BACK TO POSTS

Feb 25, 2020

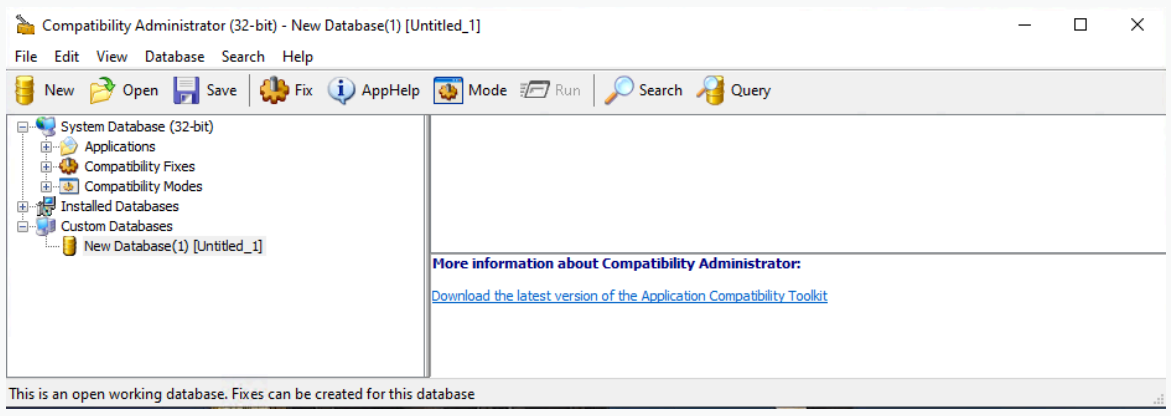
Persistence via Shims

Posted in Pentesting

Application Shimming

This'll be a quick post on the Windows Persistence method of Application Shimming - T1138.

Microsoft developed a method for users/engineers to create custom fixes for application compatibility issues on differing Windows version. These custom fixes are called shims and can be created by the end user with a free Microsoft tool called Application Compatibility Toolkit(ACT).



The user (attacker) created shims can include any options provided within the ACT, which are many - from *"Inject DLL"* to

"DisableWindowsDefender". Once the Shim is built with ACT, a Shim Database file (.sdb) is created and outputted to disk. This sdb file can then be installed on the target with **sbdinst.exe**, which will implement the shim.

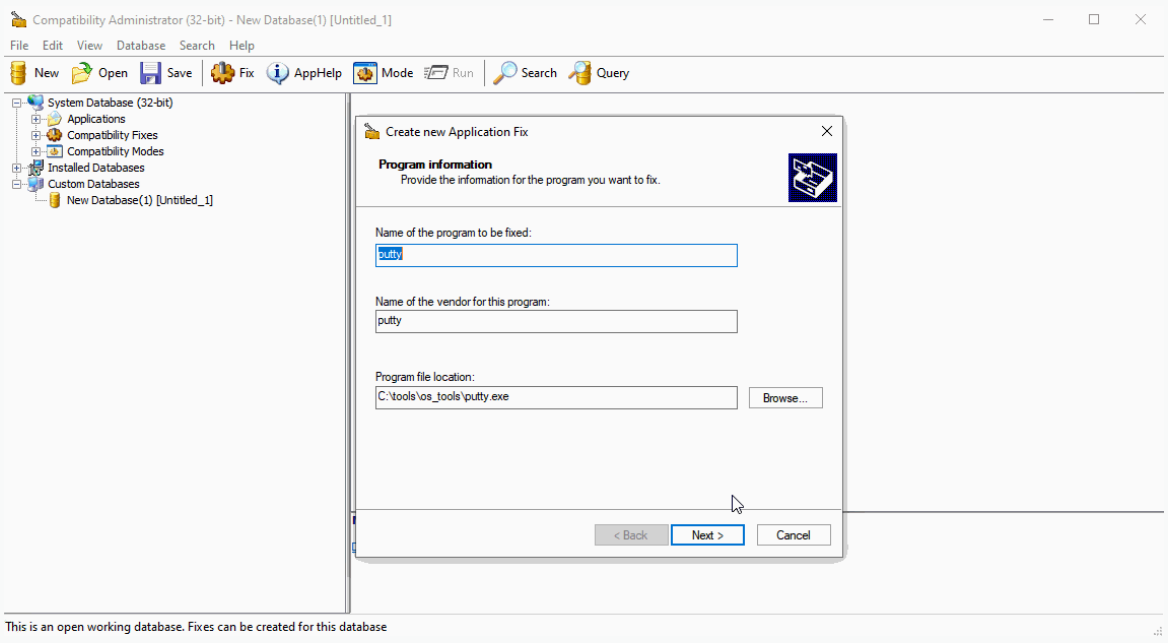
This technique is not new, and has a decent amount of research already available. From my quick research the biggest gap from publicly available resources is the shim options, which are ripe for research. For an excellent talk on shims, I'd absolutely recommend Sean Pierce's DEF CON talk on Abusing Native Shims for Post Exploitation. A lot of useful info from this talk. Checkout additional sources at the end as well.

Build a Shim

To summarize, an attacker can build (using ACT) and add (using sdbinst.exe) plenty of functionalities to any application on a target machine.

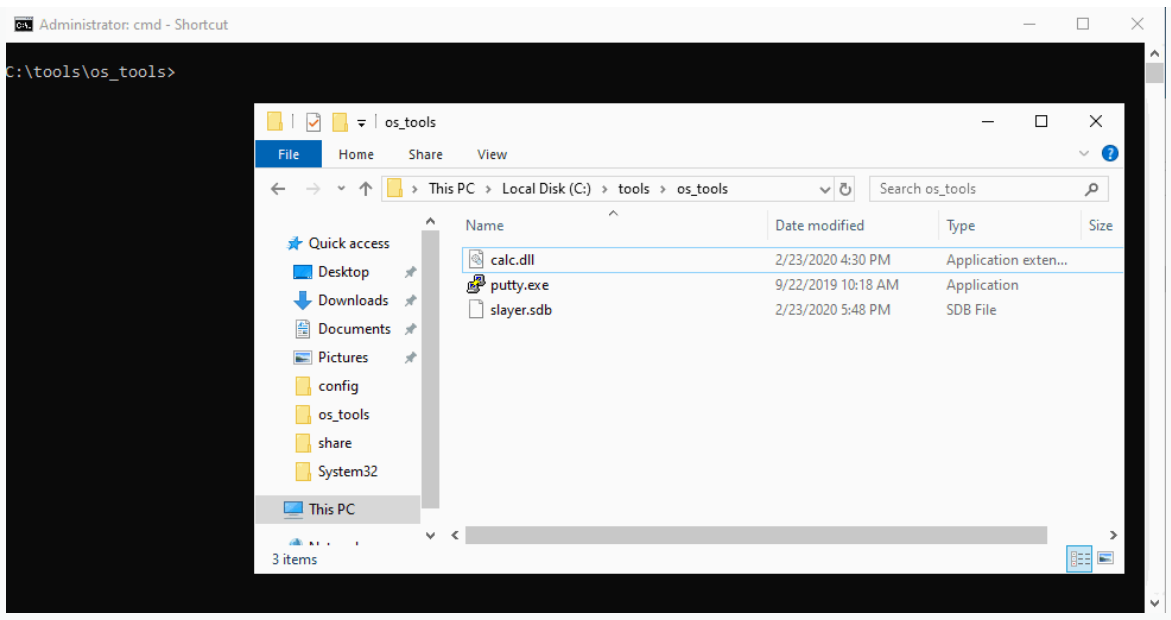
Sdbinst.exe is a built-in Windows binary, which allows you to install a pre-created sdb file from CLI (as Admin). The ACT app **does not** need to be installed and configured on the target machine, but can be installed on an attacker controlled, host, lab, etc machine.

As documented in other sources, we'll first create putty shim using the "*Inject DLL*" functionality. The DLL in this example will simply pop calc. Setting the location of the DLL is important - which will need to stay the same once the shim is pushed.



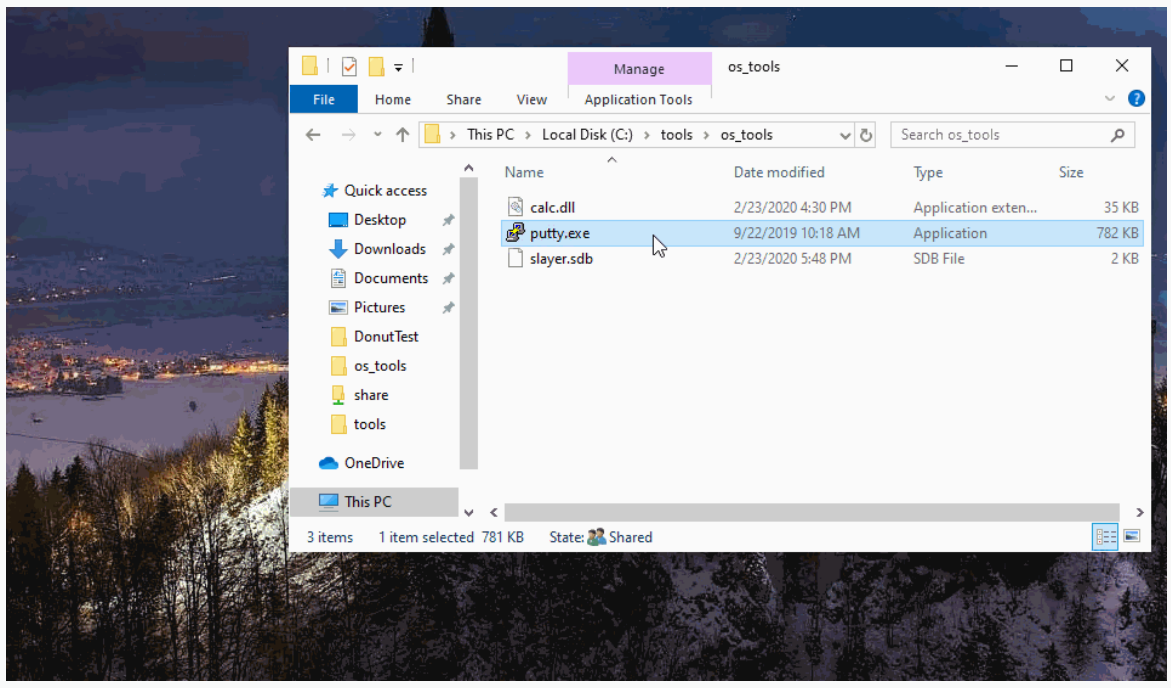
Install and Run

Once the shim is created and saved as a .sdb, you can add it using **sdbinst.exe**. The sdb, dll and exe file do not need to be in the same directory, and the sdb file can be deleted after sdbinst completes.



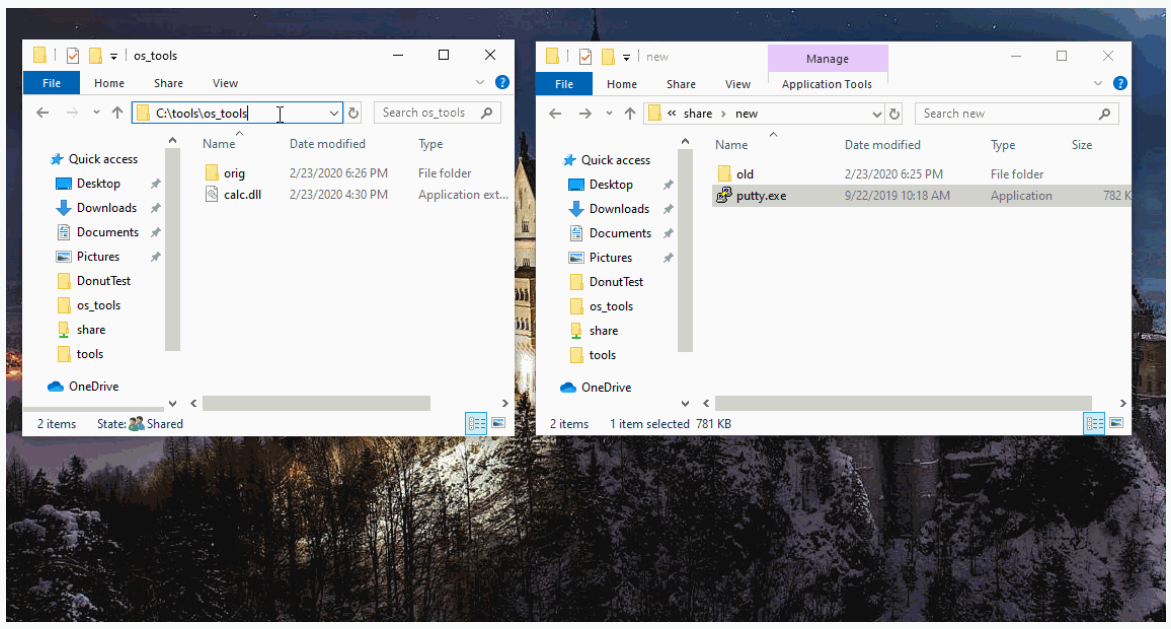
Once run, this will add an entry into **Programs and Features** (appwiz.cpl). To uninstall the shim, simply uninstall via appwiz.cpl. Otherwise the shim entry can be removed while keeping complete functionality by deleting the GUID in the uninstall reg key >:)

Now time to run...



Another interesting feature from testing is the shimmed app can be executed from a UNC path and still work. As long as the specified dll is in the **same location, and unchanged**.

In the below case, putty was already installed and shimmed on localhost, but the exe was also located on a network share - popping calc when ran from the share. Pretty neat app functionality.

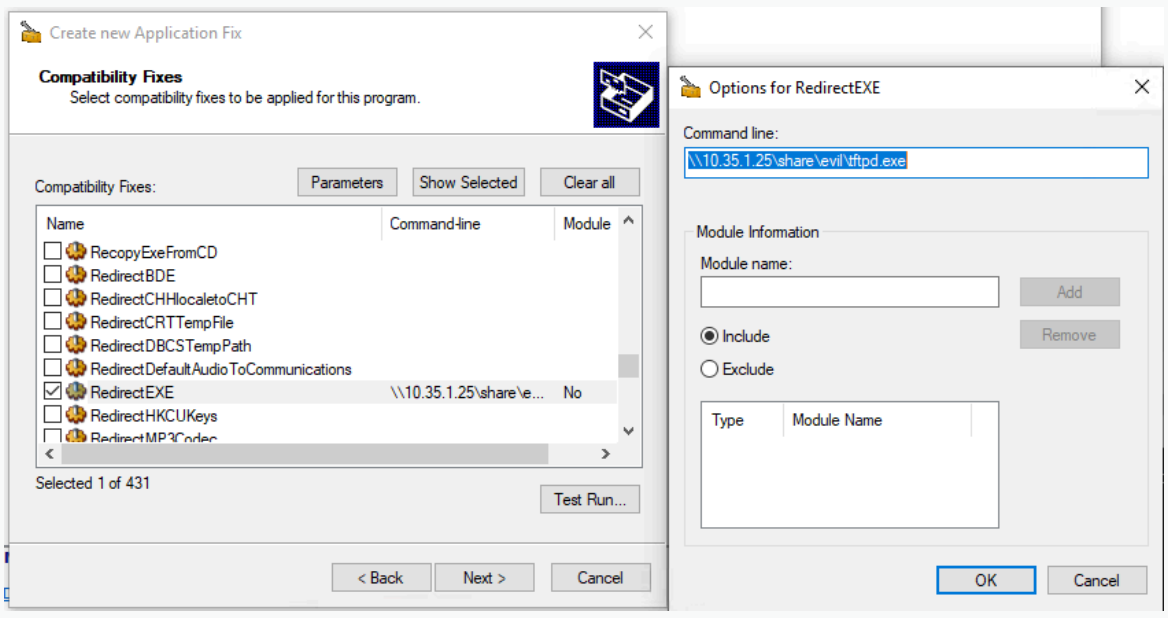


More Fun

Another Windows shim functionality with ACT is "*RedirectEXE*". This will basically execute an exe defined by the shim creator. Have chrome.exe instead open firefox.exe? Or maybe something more fun?

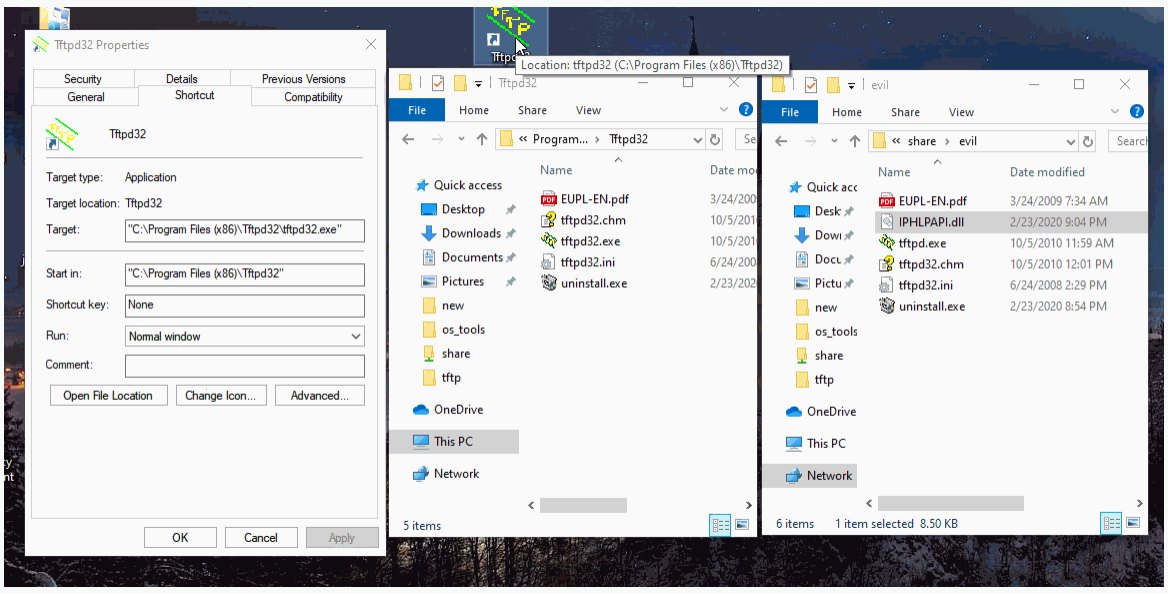
In this scenario a user has a shortcut on their Desktop to an installed app in Program Files - like many Windows workstations. The app (PE) itself will be shimmed to "*RedirectEXE*" to an evil, attacker controlled version of the same software...in a UNC network share. In this arbitrary case the attacker controlled version has a DLL Hijack which will pop calc.

First the shim is built.



So when the user clicks their usual desktop icon, a alternate "identical" version opens from the share and runs.

Once installed it'll be executed normally from the desktop .lnk. In this case the DLL hijack is using IPHLPAPI.dll - checkout my DLL Hijacking post using Ghidra on this same PE/DLL.



Much potential from application shimming.

Detection

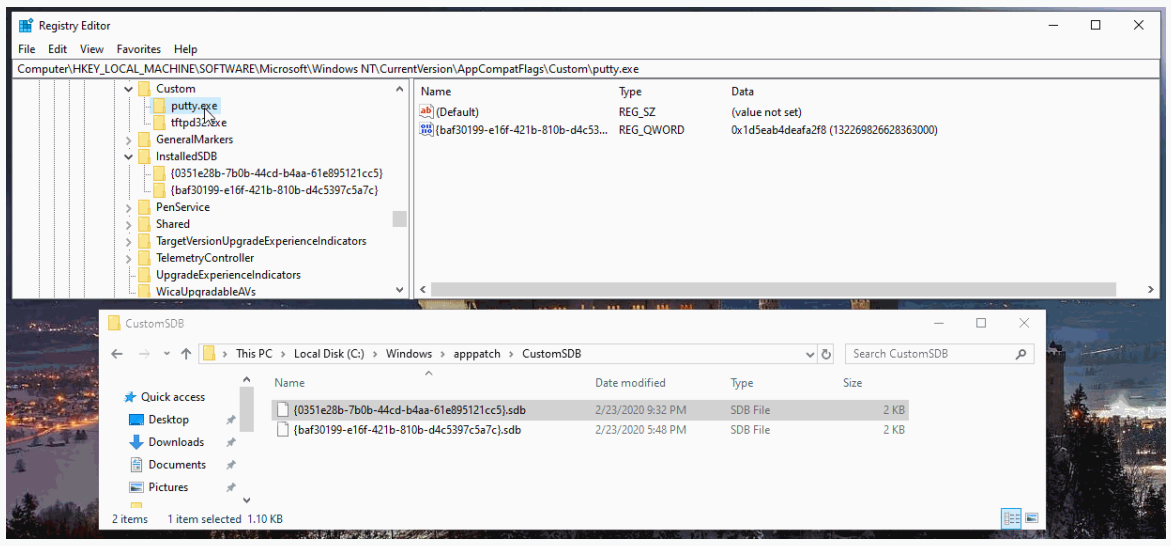
One quick way to detected shimmed apps, is to look in the registry:

```
HKLM\Software\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Custom
HKLM\Software\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Installe
```

Or checkout the custom SDB location:

```
C:\Windows\apppatch\CustomSDB
```

By default Windows 10 1903 there aren't any entries. Since they are "Custom" I'd assume they have to added manually via GPO or otherwise on all Windows versions. Andrea Fortuna has a post that dives into app shim forensics a bit more. Here's a visual of the juicy shim locations.



Further Reading

https://www.geoffchappell.com/studies/windows/win32/apphelp/sdb/shimdbc.htm

https://www.alchemistowl.org/pocorgtfo/pocorgtfo13.pdf

https://attack.mitre.org/techniques/T1138/

READ NEXT

Persistence via IFEO