BOHOPS

A blog about cybersecurity research, education, and news

WRITTEN BY BOHOPS

MARCH 10, 2018

LEVERAGING INF-SCT FETCH & EXECUTE TECHNIQUES FOR BYPASS, EVASION, & PERSISTENCE (PART 2)

QUICK LINKS

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)

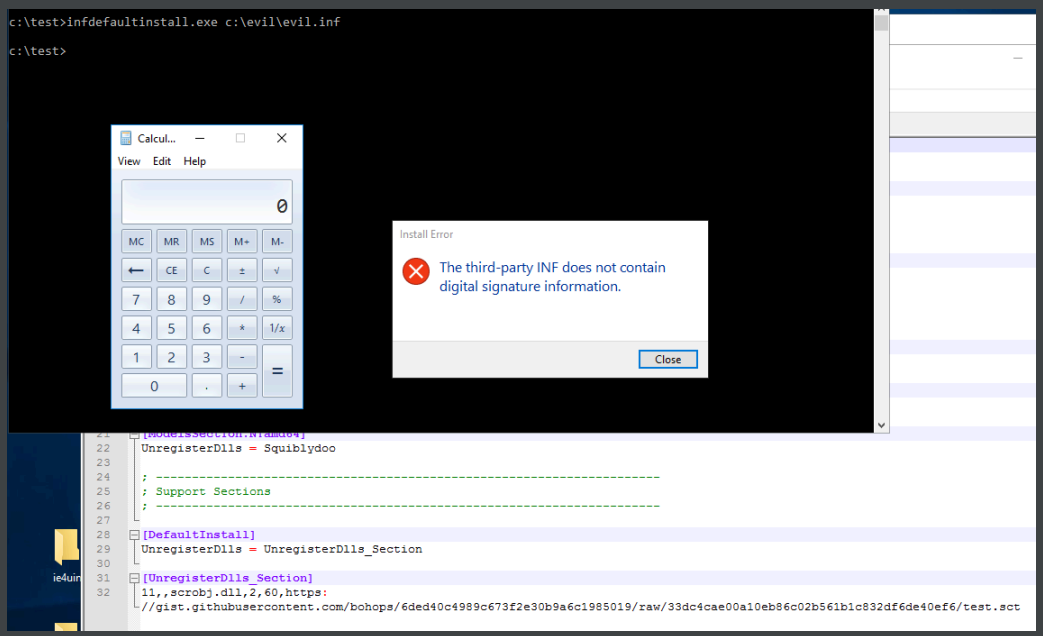
SETUPAPI.DLL (INSTALLHINFSECTION) – INFDEFAULTINSTALL.EXE

In their DerbyCon 2017 talk – Evading AutoRuns, @KyleHanslovan and @ChrisBisnett of @HuntressLabs presented several INF-SCT techniques. In particular, I highlighted that Setupapi.dll

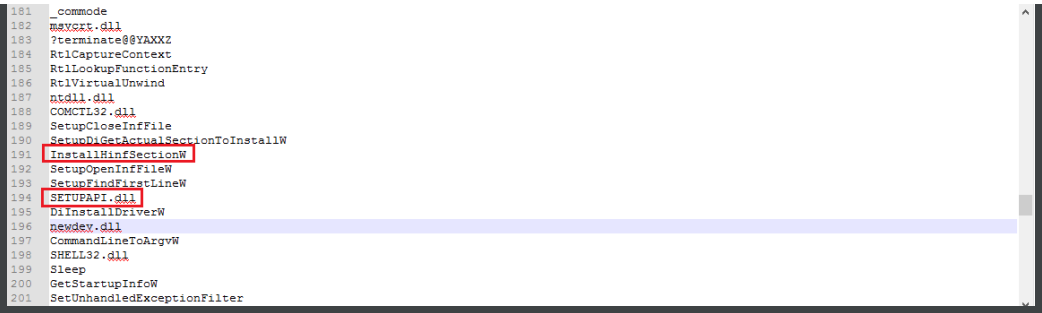
(InstallHinfSection) could be used for such invocation, but I deviated from the spirit of their presentation by failing to mention their discovery of InfDefaultInstall. Using this binary, INF-SCT payload execution can be achieved with this basic command:

```
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):

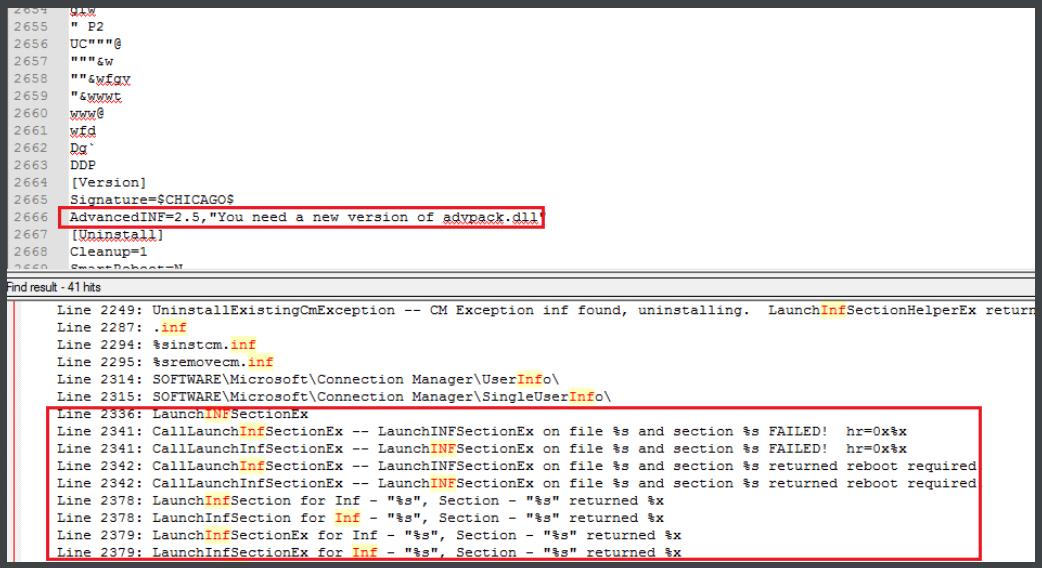


After running **SysInternals Strings** to do a quick analysis of InfDefaultInstall, it appears that this binary relies on the invocation of Setupapi.dll and a character set compatibility variant of InstallHinfSection to achieve execution:



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:

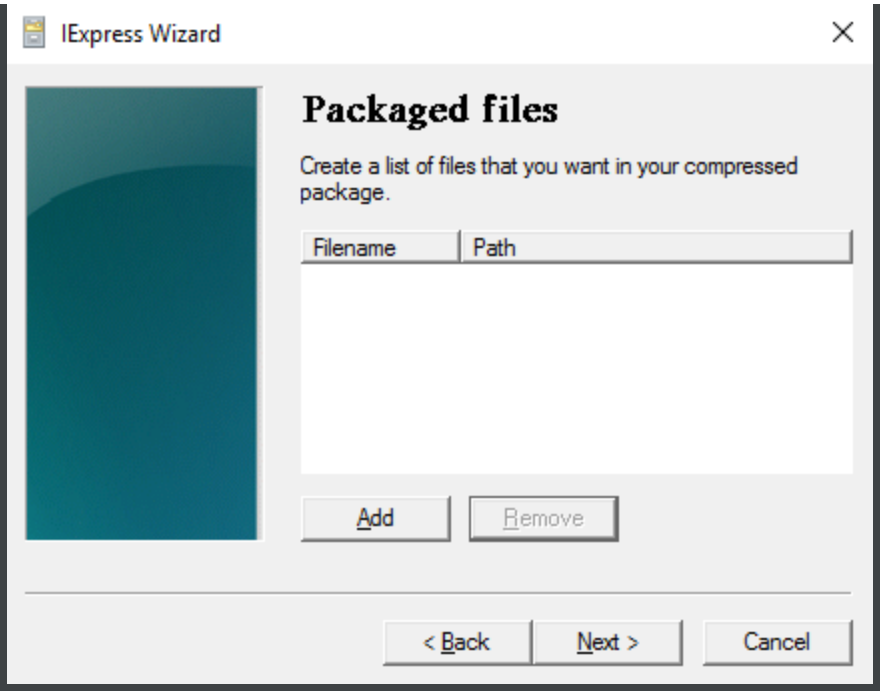


Under the hood, CMSTP leverages Advpack.dll and a variant of LaunchINFSection.

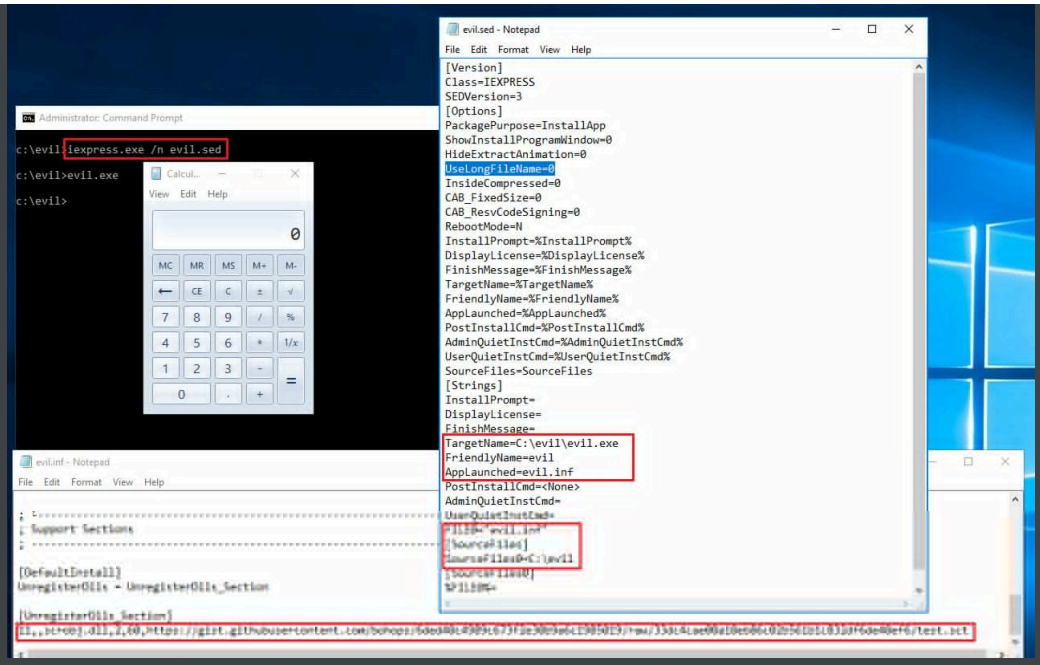
INTRODUCING INF-SCT EXECUTION WITH IEEXPRESS, IEADVPACK.DLL (LAUNCHINFSECTION), AND IE4UINIT

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.



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:



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:

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!

**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':

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:

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

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:

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:

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

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:

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

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.

TAGGED APPLOCKER, BLUETEAM, DFIR, REDTEAM.

ONE THOUGHT ON "LEVERAGING INF-SCT FETCH & EXECUTE TECHNIQUES FOR BYPASS, EVASION, & PERSISTENCE (PART 2)"

Pingback: AppLocker Bypass – CMSTP | Penetration Testing Lab

Comments are closed.

PREVIOUS POST

NEXT POST