To SDB, Or Not To SDB: FIN7 Leveraging Shim Databases for Persistence

By Mandiant

Published: 2017-05-03 · Archived: 2026-04-05 14:12:23 UTC

Written by: Matthew McWhirt, Jon Erickson, DJ Palombo

In 2017, Mandiant responded to multiple incidents we attribute to FIN7, a financially motivated threat group associated with malicious operations dating back to 2015. Throughout the various environments, FIN7 leveraged the CARBANAK backdoor, which this group has used in previous operations.

A unique aspect of the incidents was how the group installed the CARBANAK backdoor for persistent access. Mandiant identified that the group leveraged an application shim database to achieve persistence on systems in multiple environments. The shim injected a malicious in-memory patch into the Services Control Manager ("services.exe") process, and then spawned a CARBANAK backdoor process.

Mandiant identified that FIN7 also used this technique to install a payment card harvesting utility for persistent access. This was a departure from FIN7's previous approach of installing a malicious Windows service for process injection and persistent access.

Application Compatibility Shims Background

According to Microsoft, an application compatibility shim is a small library that transparently intercepts an API (via hooking), changes the parameters passed, handles the operation itself, or redirects the operation elsewhere, such as additional code stored on a system. Today, shims are mainly used for compatibility purposes for legacy applications. While shims serve a legitimate purpose, they can also be used in a malicious manner. Mandiant consultants previously discussed shim databases at both BruCon and BlackHat.

Shim Database Registration

There are multiple ways to register a shim database on a system. One technique is to use the built-in "sdbinst.exe" command line tool. Figure 1 displays the two registry keys created when a shim is registered with the "sdbinst.exe" utility.

```
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Custom
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\InstalledSDB
```

Figure 1: Shim database registry keys

Once a shim database has been registered on a system, the shim database file (".sdb" file extension) will be copied to the "C:\Windows\AppPatch\Custom" directory for 32-bit shims or "C:\Windows\AppPatch\Custom\Custom64" directory for 64-bit shims.

Malicious Shim Database Installation

To install and register the malicious shim database on a system, FIN7 used a custom Base64 encoded PowerShell script, which ran the "sdbinst.exe" utility to register a custom shim database file containing a patch onto a system. Figure 2 provides a decoded excerpt from a recovered FIN7 PowerShell script showing the parameters for this command.

```
s.y.s.t.e.m.3.2.\.s.d.b.i.n.s.t...e.x.e.....-.q. .-.p. ."%.s."
```

Figure 2: Excerpt from a FIN7 PowerShell script to install a custom shim

FIN7 used various naming conventions for the shim database files that were installed and registered on systems with the "sdbinst.exe" utility. A common observance was the creation of a shim database file with a ".tmp" file extension (Figure 3).

```
C:\Windows\Temp\sdbE376.tmp
```

Figure 3: Malicious shim database example

Upon registering the custom shim database on a system, a file named with a random GUID and an ".sdb" extension was written to the 64-bit shim database default directory, as shown in Figure 4. The registered shim database file had the same MD5 hash as the file that was initially created in the "C:\Windows\Temp" directory.

```
C:\Windows\AppPatch\Custom\Custom64\{8a2ccc5d-5332-7116-d3c1-a843714e9ad4}.sdb
```

Figure 4: Shim database after registration

In addition, specific registry keys were created that correlated to the shim database registration. Figure 5 shows the keys and values related to this shim installation.

Registry Key Path:
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\InstalledSDB\{8a2ccc5d-5332-7116-d3c1-a843714e9ad4}\

ValueName: DatabasePath
Text: C:\Windows\AppPatch\Custom\Custom64\{8a2ccc5d-5332-7116-d3c1-a843714e9ad4}.sdb

ValueName: DatabaseType
Text: 65536

ValueName: DatabaseDescription
Text: Microsoft KB2832077

ValueName: DatabaseInstallTimeStamp
Text: 131314968194067341

Registry Key Path:
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Custom\services.exe\

ValueName: {8a2ccc5d-5332-7116-d3c1-a843714e9ad4}.sdb
Text: 131314968194067341

Figure 5: Shim database registry keys

The database description used for the shim database registration, “Microsoft KB2832077” was interesting because this KB number was not a published Microsoft Knowledge Base patch. This description (shown in Figure 6) appeared in the listing of installed programs within the Windows Control Panel on the compromised system.

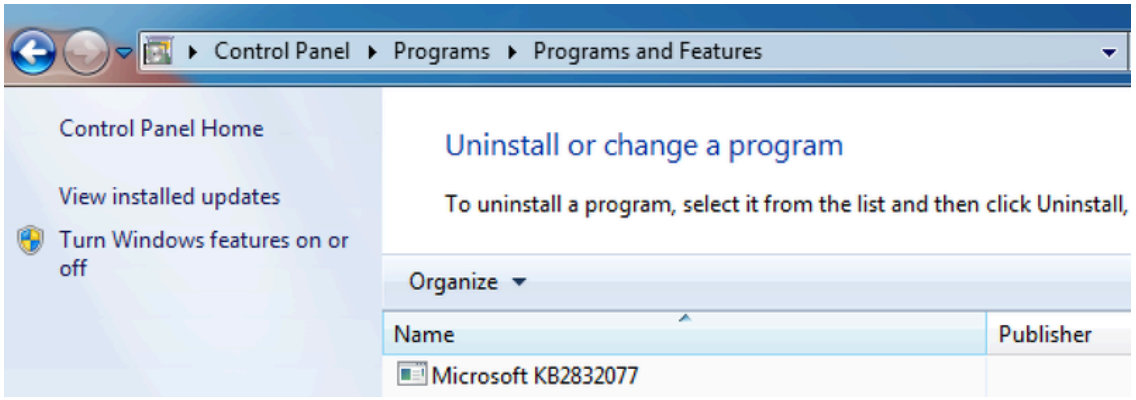

Figure 6: Shim database as an installed application

Malicious Shim Database Details

During the investigations, Mandiant observed that FIN7 used a custom shim database to patch both the 32-bit and 64-bit versions of “services.exe” with their CARBANAK payload. This occurred when the “services.exe” process executed at startup. The shim database file contained shellcode for a first stage loader that obtained an additional shellcode payload stored in a registry key. The second stage shellcode launched the CARBANAK DLL (stored in a registry key), which spawned an instance of Service Host (“svchost.exe”) and injected itself into that process.

Figure 7 shows a parsed shim database file that was leveraged by FIN7.

```
<DATABASE>
  <NAME type='stringref'>Microsoft KB2832077</NAME>
  <DATABASE_ID type='guid'>8a2ccc5d-5332-7116-d3c1-a843714e9ad4</DATABASE_ID>
  <OS_PLATFORM type='integer'>0x2</OS_PLATFORM>
  <PATCH>
    <NAME type='stringref'>ServicesFix01</NAME>
    <PATCH_BITS type='hex'>0200<REDACTED>0000</PATCH_BITS>
  </PATCH>
  <EXE>
    <NAME type='stringref'>services.exe</NAME>
    <APP_NAME type='stringref'>Microsoft Services</APP_NAME>
    <EXE_ID type='hex'>a3ec6475-d421-0249-a69a-bb2dad06e581</EXE_ID>
    <MATCHING_FILE>
      <NAME type='stringref'>services.exe</NAME>
      <COMPANY_NAME type='stringref'>Microsoft Corporation</COMPANY_NAME>
    </MATCHING_FILE>
    <PATCH_REF>
      <NAME type='stringref'>ServicesFix01</NAME>
      <PATCH_TAGID type='integer'>0x60</PATCH_TAGID>
    </PATCH_REF>
  </EXE>
</DATABASE>
```

Figure 7: Parsed shim database file

For the first stage loader, the patch overwrote the “ScRegisterTCPEndpoint” function at relative virtual address (RVA) “0x0001407c” within the services.exe process with the malicious shellcode from the shim database file.

The new “ScRegisterTCPEndpoint” function (shellcode) contained a reference to the path of “\REGISTRY\MACHINE\SOFTWARE\Microsoft\DRM”, which is a registry location where additional malicious shellcode and the CARBANAK DLL payload was stored on the system.

Figure 8 provides an excerpt of the parsed patch structure within the recovered shim database file.

```
opcode: PATCH_REPLACE
module name: services.exe
rva: 0x0001407c
unk: 0x00000000
payload:
  00000000: 40 55 53 57 48 8D 6C 24  B9 48 81 EC 90 00 00 00  @USWH.l$.H.....
  <REDACTED>
  00000220: 7C 24 18 C3 90 90 90 E8  52 00 00 00 5C 00 52 00  |$.....R...\.R.
  00000230: 45 00 47 00 49 00 53 00  54 00 52 00 59 00 5C 00  E.G.I.S.T.R.Y.\.
  00000240: 4D 00 41 00 43 00 48 00  49 00 4E 00 45 00 5C 00  M.A.C.H.I.N.E.\.
  00000250: 53 00 4F 00 46 00 54 00  57 00 41 00 52 00 45 00  S.O.F.T.W.A.R.E.
  00000260: 5C 00 4D 00 69 00 63 00  72 00 6F 00 73 00 6F 00  \.M.i.c.r.o.s.o.
  00000270: 66 00 74 00 5C 00 44 00  52 00 4D 00 00 00 58 C3  f.t.\.D.R.M...X.
```

Figure 8: Parsed patch structure from the shim database file

The shellcode stored within the registry path “HKLM\SOFTWARE\Microsoft\DRM” used the API function “RtlDecompressBuffer” to decompress the payload. It then slept for four minutes before calling the CARBANAK DLL payload's entry point on the system. Once loaded in memory, it created a new process named “svchost.exe” that contained the CARBANAK DLL.

Bringing it Together

Figure 9 provides a high-level overview of a shim database being leveraged as a persistent mechanism for utilizing an in-memory patch, injecting shellcode into the 64-bit version of “services.exe”.

Figure 9: Shim database code injection process

Detection

Mandiant recommends the following to detect malicious application shimming in an environment:

1. Monitor for new shim database files created in the default shim database directories of "C:\Windows\AppPatch\Custom" and "C:\Windows\AppPatch\Custom\Custom64"
2. Monitor for registry key creation and/or modification events for the keys of "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Custom" and "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\InstalledSDB"
3. Monitor process execution events and command line arguments for malicious use of the "sdbinst.exe" utility

Posted in

- Threat Intelligence
- Security & Identity

Source: https://www.fireeye.com/blog/threat-research/2017/05/fin7-shim-databases-persistence.html