TZWorks® Shim Database Parser (shims) Users Guide

Abstract

shims is a standalone, command-line tool that parses and extracts components from a Windows Application Compatibility database. Designed for the malware investigator, shims allows one to analyze any entry that may have been used to compromise a Windows system. ***shims*** runs on Windows, Linux and Mac OS-X.

www.tzworks.net

Contact Info: info@tzworks.net

Document applies to v0.12 of ***shims***

Updated: Sep 9, 2015

Table of Contents

TZWorks® Shim Database Parser (shims) Users Guide

Webpage: http://www.tzworks.net/prototype_page.php?proto_id=30
Contact Information: info@tzworks.net

1 Introduction

shims is a command line tool that parses and extracts components from an Application Compatibility Database (specifically referenced in this user's guide as a Shim Database or SDB file). This database is the configuration component used by the Window's Shim engine used to resolve compatibility issues between an application and how it interacts with Windows. The technology that implements this interacts between the Application Compatibility Interface (eg. shimreg.dll and apphelp.dll), the Shim engine (shimeng.dll), and various callbacks in the Portable Executable (PE) loader.

The Application Compatibility framework uses the Shim Database to identify if, and how, a process or DLL should be shimmed during process startup and/or DLL loading. The default Shim Database is located at \Windows\AppPatch\sysmain.sdb and contains thousands of entries for a normal Win7 box. In addition to the *sysmain.sdb* database, Windows can have other pre-installed databases and user-defined *custom* databases.

While the Window's Shim engine is used to enhance the user experience as well as resolve incompatibles between older binaries and operating systems they are running on, it can also be used (and has been used) as a launching point for malware. Specifically, the Shim engine allows installed applications on a Windows box to be patched 'on the fly' (ie. the term *hot-patching* is used by the community). This patch can be used to spawn other processes, or inject undesired DLLs, into the patched application. Doing this offers the malware writer another way to achieve persistence across reboots. Therefore, understanding which Shim Databases are on your system and subsequently parsing those databases to extract targeted patches per application are one of the primary purposes of this tool.

There are at least four different types of modifications that can be done with the Application Compatibility framework:

- System shims, which get implemented with an API hook to one of the libraries, AcGenrl.dll or AcLayers.dll
- Application tailored shims, which also get implemented with an API hook, but to the library AcSpecifc.dll.
- Flag shims, which specifies some flag(s) to the application, or to an installer, about the application.

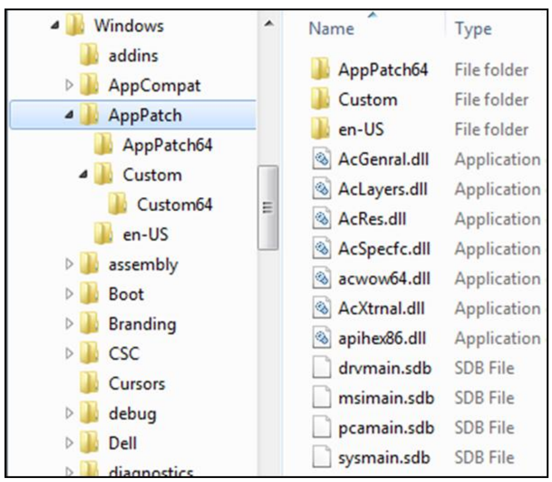
- Binary patch, which represents an 'on the fly' memory patch on the executable instead of a system API hook.

To target an application, or a family of applications, entries within the Shim Database can identify either specific internal parameters or very generic external parameters to the Application Compatibility matching algorithm. For example, below are some of the available options that can be seen when examining a Shim Database.

- Simple matching which can use file timestamp, compile timestamp and/or checksum entries
- More complex matching which can use the present of certain resources within a PE file, such as bitmaps, and/or other data.
- Generic matching which can use wildcards along with Boolean logic for other matching conditions.

2 Background Information

Shim databases are typically located in the *%windir%\AppPatch* main directory. Whether a shim database targets a 32 bit or 64 bit application and whether it is a custom shim or not, determines which subdirectory it goes into.



The 32 bit versions of the default Windows shim databases are at the root of the *%windir%\AppPatch* directory. The 64 bit versions of the default Windows shim databases are in one directory down, in the *%windir%\AppPatch\AppPatch64* directory. Custom shim databases (those that are made by anyone else or are not part of the default Windows shim databases) are stored in the *%windir%\AppPatch\Custom* directory and *%windir%\AppPatch\Custom64* directories. The 32 bit versions are stored in the former and 64 bit versions are stored in the latter. Unfortunately, these directories are only a convention and not a requirement. For example, on my Windows 8 box, the *%windir%\system32\CompatTel* directory contains a *sysmain32.sdb* Shim Database file. The good news is each *custom* shim database has a registry entry that identifies its name, path, and installation timestamp. This can be found at *HKLM\Software\Microsoft\Windows*

NT\CurrentVersion\AppCompatFlags\InstalledSDB. Below is the data taken from a sample custom shim that was installed for demo purposes. So if a Shim Database did use a different path or different extension, then it would be documented here.

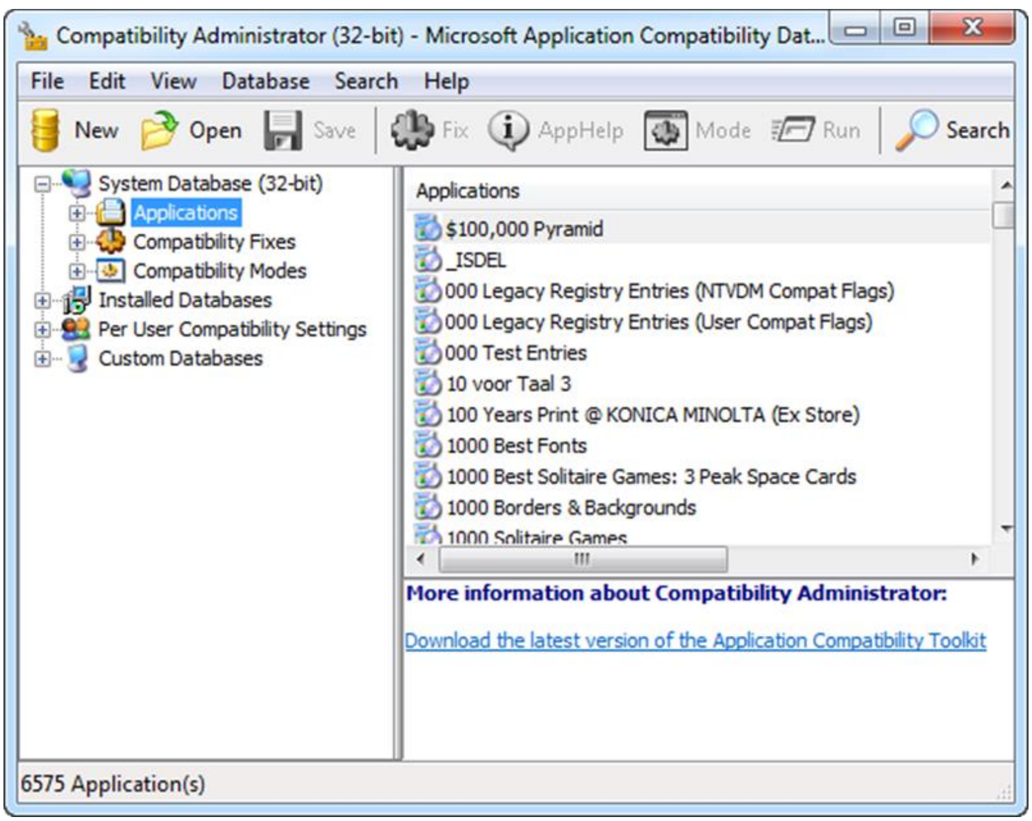
```
SOFTWARE\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\InstalledSDB\{fd241ca6-4568-4962-b66e-015cb56c27ce}

Timestamp: 0x01d0523fcb0cd508 (02/27/2015 03:45:13.677 UTC)

DatabasePath                REG_SZ       C:\Windows\AppPatch\Custom\{fd241ca6-4568-4962-b66e-015cb56c27ce}.sdb
DatabaseType                REG_DWORD    0x00010000
DatabaseDescription         REG_SZ       TestShimDB
DatabaseInstallTimeStamp    REG_QWORD    0x01d0523fcb0cd508
```

2.1 Compatibility Administrator Tool

Microsoft provides a nice GUI utility, called the Compatibility Administrator to read compatible SDB databases. Below is a screen shot of this tool examining the global *sysmain.sdb* database. This tool is very useful in breaking out the various applications that are targeted, the compatibility fixes and the modes. The tool also shows any custom database currently active as well.



When designing the ***shims*** tool, we used the above Microsoft tool to validate our output. Unfortunately, we could not verify everything, as the Microsoft tool does not show much of the internal data, which includes: patches, GUIDs, certain flags, etc. So to validate some of the other metadata, we resorted to other techniques to identify some of the fields that were not shown in the GUI tool. This gave us the enough insight to understand many of the fields that were not shown in the GUI tool and allowed use to write our own application that could work across multiple platforms. While we believe our ***shims*** tools is relatively stable, there are undoubtedly boundary conditions that still need to be discovered and fixed.

3 How to Use the *shims* Tool

To extract general purpose information from one of these databases, use the ***-stats*** option. This gives summary information of what type of compatibility fixes are in the database as well as various timestamps associated with the database.

To search a database, or find details about certain entries, one can use a variety of other options. This includes filtering on different types of compatibility fixes (such as: patches, shims, fixes), or just searching for specific target executables or DLLs.

Below is a menu which shows many of the options in summary form:

```
Administrator: Command Prompt

shims - full ver: 0.12; Copyright (c) TZWorks LLC

Usage
 shims -listsdb              = list SDB files on system volume
 shims -stats                = pull stats from SDB files on system volume
 shims -sdb <DB> [opts]      = target SDB file w/ specific option

 Enumerate options
  -apps                      = ** all apps (exes, packages, driverblocks,..)
  -exes                      = ** filter only exe tags
  -fixes                     = ** all types of fixes (shims, flags,..)
  -shims                     = ** filter only shim tag fixes
  -patches                   = ** filter only patch tag fixes
  -layers                    = ** filter only layer tag fixes
  -flags                     = ** filter only flag tag fixes
  -tag <#>                   = *** filter specific tag type
  -guids                     = ** enumerate guids
  -stringtable               = ** enumerate stringtable

 Find Options
  -strings "str1 | str2 |.." = ** finds partial strings [case insensitive]
  -guid <guid to find>       = ** syntax: 11111111-1111-1111-1111-111111111111
  -tagids "id1 | id2 |.."    = *** finds specified tagids
  -patchbytes "pattern"      = *** find patch, hex bytes w/ space delimiters
  -match                     = *** use w/ -pe <PE file> to check for shims

 Additional Options
  -vss <index>               = *** target Volume snapshop at index
  -partition <letter>        = ** target Shim DB locations in this volume
  -pipe                      = ** use stdin to identify files to process
  -stats -sdb <file>         = pull stats (on SDB File). [-reg <sw hive>]
  -stats -pe <file>          = pull stats (on PE File)
  -enumsdb                   = list SDB files

 General Examples
  shims -sdb <file> -apps    = pull all apps from DB
  shims -sdb <file> -patches = pull all patches from DB
  shims -sdb <file> -stats   = pull DB stats
  shims -pe <file> -stats    = pull stats from PE file

 Pulling stats from multiple SDB's
  dir c:\windows\AppPatch\*.sdb /b /s | shims -stats -pipe -csv > out.csv
  shims -vss 2 -stats -csv > out.csv
  shims -partition c: -exes > out.txt
```

All the compatibility fixes will be rendered in XML output, while the statistics options can be done in either unformatted text or CSV output. The various options and how they can be used, are discussed in the sections below.

3.1 Quick-look Report for a Database

When analyzing a database, one can pull the statistics about the database and its composition by running the ***-stats*** command. Below is an example of running shims on one of the Volume shadow copies and truncating the output to display the global shim database (*sysmain.sdb*).

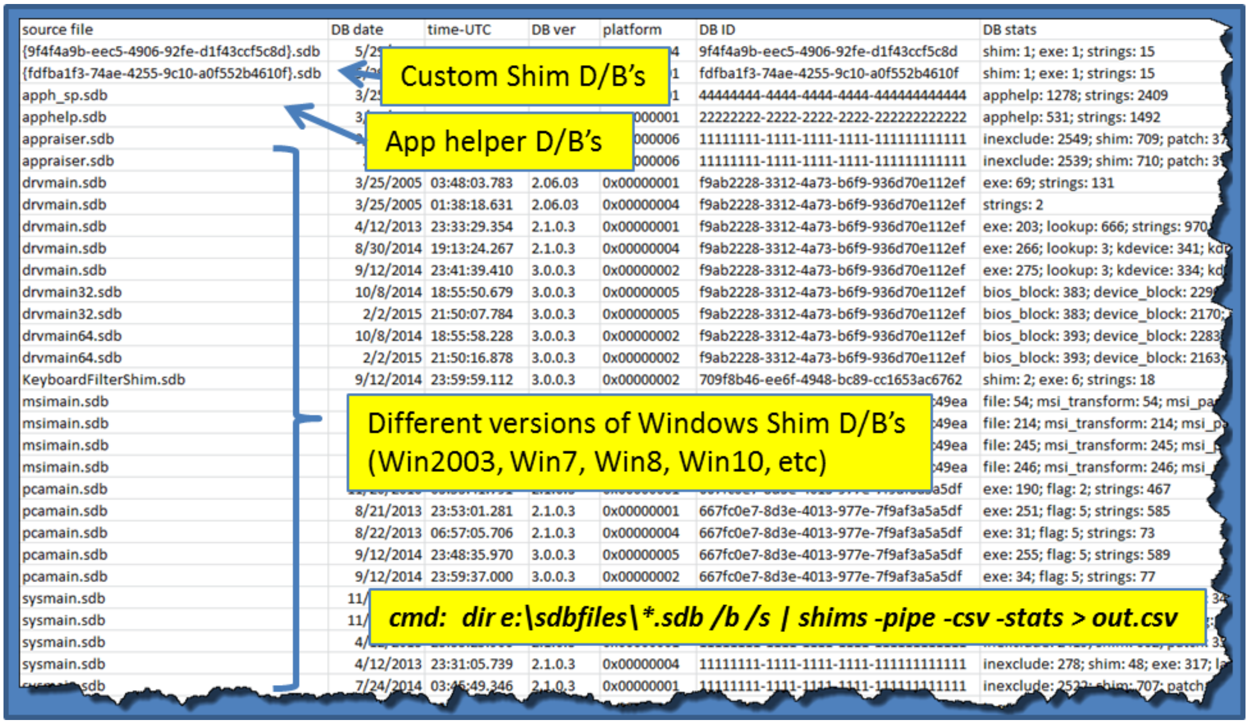
```
"cmdline: shims64 -stats -vss 1"

Database Path/File      | \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1\Windows\AppPatch\sysmain.sdb
Database MD5            | 1d8c1280d38c526c7041e72db8d70dc1
Database SHA1           | da2e372481e6cdb450091794a58f294a46be1a46
File ModTime            | 04/12/2013 23:32:33.314 [UTC]
File AccessTime         | 09/12/2014 01:00:55.654 [UTC]
File CreateTime         | 09/12/2014 01:00:55.654 [UTC]
Database ModTime        | 04/12/2013 23:33:25.906 [UTC]
Compiler Version        | 2.1.0.3
Database Version        | 2.1
Database Internal Name  | Microsoft Windows Application Compatibility Fix Database
Database Platform       | 0x00000001
Database Identifier     | 11111111-1111-1111-1111-111111111111
appname                 | tag 0x6006:  6625 items
inexclude               | tag 0x7003:  2419 items
shim                    | tag 0x7004:   662 items
patch                   | tag 0x7005:    35 items
exe                     | tag 0x7007: 13105 items
layer                   | tag 0x700b:    64 items
flag                    | tag 0x7013:   149 items
context                 | tag 0x7018:     1 item
strings                 | tag 0x8801: 39202 items
appid                   | tag 0x9011:  7013 items
--------------------------------------------------------------------------------------
Database Path/File      | \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1\Windows\AppPatch\AppPatch64\
```

The output shows the various timestamps of the SDB file as well as the last time the database was updated (via the internal database timestamp labeled *Database ModTime*). Included in the database summary are the following: the version number, MD5/SHA1 hashes, identifier, and a number of other stats about the contents within it, such as the occurrences of the differing fixes and other elements. From empirical data, the database identifier either uses a class *GUID* or uses a custom unique *GUID*. For example, both the *sysmain.sdb* and the *appraiser.sdb* databases appear to be always classified as 11111111-1111-1111-1111-111111111111. Other databases seem to have common GUIDs as well. Below is a table of some of the common GUIDs we have found from empirical analysis.

SDB name	Type	GUID
sysmain[null|32|64].sdb, appraiser.sdb, sysmain[32|64]runtime.sdb	App Compatibility Fix D/B	11111111-1111-1111-1111-111111111111
drvmain[null|32|64].sdb	Driver Compatibility D/B	f9ab2228-3312-4a73-b6f9-936d70e112ef
pcamain.sdb	Program Compatibility Assistant D/B	667fc0e7-8d3e-4013-977e-7f9af3a5a5df
msimain.sdb	System Installer Compatibility D/B	d8ff6d16-6a3a-468a-8b44-01714ddc49ea
KeyboardFilterShim.sdb	Embedded Keyboard Filter D/B	709f8b46-ee6f-4948-bc89-cc1653ac6762
apphelp.sdb	App Compatibility Message D/B	22222222-2222-2222-2222-222222222222
apph_sp.sdb	App Compatibility Message D/B - Service Pack	44444444-4444-4444-4444-444444444444

One can repeat this by collecting a number of shim databases from various versions of Windows operating systems into a directory for analysis, and then piping in the directory into the ***shims*** tool using the ***-pipe*** and ***-stats*** commands together. The ***-stats*** command also allows one to use the options: ***-csv, -csvl2t, -csv_separator, -dateformat, -timeformat***.



source file	DB date	time-UTC	DB ver	platform	DB ID	DB stats
{9f4f4a9b-eec5-4906-92fe-d1f43ccf5c8d}.sdb	5/2			4	9f4f4a9b-eec5-4906-92fe-d1f43ccf5c8d	shim: 1; exe: 1; strings: 15
{fdfba1f3-74ae-4255-9c10-a0f552b4610f}.sdb				1	fdfba1f3-74ae-4255-9c10-a0f552b4610f	shim: 1; exe: 1; strings: 15
apph_sp.sdb	3/2			1	44444444-4444-4444-4444-444444444444	apphelp: 1278; strings: 2409
apphelp.sdb	3			000001	22222222-2222-2222-2222-222222222222	apphelp: 531; strings: 1492
appraiser.sdb				000006	11111111-1111-1111-1111-111111111111	inexclude: 2549; shim: 709; patch: 3
appraiser.sdb				000006	11111111-1111-1111-1111-111111111111	inexclude: 2539; shim: 710; patch: 3
drvmain.sdb	3/25/2005	03:48:03.783	2.06.03	0x00000001	f9ab2228-3312-4a73-b6f9-936d70e112ef	exe: 69; strings: 131
drvmain.sdb	3/25/2005	01:38:18.631	2.06.03	0x00000004	f9ab2228-3312-4a73-b6f9-936d70e112ef	strings: 2
drvmain.sdb	4/12/2013	23:33:29.354	2.1.0.3	0x00000001	f9ab2228-3312-4a73-b6f9-936d70e112ef	exe: 203; lookup: 666; strings: 970
drvmain.sdb	8/30/2014	19:13:24.267	2.1.0.3	0x00000004	f9ab2228-3312-4a73-b6f9-936d70e112ef	exe: 266; lookup: 3; kdevice: 341; kd
drvmain.sdb	9/12/2014	23:41:39.410	3.0.0.3	0x00000002	f9ab2228-3312-4a73-b6f9-936d70e112ef	exe: 275; lookup: 3; kdevice: 334; kd
drvmain32.sdb	10/8/2014	18:55:50.679	3.0.0.3	0x00000005	f9ab2228-3312-4a73-b6f9-936d70e112ef	bios_block: 383; device_block: 229
drvmain32.sdb	2/2/2015	21:50:07.784	3.0.0.3	0x00000005	f9ab2228-3312-4a73-b6f9-936d70e112ef	bios_block: 383; device_block: 2170;
drvmain64.sdb	10/8/2014	18:55:58.228	3.0.0.3	0x00000002	f9ab2228-3312-4a73-b6f9-936d70e112ef	bios_block: 393; device_block: 2283
drvmain64.sdb	2/2/2015	21:50:16.878	3.0.0.3	0x00000002	f9ab2228-3312-4a73-b6f9-936d70e112ef	bios_block: 393; device_block: 2163
KeyboardFilterShim.sdb	9/12/2014	23:59:59.112	3.0.0.3	0x00000002	709f8b46-ee6f-4948-bc89-cc1653ac6762	shim: 2; exe: 6; strings: 18
msimain.sdb					49ea	file: 54; msi_transform: 54; msi_pa
msimain.sdb					49ea	file: 214; msi_transform: 214; msi_p
msimain.sdb					49ea	file: 245; msi_transform: 245; msi_
msimain.sdb					49ea	file: 246; msi_transform: 246; msi_
pcamain.sdb					a5df	exe: 190; flag: 2; strings: 467
pcamain.sdb	8/21/2013	23:53:01.281	2.1.0.3	0x00000001	667fc0e7-8d3e-4013-977e-7f9af3a5a5df	exe: 251; flag: 5; strings: 585
pcamain.sdb	8/22/2013	06:57:05.706	2.1.0.3	0x00000004	667fc0e7-8d3e-4013-977e-7f9af3a5a5df	exe: 31; flag: 5; strings: 73
pcamain.sdb	9/12/2014	23:48:35.970	3.0.0.3	0x00000005	667fc0e7-8d3e-4013-977e-7f9af3a5a5df	exe: 255; flag: 5; strings: 589
pcamain.sdb	9/12/2014	23:59:37.000	3.0.0.3	0x00000002	667fc0e7-8d3e-4013-977e-7f9af3a5a5df	exe: 34; flag: 5; strings: 77
sysmain.sdb	11/					34
sysmain.sdb	11/					
sysmain.sdb	4/					3
sysmain.sdb	4/12/2013	23:31:05.739	2.1.0.3	0x00000004	11111111-1111-1111-1111-111111111111	inexclude: 278; shim: 48; exe: 317; l
[illegible]sdb	7/24/2014	03:45:49.346	2.1.0.3	0x00000001	11111111-1111-1111-1111-111111111111	inexclude: 252[illegible] shim: 707; patch

Custom shims have some additional statistics that come from their respective registry entries. Of interest are: (a) the shim database 'install' timestamp and (b) when the subkey for the registry entry was modified. Below is an example of where these additional timestamps are populated in the stats output:

```
"cmdline: shims64 -sdb c:\Windows\AppPatch\Custom\{fd241ca6-4568-4962-b66e-015cb56c27ce}.sdb -stats"

Database Path/File      | c:\Windows\AppPatch\Custom\{fd241ca6-4568-4962-b66e-015cb56c27ce}.sdb
Database MD5            | 2d660abdf56c914a336a48a7eaa7ca68
Database SHA1           | 57d1fd6f021d8b3d88732474e2f40dff6f4c26dd
File ModTime            | 03/02/2015 19:42:24.645 [UTC]
File AccessTime         | 03/03/2015 04:39:16.900 [UTC]
File CreateTime         | 03/02/2015 19:42:24.645 [UTC]
Database ModTime        | 03/02/2015 19:42:42.176 [UTC]
Reg DB InstallTime      | 03/03/2015 04:41:06.537 [UTC]
Registry ModTime        | 03/03/2015 04:41:06.537 [UTC]
Compiler Version        | 2.1.0.3
Database Version        | 2.1
Database Internal Name  | TestShimDB
Database Platform       | 0x00000001
Database Identifier     | fd241ca6-4568-4962-b66e-015cb56c27ce
appname                 | tag 0x6006:     2 items
exe                     | tag 0x7007:     2 items
strings                 | tag 0x8801:    15 items
```

Custom shim

Times available from registry

3.1.1 Statistics for Mounted System Volume or Volume Shadow

If one just wants to enumerate all the shim databases in the conventional directories as well as any custom shim databases, one can use the ***-partition*** option and the ***-vss*** option. The first option will analyze the specified system partition, and the second option will analyze the specified volume shadow. Below are examples:

shims -partition "c" -stats -csv

shims -vss 1 -stats –csv

3.2 Searching Strings

The string search is case-insensitive and looks for partial strings. The search will default to scanning all application type tags. As an example, let's say one wanted to analyze all the entries that make up the Compatibility Fix name, such as "*InjectDLL*" or "*RunAsAdmin*". To search multiple strings, just use a pipe delimiter between the strings you want to search on. If one of the substrings is found, the application that included the substring is returned so that one can see the context of where it was used. Below is an example of performing this search on a Windows system volume.

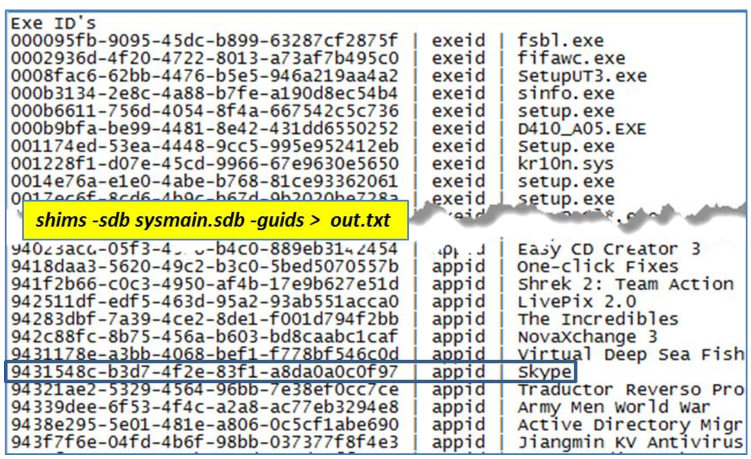
```
<shimdb>
 <header>
  cmd: shims64 –partition c:  -strings "InjectDLL | RunAsAdmin" > out.txt
  <meta info="run time: 03/03/2015 20:27:44 [UTC]" />
  <meta info="cmdline: shims64 -partition c: -strings 'InjectDll | RunA
  <meta file="c:\Windows\AppPatch\sysmain.sdb" />
 </header>
 <exe name="*.exe" wildcaname="*.exe" vendor="Big Fish Games" exeid="2
  <app appname="Big Fish Games Installer" appid="6b9992e3-b4b9-4e20-909
  <matchingfile name="*.exe" companyname="Big Fish Games" productname=
  <layer name="RunAsAdmin" layertagid="32c5c">
   <layer name="RunAsAdmin" fixid="f5ac3378-b8e4-4f9b-aa9a-d839e5b1ef
    <data name="SHIMFLAGS" data_valuetype="0x00000004" data_dword="0x00
    <flagref name="RunAsAdmin" flagtagid="2eb04">
     <flag name="RunAsAdmin" fixid="3c824c52-8f73-4a1a-81dd-19bcbe04339
    </flagref>
   </layer>
  </layer>
 </exe>
 <exe name="GLJ*.tmp" wildcaname="GLJ*.tmp" vendor="AOL" exeid="f83a0fc
  <app appname="AOL Instant Messenger" appid="d1591404-7c1c-4a8e-9395
  <matchingfile name="GLJ*.tmp" size="0x00000a00" checksum="0x49446e1a"
  <shimref name="InjectDll" shimtagid="2524a" commandline="RTvideo.dll"
   <shim name="InjectDll" dllfile="AcGenral.DLL" fixid="3432bc96-d181-4
  </shimref>
 </exe>
 <exe name="SkypeSetup*.exe" wildcaname="SkypeSetup*.exe" vendor="Skype
  <app appname="Skype" appid="9431548c-b3d7-4f2e-83f1-a8da0a0c0f97" />
  <matchingfile name="SkypeSetup*.exe" companyname="Skype Technologies S
  <layer name="VistaSetup" />
  <flagref name="RunAsAdmin" flagtagid="2eb04">
   <flag name="RunAsAdmin" fixid="3c824c52-8f73-4a1a-81dd-19bcbe043396
  </flagref>
 </exe>
 <exe name="SkypeToolbarForOutlook*.exe" wildcaname="SkypeToolbarForOut
  <app appname="Skype Email Toolbar" appid="d9fa215c-52c3-472b-b6ff-b6
  <matchingfile name="SkypeToolbarForOutlook*.exe" companyname="'Skyp
  <flagref name="RunAsAdmin" flagtagid="2eb04">
   <flag name="RunAsAdmin" fixid="3c824c52-8f73-4a1a-81dd-19bcbe043396"
  </flagref>
 </exe>
```

In this case, four application entries are found and the output is rendered in XML. Annotated are the locations of where the specified strings were found.

3.3 Searching GUIDs

The Shim database makes use of GUID identifiers for three main types of tags: executables, applications, and fixes. It should be noted that the executable GUID identifier is independent of the application GUID identifier, however all executable containers also include an application identifier. From the empirical data, the application GUID is used to group similar executables where each executable can have a different (or the same) name, but have different executable GUIDs. When viewed in the Microsoft Compatibility Administrator, the *Applications* folder contains folder instances of *application* GUID IDs, where each folder is a collection of unique *executable* GUID IDs (Note: the GUID for the *application* and *executable* are not the same). The previous screen shot shows this for the first application entry. Unfortunately, the Compatibility Administrator tool does not show the GUIDs of the items.

Instead of repeating the search using one of the GUIDs shown from the previous example, we will use the application identifiers used for the Skype application. To find which GUIDs are used in the database, one can do an initial scan for all GUIDs by using the ***-guids*** switch. Below is the type of output you would get by invoking this command:



The above output is broken out by *exeid* (for executable identifiers), *appid* (for application identifiers), and *fixid* (for fix identifiers). For this example, we will pull the Skype application identifier (eg. 9431548c-b3d7-4f2e-83f1-a8da0a0c0f97) and search on that. Below are the results. Alternatively, we could have done a string search on "*skype*", but the results most likely would have included other entries that were not designated with this application identifier.

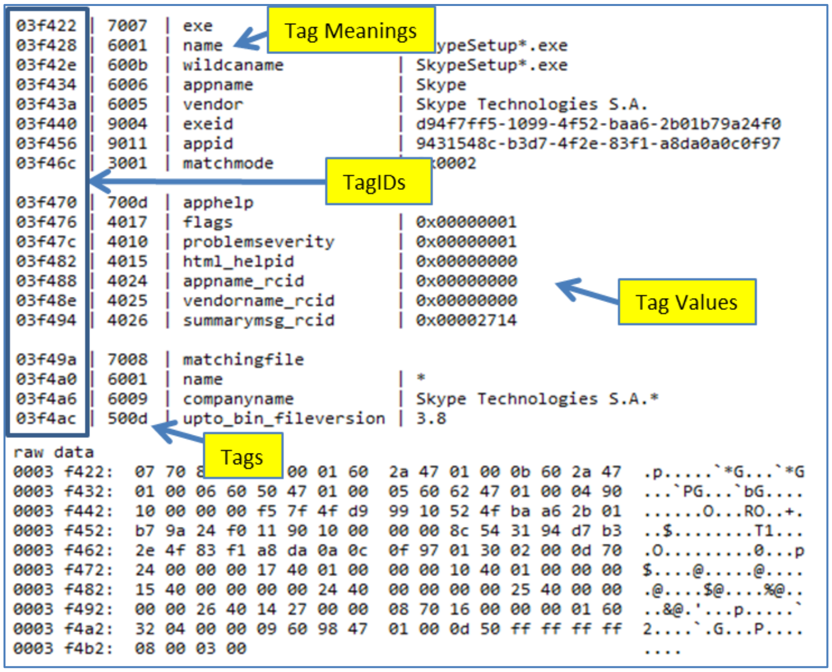
```
<?xml version="1.0" encoding="UTF-8"?>
- <shimdb>
  + <header>
  - <exe matchmode="0x0002" exeid="d94f7ff5-1099-4f52-baa6-2b01b79a24f0" vendor="Skype Technologies S.A." wildcaname="SkypeSetup*.exe"
    name="SkypeSetup*.exe">
      <app appid="9431548c-b3d7-4f2e-83f1-a8da0a0c0f97" appname="Skype"/>
      <apphelp summarymsg_rcid="0x00002714" problemseverity="NOBLOCK"/>
      <matchingfile name="SkypeSetup*.exe" upto_bin_fileversion="3.8" companyname="Skype Technologies S.A.*"/>
    </exe>
  - <exe matchmode="0x0002" exeid="ac65ebbf-77c8-4562-b031-5c07f058e0f5" vendor="Skype Technologies S.A." wildcaname="SkypeSetup*.exe"
    name="SkypeSetup*.exe">
      <app appid="9431548c-b3d7-4f2e-83f1-a8da0a0c0f97" appname="Skype"/>
      <matchingfile name="SkypeSetup*.exe" upto_bin_fileversion="2" companyname="Skype Technologies S.A.*"/>
      <layer name="VistaSetup"/>
    - <flagref name="RunAsAdmin" flagtagid="2eb04">
        <flag name="RunAsAdmin" description_rcid="0x0001117d" general="set" flag_lua="0x0000000000000004" fixid="3c824c52-8f73-4a1a-81dd-19bcbe043396"/>
      </flagref>
    </exe>
  - <exe matchmode="0x0002" exeid="275834a8-0fa1-4a9c-89bb[illegible]3799543a" vendor="Skype Technologies S.A." name="Skype.exe">
      <app appid="9431548c-b3d7-4f2e-83f1-a8da0a0c0f97" appname="Skype"/>
      <apphelp summarymsg_rcid="0x00002714" problemseverity="NOBLOCK"/>
      <matchingfile name="Skype.exe" companyname="Skype Technologies S.A.*" bin_productversion="3.8"/>
    </exe>
  - <exe matchmode="0x0002" exeid="3260c340-5299-463e-b80[illegible]3ec7bd31" vendor="Skype Technologies S.A." name="Skype.exe">
      <app appid="9431548c-b3d7-4f2e-83f1-a8da0a0c0f97" appname="Skype"/>
      <matchingfile name="Skype.exe" companyname="Skype Technologies S.A.*" bin_productversion="3"/>
    - <shimref name="RedirectDefaultAudioToCommunications" shimtagid="26d8c">
        <shim name="RedirectDefaultAudioToCommunications" description_rcid="0x0000eb58" general="set" fixid="a8fa7f99-ca13-4d56-b43c-43ed4c48beda"
          dllfile="AcGenral.DLL"/>
      </shimref>
    </exe>
</shimdb>
```

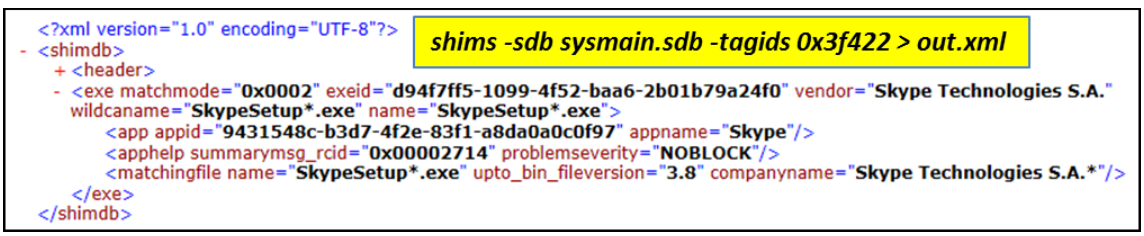

shims -guid 9431548c-b3d7-4f2e-83f1-a8da0a0c0f97 -sdb sysmain.sdb > out.xml

3.4 Searching *TagIDs*

Internally, the Shims database uses tag identifiers to identify certain elements in the database. From empirical analysis, this *TagID* turns out to be the offset into the database where the element is located. Therefore, one can arbitrarily assign the offset of the element as the TagID. This provides a unique key for each element when creating an associative array for indexing purposes. Therefore, if you know the *TagID* of an element, ***shims*** can easily look-up the element associated with that *TagID* and output the resulting data.

To visually see where *TagIDs* are used (from our perspective) and how they are lined up with a container, we will look at the first executable from the previous example, which is GUID d94f7ff5-1099-4f52-baa6-2b01b79a24f0. Using our internal (non-public) options, we show how the ***shims*** tool dissects this entry and identifies each element. The highlighted column shows the mapping of *TagID* to each element. Therefore, if a database entry used a TagID to reference a fix, shim, or whatever, it is straightforward to find it within the database and merge it. Suffice to say, using and searching on *TagIDs* is something useful to the reverse engineers.





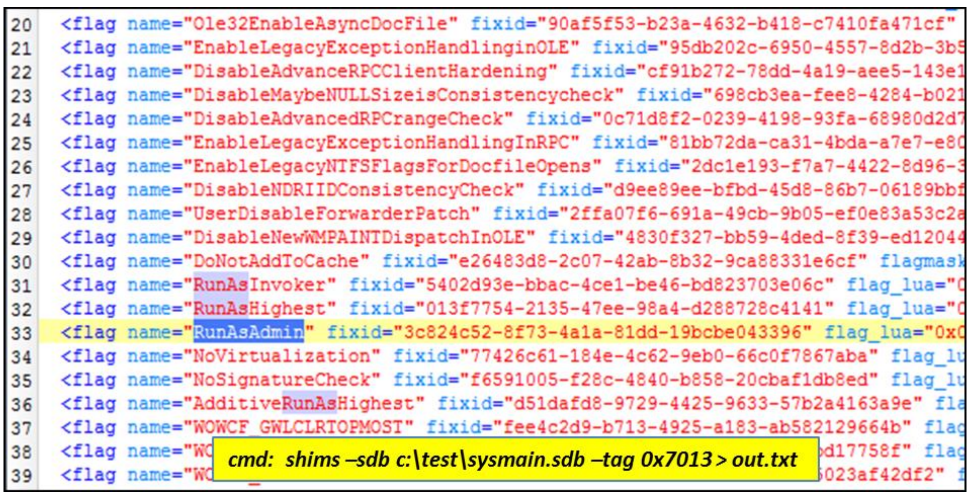
3.5 Pulling out Specific List Type Tags

A Shim database has all sorts of tags that can be searched on. The shims tool only has shortcut options for some of the more basic tags. For example: ***-exes*** for TAG_EXE, ***-apps*** for TAG_APP, ***-patches*** for TAG_PATCHES and a few others. There are many other tags that are available, such as TAG_APPHELP (0x700d), TAG_KDRIVER (0x701c), etc, which we do not have menu shortcuts. However, one can use the ***-tag*** option to enumerate some of these. Many of these are documented on the Microsoft website at: (http://msdn.microsoft.com/en-us/library/bb432487). The ***-tag <tag number>*** currently only handles some of the *TAG_TYPE_LIST* items. Below is a table of some of the ones that can be used.

TAG_TYPE_LIST types handled	Menu option	Purpose
TAG_SHIM	-shims	Shim entry
TAG_PATCH	-patches	In-memory (hot-patch) info
TAG_APP	-apps	Application entry
TAG_EXE	-exes	Executable entry
TAG_LAYER	-layers	Layer shim entry
TAG_MSI_FLAG	-flags	Flag entry to enable built-in fixes
TAG_MATCHING_FILE	-tag 0x7008	Matching file entry

TAG_FILE	-tag 0x700c	File attributed used in a shim entry
TAG_APPHELP	-tag 0x700d	Application help info entry
TAG_LINK	-tag 0x700e	Application help on-line link info entry
TAG_DATA	-tag 0x700f	Name-value mapping entry
TAG_MSI_TRANSFORM	-tag 0x7010	MSI transform entry
TAG_MSI_PACKAGE	-tag 0x7012	MSI package entry
TAG_MSI_CUSTOM_ACTION	-tag 0x7014	MSI custom action entry
TAG_LOOKUP	-tag 0x7017	Lookup entry in a driver database

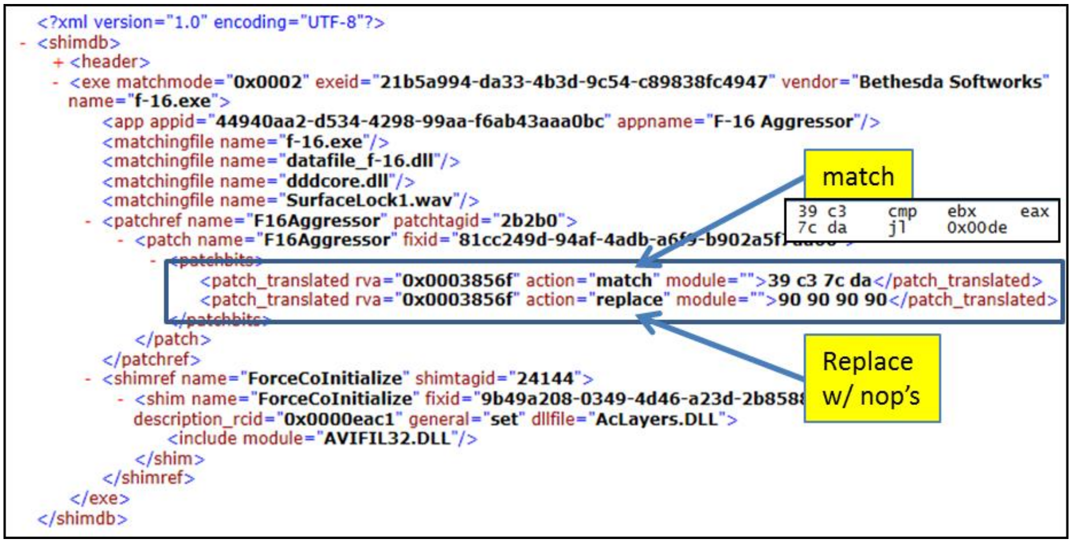
As an example, to enumerate all the TAG_FLAG's, one normally would use the ***-flags*** option, however, one could also use the option ***-tag 0x7013*** (0x7013 equates to TAG_FLAG) as part of the command. The TAG_FLAG is actually interesting, in that its presence indicates which built-in Compatibility fix to turn on. Shown below what one would see if enumerating the flag entries. Highlighted is the flag entry *RunAsAdmin* Compatibility fix.



3.6 Searching Patches

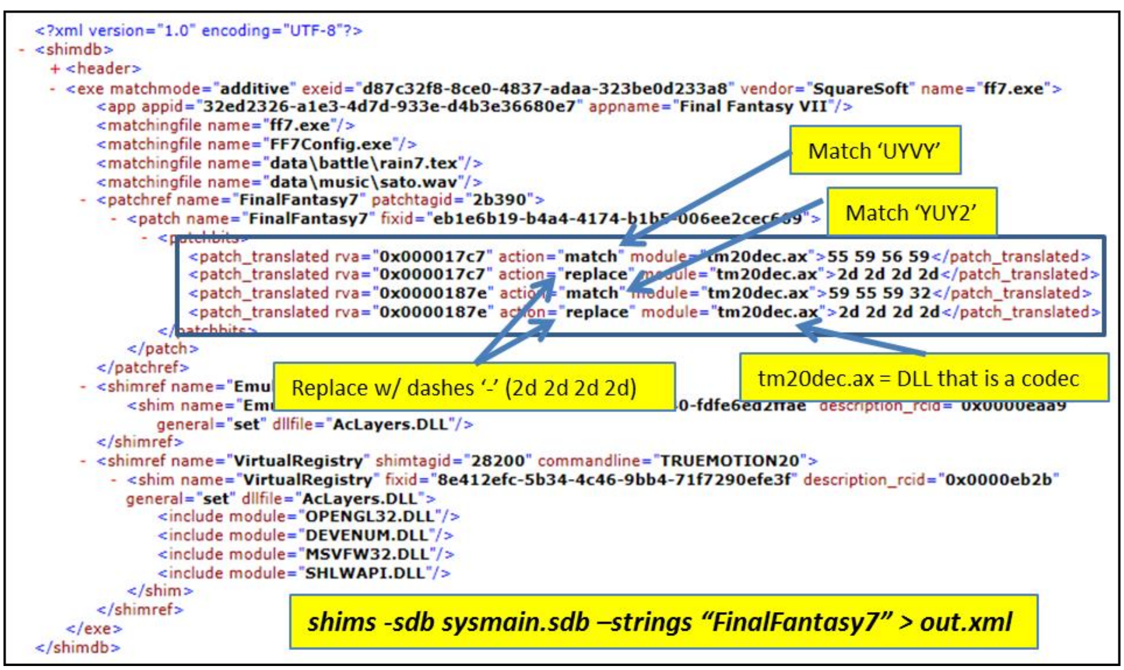
The fixes in the Shim database come in a variety of types (shims, flags, quirks, etc.), where patches are just but one. Focusing on patches, there are two types of patch entries in Shim databases: (a) Those that are patch sequences that need to be found in the target file and (b) those that are patch sequences that are meant to replace the sequence found. In addition, the patch entry has the binary location in the target file where to look and also where to apply the patch. This location is called the RVA which just equates to the relative virtual address.

Below is a simple patch example that replaces 4 bytes (39 c3 7c da) with NOPs (90 90 90 90) at the RVA of 0x0003856f. In this particular patch, the module name is not explicitly listed, which then defaults to the one of the matching file names.



```
<?xml version="1.0" encoding="UTF-8"?>
- <shimdb>
  + <header>
  - <exe matchmode="0x0002" exeid="21b5a994-da33-4b3d-9c54-c89838fc4947" vendor="Bethesda Softworks"
    name="f-16.exe">
      <app appid="44940aa2-d534-4298-99aa-f6ab43aaa0bc" appname="F-16 Aggressor"/>
      <matchingfile name="f-16.exe"/>
      <matchingfile name="datafile_f-16.dll"/>
      <matchingfile name="dddcore.dll"/>
      <matchingfile name="SurfaceLock1.wav"/>
    - <patchref name="F16Aggressor" patchtagid="2b2b0">
      - <patch name="F16Aggressor" fixid="81cc249d-94af-4adb-a6
        - <patchbits>
            <patch_translated rva="0x0003856f" action="match" module="">39 c3 7c da</patch_translated>
            <patch_translated rva="0x0003856f" action="replace" module="">90 90 90 90</patch_translated>
          </patchbits>
        </patch>
      </patchref>
    - <shimref name="ForceCoInitialize" shimtagid="24144">
      - <shim name="ForceCoInitialize" fixid="9b49a208-0349-4d46-a23d-2b858
        description_rcid="0x0000eac1" general="set" dllfile="AcLayers.DLL">
          <include module="AVIFIL32.DLL"/>
        </shim>
      </shimref>
    </exe>
</shimdb>
```


Some of the patches do not have assembly opcodes, but could just target constants or strings. For example, this next patch clears out two of the video options from a codec DLL module with the name of *tm20dec.ax*. From the patch data shown below, there are 2 pairs of match/replace entries. One can see this by looking at the matching RVA for each pair. The first pair starts by looking for the byte sequence "55 59 56 59", which equates to the ASCII characters 'UYVY'. The second pair starts by looking for the byte sequence "59 55 59 32", which equates to the ASCII characters 'YUY2'. Both of these happen to be video formats. The 'replace' portion for both of the matches are a sequence of "2d 2d 2d 2d", which equates to the ASCII characters '----', to evidently remove the video format options, should their companion match condition be satisfied.



```
<?xml version="1.0" encoding="UTF-8"?>
- <shimdb>
  + <header>
  - <exe matchmode="additive" exeid="d87c32f8-8ce0-4837-adaa-323be0d233a8" vendor="SquareSoft" name="ff7.exe">
      <app appid="32ed2326-a1e3-4d7d-933e-d4b3e36680e7" appname="Final Fantasy VII"/>
      <matchingfile name="ff7.exe"/>
      <matchingfile name="FF7Config.exe"/>
      <matchingfile name="data\battle\rain7.tex"/>
      <matchingfile name="data\music\sato.wav"/>
    - <patchref name="FinalFantasy7" patchtagid="2b390">
      - <patch name="FinalFantasy7" fixid="eb1e6b19-b4a4-4174-b1b
        - <patchbits>
            <patch_translated rva="0x000017c7" action="match" module="tm20dec.ax">55 59 56 59</patch_translated>
            <patch_translated rva="0x000017c7" action="replace" module="tm20dec.ax">2d 2d 2d 2d</patch_translated>
            <patch_translated rva="0x0000187e" action="match" module="tm20dec.ax">59 55 59 32</patch_translated>
            <patch_translated rva="0x0000187e" action="replace" module="tm20dec.ax">2d 2d 2d 2d</patch_translated>
          </patchbits>
        </patch>
      </patchref>
    - <shimref name="Emu
        <shim name="Emu                0-fdfe6ed2ffae" description_rcid="0x0000eaa9"
          general="set" dllfile="AcLayers.DLL"/>
      </shimref>
    - <shimref name="VirtualRegistry" shimtagid="28200" commandline="TRUEMOTION20">
      - <shim name="VirtualRegistry" fixid="8e412efc-5b34-4c46-9bb4-71f7290efe3f" description_rcid="0x0000eb2b"
        general="set" dllfile="AcLayers.DLL">
          <include module="OPENGL32.DLL"/>
          <include module="DEVENUM.DLL"/>
          <include module="MSVFW32.DLL"/>
          <include module="SHLWAPI.DLL"/>
        </shim>
      </shimref>
    </exe>
</shimdb>
```

As a final example, to show how the pattern matching rules allow for a pattern sequence with gaps, the byte pattern of "ff 15 20 90 ?? ?? 89 1e" is scanned for at the RVA of 0x4fe5. The '??' are just wildcards in the notation above. This wildcard sequence is implemented, in this case, by using a pair of 'match' patterns at the appropriate RVA offsets to create the gap for the wildcards. This pair of match entries is followed by one 'replace' pattern that covers the full size covered by the match-pair and substitutes NOPs in their place.

```
<patch name="NetManageViewNowTN3270" fixid="7269485b-3f58-443f-b414-68ea92795df4">
 <patchbits>
  <patch_translated module="TCPCONN.DLL" action="match" rva="0x00004fe5">ff 15 20 90</patch_translated>
  <patch_translated module="TCPCONN.DLL" action="match" rva="0x00004feb">89 1e</patch_translated>
  <patch_translated module="TCPCONN.DLL" action="replace" rva="0x00004fe5">90 90 90 90 90 90 90 90</patch_translated>
 </patchbits>
</patch>
```

Pattern: ff 15 20 90 ?? ?? 89 1e with nops

Using various combinations of 'match/replace' entries, it is relatively straight forward to come up with any number of patterns to filter and act on. While not strictly necessary, a companion part of the Application Compatibility architecture is creating hot-patch points (or stubs) within a binary for each program or library entry point.

3.6.1 Microsoft Hot-Patching

Microsoft designs some of their functions to be dynamically hot-patched. This was first seen in the early examples of 32bit functions using the byte pattern "8b ff .." at the beginning of the function. Further, the function was preceded by 5 NOPs (0x90) or breakpoints (0xcc) bytes. In fact, the Visual Studio development platform from Microsoft allows developers to build binaries with hot-patching built in as a normal course, using the /hotpatch and /functionpadmin options during compiling and linking, respectively. Since the /hotpatch option only guarantees that each function's first instruction is at least 2 bytes, the "8b ff" pattern is seen when the function starts with a 1 byte instruction. The NOP byte sequence is shown below, with the 2 byte pad added by the /hotpatch compile option:

```
90      nop
90      nop
90      nop             } Pad for jump
90      nop
90      nop
8b ff   mov    edi   edi    Function start here
55      push   ebp
8b ec   mov    ebp   esp
```

The function above starts with the byte sequence (8b ff), which translates to moving the contents of the EDI register to itself. While this is a completely meaningless statement, it acts as filler bytes. From a hot-patch standpoint, these two filler bytes can be used by replacing them with a two byte jump instruction that jumps backward 5 bytes to redirect control to the five bytes of patch space that comes immediately before the start of each function. During the hot-patch operation, the five NOP bytes (or breakpoint bytes if using 0xcc) are replaced with a full jump instruction that can go anywhere in the code execution space (a 32 bit operating system is assumed here). So if one was to do a hot-patch and call some other routine, something like this could be done. Below is what the hot-patch operation would result in if wishing to JMP to address 0xdebf9.   The arrow below shows the start of the original function.

```
e9 f4 eb 0d 00   jmp   0x000debf9 ; any relative 32 bit addr
eb f9 ←          jmp   0xfb       ; jmp -5 bytes
55               push  ebp
8b ec            mov   ebp  esp
```

3.6.2 Scanning for Patch Patterns

To assist in searches for patches, one uses the ***-patchbytes*** option. The argument is the sequence of bytes one would like to find. The bytes are represented by hexadecimal notation and each byte is separated with a space. The entire sequence of bytes is then encompassed in double quotes. To look for a certain patch, it is useful to understand assembly language, since the byte sequence could represent the mnemonic opcodes used in the patch.

3.7 PE Metadata

When it comes to finding if a fix or patch targets a particular PE file, one needs access to the PE metadata to see if there is a match.   Shims includes an option ***-pe <filename> -stats*** for looking at some of the more common PE metadata used in the matching syntax. Below is the type of data this option produces.

```
"cmdline: shims64 -pe c:\Windows\notepad.exe -stats"

Source file       | c:\Windows\notepad.exe
CompanyName       | Microsoft Corporation
CompileTimestamp  | 0x4a5bc9b3 [07/13/2009 23:56:35 UTC]
FileDescription   | Notepad
FileOS            | nt, win32
FileSize          | 0x0002f400
FileType          | app
FileVersion       | 6.1.7600.163
Filename          | notepad.exe
InternalName      | Notepad
LegalCopyright    | © Microsoft Corporation. All rights reserved.
LinkerVersion     | 0x00090000
OSMajorVersion    | 0x00000006
OSMinorVersion    | 0x00000001
OriginalFilename  | NOTEPAD.EXE
PEChecksum        | 0x0003e749
ProductName       | Microsoft® Windows® Operating System
ProductVersion    | 6.1.7600.16385
```

Pulling PE metadata that can be used for shim matching

Similar to the SDB stats, this option also allows one to use the options: ***-pipe, -csv, -csv_separator, -dateformat, -timeformat***. The ***-pipe*** option is useful if wishing to pull many PE file matching stats in one run.

3.7.1 Matching PE Metadata with Shim Entries

One of the requirements of the Application Compatibility framework is to scan the metadata in every PE file during their load operation and compare it to any of the Shim Databases active on the system at that time. This is required to see if an executable, DL,L or driver PE file needs to be considered for a fix-up operation. To test out this with the ***shims*** tool, there is an experimental ***-match*** option to take in a desired PE file with companion Shim database to see if any entries in the Shim database target this particular PE file. Since this option only covers some of the parameters identified in the Shim Database used for matching, it should be considered prototype in nature and the results should not be considered definitive.

3.8 Parsing Collections of SDB files

There are 3 basic options for parsing a collection of SDB files: (a) targeting a particular system volume, (b) targeting a Volume Shadow copy, and (c) targeting a directory and its subdirectories that has a collection of SDB file.

3.8.1 Targeting a System Volume

If desiring to just parse a system volume without the fuss of finding each Shim database, one can use the ***-partition <volume letter>*** option to look in the conventional locations for SDB databases. The volume letter would normally be the c: volume for a live system collect, but it can also be a mounted volume from a system image from another computer.

3.8.2 Targeting a Volume Shadow Copy

To target a Volume Shadow copy, use the ***-vss <#>*** option, where the <#> is the index of the targeted Volume Shadow. The shims tool will scan the registry for custom Shim database locations as well as look in the conventional locations to find SDB files and parse them all in one session.

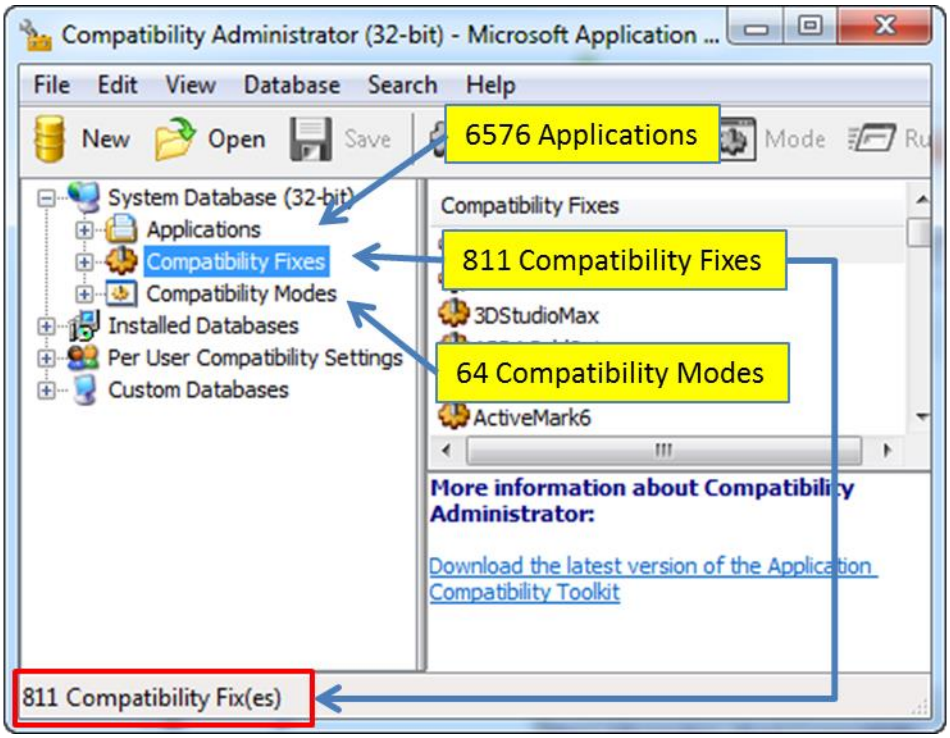
3.8.3 Targeting Directories

To target a specific directory (or a nested set of subdirectories within a parent director) that contains many SDB files, one can use the ***-pipe*** option. The first is used to gather statistics about all the SDB files and renders the output in CSV notation. The second pulls all the applications' entries from all the SDB files and renders the output in XML format.

```
dir e:\sdbfiles\*.sdb /b /s | shims -pipe -csv -stats > stats1.csv
dir e:\sdbfiles\*.sdb /b /s | shims -pipe -apps > apps.txt
```

4 Comparing the Application Compatibility Administrator to the *shims* tool

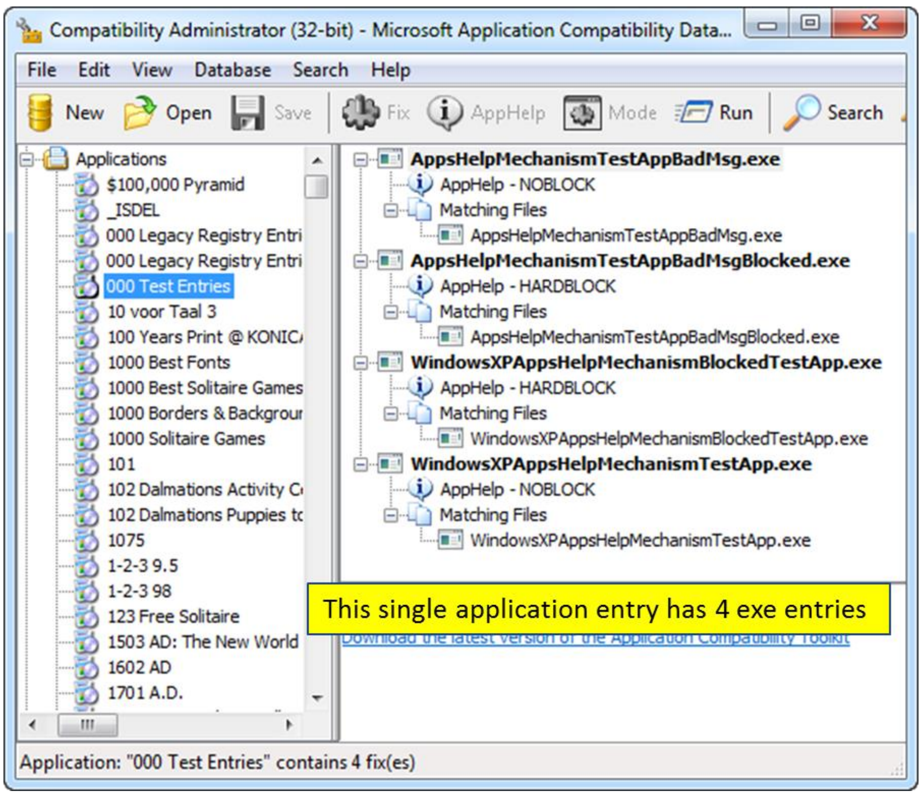
There are two Compatibility Administrator tools: (a) one for 32 bit databases and (b) one for 64 bit databases. Below is the 32 bit version of the tool, looking at the default 32 bit database on a Win7 operating system, 64 bit install. One can see the number of fixes, modes, and applications the 32 bit default database handles by looking at the stats in the lower bottom of the dialog window.



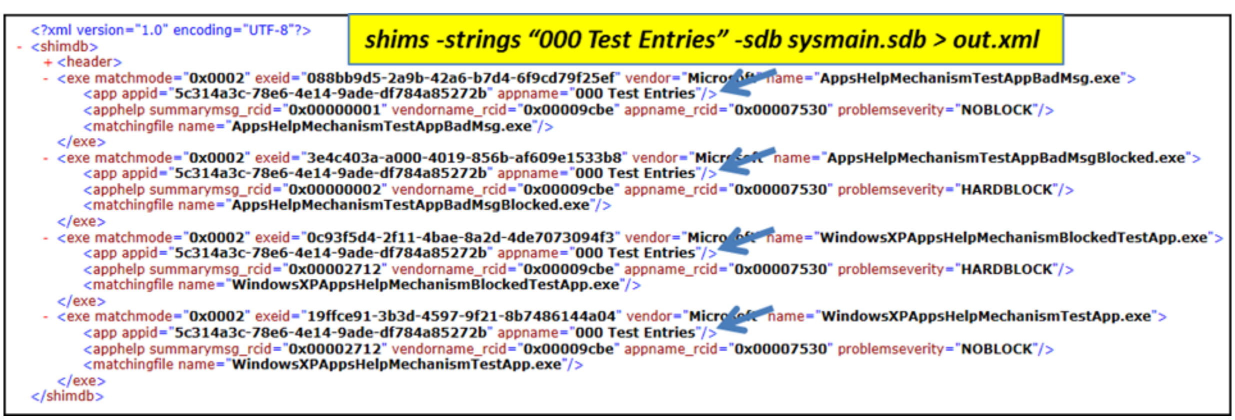
Running the ***shims*** tool against the same SDB file and using the ***-stats*** option, yields the following information.

```
Database Path/File         | E:\testcase\sdb\win7\AppPatch\sysmain.sdb
Database MD5               | 1d8c1280d38c526c7041e72db8d70dc1
Database SHA1              | da2e372481e6cdb450091794a58f294a46be1a46
File ModTime               | 04/12/2013 23:32:33.314 [UTC]
File AccessTime            | 02/25/2015 14:26:13.079 [UTC]
File CreateTime            | 02/25/2015 14:26:13.079 [UTC]
Database ModTime           | 04/12/2013 23:33:25.906 [UTC]
Compiler Version           | 2.1.0.3
Database Version           | 2.1
Database Internal Name     | Microsoft Windows Application Compatibility Fix Database
Database Platform          | 0x00000001
Database Identifier        | 11111111-1111-1111-1111-111111111111
appname                    | tag 0x6006:  6625 items    #apps = 6625
inexclude                  | tag 0x7003:  2419 items
shim                       | tag 0x7004:   662 items
patch                      | tag 0x7005:    35 items
exe                        | tag 0x7007: 13105 items
layer                      | tag 0x700b:    64 items    #layers = 64
flag                       | tag 0x7013:   149 items
context                    | tag 0x7018:     1 item
strings                    | tag 0x8801: 39202 items    #shims + #flags = 662 + 149 = 811
appid                      | tag 0x9011:  7013 items
```

Comparing the two outputs shows a couple of things: (a) the *Compatibility Fixes* in the Microsoft tool include both the entries of type *shim* entries and type *flag*, (b) the *Compatibility Mode* correlates to the entries of type *layer*, and (c) the Applications correlate to the entries of type *app name*. For the last one, the Application does not directly correlate to the entries of type exe. The reason for the mismatch is an *Application entry* can include 1 or more *exe* type entries (as well as other types). To see this, one can look at a few of the Application entries in the Compatibility Administrator tool. For the Application Entry '000 Test Entries' there contains four *exe* entries.



If one looks at the companion entry in the shims tools, one can do this by searching on the string "000 Test Entries" and examining the output. Below is an example of doing this and one can see the data that is in the Microsoft tool is a subset of the data in the ***shims*** tool.



5 Available Enumeration Options

Option	Extra	Description
-apps	**	Enumerate application category entries. This includes, but is not limited to, the following types: exe, packages, msi_packages.
-exes	**	Enumerate executable category entries (TAG_EXE)
-fixes	**	Enumerate the various types of fixes, including but not limited to: shims, patches, flags, layers, etc.
-shims	**	Enumerate shim category entries (TAG_SHIM).
-patches	**	Enumerate patch category entries (TAG_PATCH).
-layers	**	Enumerate layer category entries (TAG_LAYER).
-flags	**	Enumerate flag category entries (TAG_FLAG).
-tag	***	Enumerate the specified tag. Needs to be of type TAG_LIST_LIST. The syntax is ***-tag <#>***
-guids	**	Enumerate all GUIDs in the database along with the name associated with the GUID
-stringtable	**	Enumerate all the strings in the string table

6 Available Find Options

Option	Extra	Description
-strings	**	Search for the specified partial strings. If more than one partial string is listed, then use a pipe delimiter between each string and enclose the entire set of strings between double quotes. Will search using case-insensitive logic and will look for partial strings.
-guid	**	Search for the specified GUID. The GUID syntax is 11111111-1111-1111-1111-111111111111.
-tagids	***	Search for the specified tagid's. More than one tagid can be searched on as long as the entire set of tag identifiers are enclosed in quotes and delimited by the pipe character.
-patchbytes	***	Search for the specified byte pattern in the available patches
-match	***	Experimental. Used in conjunction with the ***-pe <PE File>*** option, to search the specified Shim DB for possible shims to the specified PE file.

7 Miscellaneous Options

Option	Extra	Description
-vss	***	Experimental. Parse SDB artifacts from Volume Shadow. The syntax is ***-vss <index number of shadow copy>***. Only applies to Windows Vista, Win7, Win8 and beyond. Does not apply to Windows XP.
-stats		Output a set of summary statistics about the Shim DB. Syntax is ***-sdb <db> -stats***. This option also is aware of the following sub-options: ***-reg <sw hive>*** (to pull stats from the hive as well), ***-csv*** (for CSV output), ***-csvl2t*** (for log2timeline output), ***-timeformat***, ***-dateformat***, and ***-csv_separator***.
-pe		Specifies the target file is a PE file vice a Shim DB file. Used in conjunction with the -stats option (eg. ***-pe <file> -stats***) and the ***-match*** option (eg. ***-pe <file> -match –sdb <shim db>***).
-pipe	**	Used to pipe files into the tool via STDIN (standard input). Each file passed in is parsed in sequence.

8 Sub Options that can be used with the *–stats* Option

Option	Extra	Description
-reg		Pull Application Compatibility data related to custom shim databases from the specified Software hive. Syntax is ***-reg <sw hive>***.
-csv		Outputs the data fields delimited by commas.
-csvl2t		Outputs the data fields in accordance with the log2timeline format.
-csv_separator	**	Used in conjunction with the ***-csv*** option to change the CSV separator from the default comma to something else. Syntax is ***-csv_separator "|"*** to change the CSV separator to the pipe character.
-no_whitespace	**	Used in conjunction with ***-csv*** option to remove any whitespace between the field value and the CSV separator.
-hostname	**	Option is used to populate the output records with a specified hostname. The syntax is ***-hostname <name to use>***.
-dateformat	**	Output the date using the specified format. Default behavior is ***-dateformat "mm/dd/yyyy"***. This allows more flexibility for a desired format. For example, one can use this to show year first, via ***"yyyy/mm/dd"*** or day first, via ***"dd/mm/yyyy"***, or only show 2 digit years, via the ***"mm/dd/yy"***. The restriction with this option is the forward slash (/) symbol needs to separate month, day and year and the month is

		in digit (1-12) form versus abbreviated name form.
-timeformat	**	Output the time using the specified format. Default behavior is ***-timeformat "hh:mm:ss.xxx"*** One can adjust the format to microseconds, via ***"hh:mm:ss.xxxxxx"*** or nanoseconds, via ***"hh:mm:ss.xxxxxxxxx"***, or no fractional seconds, via ***"hh:mm:ss"***. The restrictions with this option is that a colon (:) symbol needs to separate hours, minutes and seconds, a period (.) symbol needs to separate the seconds and fractional seconds, and the repeating symbol 'x' is used to represent number of fractional seconds. (Note: the fractional seconds applies only to those time formats that have the appropriate precision available. The Windows internal filetime has, for example, 100 nsec unit precision available.

9 Authentication and the License File

This tool has authentication built into the binary. There are two authentication mechanisms: (a) the digital certificate embedded into the binary and (b) the runtime authentication. For the first method, only the Windows and Mac OS-X (if available) versions have been signed by an X-509 digital code signing certificate, which is validated by Windows (or OS-X) during operation. If the binary has been tampered with, the digital certificate will be invalidated.

For the second (runtime authentication) method, the authentication does two things: (a) validates that the tool has a valid license and (b) validates the tool's binary has not been corrupted. The license needs to be in the same directory of the tool for it to authenticate. Furthermore, any modification to the license, either to its name or contents, will invalidate the license. The runtime binary validation hashes the executable that is running and fails the authentication if it detects any modifications.

10 References

1. Microsoft Application Compatibility Toolkit: https://msdn.microsoft.com/en-us/library/windows/desktop/dd562082(v=vs.85).aspx
2. Various MSDN articles, including but not limited to:
    a. Application Compatibility Database: http://msdn.microsoft.com/en-us/library/bb432182(v=vs.85).aspx
    b. Tag Types: http://msdn2.microsoft.com/en-us/library/bb432490
    c. Tags: http://msdn.microsoft.com/en-us/library/bb432487
3. Secrets of the Application Compatibility Database (SDB) parts 1-4, by Alex Ionescu. Ref: http://www.alex-ionescu.com/.