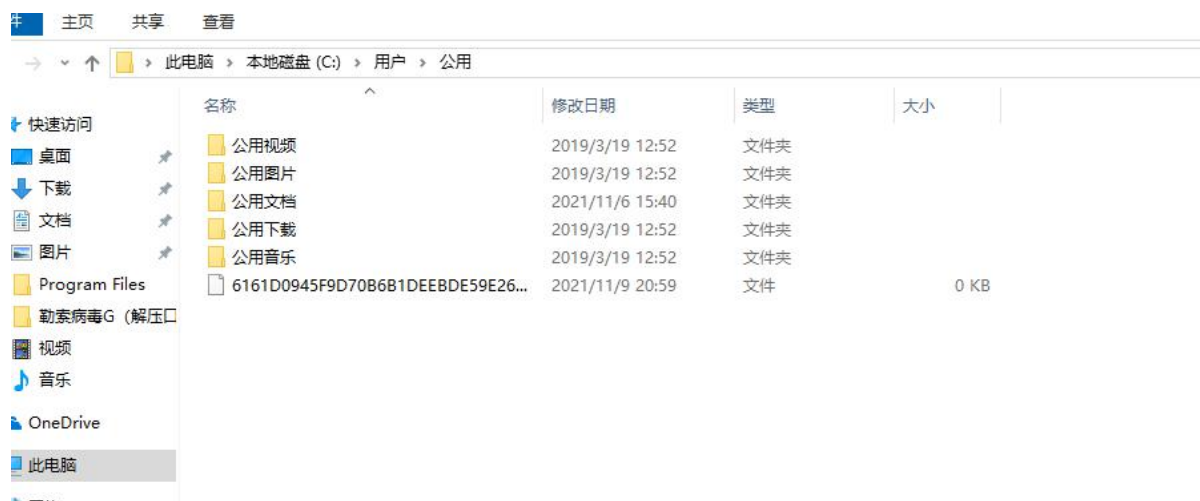
### 3) 行为 3: 在%PUBLIC%下新建文件

行为 2 结束后, 先采用和行为 1 一样的方法, 得到%public%的值, 为 c:\users\public。若%PUBLIC%为找到则继续寻找%allusersprofile%。在本虚拟机运行环境下, %PUBLIC%被成功定位。



EDI 005AB100 UNICODE "C:\Users\Public"

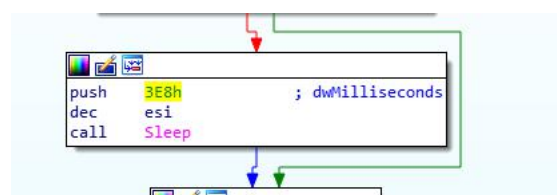
和行为 1 同理, 之后调用 PathAddBackslashW 在字符串后加上反斜线, 然后调用函数 sub\_4098B8 跳转后, 先调用 CreateFileW 函数, 在%PUBLIC%目录下生成新文件, 打开%PUBLIC%查看, 可见有新文件产生。但此时只创建了该文件, 还未向文件中写入内容。



再调用 WriteFile 向文件写入内容，可见文件大小发生变化



调用 sleep 函数，令进程进入短暂休眠状态



4) 行为 4: 对文件进行加密

利用 PEVIEW 进行静态分析，可知该程序还调用了

A) SHELL32.dll

其中，函数 SHChangeNotify 作用为：当应用程序的行为可能影响内核时，调用这个函数  
函数 ShellExecuteExW 作用为：不仅可以运行 EXE 文件，也可以运行已经关联的文件

000004DC	0000CFF8	Hint/Name RVA	007F SHChangeNotify
000004E0	0000CFE6	Hint/Name RVA	0121 ShellExecuteExW
000004E4	00000000	End of Imports	SHELL32.dll

B) SHLWAPI.dll

该动态库主要依据路径对文件进行删除操作，预计属于勒索操作的一部分，当勒索不成功，采取此操作对文件进行删除

000004E4	00000000	End of Imports	SHLWAPI.dll
000004E8	0000D02A	Hint/Name RVA	0049 PathFindFileNameW
000004EC	0000D03E	Hint/Name RVA	008B PathRemoveFileSpecW
000004F0	0000D016	Hint/Name RVA	0030 PathAddBackslashW
000004F4	00000000	End of Imports	SHLWAPI.dll

C) ntdll.dll

该动态库与 NT 内核操作相关



000004F8	0000D060	Hint/Name RVA	04FE _aulldiv
000004FC	0000D06C	Hint/Name RVA	04F6 _alldiv
00000500	0000D076	Hint/Name RVA	04FA _allrem
00000504	0000D080	Hint/Name RVA	0502 _chkstk
00000508	0000D094	Hint/Name RVA	0396 RtlUnwind
0000050C	0000D0A0	Hint/Name RVA	0135 NtQueryVirtualMemory
00000510	00000000	End of Imports	ntdll.dll

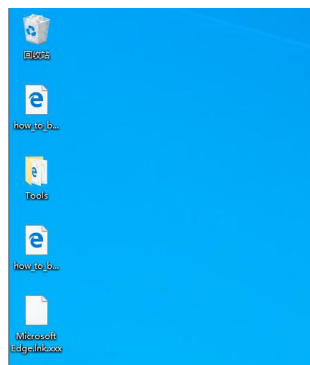
NT 内核级文件

再利用 o1lydbg 和 IDA 进行动态分析。调用 GetProcessHeap 与 GetLogicDriver，作用分别为调用堆栈与返回目前有空余空间的磁盘。

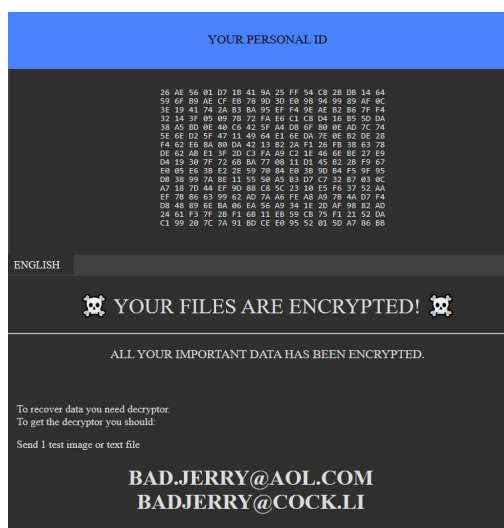
```
sub_4099A3 proc near
    Handles= dword ptr -870h
    RootPathName= byte ptr -800h
    var_7FC= byte ptr -7FC
    arg_0= dword ptr 4
    arg_4= dword ptr 8

    sub esp, 870h
    push ebx
    push ebp
    push esi
    push edi
    mov edi, GetProcessHeap
    push 70h ; dwBytes
    push 0 ; dwFlags
    call edi ; GetProcessHeap
    push eax ; hHeap
    call HeapAlloc
    mov ebp, eax
    call GetLogicalDrives
    push 7FC
    mov ebx, eax
    mov dword ptr [esp+884h+RootPathName], 5C3A41h
    xor esi, esi
    lea eax, [esp+884h+var_7FC]
    push esi
    call sub_402647
    add esp, 0Ch
    test ebx, ebx
    jz short loc_409A47
endproc
```

再通过 mov 将特定字符串存入指定位置，最后调用 WaitForMultipleObject，等待句柄所指内核对象调用结束。该调用结束后，病毒整体感染完成，文件被加密，出现 how\_to\_back\_file 的勒索文件。



打开该文件，内容为勒索内容和返还数据方式。



### 3. 样本危害性

样本会通过释放文件、修改开机自启动项最终对用户应用程序数据进行加密并勒索，造成用户数据与财产损失。

#### 4. 防护建议

1) 开启 windows 系统自带的病毒和威胁防护。在实验过程中可以发现，当试图运行病毒程序时，病毒和威胁防护会阻止程序的运行，甚至自行清理程序。在手动关闭“病毒和威胁防护”设置后，病毒程序才能继续启动。因此，如果用户遵照 windows 提示进行查杀修复，能够有效避免该病毒的执行。



2) 及时更新 windows 操作系统版本

本次虚拟机运行环境为 Wwindows10，相比于 xp 等版本，windows10 的相关更新更加及时，防护能力也更强。

3) 对网络上的文件下载保持警惕

在查阅相关资料时可发现，该病毒在之后又衍生出了多个变种，诸如通过邮件进行传播。普通用户应对该类事件保持警惕，不要随意下载网络上的不明文件。