Leveraging the Application Compatibility Cache in Forensic Investigations

By Andrew Davis, Associate Consultant, Mandiant

During keyword searches of compromised systems, Mandiant discovered known malicious file names in the Windows operating system registry. Further research showed the cache data was generated by the Windows Application Compatibility Database.[1] Along with these file names, other types of file metadata may be recovered such as file size, file last modified times, and last execution time depending on the operating system version. This data can be very useful during an incident response in order to identify what systems an attacker may have executed malware on and potentially, time information of when it may have occurred. A proof-of-concept tool was developed to extract this useful forensic evidence named Shim Cache Parser™, which can be found at https://github.com/mandiant/ShimCacheParser. This script will automatically determine the format of the cache data and output their contents. It supports a number of inputs including system registry hives, MIR XML, raw binary, or the current system's registry.

This cache is a component of the Application Compatibility Database, which is used by the Windows operating system to identify application compatibility issues. [1] The cache contains data related to this Windows feature and it is used for quick lookups to determine if modules require shimming for compatibility. [2] If a module is not found in this cache, lookups are performed which require accessing the database files stored on the system.[2] For more detail on how the Application Compatibility Database itself is implemented, Alex Ionescu has an excellent blog series examining it beginning here: http://www.alex-ionescu.com/?p=39. This document however, will focus on how the serialized cache data stored in the registry can be recovered and leveraged in a forensic investigation.

Compatibility lookups typically start in in CreateProcessInternelW function within kernel32.dll using the function BasepCheckBadApp.[2][3] File metadata of executed files may then be added to this cache for later lookups. Upon system shutdown, this data is then serialized into one of two registry keys containing "big data" values. We will examine the types of useful forensic data stored within the registry.

The Application Compatibility Shim Cache is implemented in one of four ways depending on the operating system. The type of data stored by the cache also varies per operating system. The following sections discuss the differences between these implementations. Note that this information does not cover every possible configuration of Windows and was based on limited research conducted by Mandiant. In addition, the included structure definitions are not their official names, but where derived from their usage.

Windows XP

On 32-bit versions of Windows XP, the cache is stored within shared memory in kernel32.dll and contains metadata of executed files. [2] *winlogin.exe* creates this section object during the boot process. The cache itself is maintained by API functions exported by kernel32 and synchronized between processes by the locking of a mutex named "ShimCacheMutex" whenever the cache is accessed. During system shutdown, the winlogon process will save the cache contents to the registry key named HKLM\SYSTEM\CurrentControlSet\Control\SessionManager\AppCompatibility\ AppCompatCache.

The cache, when recovered from the registry, contains a 400 byte header that must begin with the magic value of "0xDEADBEEF". This header contains the current number entries as well as indexes used by the cache manager. Each of the entry structures contain the full path of the executed file, the $STANDARD_INFORMATION last modified time of the file, file size, and the time the entry was last

updated. Shown below is the structure format of the Shim Cache entries found on 32-bit Windows XP systems. The cache will contain at most 96 of these entries and the entries are each 552 bytes in size.

```
//32-bit WinXP AppCompatCache Structure
typedef struct AppCompatCacheEntry_XP{
      WCHAR Path[MAX_PATH+4];
      FILETIME ftLastModTime;
      LARGE_INTEGER  qwFileSize;
      FILETIME ftLastUpdateTime;
} APPCOMPATCACHE_ENTRY32_XP;
```

New entries are created when an existing file's metadata has changed and re-executed, or a new file is executed. These new entries are added to the cache by use of the BaseUpdateAppcompatCache function. [3] Since the "LastUpdateTime" is updated when files are executed (regardless if the entry data has changed), this data may potentially indicate the last time that the file was executed on the system. In addition, since the relevant data is resident within each structure, the Windows XP cache entries may potentially be recovered from unallocated registry hive or disk space. 64-bit versions of Windows XP have the same cache format that 64-bit versions of Server 2003 have, which is described below.

Windows Server 2003

The Shim Cache changed a great deal beginning with Windows Server 2003. The last update time that was provided in Windows XP is no longer stored in the cache in Server 2003. The code that managed the cache was moved from user-land into the Windows kernel. When the system boots, the cache is read in from the registry by the kernel and on shutdown the cache contents are written back out to the registry path HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\AppCompatCache\ AppCompatCache. The active data is now also stored as a generic table that uses AVL self-balancing binary trees within kernel memory. The interface for managing the cache is also now exposed through a Native API named NtAppHelpCacheControl. [2]

When acquired from the registry, the data consists of a simple 8-byte header that must begin with the magic value "0xBADC0FFE". The next 4 bytes contain the number of cache entries in the data. Each entry consists of a serialized UNICODE_STRING structure, the $STANDARD_INFORMATION last modified time of the file when it was added, and file size. When the UNICODE_STRING structure is saved from memory to the registry, the string pointer is converted to an offset into the registry cache data. Shown below is the structure format of the Shim Cache entries found on 32-bit and 64-bit Windows Server 2003 systems. The cache will contain at most 512 of these entries and the entries are each 24 or 32 bytes in size depending on architecture.

```
//32-bit Win2k3 AppCompatCache Structure
typedef struct AppCompatCacheEntry32_2k3 {
        USHORT wLength;
        USHORT wMaximumLength;
        DWORD  dwPathOffset;
        FILETIME ftLastModTime;
        LARGE_INTEGER qwFileSize;
} APPCOMPATCACHE_ENTRY32_2k3;

//64-bit Win2k3 AppCompatCache Structure
typedef struct AppCompatCacheEntry64_2k3 {
        USHORT wLength;
        USHORT wMaximumLength;
        DWORD  dwPadding;
```

```
        QWORD  dwPathOffset;
        FILETIME ftLastModTime;
        LARGE_INTEGER qwFileSize;
} APPCOMPATCACHE_ENTRY64_2k3;
```

If the file metadata is updated, a new entry will be created when the modified file is executed. Additionally, if the file path name changes and it is executed again, a new entry will also be created.

Windows Vista and Windows Server 2008

The Shim Cache implementation on Windows Vista and Windows Server 2008 is similar to that on Windows Server 2003. [2] The serialized cache data is also stored in the HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\AppCompatCache\ AppCompatCache registry key. The format of the data recovered from the registry is also very similar with the exception of two entry fields. In place of the file sizes contained in the cache entries, there are now two 4-byte flags. Shown below is the structure format of the Shim Cache entries found on 32-bit and 64-bit Windows Vista and Windows Server 2008 systems. The cache will contain at most 1024 of these entries and they are each 24 or 32 bytes in size depending on architecture.

```
//32-bit Vista/2k8 AppCompatCache Entry Structure
typedef struct AppCompatCacheEntry32_Vista {
        USHORT wLength;
        USHORT wMaximumLength;
        DWORD  dwPathOffset;
        FILETIME ftLastModTime;
        DWORD  dwInsertFlags;
        DWORD  dwFlags;
} APPCOMPATCACHE_ENTRY32_VISTA;

//64-bit Vista/2k8 AppCompatCache Entry Structure
typedef struct AppCompatCacheEntry64_Vista {
        USHORT wLength;
        USHORT wMaximumLength;
        DWORD  dwPadding;
        QWORD  dwPathOffset;
        FILETIME ftLastModTime;
        DWORD  dwInsertFlags;
        DWORD  dwFlags;
} APPCOMPATCACHE_ENTRY64_VISTA;
```

Files can also be added to the cache that has not been executed. For example, when browsing a directory interactively, explorer.exe will attempt to parse executables within the directory and while doing so, add the file metadata to the Shim Cache using the Application Experience Lookup Service. It is currently unknown if any other Windows processes may add files to the cache with them having never been executed.

When processes are created by CSRSS, the BaseSrvSxsCheckManifestAndDotLocal function in basesrv.dll adds an entry to the Shim Cache and bitwise OR's the "InsertFlags" field with a value of 2. The true purpose of this flag is currently unknown however, during testing, it was possible to identify ".exe" files that were executed by testing if this bit flag was set. This is still being tested for consistency and possible scenarios where this flag may be set from other sources. The purpose of the second flag field is currently unknown but is typically set to zero.

Windows 7 and Windows Server 2008 R2

The format of the Shim Cache data underwent a significant change in Windows 7 and Windows Server 2008 R2. The first 8 bytes of the header contain the magic value "0xBADC0FEE" and the number of entries contained in the cache. The remaining 120 bytes of the header contain cache stats maintained by the kernel. Each entry consists of a serialized UNICODE_STRING structure, the $STANDARD_INFORMATION last modified time of the file when it was added, flags, and an optional offset to an additional data structure. Shown below is the structure format of the cache entries found on 32-bit and 64-bit Windows 7 and Server 2008 R2 systems. The cache will contain at most 1024 of these entries and they are each 32 or 48 bytes in size depending on architecture.

```
//32-bit Win7/2k8R2 AppCompatCache Entry Structure
typedef struct AppCompatCacheEntry32_Win7{
        USHORT wLength;
        USHORT wMaximumLength;
        DWORD  dwPathOffset;
        FILETIME ftLastModTime;
        DWORD dwInsertFlags;
        DWORD dwShimFlags;
        DWORD dwBlobSize;
        DWORD dwBlobOffset;
} APPCOMPATCACHE_ENTRY32_WIN7;

//64-bit Win7/2k8R2 AppCompatCache Entry Structure
typedef struct AppCompatCacheEntry64_Win7{
        USHORT wLength;
        USHORT wMaximumLength;
        DWORD dwPadding;
        QWORD  dwPathOffset;
        FILETIME ftLastModTime;
        DWORD dwInsertFlags;
        DWORD dwShimFlags;
        QWORD qwBlobSize;
        QWORD qwBlobOffset;
} APPCOMPATCACHE_ENTRY64_WIN7;
```

The "InsertFlags" values are updated from several sources, such as apphelp.dll and aelupsvc.dll when files are accessed by explorer or executed. Similar to Windows Vista and Windows Server 2008, one of these flags is set when processes are created by CSRSS. The function BaseSrvSxsCheckManifestAndDotLocal in sxssrv.dll will bitwise OR the "InsertFlags" field with the value of 2 when processes are created. The true purpose of this flag is currently unknown however, during testing, it was possible to identify if the ".exe" files contained within the cache were executed or not by testing if this bit flag was set. This is still being tested for consistency and possible scenarios where this flag may be set from other sources. The "ShimFlags" field may be set within apphelp.dll and is related to the Compatibility Database. The "BlobSize" and "BlobOffset" fields purpose are currently unknown and are typically both set to zero. The "BlobOffset" is an offset to an opaque data structure within the cache data, which is of size specified by the "BlobSize" field. These fields were usually found set when the file executed is a type of installer.

These structures stored in the registry can be helpful to forensic investigators and the temporal information may assist in time line analysis of a compromised system. While this data will not show every file that executed on a system, it may at least provide a window for seeing where and when an attacker executed their malware.

Mandiant would like to thank Josh Homan, an analyst with the SAIC CIRT, who contributed to this research.

---

[1] http://msdn.microsoft.com/en-us/library/bb432182(v=vs.85).aspx
[2] http://www.alex-ionescu.com/?p=41
[3] http://doxygen.reactos.org/d2/d93/appcache_8c_source.html

About MANDIANT

MANDIANT is the information security industry's leading provider of incident response and computer forensics solutions and services. MANDIANT provides products, professional services and education to Fortune 500 companies, financial institutions, government agencies, domestic and foreign police departments and leading U.S. law firms. To learn more about MANDIANT visit www.mandiant.com, read M-unition, the company blog: http://blog.mandiant.com, or follow on Twitter @MANDIANT.

	ADDRESS	PHONE	WEBSITE
MANDIANT	2318 Mill Road, Suite 500, Alexandria, VA 22314	800.647.7020	www.mandiant.com