Malicious Application Compatibility Shims

Table of Contents

Introduction

One of the many reasons why Microsoft has dominated the personal computer market is because they have gone to extraordinary lengths to ensure backwards compatibility in (most) of their products, and many third party products. This is currently implemented by the Application Compatibility Framework (ACF) which holds a position of considerable tactical value on modern computer systems because, by design, the ACF is capable of intercepting application API calls and manipulating the Portable Executable (PE) file loading process. Actors in the wild have been observed leveraging a few of the simple components of the ACF and it is expected that the number of exploitations will rise and grow in sophistication. Some of these more advanced malicious techniques will be demonstrated and I will be releasing the source code of several new tools that help mitigate this new threat.

Background

Raymond Chen is a developer on the Windows Team at Microsoft and maintains a blog called 'The Old New Thing' in which he describes many fascinating stories and deep technical insights into the history of Windows. On August 24th 2005 he wrote:

*"During the development of Windows 95, application compatibility was of course a very high priority. To make sure that coverage was as broad as possible, the development manager for Windows 95 took his pick-up truck, drove down to the local Egghead Software store (back when Egghead still existed), and bought **one copy of every single PC program in the store**... In exchange for taking responsibility for ensuring that Windows 95 was compatible with your adopted programs, you [the developer] got to keep them [the applications] after Windows 95 shipped."* (Chen, 2005)

In any given Windows 7 machine one can find hard coded references to '3DFrogFrenzy', 'Acrobat5', 'WordPerfect8', 'Works99' and many other applications dating back to the early 1990's. These programs do not correctly function on modern Windows machines, so in an effort to compensate for poor coding (on all sides), Microsoft has sacrificed security for developer convenience by allowing anyone to create and install Shim database (sdb) files. These database files contain the specific details on how Windows should manipulate (in other words 'shim') a target program with predefined 'Fixes'. The Application Compatibility Toolkit (ACT) contains hundreds of configurations for thousands of programs (like those listed above) primarily contained within the file %WINDIR%\AppPatch\sysmain.sdb. Microsoft provides a free tool called the Application Compatibility Administrator which allows users to create and apply specific Fixes such as 'DisableNX', 'ModifyShellLinkPath', 'VirtualRegistry', 'DisableAdvancedRPCClientHardening', 'ForceAdminAccess', 'InjectDll' (my favorite), 'DisableSeh', 'ShellExecuteXP' and many others.
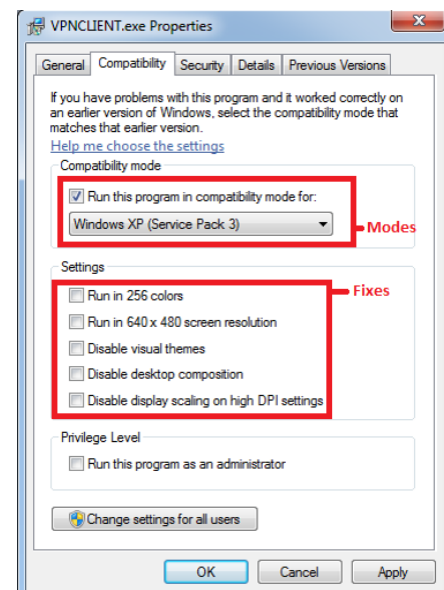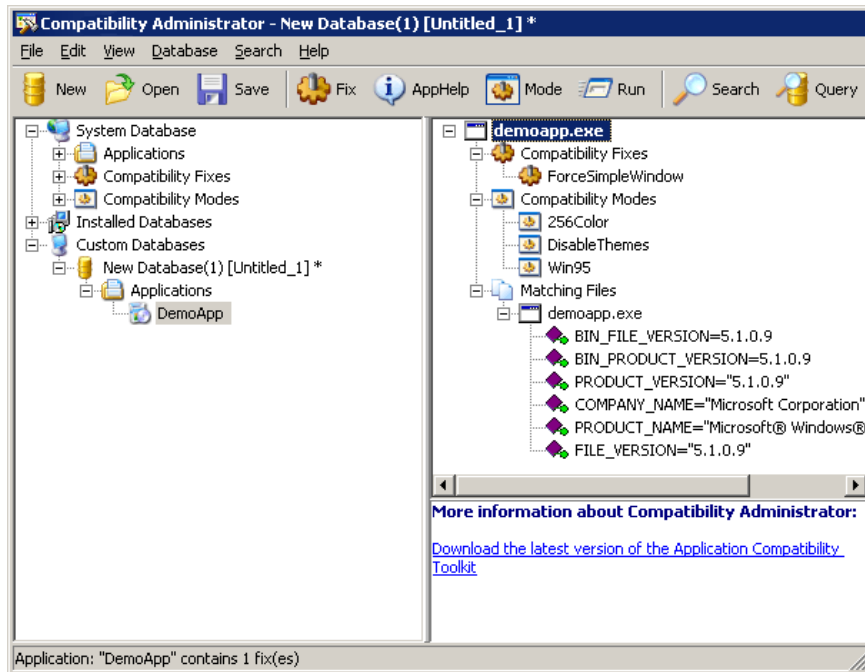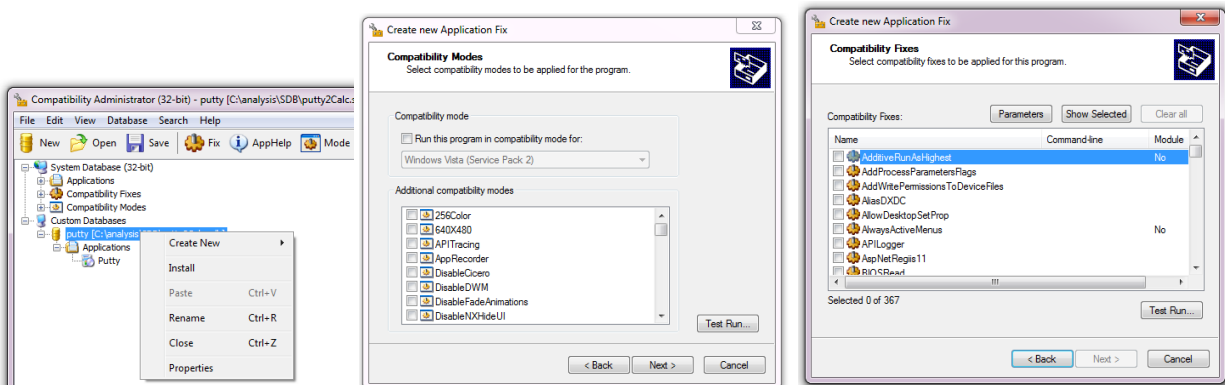
Figure 1. The Application Compatibility Toolkit (ACT) – Compatibility Administrator

The Application Compatibility Toolkit can be used to create and install a shim by guiding the user through a Wizard which first asks for a path to the target executable. The ACT will then analyze the file for criteria to filter on such as file name, file size, link date, PE checksum, etc. Then the user will be presented with a list of 'Modes' where each 'Mode' is effectively a bundle of individual Fixes that can be applied to a process and its children (A 'Mode' or 'Compatibility Mode' are sometimes referred to as a 'Layer'). An example of a Mode would be 'VistaSP1VersionLie' as opposed to a Fix such as, 'Emulate Cursor'. After the Mode screen, the wizard will show a list of Fixes. By default the 32-bit Compatibility Administrator will show a list of 367 Fixes, however if the Compatibility Administrator is executed from the command line with the '/x' parameter the list of Fixes will increase to 811 to include undocumented Fixes. After finishing the wizard, a user can save the sdb file, and then install it by right clicking on the database icon and clicking 'Install'. The installer simply executes the command line sdb installer called sdbinst.exe.

The Application Compatibility Toolkit (ACT) – Compatibility Administrator

The installer will create a GUID, copy the sdb file into %SystemRoot%\AppPatch\Custom\<GUID>.sdb then add a registry key using an internal database name in the format of HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Custom\<GUID>.sdb and \InstalledSDB\<GUID>.sdb. If everything is successful, the installer will add an entry to the Add/Remove programs list. Obviously if a user has administrative access, they could simply add the keys to the registry directly.

Registry keys:

- HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Custom
- HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\InstalledSDB
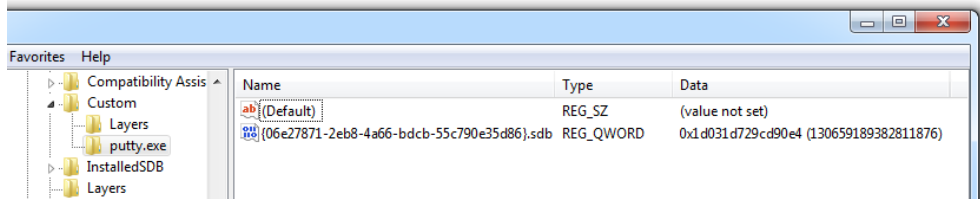
Figure 2 Install Registry Keys (1)
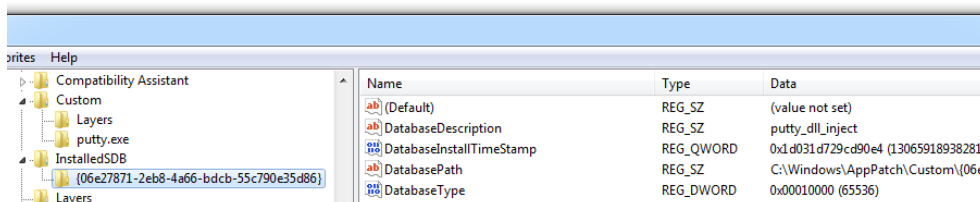
Figure 3. Install Registry Keys (2)

Default Install File Locations

- %WINDIR%\AppPatch\Custom\
- %WINDIR%\AppPatch\Custom\Custom64\
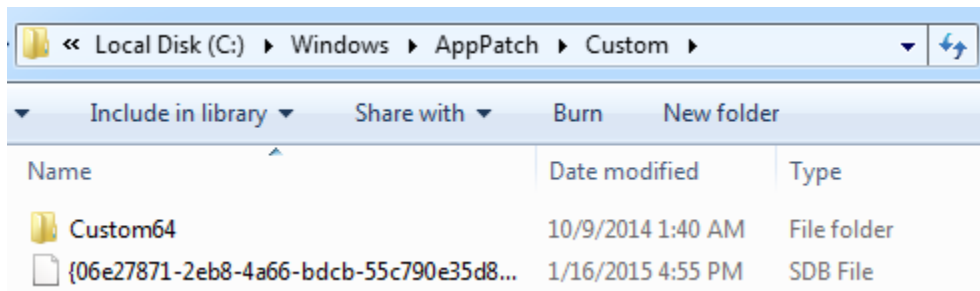
Figure 4 - Default Install Path. Note: The sdb file can be put anywhere accessible. Even a UNC path

After the sdb file installation is complete all processes launched after that point will be subjected to the file matching rules of this shim database. For efficiency reasons, Windows caches metadata for each executed file and stores that data in the Application Compatibility Cache. In effect, the cache is a record of all executed processes since the computer was last rebooted. The forensic value from this data is used by some tools such as Mandiant's

Shim Cache Parser ® and was the subject of the MS15-001 elevation of privilege vulnerability in January 2015.

Seven months prior to the MS15-001 patch, a forensics expert named Chris Graham published a blog post titled, 'Shimming Your way Past UAC'. In this entry he provided a detailed walk through (including source code) on how to generate and install an sdb file that would use the 'RedirectEXE' Fix to redirect the execution (while retaining the privileges) of a target executable (Graham, 2014). The processes that are targeted by this method rely on the fact that these executables are signed by Microsoft and have a manifest file property of `<autoElevate>true</autoElevate>` such as SndVol.exe. The MS15-001 did nothing to prevent this UAC bypass technique and as a result it is still being exploited in the wild. However, in April 2015 Microsoft did push out an optional update (KB3045645) which removed the auto-elevating manifest property from sdbinst.exe (the system sdb installer). More details can be found in the Defenses section.

Shims are heavily integrated into Windows for normal operations, such as auto elevating any executable matching the filename *instal*. Microsoft also uses shims to test and fix their own products such as Outlook and Word rather than updating/patching them the traditional way. When evaluating compatibility issues with Windows Vista, the ACT was used to collect data via shimming exeplorer.exe and other system processes (Hancke, 2009). Primarily Microsoft uses shims to implement most of the functionality in products such as the Enhanced Mitigation Experience Toolkit (EMET) and Windows Fixit patches which are used to mitigate critical vulnerabilities as a stopgap measure between patch cycles.

Shim strengths from an attacker's point of view are:
- Nearly every process is vulnerable. The shim engine (shimeng.dll) will not shim certain hard coded modules such as:
  - NT Symbolic Debugger (NTSD)
  - WinDbg
  - Software License Service (slsvc.exe) for Windows Vista (Ionescu, 2007)
- The shim engine (shimeng.dll) will intercept GetProcAddress() in the event an application attempts to dynamically call a function that the shim engine has manipulated.
- There are many ways to apply native fixes maliciously.
- There are little to no defensive measures from malicious shims (until now).
- They are not considered executable even though they can contain shellcode.
- Custom fixes can be defined in the form of a user supplied DLL file. This is useful because the 'InjectDll' is not a predefined Fix for 64-bit shims.
- With a few exceptions, all modern Windows OS versions support shims and natively ship with the auto-elevated shim database installer sdbinst.exe. The native installer will add an entry in the Add/Remove programs list but this can be bypassed by directly adding to the appropriate keys.

Shim weaknesses from an attacker's pointer of view:
- An attacker needs administrative rights to execute the shim database installer and/or write to the necessary registry keys.

- Programs only need to change their names to avoid being shimmed. There are no currently available tools that allow wildcards in file names, but the functionality does exist as seen in the native shim databases such as sysmain.sdb.
- Shims only run in user-mode so they cannot manipulate kernel-mode modules.

How the Windows Image Loader works with the Shim Engine

When a new process needs to be created, a program will use one of the functions in ntdll.dll, such as CreateProcess(). To determine if the new process needs to be shimmed, apphelp.dll is loaded and two registry keys might be opened or queried:
- `HKCM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Layers`
- `HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Layers`

If the application needs to be shimmed an undocumented __ COMPAT_LAYER environment variable will be set in the new child process. The kernel creates the various data structures, sets up the process space then loads ntdll.dll into that new process space. In that space, the kernel will set the context struct to begin executing the ntdll.dll function called LdrInitializedThunk(). This function effectively sets up everything the process will need such as:
- Initialize the heap
- Maps and initializes the appropriate DLL's
- Initializes Tread Local Storage (TLS)
- Fiber Local Storage (FLS)
- Import Address Tables (IAT's)
- Export Address Tables (EAT's)
- Process the Manifest files (Which might also have compatibility settings)
- Read the application compatibility database and loading the shim engine

After all the initialization is done, the kernel will reset the threat's context to the ntdll.dll function RtlUserThreatStart() which will in turn begin executing the actual original entry point for the executable that was specified with CreateProcess(). If the child process needs to be shimmed, both the parent and child processes will load apphelp.dll and verify the sdb file:

LaunchWithoutShim.exe	CreateFileMapping	C:\Users\Sean Pierce\Documents\Visual Studio 2013\Projects\LaunchWithoutShim\x64\Debug\HelloWorldShimmed.exe
LaunchWithoutShim.exe	RegOpenKey	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\HelloWorldShimmed.exe
LaunchWithoutShim.exe	QueryNameInformationFile	C:\Users\Sean Pierce\Documents\Visual Studio 2013\Projects\LaunchWithoutShim\x64\Debug\HelloWorldShimmed.exe
LaunchWithoutShim.exe	CreateFileMapping	C:\Windows\AppPatch\sysmain.sdb
LaunchWithoutShim.exe	RegOpenKey	HKLM\Software\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Layers
LaunchWithoutShim.exe	RegOpenKey	HKCU\Software\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Layers
LaunchWithoutShim.exe	RegQueryValue	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\InstalledSDB\{5e8564b1-053c-485a-bda2-420947aba6c7}\DatabasePath
LaunchWithoutShim.exe	CreateFileMapping	C:\Windows\AppPatch\Custom\{5e8564b1-053c-485a-bda2-420947aba6c7}.sdb
LaunchWithoutShim.exe	QueryDirectory	C:\Users\Sean Pierce\Documents\Visual Studio 2013\Projects\LaunchWithoutShim\x64\Debug
LaunchWithoutShim.exe	RegQueryValue	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\{ca37ff05-c61e-4e3f-bdaf-7f3a4cf31661}
LaunchWithoutShim.exe	RegQueryValue	HKCU\Software\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\{ca37ff05-c61e-4e3f-bdaf-7f3a4cf31661}
LaunchWithoutShim.exe	CreateFile	C:\Users\Sean Pierce\Documents\Visual Studio 2013\Projects\LaunchWithoutShim\x64\Debug\ui\SwDRM.dll
LaunchWithoutShim.exe	RegQueryValue	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\SideBySide\PreferExternalManifest
LaunchWithoutShim.exe	CreateFileMapping	C:\Windows\AppPatch\sysmain.sdb
LaunchWithoutShim.exe	CreateFileMapping	C:\Windows\AppPatch\sysmain.sdb
LaunchWithoutShim.exe	RegQueryValue	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\InstalledSDB\{5e8564b1-053c-485a-bda2-420947aba6c7}\DatabasePath
LaunchWithoutShim.exe	CreateFileMapping	C:\Windows\AppPatch\Custom\{5e8564b1-053c-485a-bda2-420947aba6c7}.sdb
HelloWorldShimmed.exe	CreateFile	C:\Windows\Prefetch\HELLOWORLDSHIMMED.EXE-995A7E9E.pf
HelloWorldShimmed.exe	RegQueryValue	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\DisableUserModeCallbackFilter
HelloWorldShimmed.exe	RegOpenKey	HKLM\System\CurrentControlSet\Control\Session Manager
HelloWorldShimmed.exe	RegQueryValue	HKLM\System\CurrentControlSet\Control\SESSION MANAGER\CWDIllegalInDLLSearch
HelloWorldShimmed.exe	CreateFile	C:\Users\Sean Pierce\Documents\Visual Studio 2013\Projects\LaunchWithoutShim\x64\Debug\MSVCR120.dll
HelloWorldShimmed.exe	CreateFileMapping	C:\Windows\SysWOW64\msvcr120.dll
HelloWorldShimmed.exe	CreateFileMapping	C:\Windows\SysWOW64\apphelp.dll
HelloWorldShimmed.exe	CreateFileMapping	C:\Windows\AppPatch\sysmain.sdb
HelloWorldShimmed.exe	RegQueryValue	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\InstalledSDB\{5e8564b1-053c-485a-bda2-420947aba6c7}\DatabasePath
HelloWorldShimmed.exe	CreateFileMapping	C:\Windows\AppPatch\Custom\{5e8564b1-053c-485a-bda2-420947aba6c7}.sdb

Figure 5 Confusing procmon log output of a program named 'LaunchWithoutShim.exe' executing a program called, 'HelloWorldShimmed.exe'. Procmon is a part of SysInternals by Mark Russinovich

If the sdb file cannot be found or opened the Application Event Log will record, 'Unable to open shim database version registry key'. Assuming the sdb file is found and loaded without error, it is then parsed to further verify that the file should indeed be shimmed by checking the sdb file for more name filters and other file criteria; there is always at least one filter, but by default it is a match-all wildcard character: *. The loaded sdb file will be further parsed to find the specific Fix (or Fixes) that will be applied along with a reference to a DLL where that Fix is implemented. The specified DLL is then loaded into the suspended process, and allowed to execute DllMain() following the initialization of the other system DLLs. If the Fix does not reference a DLL then the native system shim database(s) are loaded in order find the appropriate DLL where that particular fix is implemented. Most of the predefined Fixes are found in %WINDIR%\AppPatch\sysmain.sdb and most of those fixes are implemented in %WINDIR%\AppPatch\AcGeneral.dll.

If the Fix is not predefined in sysmain.sdb it is called a Custom Fix. Anyone can create a custom fix by creating a DLL which exports the following functions:

- `GetHookAPIs(char *,ushort *,ulong *)`
- `NotifyShims(char *, unsigned __int16 *, unsigned __int32 *)`

The DLL containing the Fix is typically hooks the IAT of the target to intercept specific functions. For example the VirtualRegistry Fix will hook all registry related calls and manipulate them accordingly. GetProcAddress() is also hooked to anticipate dynamic calls to the would-be hooked functions. Sdb files have undocumented features such as hot patching which is currently only used by Microsoft to implement the functionality in Fixit patches (discussed later). After ntdll.dll loads the AppInit DLLs in the child process, it then loads the shim engine which implemented with apphelp.dll. This shim engine will clear the environment variable __COMPAT_LAYER then apply the Fixes and patches (if any) to the target module. It is possible to have more than one sdb apply Fixes to the same process but this leads to undefined behavior due to the uncertain order in which the shims are applied.

Prior Work

The earliest mention of shims relating to security was in 2007 when a software engineer named Alex Ionescu started to publish a number of blog posts titled, 'Secrets of the Application Compatilibity Database (SDB)'. This was planned to be a 7 part series but unfortunately Ionescu stopped after the fourth post. In the years preceding and following, a small number of disparate bloggers posted various technical details and mentions of shims, but typically with the focus of application compatibility. Below is a general timeline of major shim references:

1. June 2005. A Microsoft Employee named Chris Jackson's started an MSDN blog called, 'The App Compat Guy'. Around January 2007 he began posting information about the ACT and blogging about shims.
   http://blogs.msdn.com/b/cjacks/
2. May 2007. Ionescu 'Secrets of the Application Compatilibity Database (SDB)'
   www.alex-ionescu.com/?p=39

3. August 2008. A blogger with the handle 'wedday' posted some technical details about how the shim engine worked based on the shim engine's debug output. http://wedday.blogspot.co.uk/2008/08/shimeng.html
4. July 2009. Another Microsoft Employee named 'Maarten van de Bospoort' wrote a blog post titled 'Disabling a Shim'. However his solution was to disable the shim engine via group policy which is not advised because Windows relies on the shim engine for EMET and Fixit patches (discussed later). (Bospoort, 2009) http://blogs.msdn.com/b/maartenb/archive/2009/07/24/disabling-a-shim.aspx
5. February 2010. A developer who goes by the name, 'Jochen Kalmbach' posted a tool that will display the shims that would be activated by a particular file. http://blog.kalmbach-software.de/2010/02/22/the-shim-database/
6. April 2012. A company named 'Recx' posted a compilation of research notes titled 'Windows AppCompat Research Notes' where deep technical insight and control flow of the shim engine was discussed. (Ollie, 2012) http://recxltd.blogspot.co.uk/2012/04/windows-appcompat-research-notes-part-1.html
7. May 2012. Microsoft updated its Application Compatibility Toolkit (ACT) Technical Reference where background information is given on the ACT along with details of individual fixes. This article also links to the AppHelp.dll documentation. https://technet.microsoft.com/en-us/library/hh825181.aspx
8. October 2013. Mark Baggett gave a talk at DerbyCon titled, 'Windows – Own3d by Default'. He is the first to discuss the use of shims in a post exploitation context. http://www.irongeek.com/i.php?page=videos/derbycon3/4206-windows-0wn3d-by-default-mark-baggett
9. March 2014. Jon Erickson presented at Black Hat Asia 'Persist-It – Using and Abusing Microsoft Fix It Patches' where he describes how Fixit patches work and how his tool can be used to analyze them. (Erickson, 2014) https://www.blackhat.com/docs/asia-14/materials/Erickson/Asia-14-Erickson-Persist-It-Using-And-Abusing-Microsofts-Fix-It-Patches.pdf https://www.blackhat.com/docs/asia-14/materials/Erickson/WP-Asia-14-Erickson-Persist-It-Using-And-Abusing-Microsofts-Fix-It-Patches.pdf
10. May 2014. Graham Posts 'Shimming your Way Past UAC' (Graham, 2014). This provided the first public source code reference for how the ACT can be used to bypass a UAC prompt. http://blog.ddifrontline.com/4#more-4
11. December 2014. Jon Erickson presents 'The active use and exploitation of Microsoft's Application Compatibility Framework' at SysScan 360 and CodeBlue in Japan. http://sdb.io/erickson-codeblue.pdf
12. Jan 2015. Sean Pierce (me) Shmoocon Epilogue "Forensic Analysis of Advanced Persistence through the Application Compatibility Toolkit"

In the wild

The frame of reference for leveraging malicious shims is centered around post exploitation where it is assumed the attacker has administrative privileges in order to write the necessary registry keys to install shims on a system wide basis. Listed below are numerous examples of malicious shims being used in cyber-espionage and cyber-crime activities. While the ACT offers many avenues to perform arbitrary code execution, attackers routinely use only a few of the basic Fixes:

BlackEnergy: There are variants of this malware that have been used in cyber-crime & cyber-espionage and has been under continuous development since June 2007. Modern variations support components such as rootkits and 64-bit plugins. In April 2013 the authors responsible for BlackEnergy added the UAC bypass 'RedirectEXE' Fix making BlackEnergy the first publically known malicious use of the ACT. They also used the 'DisableNXShowUI' Fix to disable Data Execution Prevention (DEP) in its host process. (F-Secure, 2015) (Aquilino, 2014) (Cherepanov R. L., 2014)
BlackEnergy 2 hash: 26b9816b3f9e2f350cc92ef4c30a097c6fec7798
BlackEnergy 3 hash: 78636f7bbd52ea80d79b4e2a7882403092bbb02d

GootKit: This is a backdoor that was initially observed in April of 2014 and is purportedly created by the actor 'MzH' (hFireF0X, 2015). It used the UAC bypassing 'RedirectEXE' Fix to target the auto-elevated 'cliconfg.exe' (SQL Server Client Network Utility). In March 2015, the malware author(s) leveraged the in-memory patching as discussed by Jon Erickson at his talks at BlackHat Asia in August 2014. The authors of Gootkit most likely used Erickson's publically available sample code to generate the sdb files (23 in all), but for unknown reasons the Gootkit authors did not use Erickson's stealth installation technique (they still use the traditional native sdb installer, sdbinst.exe). (Pernet, 2015) (Dr. WEB, 2014) (Erickson, 2014)
GootKit: f9772fcfbcaac9c4873989a1759a5c654eec440e

Roaming Tiger: This is a cyber-espionage group outed by Eset's Anton Cherepanov at ZeroNights 2014. Cherepanov observed Roaming Tiger targeting countries belonging to the Russia and Commonwealth of Independent States (CIS) with phishing documents to get initial code execution which then used the ACT UAC bypass technique and then installed ACT enabled persistence. What makes this case unique is that Roaming Tiger registered specially crafted 32-bit and 64-bit shim databases for 'explorer.exe' and two other processes. These sdb files described custom Fixes named 'AcProtect_Apps' and specified that those Fixes were contained within the library 'AcProtect.dll' as seen in Figure 6. As a result the AcProtect.dll was injected into every target process that matched the sdb criteria. (Cherepanov A. , 2014)

```
44e TAG 7001 - DATABASE
      454 TAG 4023 - OS_PLATFORM: 1 (0x1)
      45a TAG 6001 - NAME: AcProtect_Database
      460 TAG 9007 - DATABASE_ID: {F8C4CC07-6DC4-418F-B72B-304FCDB64052} NON-S
      476 TAG 7002 - LIBRARY
      47c TAG 7004 - SHIM
              482 TAG 6001 - NAME: AcProtect_Shim
              488 TAG 600a - DLLFILE: AcProtect.dll
      48e TAG 7007 - EXE
              494 TAG 6001 - NAME: splwow64.exe
              49a TAG 6006 - APP_NAME: AcProtect_Apps
              4a0 TAG 9004 - EXE_ID: {1DAC33EB-986E-4BC5-B7D8-CB9B0B7F6555}
              4b6 TAG 7008 - MATCHING_FILE
                      4bc TAG 6001 - NAME: *
              4c2 TAG 7009 - SHIM_REF
                      4c8 TAG 6001 - NAME: AcProtect_Shim
                      4ce TAG 4004 - SHIM TAGID: 1148 (0x47c)
      4d4 TAG 7007 - EXE
              4da TAG 6001 - NAME: explorer.exe
              4e0 TAG 6006 - APP_NAME: AcProtect_Apps
              4e6 TAG 9004 - EXE_ID: {D9B74E19-6919-4C67-8DE8-3D64B72F9CFA}
              4fc TAG 7008 - MATCHING_FILE
                      502 TAG 6001 - NAME: *
              508 TAG 7009 - SHIM_REF
                      50e TAG 6001 - NAME: AcProtect_Shim
                      514 TAG 4004 - SHIM TAGID: 1148 (0x47c)
```

Figure 6 sdb-explorer showing the shim (Fix) name and DLL reference (bottom two boxes), and the DLL file reference (top)

Search Protect by Conduit: This has been classified by multiple anti-virus venders as a Potentially Unwanted Program (PUP) starting in August 2013. In January of 2015 Search Protect was observed using the ACT for targeted persistence by creating a 32-bit shim layer named 'VC32Ldr♦☻' which applies the 'InjectDll' Fix to 'chrome.exe', 'firefox.exe', 'iexplorer.exe', 'software_remove_tool.exe', and 'software_reporter_tool.exe'. For the 64-bit processes, a custom Fix named, 'VC64Shim' was defined as being in the 'VCLdr64.dll'; similar to Roaming Tiger. This resulted in that DLL being injected into 'iexplore.exe', 'chrome.exe', and 'explorer.zza'. The custom fix was most likely used because there is no predefined 'InjectDll' Fix for 64-bit processes.

By using the sdb-explorer.exe tool developed by Jon Erickson of iSIGHT Partners, the output below will show that the Fix injected \\.\globalroot\systemroot\apppatch\nbin\vc32loader.dll into every process that matched one of the target filenames listed above.
Search Protect shim: 175abbeaf62f08716f7975190e550b95

```
15c TAG 7001 - DATABASE
    162 TAG 5001 - TIME
    16c TAG 6022 - COMPILER_VERSION: 2.1.0.3
    172 TAG 6001 - NAME: Apps32
    178 TAG 4023 - OS_PLATFORM: 1 (0x1)
    17e TAG 9007 - DATABASE_ID: {8A4D5A43-C64A-45AB-BDF4-804FE18CEAFD} NON-STANDARD
    194 TAG 7002 - LIBRARY
    19a TAG 700b - LAYER
            1a0 TAG 6001 - NAME: UC32Ldr◆▯
            1a6 TAG 7009 - SHIM_REF
                    1ac TAG 6001 - NAME: InjectDll
                    1b2 TAG 6008 - COMMAND_LINE: \\.\globalroot\systemroot\apppatch\nbin\vc32loader.dll
    1b8 TAG 7007 - EXE
            1be TAG 6001 - NAME: chrome.exe
            1c4 TAG 6006 - APP_NAME: ch
            1ca TAG 6005 - VENDOR: <Unknown>
            1d0 TAG 9004 - EXE_ID: {F0E1DBDD-6215-4E9D-9FA6-D0A24D6DAEC3}
            1e6 TAG 7008 - MATCHING_FILE
                    1ec TAG 6001 - NAME: *
            1f2 TAG 700b - LAYER
                    1f8 TAG 6001 - NAME: UC32Ldr◆▯
                    1fe TAG 401a - LAYER_TAGID
    204 TAG 7007 - EXE
            20a TAG 6001 - NAME: explorer.xxx
            210 TAG 6006 - APP_NAME: ex
            216 TAG 6005 - VENDOR: <Unknown>
            21c TAG 9004 - EXE_ID: {185E4C39-8492-41A9-8ABD-28E83D921E4A}
            232 TAG 7008 - MATCHING_FILE
                    238 TAG 6001 - NAME: *
            23e TAG 700b - LAYER
                    244 TAG 6001 - NAME: UC32Ldr◆▯
                    24a TAG 401a - LAYER_TAGID
380 TAG 7801 - STRINGTABLE
    386 TAG 8801 - STRINGTABLE_ITEM: 2.1.0.3
    39c TAG 8801 - STRINGTABLE_ITEM: Apps32
    3b0 TAG 8801 - STRINGTABLE_ITEM: UC32Ldr◆▯
    3ca TAG 8801 - STRINGTABLE_ITEM: InjectDll
    3e4 TAG 8801 - STRINGTABLE_ITEM: \\.\globalroot\systemroot\apppatch\nbin\vc32loader.dll
    45e TAG 8801 - STRINGTABLE_ITEM: chrome.exe
    47a TAG 8801 - STRINGTABLE_ITEM: ch
    486 TAG 8801 - STRINGTABLE_ITEM: <Unknown>
    4a0 TAG 8801 - STRINGTABLE_ITEM: *
    4aa TAG 8801 - STRINGTABLE_ITEM: explorer.xxx
    4ca TAG 8801 - STRINGTABLE_ITEM: ex
    4d6 TAG 8801 - STRINGTABLE_ITEM: firefox.exe
    4f4 TAG 8801 - STRINGTABLE_ITEM: ff
    500 TAG 8801 - STRINGTABLE_ITEM: iexplore.exe
    520 TAG 8801 - STRINGTABLE_ITEM: ie
    52c TAG 8801 - STRINGTABLE_ITEM: software_removal_tool.exe
    566 TAG 8801 - STRINGTABLE_ITEM: sr
    572 TAG 8801 - STRINGTABLE_ITEM: software_reporter_tool.exe
    5ae TAG 8801 - STRINGTABLE_ITEM: sr2
```

Figure 7 sdb-explorer output for Search Protect shim

Upatre: This was first observed in August 2013 (msft-mmpc, 2013) and acts as a downloader for other malware such as Zbot, Dyre, and Dridex. As early as February 2015 Upatre started installing other malware with administrative privileges by using Graham's UAC bypass technique (Inocencio, 2014) (Author, 2015).
Upatre hash: 1d38c362198ad67329fdf58b4743165e

DRIDEX: This is the successor to CRIDEX, both of which focus on stealing personal and banking information. This latest variant was upgraded to use webinjects, more evasion techniques and relies on spam or Upatre for delivery. At the same time Upatre implemented Graham's code, Dridex also started using it but in a slightly different manner: Unlike Upatre, Dridex targeted 'iscsicli.exe' and used the 'RedirectEXE' Fix to execute a batch script to simply open up a port in the Window's Firewall. After that, the batch script uninstalled the shim and destroyed all evidence of there being a shim (Drapeau, 2015). This is most likely due to the fact that Graham's technique used the native 'sdbinst.exe' installer, which adds an entry into the add/remove programs' list. The commands used are:

```
"C:\Windows\system32\sdbinst.exe" /q
"C:\Users\%USERNAME%\AppData\Local\Temp\\..\..\LocalLow\com.%USERNAME%.
sdb"
```

The batch script:
```
start C:\Users\user_name\AppData\Local\edg3FAC.exe
C:\Users\user_name\Desktop\malware.exe
sdbinst.exe /q /u "C:\Users\user_name\AppData\LocalLow\$$$.sdb"
reg delete "HKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\AppCompatFlags\Custom\iscsicli.exe" /f
reg delete "HKLM\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\AppCompatFlags\InstalledSDB\{f48a0c57-7c48-461c-9957-
ab255ddc986e}" /f
del C:\Windows\AppPatch\Custom\{f48a0c57-7c48-461c-9957-
ab255ddc986e}.sdb
del %LOCALAPPDATA%Low\$$$.sdb
del %LOCALAPPDATA%Low\$$$.bat
```

Dridex hash: 2dcf82bfc86397969059e118f3feab10

Timeline of ACT usage in malware
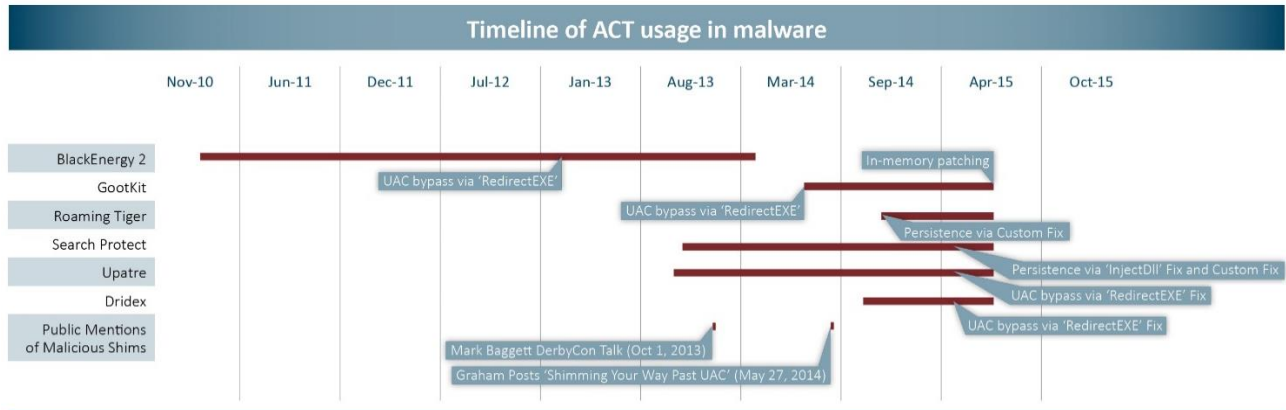
Other Offensive Uses

Attackers have been observed using the ACT predominantly as a UAC bypass, as a persistence mechanism, and only recently has one family (GootKit) used the in-memory patching capability. However there are still over 800 more 'Fixes' and 'Modes' that can be used for an array of offensive purposes, the examples listed below will be demonstrated during the talk. Incident Responders and Malware Analysts need to be aware of how shims can modify a process before and during execution, because malicious shims are capable of leveraging many rootkit-like functions:

- *Evasion, and System Integrity Subversion*: The autoruns.exe utility (part of the SysInternals Suite) is a tool created by Mark Russinovich which displays many of the common auto-start locations for executables in Windows. Autoruns does not detect shims as having any kind of auto-start capability and furthermore can be shimmed to hide auto-start locations such as registry keys with the 'VirtualRegistry' Fix. This type of tool subversion is not unheard of but what makes the ACT uniquely powerful is that the malicious shim could be developed in as little as 30 seconds. The same process can be reapplied for both 64-bit and 32-bit versions of regedit.exe (the native Windows registry editor).
- *Manipulating a Process with Redirections*: Fixes such as 'VirtualRegistry', 'CorrectFilePaths', 'VirtualizeDeleteFile', 'RedirectShortcut' and 'APILogger'. Depending on the application, these fixes could be used to intercept and manipulate sensitive information such as credentials, and debugging logs. The primary example being web browsers.
- *Persistence & Trojanizing*: Almost any process can be a host for malicious activity with 'InjectDll', 'LoadLibraryRedirect', 'RedirectShortcut', 'RedirectEXE', 'ShimViaEAT', 'LoadLibraryfromCWD' and much more can be accomplished with in-memory patching.
- *Denial of Service*: The "TerminateExe" Fix is the most trait forward way to prevent processes from running, however many of the fixes can be used to achieve the same goals such as 'IgnoreLoadLibrary', 'RedirectEXE', and 'DeprecatedServiceShim'. The last shim will allow the denial or redirection of any service that the process was attempting to start.
- *Disabling Defenses*: Shims can also incorporate simple Fixes such as 'DisableWindowsDefender', 'DisableASLR', 'DisableAdvancedRPCClientHardening', 'DisableSeh', and 'DisableNX' which would be useful when executing malware or attempting to exploit a vulnerability. This is especially powerful when one considers that vulnerabilities like MS08-067 can simply be restored as Mark Baggett mentioned in his 2013 DerbyCon talk (Baggett, 2013).
- *Malware Obfuscation*: As a malware analyst this section has been divided into three categories based on the target/host executable:
    - Benign Executables – The shimmed executable can be a generic program that has been signed and/or whitelisted. The classic 'InjectDll' and 'LoadLibraryRedirect' Fixes could be leveraged via a UNC path but patching in new code and/or utilizing existing code akin to ROP (Return Oriented Programming) chains would be a much stealthier and would be very difficult to analyze.
    - Dependently Malicious Executables – The target executable is purposefully built to work in tandem with its shim for malicious functionally. As a result the target would fail to display any malicious behaviors without the malicious shim

to interfere. A classic example would be a subtle 'kill switch' being disabled via a shim like the 'IgnoreException' Fix. Another example would be hot patching instructions to redirect program flow to malicious code existing elsewhere in the binary.

- o Obfuscated Executable – The target executable will fail completely without the shim. This can be done with code injection and/or hot patching. Such an executable could be partially or completely encrypted.

- *In-Memory Patching*. This can offer the most wide array of stealth, defense and general anti-analysis capabilities:
  - o Hooking any function
  - o Add/Remove/Edit Instructions
  - o Inserting strings/data where needed
    - ▪ Changing destination IP's/hostnames
    - ▪ Changing/Randomizing Indicators
  - o Early detection of debugging and/or other analysis tools.

One can learn more about what each documented Fix does here:
https://technet.microsoft.com/en-us/library/cc765984(v=ws.10).aspx

Tools

Current Analysis Tools

There are a number of publicly available tools that are indispensable when analyzing a suspicious sdb file. Below is an inventory of the major ones:

- *Compatibility Administrator*: The free tool by Microsoft to aid customers in creating, testing, and installing shim database files as mentioned above. It is the only tool with a graphical user interface (GUI) and uses that to display sdb files in a logical manner. However it fails to open most custom made sdb files (such as those made with sdb-explorer) and fails to display most of the technical details or even the existence of in-memory patches and custom Fixes.
  Link: http://www.microsoft.com/en-us/download/details.aspx?id=7352
- *sdbinst.exe*: The native installer that the Compatibility Administrator uses to install sdb files.
  Link: https://support.microsoft.com/en-us/kb/308235
- *sdb-explorer.exe*: Created by Jon Erickson and for his BlackHat Asia talk 'Using and Abusing Microsoft's Fix It Patches' in March 2014. Unlike the Compatibility Administrator sdb-explorer displays tremendous amounts of technical information for a given shim database including disassembling the shims patch code for x86. sdb-explorer can also create in-memory patch code and stealthily install a shim by manually adding the required registry keys. This technique bypasses the sdbinst safe guard of creating an entry in the Add/Remove Programs list.
  Link: https://github.com/evil-e/sdb-explorer
- *sdb2xml.exe*: - Created by Heath Stewart of Microsoft in November 2007 extracts some data as XML from a shim database. (Stewart, 2007)
  Link: http://blogs.msdn.com/b/heaths/archive/2007/11/02/sdb2xml.aspx

- *shims.exe*: - The only non-free application on this list. Created by David Tomczak in March 2015, this tool similarly displays information about shim database files, shim statistics, and patch information in a much prettier format than the other tools, but does not automatically disassemble the patched bytes.
Link: https://tzworks.net/prototype_page.php?proto_id=33

Defense Tools

Targeted persistence is currently not detected as a form of persistence by any analysis tools (notably SysInternals' autoruns). As a result, the code that is being released attempts to uninstall malicious shims whenever they are detected. Some tools even have built-in shim-defenses as proof of concept code for developers. The following are several examples of these tools:

Shim-Detector: Shims have total control of file redirection and Registry re-mapping for the target executable so this tool checks in-memory data structures for undocumented Application Compatibility Flags. Features:
- Will search for shim flags.
- Checks for Shim App Helper DLLs in the process space.
- Future:
    - Checks the shim engine to see if it should have been shimmed

Shim-Process-Scanner: This scanner checks the memory of every running process for Application Compatibility Flags. Features:
- Will search other processes for shim flags
- WMI event trigger on install registries
- Future:
    - Hash check VirusTotal

Shim-Guard-Service: This tools starts as a service when Windows starts up and monitors the registry for any shim installations, then promptly uninstalls them. Features:
- Runs as a service
- Detects and alerts on newly installed shims
- Uninstalls detected shims
- Future:
    - Blocks installs
    - Install a shell extension for 'No shim execute'

Shim-Guard-Lite
- Flexible WMI based script
- Alerts on newly installed shims
- Uninstalls detected shims

Shim-File-Scanner
- Scans Files/Folders for non-default shims and analyzes them
- Checks registry for installed shims
- Tests a file to see if any shim can be applied to them

- Future:
    - Shell Extension for file/folder scanner

Forensics Tools

The tools listed below focus on a forensic perspective for which I have created two plugins that will assist with both the *live* and *dead* analysis of a system afflicted by malicious shims:

shimScanner: Attempts to find evidence for active sdb files in-memory via the open source memory analysis framework Volatility. Features:
- Searches the Registry for installed sdb files
- Searches each process for shim flags
- Future:
    - Pull request for Volatility autoruns plugin
    - Add analyzing ability

sdbFileIngestModule: Is a File Ingest Module that detects and analyzes sdb files for the open source digital forensics tool Autopsy which is a graphical interface for the Sleuth Kit tools. Features:
- Searches disk image for sdb files
- Parses Registry for installed sdb files
- Uses results from sdb-explorer (windows only tool)

Penetration Testing Tools

Run Self as Admin
- Builds an shim database, installs it, bypasses UAC

Subversion-Shims
- Hides auto run keys from autoruns
- Hides auto run keys from regedit
- Prevents SysInternals tools from launching

Stealth-Shim
- Hot patching code that will conceal flags that the above tools search for
- Unload custom App Comp DLL Helper with FreeLibraryAndExitThread()

PuttyRider-Shim
- Injects the putty rider DLL into Putty

During the course of this research it became obvious that to ensure compatibility, Microsoft built Windows to effectively lie to all non-native code. Therefore 32-bit and 64-bit versions are available but will only work in their native code environment. All tools have been tested

primarily on 64-bit Windows 7 Professional and should work on x86/x64 on Windows XP, Vista, Win7, and Win 8 depending on certain *compatibility* issues (joke).

Defenses

Aside from using the given tools above, the only other technical defense from a malicious shim is by disabling the Shim Engine via Group Policy. However, that is not recommended because shims are heavily integrated into Windows for normal operations; such as auto elevating any executable matching the filename *instal*.

In late April 2015, Microsoft released an optional update named KB3045645 for Windows 7 and above which, according to their description will "force a UAC prompt when a customized .sdb file is created in Windows" (Microsoft Support , 2015). Contrary to the official description, this patch effectively removes the `<autoElevate>true</autoElevate>` property from the sdbinst.exe file and does not present a UAC prompt when a customized sdb files are created or applied to a process. This prevents the traditional UAC bypass method used by GootKit and other malware families. However the update is classified as Optional so it will not install by default and sdbinst.exe was merely the official installer for sdb files, so a UAC bypass is still possible by manually adding the necessary registry keys.
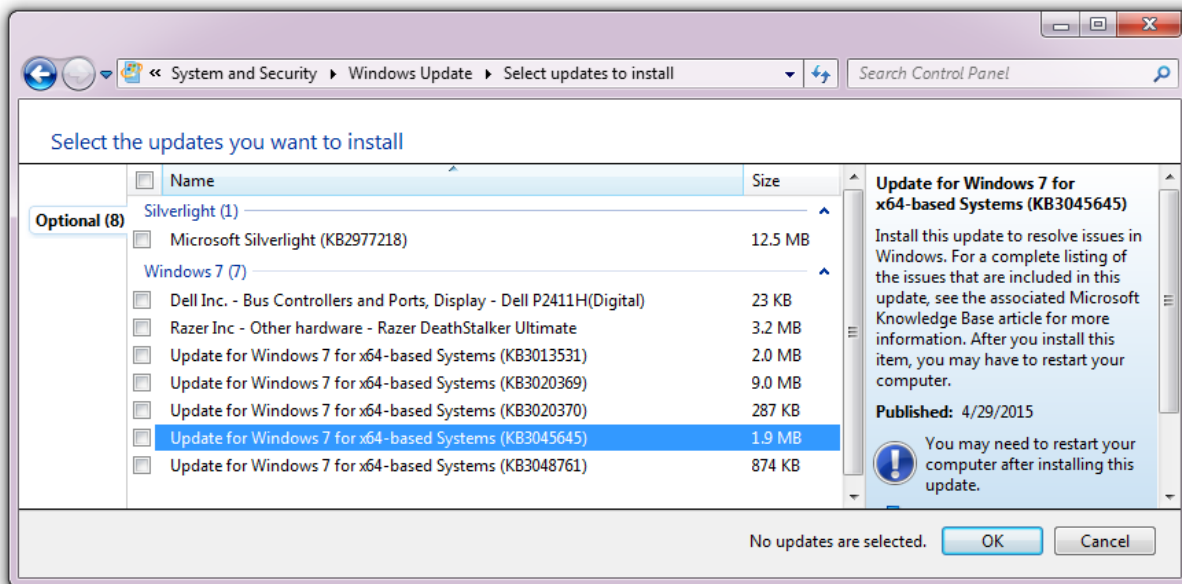

Figure 8. Optional Patch which will disable sdbinst.exe auto-elevation

Prior to KB3045645 malware relied on sdbinst.exe auto-elevating to install a 'RedirectEXE' fix in order to bypass a UAC prompt. Barring the use of that patch, some system administrators may choose to increase the UAC security setting to "Always notify" but this will of course annoy users into allowing everything especially over long periods of time. Note: one particularly annoying part of Dridex is that the UAC prompt will appear in an infinite loop until the user clicks 'Yes'.
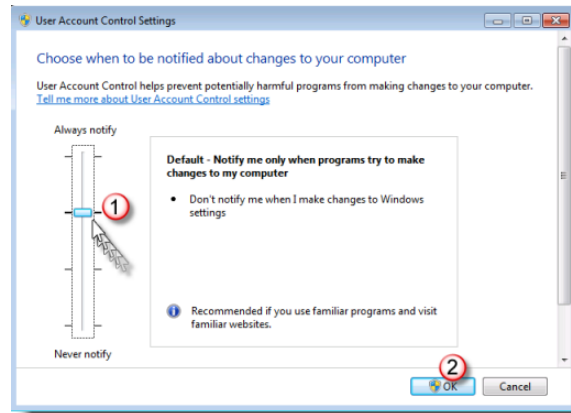
Figure 9 More details see http://support.microsoft.com/en-us/kb/975787

Moving Forward

I would recommend that Microsoft treat sdb files as they do any other executable file and enable and enforce code signing. I also recommend Windows Defender and other security software vendors to scan sdb files for suspicious attributes such as, 'RedirectEXE' for auto-elevated apps, 'TerminateExe' for security apps, and 'InjectDll' for any app. Currently only the security venders GData and Kaspersky appear to be aware of malicious shims and will sometimes detect sdb files that are dropped by malware or sdb files attempting a UAC bypass. In February 2015 a malware analyst named Takahio Haruyama created a generic Indicator of Compromise (IOC) for the for the Volatility plugin openioc_scan to trigger on the UAC bypass used by Dridex. (Haruyama, 2015)

Awareness of malicious shims capabilities combined with the tools described above, are good but it is expected that in the near future more sophisticated ACT enabled in-memory patching and malware obfuscation techniques will become more prevalent and beyond that I hypothesize more advanced attacks will include insertion of malicious sdb configurations into the native system sdb files, and the use of wildcards in file names (but this would require much more reverse engineering). This will likewise require defenders to continue to adapt and innovate to better defend their environments.

Conclusion

Defenders such as Incident Responders and Malware Analysts should be aware of how shim databases can be used to leverage rootkit-like functionality from the features found in the Application Compatibility Toolkit. Malicious shim database files have been observed in the wild with increasing frequency and complexity. Given the features and ease-of-use found in the ACT, attackers will soon start utilizing the more advanced methods demonstrated here, which has prompted me to create and distribute the tools necessary to prevent, detect, and combat this new avenue of post-exploitation.

Bibliography

Aquilino, B. (2014, 6 30). *Beware BlackEnergy If Involved In Europe/Ukraine Diplomacy*. Retrieved from F-Secure: https://www.f-secure.com/weblog/archives/00002721.html

Author. (2015, 2 8). *Upatre downloader malware using AppCompat for automatic UAC elevation*. Retrieved from Bleen. All the integers between 6 and 7: http://lupwa.org/bleen/2015/02/08/upatre-downloader-malware-using-appcompat-for-automatic-uac-elevation/

Bospoort, M. v. (2009, 7 23). *Disabling a Shim*. Retrieved from Maarten's blog: http://blogs.msdn.com/b/maartenb/archive/2009/07/24/disabling-a-shim.aspx

Chen, R. (2005, 8 24). *Buying an entire Egghead Software store*. Retrieved from The Old New Thing - MSDN Blogs: http://blogs.msdn.com/b/oldnewthing/archive/2005/08/24/455557.aspx

Cherepanov, A. (2014). Roaming tiger. *Zeronights 2014* (p. 20). Moscow: http://2014.zeronights.org/assets/files/slides/roaming_tiger_zeronights_2014.pdf.

Cherepanov, R. L. (2014, 1 25). *Last-minute paper: Back in BlackEnergy: 2014 targeted attacks in the Ukraine and Poland*. Retrieved from Virus Bulletin: https://www.virusbtn.com/conference/vb2014/abstracts/LM3-LipovskyCherepanov.xml

Dr. WEB. (2014, 4 9). Retrieved from BackDoor.Gootkit.112—a new multi-purpose backdoor: https://news.drweb.com/show/?i=4338&lng=en

Drapeau, P. (2015, 1 16). *Firewall Rule Changes and Compatibility Trickery*. Retrieved from Kill Chain. The Confer Blog: http://www.confer.net/kill-chain-the-confer-blog/80-firewall-rule-changes-and-compatibility-trickery

Erickson, J. (2014). Persist It: Using and Abusing Microsoft Fix It Patches. *BlackHat Asia* (p. 14). Singapore: https://www.blackhat.com/docs/asia-14/materials/Erickson/WP-Asia-14-Erickson-Persist-It-Using-And-Abusing-Microsofts-Fix-It-Patches.pdf.

F-Secure. (2015). *BLACKENERGY & QUEDAGH*. Retrieved from f-secure.com: https://www.f-secure.com/documents/996508/1030745/blackenergy_whitepaper.pdf

Graham, C. (2014, 5 27). *Shimming Your Way Past UAC*. Retrieved from Digital Defense Incorporated: http://blog.ddifrontline.com/4#more-4

Hancke, P. (2009). *The Windows 7 App Compat Story*. Retrieved from Microsoft Tech-ed Afraca 2009: http://download.microsoft.com/download/6/7/3/673B71D0-6D4F-4591-9940-D33DF9135487/WCL304_Win7_app_compat_story_Hancke.pptx

Haruyama, T. (2015, 2 26). *Github*. Retrieved from TakahiroHaruyama / openioc_scan: https://github.com/TakahiroHaruyama/openioc_scan/blob/master/IOCs/generic/cdcd5fdb-fcd3-4947-8c76-d2fbdc1b5f82_UAC_COM.ioc

hFireF0X. (2015, 1 9). *FireF0X*. Retrieved from Twitter: https://twitter.com/hfiref0x/status/553777371682443264

Inocencio, R. (2014, 11 5). *Banking Trojan DRIDEX Uses Macros for Infection*. Retrieved from Security
Intelligence Blog: http://blog.trendmicro.com/trendlabs-security-intelligence/banking-trojan-
dridex-uses-macros-for-infection/

Microsoft. (2009, 10 29). *Deploy an Application Compatibility Database by Using SDBInst*. Retrieved from
TechNet: https://technet.microsoft.com/en-us/library/ee732408(v=WS.10).aspx

Microsoft Support . (2015, 4 29). Retrieved from Update to force a UAC prompt when a customized .sdb
file is created in Windows: https://support.microsoft.com/en-us/kb/3045645

msft-mmpc. (2013, 10 31). *Upatre: Emerging Up(d)at(er) in the wild*. Retrieved from Microsoft Malware
Protect Center: http://blogs.technet.com/b/mmpc/archive/2013/10/31/upatre-emerging-up-d-
at-er-in-the-wild.aspx

Ollie. (2012, 4 28). *Windows AppCompat Research Notes - Part 1*. Retrieved from Research, Develop,
Assess, Consult & Educate: http://recxltd.blogspot.co.uk/2012/04/windows-appcompat-
research-notes-part-1.html

Pernet, C. (2015, 3 30). *Fake Judicial Spam Leads to Backdoor with Fake Certificate Authority*. Retrieved
from TrendLabs Security Intelligence Blog: http://blog.trendmicro.com/trendlabs-security-
intelligence/fake-judicial-spam-leads-to-backdoor-with-fake-certificate-authority/

Stewart, H. (2007, 11 3). *Shim Database to XML*. Retrieved from Setup & Install by Heath Stewart:
http://blogs.msdn.com/b/heaths/archive/2007/11/02/sdb2xml.aspx

*Baggett, M. (2013, February 23). 2013 Posts and Publications. Retrieved October 23, 2013, from In
Depth Defense: http://www.indepthdefense.com/2013/02/2013-posts-and-publications.html*

*Ionescu, A. (2007, May 20). Secrets of the Application Compatilibity Database (SDB) – Part 1. Retrieved
April 15, 2015, from Alex Ionescu's Blog: http://www.alex-ionescu.com/?p=39*

*Ionescu, A. (2007, May 21). Secrets of the Application Compatilibity Database (SDB) – Part 2. Retrieved
April 15, 2015, from Alex Ionescu's Blog: http://www.alex-ionescu.com/?p=40*

*Ionescu, A. (2007, May 26). Secrets of the Application Compatilibity Database (SDB) – Part 3. Retrieved
April 15, 2015, from Alex Ionescu's Blog: http://www.alex-ionescu.com/?p=41*

*Mark Russinovich, B. C. (2013, August 1). Autoruns for Windows v11.70. Retrieved September 5, 2013,
from Windows Sysinternals: http://technet.microsoft.com/en-us/sysinternals/bb963902.aspx*

*Microsoft. (2013, September 6). !chkimg. Retrieved October 2, 2013, from Dev Center:
http://msdn.microsoft.com/en-us/library/windows/hardware/ff562217%28v=vs.85%29.aspx*

Microsoft. (2013, October 1). Application Compatibility Database. Retrieved October 23, 2013, from Microsoft Developer Network:http://msdn.microsoft.com/library/bb432182.aspx

Microsoft. (2013). Fix it Solution Center. Retrieved 2013 24-October from Microsoft Support: http://support.microsoft.com/fixit/

Microsoft. (2012, October 1). Microsoft Security Advisory: Vulnerability in Microsoft XML Core Services could allow remote code execution. Retrieved September 5, 2013, from Microsoft Support: http://support.microsoft.com/kb/2719615

Microsoft. (2012, December 7). Shim Database Types. Retrieved September 5, 2013, from Microsoft Developer Network: http://msdn.microsoft.com/en-us/library/bb432483%28v=vs.85%29.aspx

Sikka, N. (2013, September 17). CVE-2013-3893: Fix it workaround available. Retrieved October 02, 2013, from Security Research & Defense:

http://blogs.technet.com/b/srd/archive/2013/09/17/cve-2013-3893-fix-it-workaround-available.aspx

Stewart, H. (2007, November 3). Shim Database to XML. Retrieved September 5, 2013, from Setup & Install by Heath Stewart: http://blogs.msdn.com/b/heaths/archive/2007/11/02/sdb2xml.aspx