3

ANALYZING THE SYSTEM HIVES

INFORMATION IN THIS CHAPTER

- Artifact Categories
- Security Hive
- SAM Hive
- System Hive
- Software Hive
- AmCache Hive

Introduction

While I was working on the second edition of this book, I read through the introduction of this chapter, and realized that, for the most part, nothing about the content really changed. Most of what I'd written in the first edition has remained, for the most part, true and valid. This time, however, I wanted to present the information in a different manner, one that I hope would eventually make sense to the reader. You see, I'd found that just presenting lists of Registry keys and values wasn't terribly useful; most volumes that attempt to do so are far too long to be any real use, and most analysts don't read them, and for those who attempt to do so tend to not remember a great deal of what they've read. As such, my thought was to present the material in "artifact categories," so that analysts who've decided upon their analysis goals could then go to the section that describes the particular category of artifact that they're interested in and begin there.

As with the previous edition, this chapter (and the one after it) will not be a comprehensive list of all possible Registry keys and values that would be of interest to an analyst, mapped into artifact categories. I don't think that something like that is possible, and in the time it takes for a book to be published, the information will likely have been extended, as research into the topic of Registry analysis is continual. My hope is to present a process for conducting analysis while focusing specifically on the Windows Registry, a process that can be used and carried forward to the entire operating system, as well as to other platforms. Throughout this chapter, I will attempt to provide use cases and examples, hopefully to

Windows Registry Forensics. http://dx.doi.org/10.1016/B978-0-12-803291-6.00003-6

illustrate how the artifacts (ie, keys and values) have been found to be valuable during an investigation. Ultimately, what I'd like to see is more analysts investigating artifacts within the Registry.

Artifact Categories

An approach I wanted to take in this edition was to present various Registry keys and values, as well as use cases and analysis processes, in terms of "artifact categories." No one seems to find any value in a long list (or a spreadsheet) of Registry keys and values, particularly if that list does not include any sort of context as to the specific value associated with each of the items listed. An alternative to such lists is to provide context around the value that a Registry key or value may have to an analyst by grouping them into categories that correlate to areas of interest during analysis. For example, one category is "auto-start," which comprises locations within the Registry that allow applications to start automatically when the system is booted or when a user logs in. As discussed in chapter "Processes and Tools," Microsoft's own "Autoruns" tool (found online at https://technet.microsoft.com/en-us/sysinternals/bb963902) is an excellent tool for locating artifacts within the "auto-start" category (both in the Registry and within the file system) on live systems.

Another category is "program execution," and this category comprises artifacts that indicate that programs and applications were launched on a system.

As you can see, categories can overlap. For example, "auto-start" locations are those locations within the Registry that permit applications to be started automatically when the system starts or when a user logs in. As such, when associated with a system boot or a user logging in, these artifacts can also be indicators of program execution. The "program execution" category also applies to indications of programs being launched in other ways, such as through user interaction.

Artifacts within a category can vary depending upon the version of Windows being examined. As many analysts are aware, earlier versions of Windows, such as [Windows2000] and XP, were not nearly as prolific as later versions. As the versions of Windows has progressed, analysts have become aware of more and more locations, particularly within the Registry, that seem to record a variety of artifacts. In some cases, the locations have persisted, changing slightly in the structure of the stored data, and therefore how it is parsed. Two examples of this include the *AppCompatCache* data (discussed later in this chapter) and the *UserAssist* data (discussed in chapter: Case Studies: User Hives).

As functionality has been added to Windows as the versions have progressed, this functionality has been associated with more and more artifacts being available to for digital forensic analysis. More information has been recorded regarding the system state and configuration, as well as both system and user activity (files accessed, images viewed, etc.).

Where do these categories come from? The categories appeared, in part, from the 2012 SANS DFIR poster (found online at http://digital-forensics.sans.org/blog/2012/06/18/sans-digital-forensics-and-incident-response-poster-released), as well as the work done by Corey Harrell; go to his blog at http://journeyintoir.blogspot.com and search for the term "artifact categories." I've also described artifact categories at considerable length on my blog, found at http://windowsir.blogspot.com. During July of 2013, I posted 14 articles to the blog, several of which were directly related to enumerating artifacts within specific categories. In some ways, these artifacts may seem to be completely arbitrary; however, the fact is that analysts such as Corey and I have looked back over the type of examinations we've conducted and incidents we've responded to and compiled lists of categories that apply broadly across the spectrum. That is not to say the categories discussed within this and the following chapter are the only categories that exist; they are simply exemplar categories used by some analysts. Each analyst is free, and even encouraged, to create their own artifact categories.

Security Hive

As with the other Registry hives we will be looking at in this chapter, the Security hive is one of the Registry hives within Windows that contains information that is specific and pertinent to the running and operations of the system itself; that is to say, the information available in this and other hives tends to pertain to the system, rather than to specific users on that system. That being said, the Security hive contains some useful information regarding the system configuration and settings, but there's also one particular place we can look for an artifact that falls within the "program execution" category.

As strange as it may seem, the Security hive also contains some interesting data that pertains to the "program execution" artifact category. This particular indicator comes from the authors of *The Art of Memory Forensics* (which can be found online at http://www.memoryanalysis.net/#!amf/cmg5) and is associated with the use of the tool GSecDump, a credential theft tool that is available online from http://www.truesec.se. However, a public

link is no longer available, reportedly due to the fact that "various social media have decided to classify them as dangerous"; you need to submit through the site's contact page in order to receive a link to the tool.

Anyone experienced with working on incident response engagements where a dedicated adversary has targeted an organization has likely seen systems on which GSecDump has been used. Around the time *The Art of Memory Forensics* became available, Jamie Levy shared with me that when the tool is used to dump the LSA secrets from a system, the LastWrite time of the *Policy/Secrets* key within the Security hive is modified. This is also listed on pages 552 and 553 of the *Memory Forensics* book. As soon as she shared this, I wrote the *secrets.pl* and *secrets_tln.pl* plugins, so that I could check the LastWrite time of the key or include the time stamp in a timeline, respectively. The following output from the *secrets.pl* plugin illustrates what the data looks like when extracted from the Security hive of a Windows 7 system:

```
Launching secrets v.20140730
secrets v.20140730
(Security) Get the last write time for the Policy\Secrets key
Policy\Secrets
LastWrite Time Thu Dec 30 21:42:38 2010 (UTC)
```

What this output shows is that the key was last modified on December 30, 2010. This can be very useful information, particularly when combined with other data sources that can illustrate what else occurred "around" that time, and provide additional context to the event itself. For example, while there may be legitimate reasons for this key being updated at the time in question, including additional data sources in your analysis may illustrated that the GSecDump tool had been run, which led to the key being modified.

Since incorporating this artifact into my analysis process, I've found that this "temporal fingerprint" has been a very reliable indicator of the fact that GSecDump had been executed on the system (with specific arguments, of course). As with other Registry artifacts, this one is most valuable when combined with other indicators, such as AppCompatCache data (discussed later in this chapter) and application prefetch files. However, this artifact has proved to be reliable even in the absence of application prefetch files.

At the time of this writing, I am aware of little data within the Security hive that might be relevant to an examination that has been discussed or shared publicly; however, there are a few keys and values that are of interest. One Registry key that can be found

on the Wikipedia page for "security identifiers" (SIDs) (found online *at* http://en.wikipedia.org/wiki/Security_Identifier) is the PolAcDms key. The "Default" value within this key contains the SID for the system and is a unique identifier for that system. As we will address later in this chapter, this information can be used to determine which users on a system are local users, and which are domain users, which is something that can be very useful with respect to a domain-connected (as opposed to stand-alone) system, and in particular a system with multiple domain trusts. Parsing the SID from the binary data is not an arduous task and is included in the RegRipper polacdms.pl plugin, the output of which (when run against a Security hive extracted from a Vista system) is shown below:

```
Launching polacdms v.20100531
PolAcDmS
Policy\PolAcDmS
LastWrite Time Fri Aug 31 15:14:53 2007 (UTC)
Machine SID: S-1-5-21-3831915772-716441274-3601324335
PolPrDmS
Policy\PolPrDmS
LastWrite Time Thu Nov 2 12:48:01 2006 (UTC)
Primary Domain SID: S-1-5-
```

Not only does this plugin extract and parse the machine SID from the PolAcDmS key, but it also extracts and parses the domain SID (for the domain to which the system was connected) from the PolPrDmS key. In this example, the Security hive was extracted from a stand-alone system used by a home user. In instances where the system was connected to a domain, the primary domain SID can be parsed from the "Default" value of that key and will be visible following "Primary Domain SID:". Later in this chapter we'll discuss local user accounts found in the SAM hive, as well as the ProfileList key from the Software hive, and see how an analyst can use this information.

Another key that is of use and interest to analysts from the Security hive is the "PolAdtEv" key. Parsing the binary data retrieved from this value is not a trivial task. However, our understanding of how this data can be parsed and understood can be helped along with Microsoft (MS) Knowledge Base (KB) article 246120 (found online *at* http://support.microsoft.com/en-us/kb/246120). As stated, this article applies to Windows NT 4.0, and there are only seven areas of auditing listed in the article. However, Windows XP has nine areas of auditing, as illustrated in Fig. 3.1.

In order to view the information illustrated in Fig. 3.1, all we need to do is open the Administrative Tools Control Panel applet and select the Local Security Policy shortcut. Another

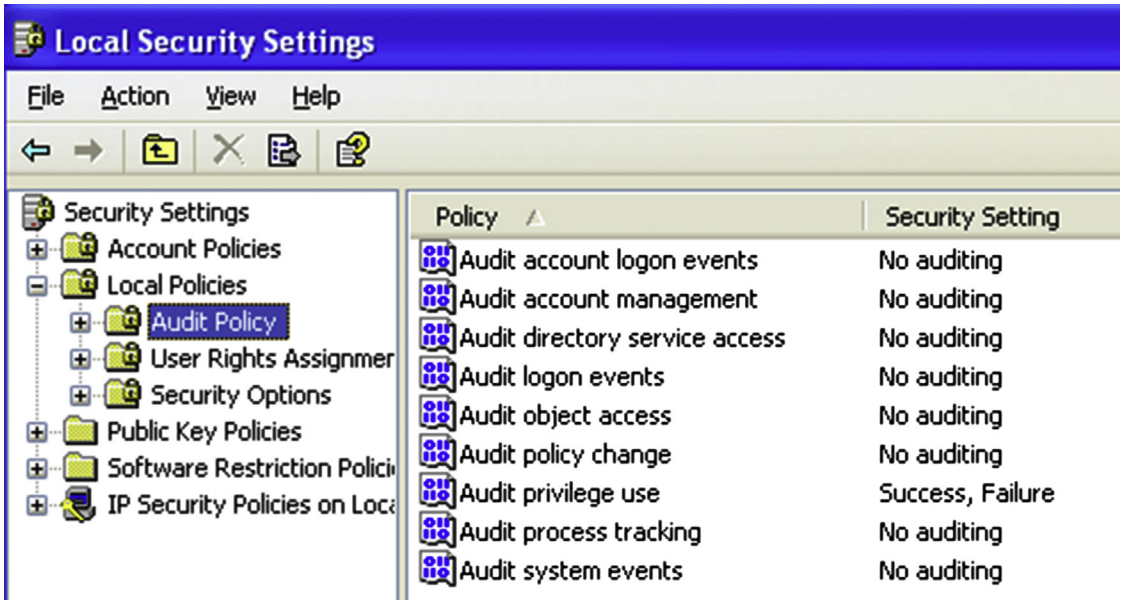


Figure 3.1 Audit policy via Local Security Settings (Windows XP).

way to view this information on Windows XP systems (one that is useful during live response, as it can be added to a batch file) is to run auditpol.exe; running it on that same live system, we see:

```
D:\tools>auditpol
Running ...
(X) Audit Enabled
System = No
Logon = No
Object Access = No
Privilege Use = Success and Failure
Process Tracking = No
Policy Change = No
Account Management = No
Directory Service Access = No
Account Logon = No
```

For Windows 7 and beyond systems, using auditpol.exe is a bit more involved, as the information being audited by default is much more extensive than on previous versions of Windows (specifically, Windows XP and 2003). An in-depth discussion of the various command line arguments for this tool is beyond the scope of this book, but an extensive list of the available options can be found online at https://technet.microsoft.com/en-us/library/cc731451.aspx.

We know how to extract the audit policy information from a live system, but what about from an acquired image? Using MS KB

article 246120 as a basis and toggling various settings on and off, we can see what modifications affect which areas of the data and develop an extrapolation of the data to our Windows XP system. Or, the RegRipper plugin *auditpol.pl* can be used to extract and parse the necessary information from Windows XP and 2003 systems, as illustrated below:

```
Launching auditpol v.20080327
auditpol
Policy\PolAdtEv
LastWrite Time Mon Jul 12 18:09:46 2010 (UTC)
Auditing is enabled.
 Audit System Events = N
 Audit Logon Events = N
 Audit Object Access = N
 Audit Privilege Use = S/F
 Audit Process Tracking = N
 Audit Policy Change = N
 Audit Account Management = N
 Audit Dir Service Access = N
 Audit Account Logon Events = N
```

This information can be very valuable, as it tells us a lot about the state of auditing on the system at the time that an image was acquired. First, the LastWrite time of the key lets us know when the settings were last modified (the time is listed in Universal Coordinated Time, or UTC). This can be very helpful in understanding why we see, or don't see, certain events in the Event Log, as well as provide an indication of when the audit policy was changed. There've been a number of examinations where I've created a timeline and seen clearly when the incident occurred, and seen that as a result of response and remediation actions taken by local IT staff, antivirus scans have been run and the audit policy has been updated, just prior to an image being acquired from the system.

Next, we see whether or not auditing is enabled on Windows XP and 2003 systems, and if so, which events are audited. This will also provide us with some indication of what we can expect to see in the Event Log. For example, if auditing of successful logon events isn't enabled, then we wouldn't expect to be able to see when someone logged into the system using a user account, either legitimately or as a result of compromised credentials. I have used this information during examinations quite extensively; during one instance, I used the fact that auditing for both successful logins and failed login attempts was enabled, but there were no indications of remote logins via the remote desktop protocol (RDP) to further illustrate that a particular user account had been accessed locally and used to view illegal images.

Tip

If successful use of privilege events are being audited (ie, Audit Privilege Use = S) on a Windows XP system, and a user modifies the system time via the "Date and Time" Control Panel applet (this can also be done by right-clicking on the time display on the Task Bar and choosing "Adjust Date/Time"), an event ID 577 appears in the Security Event Log, indicating the use of the "SeSystemtimePrivilege" privilege.

It is important to note that while this key and value exist on Windows Vista and higher systems, there has yet to be extensive testing of parsing the value on these systems. Fig. 3.2 illustrates the audit policy on a Windows 7 Ultimate system.

As you can see from Fig. 3.2, there are 9 areas of auditing listed, just as there are with Windows XP. In fact, the audit policies in Figs. 3.1 and 3.2 look very similar. However, the "Default" value for the PolAdtEv key on Windows XP contains data that is 44 bytes long, whereas on Windows Vista and 2008 systems that I've had access to, the data is 136 bytes long, and 138 bytes on some Windows 7 systems. In early December 2015, I ran across a reference to a document that outlined the structure of the value data for Vista and above systems, through Windows 10. The document even listed default values for the workstation and server operating systems at each level. I happened to have two systems, one

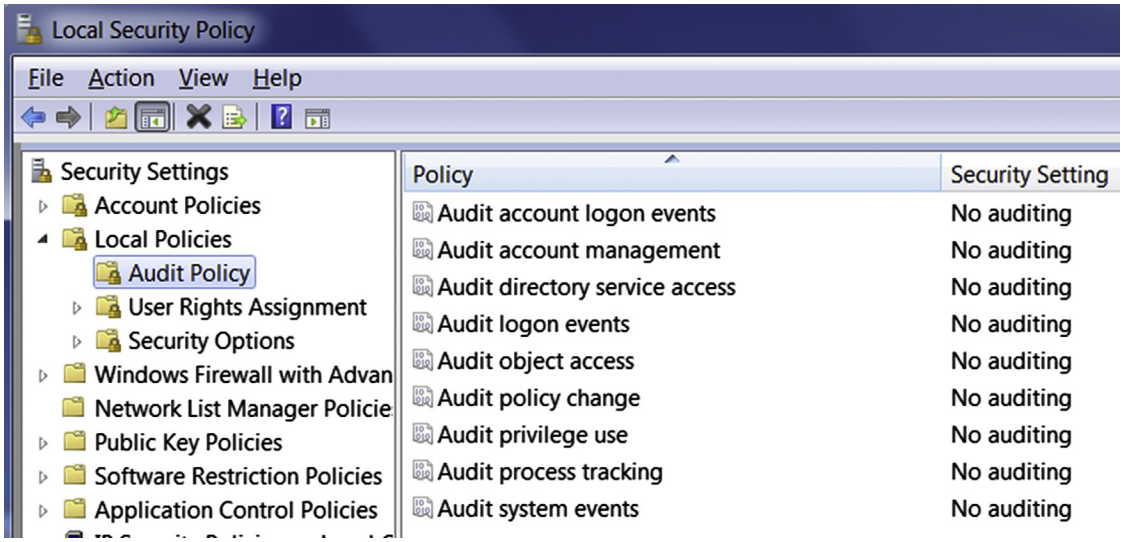


Figure 3.2 Audit policy on a Windows 7 Ultimate system.

Windows 7 and one Windows 10, so I ran the following command on each system and saved the output:

```
auditpol /get /category:*
```

I then used FTK Imager to export the Security hive off of each system and was able to write a RegRipper plugin that correctly parsed the data to display the effective audit policy from each hive file. Once this was done, I renamed the plugin that parses the data from Windows XP systems to *auditpol_xp.pl*, and named the new plugin *auditpol.pl*, and added both to the GitHub repository. But again, this plugin is based on an extremely limited sample set, and considerable testing needs to be performed in order to ensure that plugin that works correctly and is updated for other systems.

SAM Hive

Most administrators and analysts are aware that information about local users on a system is maintained in the SAM "database" or hive file. In corporate environments, the SAM hive may not have a great deal of useful information (that information may be found on a domain controller, for example), but for environments where the users will access systems using local accounts (home users, laptops, etc.), this hive file can provide a great deal of valuable data. However, this does not mean that the SAM hive is useless during incident response engagements within a corporate infrastructure, as there have been a number of times when I've seen an intruder create a local administrator account on a system, as opposed to a domain administrator account. We'll also see later in this chapter how the SAM hive can be used in other ways.

Tip

While information about user accounts local to the system is maintained in the SAM hive, the Software hive contains the ProfileList key (the full key path is *HKLM\Software\Microsoft\Windows NT\CurrentVersion\ProfileList*) which is a list of all the profiles on the system. This can show you remote or domain users who have logged into the system. We will discuss the ProfileList key later in this chapter.

The *samparse.pl* plugin extracts both user and group information from the SAM hive. Most of the information specific to each user is available beneath the *SAM\Domains\Account\Users\RID* key for each user, where *RID* is four zeros followed by the user's relative identifier (RID) in hexadecimal format. For example, the

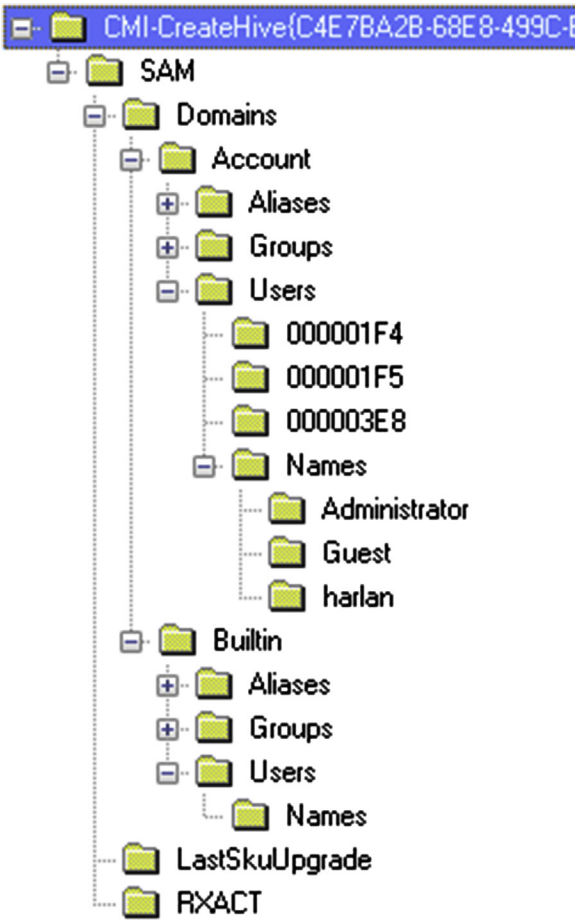


Figure 3.3 Windows 7 SAM hive, open in WRR.

Administrator account has an RID of 500, which would appear as 000001F4 in the SAM, as illustrated in Fig. 3.3.

The key for each user contains at least two values, F (contains several time stamps, etc.) and V (contains username, comment, etc.), which are binary data types and contain information about the user account. I have relied heavily on the source code for Peter Nordahl-Hagen's ntpasswd utility (found online at http://www.pogostick.net/~pnh/ntpasswd) to understand and decode this data into something usable. Sometimes within the user's key you will also find a value name "UserPasswordHint", which contains a string value if a user has entered a password hint. Many Windows systems (including Windows XP and Windows 7) allow the option to add a password hint to the user account, as illustrated in Fig. 3.4.

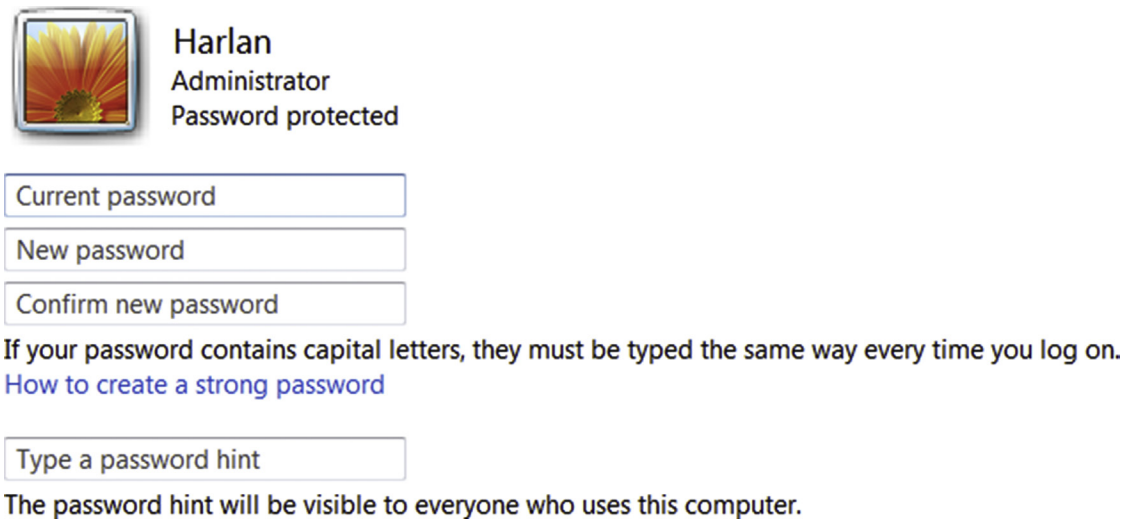


Figure 3.4 Add a password hint to a Windows 7 user account.

Tip

I was once examining an image and found an odd entry in the UserPasswordHint value within the SAM hive for a user. I used LiveView (found online at http://liveview.sourceforge.net) to create a virtual machine from the image and booted it. Once the login screen came up, I tried the string from the UserPasswordHint value and was able to log into the virtual machine. Be sure to NOT enter your password as the password hint!

An excerpt of the user information extracted from the F and V values in the SAM hive by the RegRipper *samparse.pl* plugin appears as follows:

```
  User Information
  ------------------------
  Username          : Administrator [500]
  Full Name         :
  User Comment      : Built-in account for administering the
computer/domain
  Account Created   : Tue Sep 11 14:26:13 2007 Z
  Last Login Date   : Fri Aug 31 15:52:42 2007 Z
  Pwd Reset Date    : Thu Nov 2 13:09:52 2006 Z
  Pwd Fail Date     : Never
  Login Count       : 4
   --> Password does not expire
   --> Account Disabled
   --> Normal user account
```

```
  Username          : Guest [501]
  Full Name         :
  User Comment      : Built-in account for guest access to
the computer/domain
  Account Created   : Tue Sep 11 14:26:13 2007 Z
  Last Login Date   : Never
  Pwd Reset Date    : Never
  Pwd Fail Date     : Never
  Login Count       : 0
   --> Password does not expire
   --> Account Disabled
   --> Password not required
   --> Normal user account
  Username          : Harlan [1000]
  Full Name         :
  User Comment      :
  Account Created   : Tue Sep 11 14:26:01 2007 Z
  Password Hint     : usual plus a bit more
  Last Login Date   : Mon Jan 12 12:41:35 2009 Z
  Pwd Reset Date    : Tue Sep 11 14:26:02 2007 Z
  Pwd Fail Date     : Fri Jul 11 19:54:07 2008 Z
  Login Count       : 16
   --> Password does not expire
   --> Password not required
   --> Normal user account
```

As you can see, a great deal of information is available in the user's keys within the SAM. This information can be used to demonstrate activity on the system (ie, Last Login Date, Login Count values) for a specific user account, as well as tell you a number of other things, such as if the Guest account has been enabled and used. You can also use the samparse_tln.pl RegRipper plugin to incorporate the time stamped information associated with the user accounts into a timeline in order to provide additional context to your investigation.

PASSWORD NOT REQUIRED

Of particular note in the output of the samparse.pl plugin is the entry for "Password not required". In some cases, analysts have taken this flag value to mean that the account does not have a password, and that is not the case. Rather, it means that password policies (length, complexity, etc.) applied to the user accounts on the system do not apply to those accounts for which the "Password not required" flag is set. I had posed the question to someone knowledgeable in this area and had been informed, "That specifies that the password-length and complexity policy settings do not apply to this user. If you do not set a password then you should be able to enable the account and logon with just the user account. If you set a password for the account, then you will need to provide that password at logon. Setting this flag on an existing account with

PASSWORD NOT REQUIRED—Cont'd

a password does not allow you to logon to the account without the password." This is somewhat corroborated by MS KB article 305144 (found online at http://support.microsoft.com/kb/305144), which indicates that enabling the flag means that a password is not required.

So, again…having the "Password Not Required" flag set on a user account does not mean that it does not have a password; instead, it means that any policies set regarding passwords will not apply to that account.

THE CASE OF THE DISAPPEARING USER ACCOUNT

I was examining an image sent to me, looking for indications of malicious activity. As is often the case, I didn't have a really good idea of the specific activity of interest, nor of the time frame in question. I had created a timeline of activity on the system, using the file system metadata, Prefetch file metadata, Event Log record data, etc., as sources and had started to see some unusual activity. In one instance, I found that a particular user account had logged in about a year prior to the image being acquired, but I didn't find any indication of that user account in the SAM. I used regslack.exe to extract deleted keys and values, and unallocated space from the SAM hive, and found an account with the same RID as the account I was interested in, but in the deleted data, the key had a different username associated with it. I also noted that the LastWrite time on the deleted key was very close to the time that the image of the system had been acquired. As it turned out, a system administrator had logged into the system, changed the name on the account when they heard that "someone was coming to acquire the system," and then deleted the account. This was confirmed by that same system administrator.

The *samparse.pl* plugin will also extract information about groups local to the system from the SAM hive, including the group name, comment, and the SIDs for the users in the group. An excerpt of this output from a Windows 10 Technical Preview system, installed as a virtual machine, is illustrated below:

```
  Group Name : Administrators [2]
  LastWrite : Tue Jan 20 16:17:15 2015 Z
  Group Comment : Administrators have complete and
unrestricted access to the computer/domain
  Users :
   S-1-5-21-3106756646-2393192417-535628121-500
   S-1-5-21-3106756646-2393192417-535628121-1000
  Group Name : Power Users [0]
  LastWrite : Tue Jan 20 19:08:25 2015 Z
  Group Comment : Power Users are included for backwards
compatibility and possess limited administrative powers
  Users : None
  Group Name : Cryptographic Operators [0]
```

```
   LastWrite : Tue Jan 20 19:08:25 2015 Z
   Group Comment : Members are authorized to perform
cryptographic operations.
   Users : None
```

As you can see from the sample output from the samparse.pl plugin, the samparse.pl plugin works on Windows systems up to Windows 10 Technical Preview (what was available at the time of this writing). The information derived from the plugin can be very helpful in determining the level of access that a particular user account had on a system at the time that system was acquired, in order to determine what actions that user could take on the system, such as submit Scheduled Tasks (which is one way that a user could obtain elevated privileges), etc.

Also, the *samparse.pl* plugin is very convenient as it allows you to obtain and view a great deal of local user and group information from a system, all in one easy-to-reference location.

Cracking User Passwords

There are a number of times during investigations where you would want to determine a user's password. For example, in a number of examinations, law enforcement officials have wanted to know if the user account had a password at all. In most instances, I have seen this sort of query associated with cases where something suspicious (or illegal) is associated with the user account of another family member, and law enforcement officials want to determine if the suspect had free access to that account; an account with no password is extremely vulnerable. In other cases, the "Password not required" flag in the user account settings (mentioned previously in this chapter) can be very confusing to some analysts, and determining if the user account had a password at all, and attempting to determine what that password is, is paramount to the investigation. Finally, there may be a time during an investigation where, after you've acquired an image of the system, you may want to boot the system, either the original system or the acquired image, which can be "booted" in a virtual environment via LiveView (found online at http://liveview.sourceforge.net) in order to "see" what the user saw or had access to while logged into the system.

There are a number of free, GUI-based password cracking tools available, such as Cain & Abel (available online at http://www.oxid.it/cain.html), OphCrack (found online at http://ophcrack.sourceforge.net), and John the Ripper (found online at http://www.openwall.com/john). Going into detail about how to use each of these tools is beyond the scope of the book, but don't worry, the programs are very easy and straightforward to use. While you can

use these tools to crack passwords, keep in mind that they can also be used to do a quick check to see if a user account *has* a password (as opposed to being blank).

Again, a detailed discussion of password cracking attacks or of the Cain or OphCrack applications is beyond the scope of this book. My purpose in mentioning the tools in this chapter has been to point to freeware tools that can be used to derive (and validate) information from Registry hive files; in this case, to illustrate information about user accounts extracted from the SAM database and to validate whether or not a user account actually has a password associated with it that needs to be typed in by a user. As I mentioned previously, simply determining whether or not an account has a password can be very valuable to an investigation.

System Hive

So far in our discussion in this chapter, we've touched a very little bit on how the System hive can be useful during an examination, with some references to information found in the hive. The System hive contains a great deal of configuration information about the system and devices that were included in and have been attached to it, so let's take a look at how to derive and interpret some of that data.

Throughout this section, as well as the rest of this chapter, I'm going to be presenting and discussing Registry keys and values that are most often seen, viewed, and accessed during incidents, and subsequently, during analysis. Neither this chapter nor this book is intended to be an all-inclusive listing of Registry keys, as that would be impossible and quite boring. Rather, I'd like to offer up some insight into specific keys and values, and how what you find (or, in some cases, don't find) can be used to further your examination.

Finding the "Current" ControlSet

From chapter "Registry Analysis," we know that there are portions of the Registry that are volatile, in that they only exist when the system is running. One such portion is the CurrentControlSet key in the System hive. Microsoft states in MS KB article 100010 (found online at http://support.microsoft.com/kb/100010) that a ControlSet, "contains system configuration information, such as device drivers and services." When we access the Registry on a live system, we may see two (or more) ControlSet keys (as illustrated in Fig. 3.5), in addition to the CurrentControlSet key.

Figure 3.5 SYSTEM hive via RegEdit, showing the CurrentControlSet.

Name	Type	Data
(Default)	REG_SZ	(value not set)
Current	REG_DWORD	0x00000001 (1)
Default	REG_DWORD	0x00000001 (1)
Failed	REG_DWORD	0x00000000 (0)
LastKnownGood	REG_DWORD	0x00000003 (3)

Figure 3.6 Contents of Select key in the System hive.

During a postmortem examination, we may need to determine which ControlSet was loaded as the CurrentControlSet when the system was running. In order to do so, all we need to do is view the values within the Select key in the System hive, as illustrated in Fig. 3.6.

Within the Select key, the Current value tells us which ControlSet was loaded as the CurrentControlSet when the system was running. This helps us understand a bit about the system state when it was running; for example, each ControlSet contains a list of Services installed on the system and how they are set to run (ie, automatically at boot, disabled, etc.), among other settings.

Note

All of the current RegRipper plugins that access the System hive will first check the "Current" value within the Select key and then extract information from the appropriate ControlSet based on the value data. This is simply a matter of preference and not a hard-and-fast requirement; plugins can be written to access all of the available ControlSets (I have seen System hives with three ControlSets listed) and search for/extract the desired information from each one. This may be useful for comparison, particularly if the LastWrite times on the keys themselves differ.

System Configuration Information

The System hive maintains a great deal of information that pertains to the configuration of the system, configuration settings that can have an impact on your investigation.

System Name

One of the first pieces of information I generally tend to look for within the System hive is the system name. I tend to do this in order to include the information in my case notes and documentation and to keep track of the systems I'm looking at, particularly if there are several systems involved in a particular case. Many times, this information can be helpful when correlating systems during a large breach, particularly ones that involved a great deal of lateral movement by the adversary. While we generally look for system IP address in logs, when looking at other artifacts such as the RunMRU or shellbags (both will be discussed in chapter: Case Studies: User Hives), we will often be able to track a user's movements using the system name.

IS THIS THE RIGHT SYSTEM?

Having the system name as part of your case documentation can be very important, as I've actually received the wrong system before. That's right…I've been asked to analyze a web server for signs of an intrusion and was provided the system name during the initial conversations, and the same system name was included on the chain of custody documentation. However, when I began my examination of the provided image, the system name was not correct. It turned out that the wrong image had been sent.

I've also heard of incidents where the right hard drive was sent to an analyst, but it was the wrong system. It turned out that between when the incident was identified and action was taken, an administrator had wiped the hard drive and completely reinstalled the operating system and applications and then copied the data back over. As such, the correct hard drive was provided to the analyst, but the actual system that needed to be analyzed was no longer available.

ClearPagefileAtShutdown

Some of the systems configuration settings on a Windows system may have a significant impact on an analyst's investigation; one such setting is the ClearPagefileAtShutdown value setting. This value is found within the *HKLM\System\CurrentControlSet\Control\Session Manager\Memory Management* key (see the section earlier in this chapter where we discussed how to determine the CurrentControlSet for a System hive extracted from the system

image). If the value exists and is set to "1", then the page file will be cleared when the system is shut down. This can impact an investigation, as the page file can be combed for strings and even carved for a variety of data that may prove to be valuable to the analyst.

Tip

More modern versions of Windows are capable of maintaining multiple paging files. Within the *Memory Management* key on Windows 7 and 10 systems, you will find values named "ExistingPageFiles" and "PagingFiles" that can contain a list of page files.

Network Interfaces

Much like other devices, information about the network interfaces available on the system is maintained in the System hive. The main path for information about the network interfaces available on a system is the *ControlSet00n\Services\Tcpip\Parameters\Interfaces* ("ControlSet00n" refers to the ControlSet marked as "Current" when the system is offline) key. Beneath this key, you'll find a number of subkeys whose names are globally unique identifiers (or GUIDs, pronounced *goo-idz*). Each of these subkeys refers to a specific interface, and the GUID names can be mapped to more easily readable names for the interfaces (see the "Network Cards" subsection later in this chapter).

The interface subkeys contain information about IP addresses assigned (static assignments or via DHCP), gateways, domains, as well as when DHCP leases were assigned, and when they terminate. This information can be extremely helpful during a wide variety of examinations, particularly when attempting to tie a particular system to entries found in router or web/FTP server logs. An excerpt of what this information looks like in the Registry is illustrated in Fig. 3.7.

The RegRipper plugin *nic2.pl* does a really good job of extracting this information and even goes so far as to translate some of the 32-bit time stamp values (LeaseObtainedTime, LeaseTerminatesTime, etc.) into something a bit more human readable.

Routes

One of the tricks that malware authors have used to "protect" their tools is to add entries to the hosts file so that critical assets (update sites for the operating system, applications, antivirus, etc.) cannot be reached. By forcing the query for a host or domain to resolve to a specific IP address, malware authors can inhibit the

DhcpDefaultGateway	REG_MULTI...	192.168.1.1
DhcpDomain	REG_SZ	chvlva.adelphia.net
DhcpIPAddress	REG_SZ	192.168.1.10
DhcpNameServer	REG_SZ	192.168.0.1
DhcpRetryStatus	REG_DWORD	0x00000000 (0)
DhcpRetryTime	REG_DWORD	0x0000a8bd (43197)
DhcpServer	REG_SZ	192.168.1.1
DhcpSubnetMask	REG_SZ	255.255.255.0
DhcpSubnetMaskOpt	REG_MULTI...	255.255.255.0
Domain	REG_SZ	
EnableDeadGWDetect	REG_DWORD	0x00000001 (1)
EnableDHCP	REG_DWORD	0x00000001 (1)
IPAddress	REG_MULTI...	0.0.0.0
IPAutoconfigurationAddress	REG_SZ	0.0.0.0
IPAutoconfigurationMask	REG_SZ	255.255.0.0
IPAutoconfigurationSeed	REG_DWORD	0x00000000 (0)
IsServerNapAware	REG_DWORD	0x00000000 (0)
Lease	REG_DWORD	0x00015180 (86400)
LeaseObtainedTime	REG_DWORD	0x4c739caa (1282645162)
LeaseTerminatesTime	REG_DWORD	0x4c74ee2a (1282731562)

Figure 3.7 Excerpt of network interface values (Windows XP).

functionality. After all, you wouldn't want the installed antimalware product to update itself and then detect the presence of your malware, would you?

This is also something that can be used legitimately. According to the MS KB article on name resolution order (found online at https://support.microsoft.com/en-us/kb/172218), after checking to see if a name is its own, a Windows system will then check the hosts file. System administrators can add entries that redirect traffic to specific sites, and even some antimalware and anti-spyware applications will modify this file to force known-bad hosts/domains

to resolve to the local host (ie, 127.0.0.1). Parents can also do this with Facebook and MySpace!

Another technique that can be used is to modify persistent routes on the system. One command that many incident responders run when collecting information is *route print*, which displays the current routing table for TCP/IP communications on the system. This facility also has the ability to add persistent routes that will remain in place between reboots through the *route add* command (more information about this command can be found online at https://technet.microsoft.com/en-us/library/cc757323(v=ws.10).aspx). If an added route is designated as "persistent" through the use of the "-p" switch, the command adds the routes to a Registry key within the System hive (which can be extracted using the *routes.pl* RegRipper plugin). Interestingly enough, malware such as Backdoor.Rohimafo (a description of this malware is available at the Symantec website) appears to add persistent routes to the system in order to prevent the system from accessing sites that may result in updates that allow the malware to be detected.

File System Settings

The System hive also maintains information about the configuration of the file system beneath the *ControlSet00n\Control\FileSystem* key, and there are several settings that may affect your analysis. For example, there is a value named "NtfsDisableLastAccessUpdate" (a description of the value can be found online at https://technet.microsoft.com/en-us/library/cc959914.aspx) which, back in the early days of Windows XP and 2003, was intended as a setting that could be used to enhance the performance of the system. The intention was that on high-volume file servers, disabling the updating of last access times on files would improve overall performance; however, this was an optional setting at the time, as the value did not exist by default.

Interestingly enough, one of the surprises with the release of Windows Vista was that not only did this value exist, but updating of last access times on files was disabled by default! Consider for a moment the effect that had on a lot of traditional computer forensic methodologies. Actually, I'd say that this has been very helpful, as it has forced us to look for alternative methods of demonstrating that a user accessed a file, and this search has proven to be extremely fruitful.

Beneath the same key is a value named "NtfsDisable8dot3NameCreation"; if this value is set to 1 (and the file system is NTFS), then the creation of short file names will be disabled. This may be an issue if you expect to see file names on the system similar to "PORTER~!.PPT" rather than "porter's latest widgets sales

presentation.ppt". Enabling this functionality tells the file system to not create the shorter file names.

Analysis Tip

Part of computer forensic analysis is not just recognizing what is out of place or unusual; it's also recognizing when some artifact should be present, but isn't.

Prefetch Settings

All Windows systems, beginning with Windows XP, have had the ability to perform both boot and application prefetching. The intent of this capability is to speed up the boot or application loading process, and the result of application prefetching for forensic investigators has been valuable information that illustrates that an application was executed.

Now, I said that Windows systems have the ability to perform both boot and application prefetching, but not all do. By default, only workstation versions of Windows (XP, Windows 7, and Windows 8/8.1/10) perform application prefetching. Server versions (Windows 2008 R2, Windows 2012) do not perform application prefetching by default.

A Windows system's ability to perform boot or application prefetching, or neither, or both, is controlled by a single value within the Registry; specifically, the "EnablePrefetcher" value beneath the *ControlSet00n\Control\Session Manager\Memory Management\PrefetchParameters* key. The RegRipper *prefetch.pl* plugin will extract the Prefetch configuration setting from the System hive and display it in a human-readable format.

AutoStart

Autostart settings are those that allow applications and programs to start with no interaction from the user beyond booting the system or simply logging in. Registry keys and values within this category are most often associated with malware, as the intention of the authors is to get their malware onto a system and to get it to start automatically when the system is started, with no interaction or notice from the user.

Windows Services

Perhaps the most referenced and analyzed pieces of information in the System hive, particularly during incident response

activities, are the Windows services. Windows services are programs that run automatically when the system is booted and are started by the system and with no interaction from the user (however, users with the appropriate privileges can install, start, and stop services). Windows services can be very useful; web and FTP servers, as well as DNS and DHCP servers, are all Windows services. However, the nature of Windows services (run automatically with no user interaction, as well as with elevated privileges) makes them a target for malware authors as well, and a great number of bits of malware install as Windows services.

Services on Windows systems can be extremely powerful; they generally run with elevated privileges and start without any interaction from the user beyond booting the system. Is there any wonder why services are targeted so often by malware authors and intruders? Not so much to exploit a vulnerability (yes, that does happen), but instead to use Windows services as a persistence mechanism, ensuring that the malware or backdoor or remote access Trojan (RAT) is started each time the system is booted.

Warning

Creating services (and other actions, such as submitting Scheduled Tasks) on Windows systems requires Administrator-level privileges; as such, the fact that new services are created tells you something about the level of access that the malware or the intruder had on the system. Analysts often see partial infections by malware, where the infection process was hindered by the fact that user context that was involved did not have Administrator privileges on the system. So while limiting user privileges can prevent or hamper the effects of a compromise, the flip side is that the artifacts of a compromise that you do find can tell you a lot about what may have happened.

In many cases, experienced incident responders will be able to look at a system Registry and "magically" pick out the obscure or malicious services. Some malware creates services with random names (either the service name itself, or the DisplayName value), so a quick look at the Registry is all it takes to find the offending service. Other techniques that can be used by incident responders and analysts are to look for services that have odd paths to the executable images (that is, point to executables in the ProgramData folder, within a user's profile folder, etc.) or do not have a *Description* value; many legitimate services have descriptions, and some of them can be long, depending on the vendor. The bad guys learned from these techniques and began using services names that looked a bit more legitimate and began filling in the various values to make the service itself look more legitimate, at

least when the values were seen via a Registry viewer. For instance, there have been Description values that appear legitimate, and I have seen others that have had some misspellings (ie, "down load" spelled as two words) which was enough for me to take a closer look.

Another value beneath a service key that can provide a good deal of context and perspective to an examination is the Start value. A description of the various Start values can be found in MS KB article 103000 (found online at https://support.microsoft.com/en-us/kb/103000). In most instances, you'd expect a Start value of "0x02", indicating that the service is autoloaded or run automatically. Every now and again, I see malware services that have a Start value of 0x03, which indicates that they're set to start manually, meaning that a user must do something, take some action, for the service to be started. This can be critical when attempting to determine the "window of compromise" for a customer. Basically, if the malware service was installed with a Start value of 0x03, started and run, and then the system shut down, when the system was started again, the service would not start automatically. This may play a significant role in your examination.

RegRipper includes a number of plugins for extracting service key information from the System hive, and to be honest, because RegRipper is open source, there's really no limit to how you parse and display the information. Most of the plugins will start off by locating the ControlSet00*n* marked "Current" in the Select key of the System hive; however, this is not a hard-and-fast requirement. Plugins can be written that will display the same key/value information from all of the available ControlSets, or you can write a plugin to display the information from both ControlSets if the information itself is not the same in both (or all…I've seen hives with more than two ControlSets) locations.

Warning

I was performing emergency incident response for an organization that had some issues with malware. The malware wasn't widespread and didn't seem to be infecting systems; in fact, all indications were that the malware was isolated to just a few systems, and the organization simply wanted it gone. Using regedit.exe, I found a service that appeared to be suspicious, deleted it and rebooted the system…but the malware wasn't gone. In this case, the malware used *two* services for persistence…one that was the malware, and the other that checked for the existence of the malware, and if it didn't find it, installed it.

During another incident response engagement, we had located a malicious service that had a Start value of 0x02, and would dump the virtual memory from credit card back office processing software and collect track data from the

Continued

Warning—Cont'd

memory dump. Using some commercial tools, we found that the code within the executable for the service included a *sleep()* function; it used this because when the system is first started, there is no credit card data in memory. Instead, it would read the contents of a register, shift the value to the right four times, and then *sleep()* that number of seconds; based on other artifacts, it appeared at one point to *sleep()* for several days. Under the circumstances, understanding the interaction of the malware on the system, taking all factors into account, helped us provide the customer with a more accurate window of compromise.

In another instance, the first real indicator I'd seen of malicious services was an Event Log record. The source was "Service Control Manager" and the event ID was 7035, indicating that a service had started…even though our findings indicated that the system had been running for quite some time. Further examination indicated that the service was set to start when the system was booted. All other information about the service appeared to be legitimate, even down to the executable file appearing to be a legitimate Windows file.

The point is that it's not always easy to locate a suspicious service or process, particularly when the bad guy is trying really hard to not be discovered.

Not long ago, the bad guys were found to be employing an even trickier technique to hide and maintain the persistence of their malware or backdoors. Instead of creating a service with an ImagePath value that pointed directly to the malware executable file, what they were doing was creating a service that was loaded by the venerable svchost.exe process. Per MS KB article 314056 (found online at http://support.microsoft.com/kb/314056), the Svchost.exe process is essentially a "service host," in that multiple copies of svchost.exe can be running, each "hosting" multiple services running from DLLs. When the svchost.exe process starts, it reads through the Registry to see which services it needs to be running, under which instances. Services that run under svchost.exe have ImagePath values that contain references to svchost.exe itself, such as:

```
%SystemRoot%\system32\svchost.exe -k netsvcs
```

Then, beneath the service key, there will be a "Parameters" subkey that contains a value named "ServiceDll", which points to the DLL from which the service is run. Conficker is an example of a worm that used this technique for persistence. By creating a service in this manner, it makes the malware a bit harder to find, but not impossible. All we have to do is drop down to the Parameters subkey beneath the malicious service, and the ServiceDll value will point us to the offending malware. Some of the things we'd want to look for with respect to the listed DLL are unusual paths (ie, the path name includes "temp", etc.), odd looking or apparently names for the DLL itself, etc. Looking at the referenced DLL

itself, misspelled or missing file version information, evidence of the use of a packer to obfuscate the executable code, etc., are indicators of possibly malicious files.

Note

The *Microsoft\Windows NT\CurrentVersion\SvcHost* key within the Software hive can also provide information about services that should be running "under" svchost.exe.

The RegRipper *svcdll.pl* plugin combs through the services keys within the System hive and displays all of those that are loaded by svchost.exe, sorting them based on their key LastWrite times. The *svchost.pl* plugin extracts the values and data from the SvcHost key within the Software hive. Because RegRipper and its plugins are open source, anyone with a modicum of Perl programming skill (or even the ability to use copy-paste) can easily create new plugins that perform different functions or display the output in a more meaningful manner. See chapter "RegRipper" of this book for more information regarding writing RegRipper plugins.

A side effect of the use of services as a persistence mechanism for malware is that the Windows XP and 2003 operating systems (similar activity has not been widely observed on Vista systems and above, including Windows 7, etc.) tend to record information that can make an analyst's task of locating the malware, or the initial date that the system was compromised, a bit easier. In particular, when a service or device driver is actually "run," in many cases, an entry beneath the *System\CurrentControlSet\Enum\Root* key appears; specifically, a subkey whose name is "LEGACY_<*service name*>", as illustrated in Fig. 3.8.

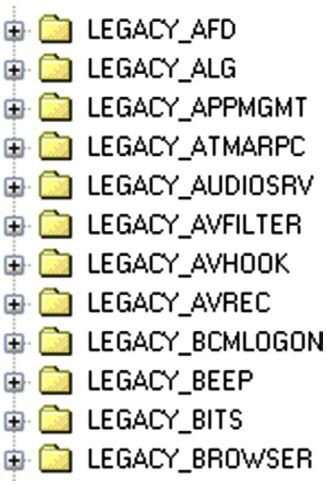


Figure 3.8 Enum\Root\LEGACY_* keys (Windows XP).

Again, these keys appear to be created relatively close to the time that the service is first run. During multiple malware and intrusion examinations involving the creation of services (particularly those that are loaded and run via svchost.exe), there appears to be a correlation between when the file was first created on the system, an Event Log entry indicating that the service was started, and the LastWrite time on the LEGACY_* subkey related to the service. This information can be very valuable when attempting to determine and/or validate the time frame of the initial compromise, or an overall window of compromise. In my experience, this information applies primarily to Windows XP and 2003 systems, as I haven't seen a similar correlation, to a great extent, on Windows 7 systems.

Beneath each of these LEGACY_* keys you will often find a subkey named "0000", which also appears to be created and modified in some way when a service is launched. Therefore, the LastWrite time on the LEGACY_*\0000 key for a particular service should closely approximate the last time the service was run. For example, on a Windows XP Service Pack 3 system I was examining, the Browser service was configured to start automatically when the system booted. The LastWrite time on the Browser service key was August 11, 2010, at approximately 08:10:28 UTC, and the LastWrite time on the *LEGACY_BROWSER\0000* key was 08:11:23 UTC on the same day. As it turned out, the system had last been booted at approximately 08:08 UTC on August 11, 2010. The LastWrite time on the *LEGACY_BROWSER* key was May 9, 2008 at approximately 01:56:17 UTC, which approximates to the time that the system was installed. This same sort of analysis applies to services that are started manually and should be carefully considered as part of your analysis, including correlating this information with other artifacts from the image, such as Event Log entries, etc.

During an examination I was working on some time ago, I found what turned out to be a service installed within the time frame of an incident. I say "an incident" because, as is sometimes the case, when examining a system to determine the root cause of one incident, I run across indications of a previous or multiple incidents. In some instances, I've found indications of multiple different bits of malware, as well as one or more intrusions. In this case, I found a service that had been installed, and the file system metadata (ie, time stamps) for the executable file indicated that it had been created on the system in February 2009, which was 15 months prior to the incident I had been asked to look into. The LastWrite time on both the *LEGACY_** and *LEGACY_*\0000* subkeys for the service indicated that it had been first launched shortly after the executable file had been created on the system, and that was the only time that the service had been launched.

Further analysis determined that the service was not configured to start automatically when the system was booted, but instead was set to be started manually.

Note

The *legacy.pl* plugin extracts the names of the *LEGACY_** subkeys from the *Enum\Root* key and displays them sorted based on their LastWrite times. Correlating this information with the output from other plugins (or any others that extract information about services) can prove to be very beneficial in locating malware, as well as establishing at time frame for the initial intrusion.

Another way that the LastWrite time for the *LEGACY_** key can be useful in determining the time frame of an incident or intrusion is when the executable file (.exe or .dll file) itself is subject to "time stomping." That is, there is malware that, when it is installed, the executable file MAC times are modified so that it remains hidden from rudimentary detection techniques, such as searching for new files on a system based on creation dates or creating a timeline of system activity for analysis. In this case, an anomaly may be detected if the creation date for the executable file were sometime in 2004, but the LastWrite time for the service's *LEGACY_** key were, say, in 2009.

Warning

Similar to what has been observed with respect to modifying file MAC times (ie, *SetFileTime()*) to arbitrary times (referred to as "time stomping"), the tool SetRegTime (found online at https://code.google.com/p/mft2csv/wiki/SetRegTime) can be used to modify key LastWrite times.

I, and others, have used this technique to great effect. There have been a number of examinations during which I have found a suspicious file, or an unusual service referenced in the Event Log, and locating the *LEGACY_** entry has led me to other interesting information in my timeline. In most cases, I've seen file creations "nearby" in the timeline that provide me with a clear indication of the initial indicators of the incident.

Program Execution

Program execution artifacts are those artifacts that demonstrate that a program was actually executed or launched at some

point. There are times during an examination where you may find an executable program file on a system, but the existence of the file isn't as important as determining if the program had been executed. There are other artifacts that we can look to for indications that the program may have been executed. As you might think, program execution artifacts may be a subset of autostart artifacts, particularly if the condition of the autostart mechanism (system booting, user logging in, etc.) has been met. Again, we're focusing in this book on those artifacts found in the Windows Registry.

AppCompatCache

In 2012, members of the consulting firm Mandiant (now part of FireEye) published an article to their company blog (found online at https://www.mandiant.com/blog/leveraging-application-compatibility-cache-forensic-investigations) describing their findings regarding the Windows "Application Compatibility Cache," or shim cache.

Tip

I should note that the Mandiant analysts also authored a five-page white paper that explains their findings. It's an easy read and does a very good job of describing what led them to the data and what they were able to determine from the data.

One aspect of the AppCompatCache, or Shim Cache, data to be aware of is that very often the time stamp that is included in the data is misinterpreted. So far, the data from all versions of Windows (except 32-bit versions of Windows XP) have a single time stamp associated with the individual file path entries in the data. In all of these cases, this time stamp has been determined to be the last modified time for the file, from the file system metadata (specifically, the $STANDARD_INFORMATION attribute within the master file table). I've seen many times during presentations and in reports that this time stamp is misinterpreted and misunderstood to be when the file was last executed.

Timeline Tip

When creating a timeline of system activity, it can be very beneficial to include AppCompatCache data entries in the timeline. If the file still exists within the file system, you will see that time stamp in the AppCompatCache data will line up with the file system last modified time for the file.

As of this writing, the RegRipper *appcompatcache.pl* plugin can parse the AppCompatCache value from Windows XP through Windows 10 and Windows [Server2012] systems, and the *appcompatcache_tln.pl* plugin will output the data in a format suitable for inclusion in a timeline.

Malware

I generally reserve the "malware" artifact category for indicators of malware that do not fall into other categories, such as "auto-start" or "program execution." There are malware variants, and even some families, which leave artifacts within the Windows Registry that have nothing to do with the persistence of the malware, and by looking for the artifacts specifically, we may be able to detect the malware, where other mechanisms failed to do so. Some examples of RegRipper plugins that can be useful in this artifact category include *fileless.pl*, *rlo.pl*, *routes.pl*, and *sizes.pl*.

I originally wrote the *fileless.pl* plugin as a means of detecting the Poweliks malware; this malware was referred to as "fileless" because it didn't persist by writing a file to the file system. Instead, this particular malware persists via what amounts to a script in an autostart Registry value, and when that script is launched, it reads a Registry value with encoded data that performs the functions of malware. While early versions of this malware were found in the user's USRCLASS.DAT hive and within the Software hive, once the malware was found installed as a Windows service, I adapted the plugin to be much more general.

The *rlo.pl* plugin resulted from some research involving the Unicode "right-to-left override" (RLO) control character, which essentially reverses the characters that follow the Unicode control character, a technique that has been used to hide malware on systems. From the perspective of a user or investigator looking at a Registry hive file in a viewer, a key or value name would appear normal, so much so that it might even appear to be a legitimate name. However, from the computer's perspective, the sequence of bytes that comprise the name are markedly different and, as such, searching for a specific key or value name without taking the RLO character into account would fail. As such, I wrote a plugin that would run through a Registry hive and identify any key or value names that included this specific character. The issue of the RLO character being used was discussed in the "How to Hide Malware in Unicode" article published to the Dell Secureworks Security and Compliance Blog on Oct 1, 2013. That article can be found online at http://www.secureworks.com/resources/blog/how-to-hide-malware-in-unicode.

I wrote the *sizes.pl* plugin originally as a testing plugin; I wanted to see how many values that contain binary data were larger than a specific size (the AppCompatCache data can be "large," on the order of tens or even hundreds of kilobytes) within hive files. I had read online about a particular malware variant that had been modified from maintaining its configuration data in a file to keeping it in a value with binary data. Knowing that key paths and value names can (and do) change, I wanted a way to look for all values with binary data of specific minimum size or larger. This plugin was instrumental in my testing process and essentially allows a user to create a whitelist of Registry values with "large" data within their environment. As this data was specific to the malware configuration and not related to its persistence, I included it in the "malware" artifact category.

The *routes.pl* plugin is, at the time of this writing, 5 years old, originally written in August 2010. The "route" command is often used in batch scripts that provide for automated volatile data collection during incident response activities, as the command "route print" outputs the network routing tables for the system in question. During my career as an incident responder, I have seen systems with specific persistent routes that had been set up for a legitimate business purpose; information about persistent routes is maintained in the Registry. Interestingly, the *routes.pl* plugin was featured in recipe 10.8 of the *Malware Analyst's Cookbook*, which was published in 2011.

I should note that, with the exception of the *routes.pl* plugin, these plugins are not specific to the System hive but can be run across all hives, including those found within the user profile.

USB Devices

Another item of interest to analysts will often be the devices (particularly USB devices) that had been attached to the system. Research into this area has been going on for some time; Cory Altheide and I published some of our joint research in this area in 2005, and some more recent analysis findings have been documented by Rob Lee on the SANS Forensic Blog (found online at http://blogs.sans.org/computer-forensics) on September 9, 2009. In short, the System hive maintains a great deal of information about the devices and when they were attached to the system. Additional information regarding user-specific artifacts of USB devices will be covered in chapter "Case Studies: User Hives" of this book.

In short, when a USB device is connected to a Windows system, the Plug-and-Play (PnP) manager receives the notification and queries the device. Information about the device, extracted from the device descriptor (which is *not* part of the memory area of the device),

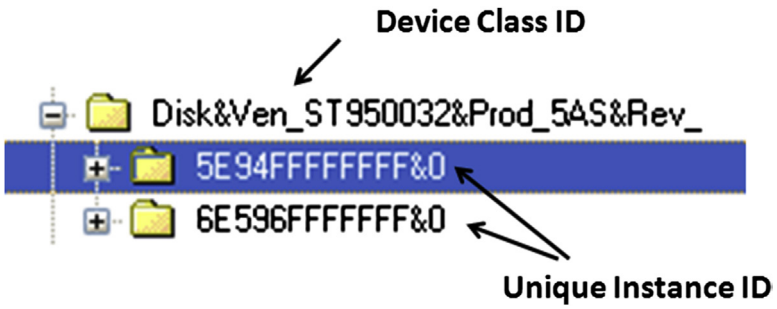


Figure 3.9 USB device in the Enum\USBStor key.

is then stored in the System hive beneath the *ControlSet00n\Enum\USBStor* and *…\USB* subkeys. The storage device is then (most often) recognized as a disk device and mounted as a drive letter or volume on the system. As such, additional information related to the device is recorded in the *MountedDevices* key within the System hive, as well as two subkeys beneath the *Control\DeviceClasses* key.

Let's take a look at what this looks like in the System hive. First, beneath the *Enum\USBStor* key, we can see where devices are listed, first by a key known as a device class identifier (ID), and by a subkey beneath the device ID known as the unique instance ID, as illustrated in Fig. 3.9.

As you can see in Fig. 3.9, the device class ID tells us a little bit about the device itself (in this case, the device is a 500 GB Seagate "wallet" drive). Beneath the device class ID, we see two unique instance IDs, which are the device serial numbers extracted from the device descriptor of each device. In each case, the unique instance ID key contains information about the devices within Registry values, including the device "FriendlyName" (in both cases, "ST950032 5A2 USB Drive").

Now, not every USB device has a serial number in its device descriptor. In such cases, Windows will assign a unique instance ID to the device. In order to tell when this is the case, take a look at the unique instance ID for the device, and if the second character (*not* the second to last character, but the second character in the string) is an "&" (as illustrated in Fig. 3.10), then the unique instance ID was created and assigned by the operating system, rather than extracted from the device descriptor of the device.

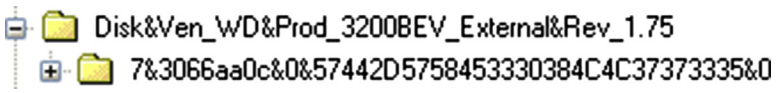


Figure 3.10 Unique instance ID assigned by Windows.

Note

The *usbstor.pl* RegRipper plugin extracts information from the *Enum\USBStor* key; specifically, for each device class ID, it lists the FriendlyName value (and on Windows XP and 2003 systems, the ParentIdPrefix value) for each unique instance ID (listed as "S/N" for "serial number" in the plugin output). The *Enum\USB* key contains information about all USB devices that had been connected to the system (quite naturally, on some systems, I have entries for "Tableau USB-to-SATA" device), and the *usbdevices.pl* plugin will extract this information.

Mapping Devices to Drive Letters

Once we have information about the USB devices attached to the system, we can attempt to map that device to a drive letter. This may not always be possible, particularly if multiple devices had been connected to the system successively. For example, I've connected a thumb drive to my system that has been mounted as the drive letter F:\. Later, I disconnect the device and then at some point connect another device, which is also mounted as the F:\ drive.

Before continuing, we need to understand that Windows treats external USB drives (hard drives in enclosures, such as "wallet" drives) and thumb drives or USB keys differently. Specifically, thumb drives contain a value within their unique instance ID key called the *ParentIdPrefix*; external drives do not contain this value. I have also seen where neither the storage component of my Motorola MB300 BackFlip smartphone nor a Garmin Nuvi (both the SD card and the flash device) will have a *ParentIdPrefix* value populated beneath the unique instance ID key. The usbstor.pl RegRipper plugin will display the *ParentIdPrefix* value for those devices that have the value, as illustrated as follows:

```
  Disk&Ven_Generic-&Prod_Multi-Card&Rev_1.00 [Sat Jan 2
12:56:01 2010]
  S/N: 200711141734000000&0 [Sun Aug 1 10:06:03 2010]
   FriendlyName : Generic- Multi-Card USB Device
   ParentIdPrefix: 7&24e8d74f&0
```

However, as indicated, external drives (usually, those in enclosures, produced by Maxtor, Western Digital, etc.) will not have *ParentIdPrefix* values, as illustrated as follows:

```
  Disk&Ven_Maxtor&Prod_OneTouch&Rev_0125 [Thu Mar 4 15:50:13
2010]
   S/N: 2HAPT6RQ____&0 [Wed Jun 30 01:27:21 2010]
     FriendlyName : Maxtor OneTouch USB Device
   S/N: 2HAPT6VY____&0 [Thu Jul 8 00:34:48 2010]
     FriendlyName : Maxtor OneTouch USB Device
```

\??\Volume{d99297b2-0b5d-11df...	REG_BINARY	D9 60 41 F7 00 7E 00 00 00 00 00 00
\??\Volume{daca9310-8f32-11df-...	REG_BINARY	71 75 43 51 00 7E 00 00 00 00 00 00
\??\Volume{e26e3ff7-f948-11de-...	REG_BINARY	5C 00 3F 00 3F 00 5C 00 53 00 54 00 4F 00
\??\Volume{fec5eece-f71f-11de-...	REG_BINARY	00 00 00 D0 00 7E 00 00 00 00 00 00
\DosDevices\C:	REG_BINARY	00 00 00 D0 00 7E 00 00 00 00 00 00
\DosDevices\D:	REG_BINARY	00 00 00 D0 00 84 12 4C 1D 00 00 00
\DosDevices\E:	REG_BINARY	5C 00 3F 00 3F 00 5C 00 49 00 44 00 45 00
\DosDevices\F:	REG_BINARY	71 75 43 51 00 7E 00 00 00 00 00 00
\DosDevices\G:	REG_BINARY	23 48 3D D4 00 7E 00 00 00 00 00 00
\DosDevices\H:	REG_BINARY	5C 00 3F 00 3F 00 5C 00 53 00 54 00 4F 00
\DosDevices\I:	REG_BINARY	2A 24 56 20 00 00 90 0C 00 00 00 00

Figure 3.11 Excerpt of values from MountedDevices key.

This is important because we may be able to use this information to map a thumb drive or key to a drive letter. I say "may be able to," because it really depends on how soon after the device being connected to the system that an image (or just the System hive) is acquired from the system. As I mentioned previously, drive letters will very often be reused, so disconnecting one device and connecting another may result in both devices being assigned the same drive letter.

All of the values within the MountedDevices key have binary data. However, different data can mean different things. For instance, Fig. 3.11 illustrates an excerpt of values from the MountedDevices key of a System hive file.

As you can see from Fig. 3.11, there are two basic types of value names; those that begin with "\DosDevices\" and refer to a drive or volume letter, and those that begin with "\??\Volume" and refer to volumes. These values have data of different lengths; some are 12 bytes long, others are longer. Many of the longer ones are actually Unicode strings that refer to devices, strings that we can read by double-clicking the value. The contents of the data for "\DosDevices\H:" (highlighted in Fig. 3.11) is illustrated in Fig. 3.12.

```
\.?.?.\.S.T.O.R.
A.G.E.#.R.e.m.o.
v.a.b.l.e.M.e.d.
i.a.#.7.&.2.4.e.
8.d.7.4.f.&.0.&.
R.M.#.{.5.3.f.5.
6.3.0.d.-.b.6.b.
f.-.1.1.d.0.-.9.
4.f.2.-.0.0.a.0.
c.9.1.e.f.b.8.b.
}.
```

Figure 3.12 MountedDevices key value data showing ParentIdPrefix.

The Unicode string in Fig. 3.12 refers to a removable storage device ("\??\Storage#RemovableMedia#", in this case, a USB device), and the highlighted substring "7&24e8d74f&0" is the *ParentIdPrefix* value for one of the USB devices that had been connected to the system. Therefore, we can use the *ParentIdPrefix* value to map a USB thumb drive from the *Enum\USBStor* key to a volume identifier within the MountedDevices key, and possibly even to a drive letter. An important factor to keep in mind, however, is that if you plug in one device that is mapped to drive H:\, disconnect it, and then connect another device that is mapped to drive H:\, the previous data for "\DosDevices\H:" is replaced.

GETTING HISTORICAL INFORMATION

Historical information about drive mappings in the hive files can be found in Windows XP System Restore Points, as well as within hive files from Volume Shadow Copies on Vista and above systems.

Using the *usbstor.pl* RegRipper plugin, we can obtain information about USB removable storage devices attached to the system (note that the key LastWrite times are displayed but are irrelevant to this example), an excerpt of which is illustrated as follows:

```
  Disk&Ven_Generic-&Prod_Multi-Card&Rev_1.00 [Sat Jan 2
12:56:01 2010]
   S/N: 200711141734000000&0 [Sun Aug 1 10:06:03 2010]
     FriendlyName : Generic- Multi-Card USB Device
     ParentIdPrefix: 7&24e8d74f&0
```

From the mountdev.pl plugin, we can get information about the values listed in the MountedDevices key, which appears as follows:

```
  Device: \??\STORAGE#RemovableMedia#7&24e8d74f&0&RM#{53f563
0d-b6bf-11d0-94f2-00a0c91efb8b}
   \??\Volume{47042c43-f725-11de-a8a5-806d6172696f}
   \DosDevices\H:
```

So now, we're able to map a USB thumb drive to a drive letter. But what about the USB external drives, such as those in enclosures (ie, "wallet" drives, etc.)? If you remember from Fig. 3.11, several of the values have data that is only 12 bytes long. These are volume identifiers and drive letters that refer to the external drives. In these cases, the first 4 bytes (DWORD) are the drive signature (also known as a volume ID) from the hard drive itself. This signature is written to a hard drive, beginning at offset 0x1b8 (440 in decimal) within the master boot record (MBR) when Windows formats the drive. You can view this value by opening the first 512 bytes of the

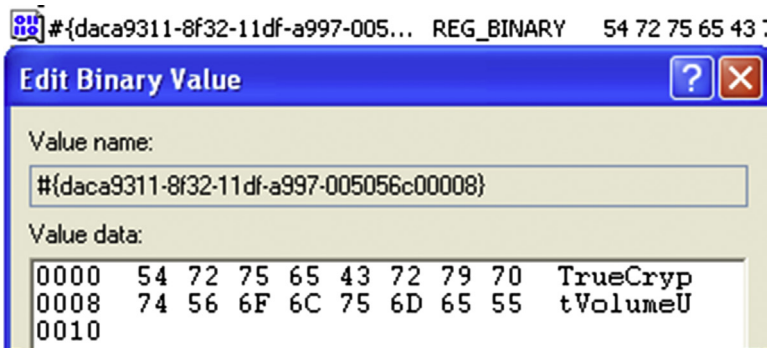


Figure 3.13 TrueCrypt volume listed in the MountedDevices key.

hard drive (MBR) in a hex editor and navigating to offset 0x1b8. The remaining 8 bytes of the data are the partition or volume offset. In Fig. 3.11, we see two drive letters (\DosDevices\C: and \DosDevices\F:) with partition offsets of 0x7e00, which is 32256 in decimal; dividing by 512 byte sectors, this means that the partitions or volumes start at sector 63 on their respective hard drives (note that \DosDevices\C: refers to the hard drive installed in the system and is used as an example).

What this means is that there is not a direct method for mapping a USB external hard drive listed in the *Enum\USBStor* key to a drive letter listed in the MountedDevices key.

While not specifically recognized as a device, per se, the MountedDevices key also maintains information about TrueCrypt volumes (may still be in use) that had been mounted on the system, as illustrated in Fig. 3.13.

As you can see, the value name is a bit different from other entries within the MountedDevices key, and the binary data is 16 bytes long and spells out "TrueCryptVolumeU". I have seen other similar values where the data spells out "TrueCryptVolumeT" or "TrueCryptVolumeS". While this will give you an indication of a user accessing TrueCrypt volumes, it does not explicitly tell you where those volumes exist.

PORTABLE DEVICES

On Windows versions beyond Vista (including Windows 7 and 10) even more information about attached (portable) devices is maintained in the Registry, albeit in the Software hive. Beneath the *Microsoft\Windows Portable Devices\Devices* key, you will see a number of subkeys that refer to devices. The subkey names can be parsed to get the name of the device and, if available, the device serial number. These subkeys also contain a value named "FriendlyName", which in many instances will

Continued

PORTABLE DEVICES—Cont'd

include the drive letter to which it was mounted, such as "Removable Disk (F:)". Further testing is required, but in some limited sample cases, the LastWrite time for the device subkey seems to correlate closely to the time that the device was last connected to the system. For example, on one Vista test system, a device (DISK&VEN_BEST_BUY&PROD_GEEK_SQUAD_U3&REV_6.15, with serial number 0C90195032E36889&0) had a subkey beneath the Devices key with the LastWrite time of Thu Feb 7 13:26:19 2008 (UTC). The corresponding subkey for the same device, beneath the DeviceClasses subkey (we will discuss this key later in the chapter), had a LastWrite time of Thu Feb 7 13:26:02 2008 (UTC).

When a USB device is first plugged into a Windows system, the PnP manager queries the device to determine information about the device in order to figure out which drivers to load for that device. On Windows XP and 2003 systems, this information is maintained in the setupapi.log file (for Vista systems and beyond, the file is setupapi.dev.log). Once the device is loaded, two additional keys are created for the device beneath the DeviceClasses key within the System hive. Both of these keys are GUIDs; one refers to disks, and the other refers to volumes, as shown below:

```
Disk GUID - {53f56307-b6bf-11d0-94f2-00a0c91efb8b}
Volume GUID - {53f5630d-b6bf-11d0-94f2-00a0c91efb8b}
```

Both of these GUIDs are defined in the ntddstor.h header file used in Windows. The first GUID, which begins with "53f56307", is defined as GUID_DEVINTERFACE_DISK, or DiskClassGUID, and refers to disk devices. An example of what the DiskClassGUID subkeys look like is illustrated in Fig. 3.14.

As illustrated in Fig. 3.14, we see keys whose names begin with "##?#USBSTOR#"; these keys go on to contain device names that look very much like the device descriptor names from the USBStor key mentioned previously in the chapter. The key name also contains the unique device descriptor or serial number for the device. According to research conducted and published by Rob Lee (of Mandiant and

Figure 3.14 DiskClassGUID keys in Windows XP System hive.

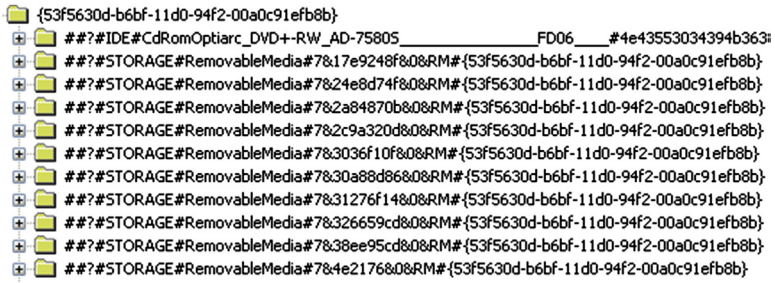


Figure 3.15 VolumeClassGUID keys in Windows XP System hive.

SANS fame), the LastWrite time for this key indicates the first time that the device was last connected to the system during the most recent boot session. What this means is that if the system was booted at 8:30 am and the device connected to the system at 9:00 am, disconnected, and then reconnected later that day, the LastWrite time of the device's subkey beneath the DiskClassGUID key will be 9:00 am. This should remain consistent regardless of the number of times the device is disconnected and reconnected to the system.

Tip

According to Rob Lee's research, the time that a USB device was last connected to a Vista system can be correlated to the LastWrite time of the *ControlSet00n\Enum\USB* key for the device. For Windows 7 systems, the LastWrite time of the *ControlSet00n\Enum\USBStor* key for the device will tell you when it was last connected to the system.

The other GUID is defined as GUID_DEVINTERFACE_VOLUME, or VolumeClassGUID, and refers to volumes. The subkeys beneath this key are associated with volumes that are mounted on the system, as illustrated in Fig. 3.15.

As illustrated in Fig. 3.15, the device's key name contains the ParentIdPrefix value for the device, mentioned earlier in this chapter.

USB DEVICES

According to research conducted and presented by Rob Lee, additional information regarding determining the last time that a USB device was connected to a system is available in the user's NTUSER.DAT hive, specifically beneath the MountPoints2 key. This will be discussed in greater detail in chapter "Case Studies: User Hives," but this provides an analyst with two important pieces of information; one is, of course, the

Continued

USB DEVICES—Cont'd

last time that the device was connected to the system. The other is that by the presence of the key within the user's hive, there is now an association with a specific user. While a device may have been connected to a system, the analyst will be able to determine the time frame that it was last connected, which may be important when developing a timeline of activity on the system, as well as which user account was logged in when the device was connected.

Finally, to close out this discussion of USB devices, I wanted to include a caveat. You'll notice that throughout the discussion of USB device artifacts, I used terminology that referred to someone else's research; I did so because that's the state of the art at the moment. A great deal of the research that has gone into developing a method for tracking USB devices on (and across) Windows systems has been thorough, but short-lived. What I mean by that is that much of the research has been conducted within a short time frame, such as a few minutes, or maybe up to a few days. Very often what happens then is that we get a lot of questions from real world investigations, where analysts have run across anomalies to what has been revealed through the research. In particular, one question I see time and again has to do with time stamps (Registry key LastWrite times or values whose data contains a time stamp) not being what they "should be," or are expected to be, based on what the analyst knows about the case. Very often, these exams occur months after the device had been connected to the system, and in the interim, drivers may have been updated, patches to the operating system installed, etc. As such, my recommendation to you, dear reader, is that particularly with the number of different artifacts that you have available for something like tracking USB devices, carefully consider what you're seeing in the data. Do not just think, "…well, Harlan said in his book…" and go about your analysis. Consider what you're seeing in the data, as there is always more research to be done, and that needs to be done, especially as new versions of Windows operating systems and applications become available.

Software Hive

The Software hive maintains a great deal of configuration information for the overall system as well as applications and can provide indications to a knowledgeable analyst of how the system and installed applications may have appeared, behaved, and responded when the system was running. Understanding the role

that these keys and values play in configuration of applications and the operating system can provide the analyst with a great deal of insight into a variety of different types of examinations. Throughout this section, we will discuss various keys and values from the Software hive that play a significant role in the overall configuration of the system and applications. Keep in mind, though, that we cannot cover every possible key and value because, quite simply, I need to finish this book at some point and send it to the printer! Also, there are constantly new applications being developed, as well as current applications (and the operating system) being updated to include new keys and values. What I hope to do is provide you with insight into some of the keys and values that you can expect to find on a wide range of systems, including Windows XP all the way up through Windows 7, with some information regarding Windows 8 and Windows 10.

REDIRECTION

In order to handle some differences between 64- and 32-bit systems, Windows system use Registry redirection in order to maintain different logical "views" of the Registry, as different versions of software may use different Registry keys and values. In short, the major difference (from the perspective of Registry analysis) is that 32-bit applications run in WOW64 mode will access and write to keys and values beneath the Wow6432Node key within the Software hive. As such, rather than the usual key path that appears as follows:

```
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\
CurrentVersion
```

…you would then see the key path as appears below:

```
HKEY_LOCAL_MACHINE\Software\WOW6432Node\Microsoft\
Windows\CurrentVersion
```

However, not all Registry keys are redirected on a 64-bit system; some are shared by both 32- and 64-bit versions of the operating system. Microsoft maintains a list of redirected and shared keys in the article "Registry Keys Affected by WOW64" (found online at https://msdn.microsoft.com/en-us/library/aa384253(VS.85).aspx). What this means is that when analyzing the Registry from 64-bit systems, you'll need to be cognizant of the updated key path and how it applies when viewing hives via a Registry viewer, or be sure to modify your RegRipper plugins to take this into account. Many of the RegRipper plugins that access keys and values that can be found beneath the "Wow6432Node" key have already been modified; however, as versions of the Windows operating system progress from Windows 7 through Windows 10 and beyond, it's important to continue researching and keep up with any new keys that appear, as well as new keys that may appear beneath the Wow6432Node key.

System Configuration Information

There are a number of configuration settings that could affect your analysis, and ultimately, your case; in previous books, I have referred to these as "time bombs," because at the time, they weren't something that I (or others) had seen on a regular basis. We've already mentioned some of these settings in the System hive section of this chapter, and we'll be discussing some of the settings in the Software hive here in this section.

Windows Version

Perhaps the most important values that you can extract from the Software hive is the version of the operating system. Why does this matter? This information alone can have a profound impact on the direction of your examination, as it sets our expectations as to what we *should* see on the system, and the truth is that not all versions of Windows are the same. For example, when someone wants to know what forensic artifacts they should be looking at in order to achieve the goals of their exam, that list is going to be significantly different between Windows XP and Windows 7. There is an abundance of artifacts available on Windows 7 systems that are simply nonexistent on Windows XP systems (which, to some extent, is what I've been trying to illustrate throughout this book, and other books as well).

The values we're interested in are found beneath the *Microsoft\Windows NT\CurrentVersion* key; specific values include "CurrentVersion", "ProductName", and "CSDVersion" (that last one tells us which service pack is installed). These values will tell us what version of Windows we're dealing with and provide information that we can use to guide our investigation.

The RegRipper *winver.pl* plugin extracts information from the Software hive that will tell you exactly which version of Windows you're working with; this is one of the first plugins that I run when I first start each examination, as I have seen instances where the paperwork accompanying an acquired image incorrectly stated the version of the operating system.

Tip

There are a number of Registry keys that exist in both the Software hive as well as within the user's NTUSER.DAT hive and have identical paths. One example is the Run key. The precedence of these entries will depend upon the key itself and what is specified in vendor documentation. Just as with the key in the Software hive, the Run key in the user's NTUSER.DAT hive is also used as a persistence mechanism for malware. In some cases, the key paths are the same, but very different information is maintained within the keys. For example, with the Software hive, the key may maintain

Tip—Cont'd

configuration information, while within the NTUSER.DAT hive, the key will contain settings, most recently used (MRU) lists, etc. The Internet Settings values described in MS KB article 323308, for example, allow the system administrator to set the described functionality on a system-wide basis via the Software hive or on a per-user basis by applying those settings to the appropriate user profile.

ProfileList

The Software hive maintains a list of the various profiles that are resident on the system, which includes both local and domain users. Now, let's be clear about this…this is not a list of accounts on the system, it's a list of user profiles. The SAM hive maintains a list of the user accounts on the system, but the ProfileList key maintains a list of profiles on the system, which can include both local and domain users.

When a user logs into a Windows system, the system first checks to see if that user account has a profile on the system. This is located in the *Software\Microsoft\Windows NT\CurrentVersion\ProfileList* key, as illustrated in Fig. 3.16.

Each subkey beneath the ProfileList key is an SID, and you can find a list of well-known SIDs in MS KB article 243330 (found online at https://support.microsoft.com/en-us/kb/243330). Each of the keys visible in the ProfileList key contains information about the user profile, as illustrated in Fig. 3.17.

Some of the information visible in Fig. 3.17 can be very useful for, well, some pretty obvious reasons. For example, the ProfileImagePath value tells us where the user profile and all of its associated files (NTUSER.DAT, for example) and subdirectories are located. On [Windows2000], XP, and 2003 systems, the default or usual path where we expect to see user profiles is in the path, "C:\Documents and Settings"; for Vista and above systems (including

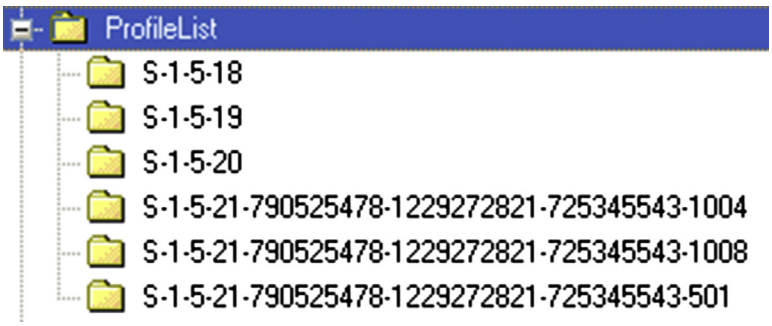
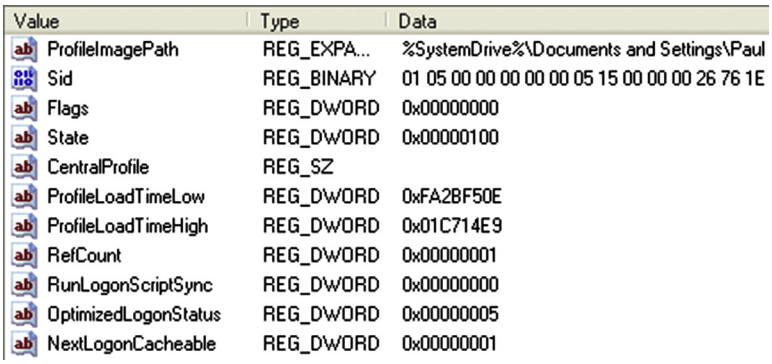


Figure 3.16 Example ProfileList key contents.



Value	Type	Data
ProfileImagePath	REG_EXPA...	%SystemDrive%\Documents and Settings\Paul
Sid	REG_BINARY	01 05 00 00 00 00 00 05 15 00 00 00 26 76 1E
Flags	REG_DWORD	0x00000000
State	REG_DWORD	0x00000100
CentralProfile	REG_SZ	
ProfileLoadTimeLow	REG_DWORD	0xFA2BF50E
ProfileLoadTimeHigh	REG_DWORD	0x01C714E9
RefCount	REG_DWORD	0x00000001
RunLogonScriptSync	REG_DWORD	0x00000000
OptimizedLogonStatus	REG_DWORD	0x00000005
NextLogonCacheable	REG_DWORD	0x00000001

Figure 3.17 Contents of a ProfileList subkey.

Windows 7 through 10), it's "C:\Users". This value, in combination with the key name (ie, the user's SID), provides a quick and easy means for associating the long SID to a username. This also allows us to quickly see if the system was at one time part of a domain (refer section Security Hive), because if it was and domain users logged into the system, some of the SID key names would be different (as opposed to just the last set of numbers…the RID…being different). Further, if the ProfileImagePath value points to some path other than what is expected, then that would tell us a couple of things, the first of which would be where to look for that user profile. The second thing it would tell us is that someone took steps to modify the default behavior of the operating system, possibly in an attempt to hide certain user activity.

Tip

It's important to remember that a user profile is not created until the account is used to log into the system. This is true not only for local accounts found in the SAM database but also for domain accounts and domain-connected systems. This may prove to be important during incident response activities, as an intruder may create local accounts on systems that they can then access at a later date, if they need to for any reason.

The CentralProfile value is discussed in MS KB article 958736 (found online at https://support.microsoft.com/en-us/kb/958736). Research conducted on the Internet (okay… "Googling") indicates that the State value may be a bit mask, whose value provides information regarding the state of the locally cached profile. However, MS KB article 150919 (found online at https://support.microsoft.com/en-us/kb/150919) indicates that this value has to do with the profile type, which can be changed via the System Control Panel applet, by going to the Advanced tab

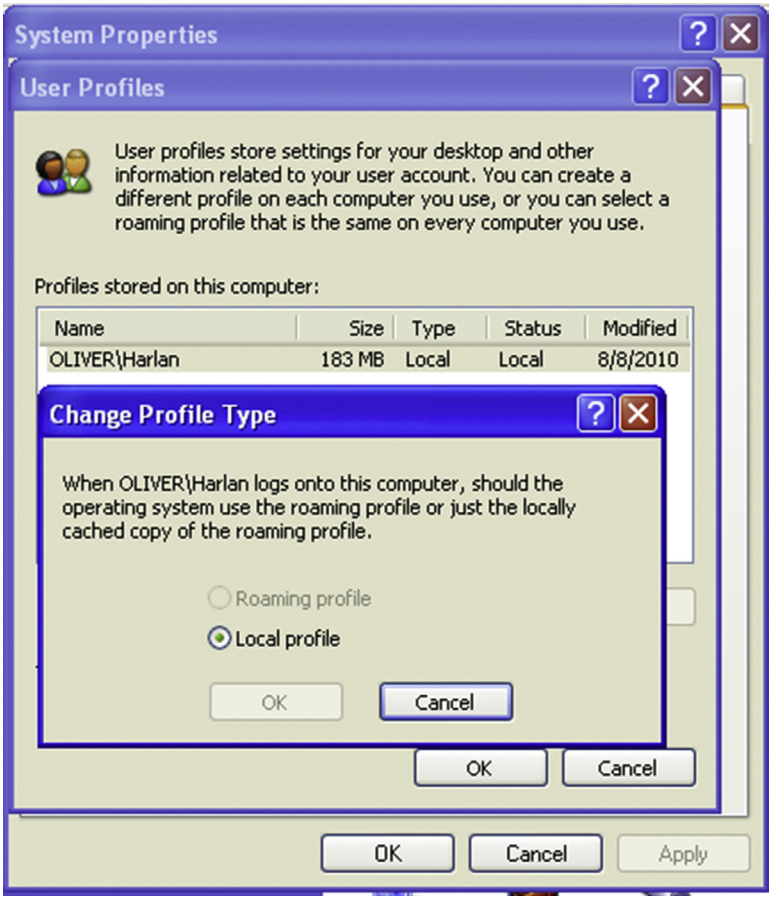


Figure 3.18 Changing the user's profile type.

and clicking the Settings button in the User Profiles section, as illustrated in Fig. 3.18.

CHANGING USERNAMES

I've seen questions posted to forums where someone has asked how to determine when a user account name was changed. I've been fortunate in the past and examined systems where auditing for "User Account Management" was enabled and found a record in the Event Log that indicated when the change was made. However, this isn't always the case. Another way to determine when this may have occurred would be to compare the LastWrite time for the user's key in the SAM (the one with the user RID in hexadecimal; for the Administrator, 000001F4 = 500) beneath the *SAM\Domains\Account\Users* key to the LastWrite time on the user's ProfileList key. Changing the username will cause the appropriate value in the SAM hive to be modified, and the key's LastWrite time will be updated.

Figure 3.19 Windows XP NetworkCards key.

Network Cards

Information about network adapters is also maintained in the Software hive. Beneath the *Microsoft\Windows NT\CurrentVersion\NetworkCards* key path, you may see several numbered subkeys, as illustrated in Fig. 3.19.

Each of these subkeys refers to an interface, and the subkey generally contains two values, ServiceName and Description. The ServiceName value refers to the GUID for the interface, and the Description value contains a description of the interface, as illustrated in the output of the *networcards.pl* plugin below:

```
  Launching networkcards v.20080325
  NetworkCards
  Microsoft\Windows NT\CurrentVersion\NetworkCards
  ADMtek AN983 10/100 PCI Adapter [Mon Sep 30 21:01:28 2002]
  Siemens SpeedStream Wireless USB [Sat Apr 22 08:17:30 2006]
  1394 Net Adapter [Mon Sep 30 21:02:04 2002]
  Instant Wireless USB Network Adapter ver.2.5 [Fri Jan 20
07:30:12 2006]
```

The output of the plugin provides an indication of the various interfaces on the system; in this case, we can see a PCI adapter and two wireless adapters (one of which is USB). This information can provide an analyst with clues as to where to look for additional information, as the information from the Software hive supports that information about network interfaces available in the System hive (discussed earlier in this chapter).

Wireless Connections

The Windows operating system maintains information about wireless access points (WAPs) to which the system has been connected. On Windows XP, this information is visible in the Preferred Networks box in the Wireless Network Connection Properties as illustrated in Fig. 3.20.

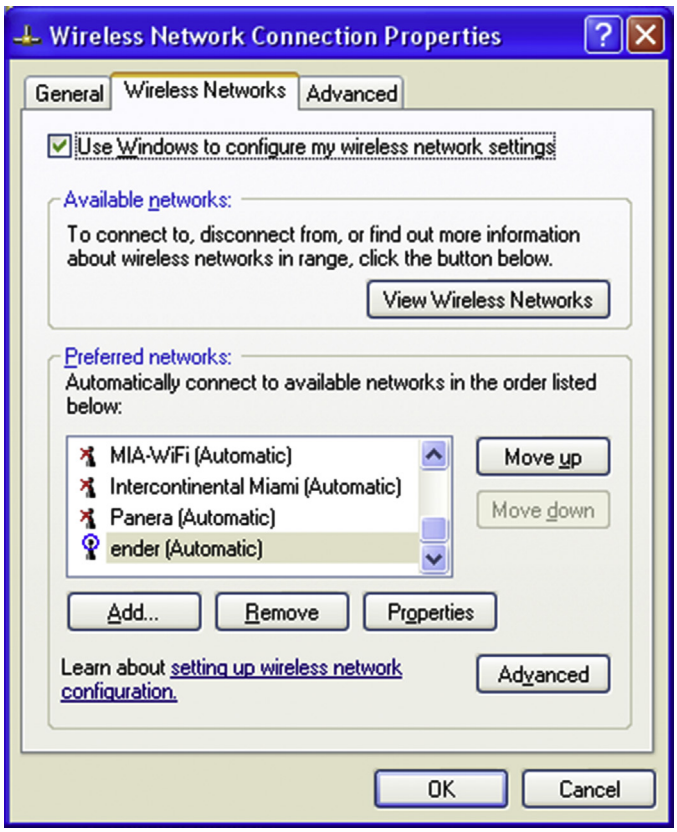


Figure 3.20 Wireless Network Connection Properties (XP).

How and where this information is maintained and visible depends on which process or application manages the wireless network connections. For example, some systems will use the Dell Wireless WLAN Card utility, and other systems may have their wireless connections and setting managed by an Intel application.

Information about wireless connections managed by Windows, such as those illustrated in Fig. 3.20, can be found in the *Microsoft\WZCSVC\Parameters\Interfaces* key. Beneath this key, you will find a subkey with the GUID for the wireless interface, and beneath that key you'll find several values that start with "Static#00xx", where "xx" is the number of the connection. Each of these values contains binary data that appears to be a structure that is very similar to the WZC_WLAN_CONFIG structure. The *ssid.pl* plugin for RegRipper parses this structure and displays the SSID of the WAP, its MAC address, and the date that the system last connected

to the WAP, as illustrated below (extracted from a Windows XP Software hive using the *ssid.pl* RegRipper plugin):

```
NIC: 11a/b/g Wireless LAN Mini PCI Express Adapter
Key LastWrite: Thu Feb 7 10:38:43 2008 UTC
Wed Oct 3 16:44:25 2007 MAC: 00-19-07-5B-36-92 tmobile
Mon Oct 8 10:12:46 2007 MAC: 00-16-B6-2F-5B-16 ender
```

If you open the ssid.pl plugin in an editor, you'll see that it also queries the *Microsoft\EAPOL\Parameters\Interfaces* key. In some cases, wireless SSIDs have also been found in this key. It's unclear to me as to how they end up there, which process is responsible for maintaining this information, or why some systems have the information while others don't, but now and again I've found information that has lead me to look in other locations for additional artifacts. An example of the output of the plugin from that key appears below:

```
NIC: 11a/b/g Wireless LAN Mini PCI Express Adapter
LastWrite time: Thu Sep 27 14:59:16 2007 UTC
1  ender
2  tmobile
```

Beginning with Windows Vista, this information was maintained in quite a different manner. This information is now maintained in GUID subkeys beneath the *Microsoft\Windows NT\CurrentVersion\NetworkList\Profiles* key in the Software hive, as illustrated in Fig. 3.21.

The values and data within these keys provide a good deal more information than what is available on earlier systems. For example, there is the profile name, the date that the profile was created (the first time that the system connected to the WAP), and the last

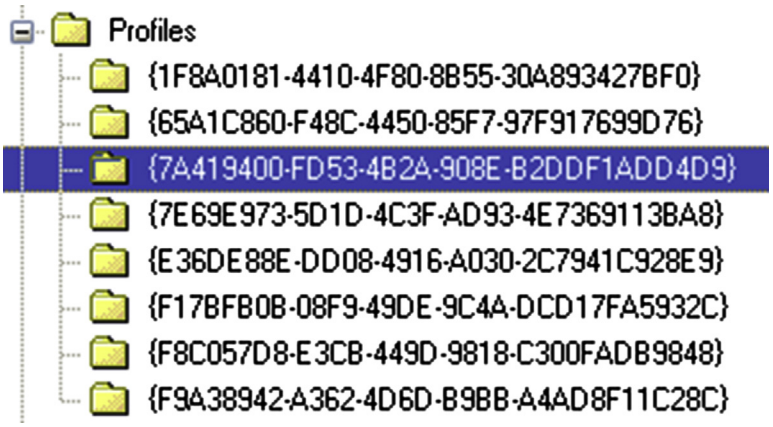


Figure 3.21 Wireless interface profile keys.

Value	Type	Data
ProfileName	REG_SZ	ender2
Description	REG_SZ	ender2
Managed	REG_DWORD	0x00000000
Category	REG_DWORD	0x00000000
DateCreated	REG_BINARY	D7 07 09 00 01 00 18 00 16 00 10 00 0A 00 52 00
NameType	REG_DWORD	0x00000047
DateLastConnected	REG_BINARY	D8 07 08 00 06 00 10 00 0A 00 35 00 10 00 AD 03

Figure 3.22 Wireless interface values.

Value	Type	Data
ProfileGuid	REG_SZ	{7A419400-FD53-4B2A-908E-B2DDF1ADD4D9}
Description	REG_SZ	ender2
Source	REG_DWORD	0x00000008
DnsSuffix	REG_SZ	<none>
FirstNetwork	REG_SZ	ender2
DefaultGatewayMac	REG_BINARY	00 15 E9 EA 39 D2 00 00

Figure 3.23 Values from *Signature\Unmanaged* subkeys.

time that the system was connected to the WAP. These values, and others, are illustrated in Fig. 3.22.

You'll notice that the data within the DateCreated and DateLastConnected values are a bit bigger than we'd expect for Unix time (32-bit) or FILETIME (64-bit) objects. The data within these values is actually 128-bit SYSTEMTIME objects. This is important as it's yet another time stamp format we have to be prepared to handle when analyzing Windows systems.

One piece of data that we don't see in these values is the WAP MAC address. If we go back to the Software hive and locate the *Microsoft\Windows NT\CurrentVersion\NetworkList\Signatures* key, we'll see *Managed* and *Unmanaged* subkeys. As the data for the Managed value illustrated in Fig. 3.22 is 0, we go to the *Unmanaged* key, and we'll see a number of subkeys whose names are long strings. Within each of these keys we find additional data that we can correlate to the data from the Profile key, using the ProfileGuid value, as illustrated in Fig. 3.23.

I wrote the RegRipper *networklist.pl* plugin to access a Vista or Windows 7 Software hive and extract and correlate the above information, an excerpt of which (run against a Vista system Software hive) is shown below:

```
Launching networklist v.20090811
Microsoft\Windows NT\CurrentVersion\NetworkList\Profiles
linksys
 Key LastWrite : Mon Feb 18 16:02:48 2008 UTC
 DateLastConnected: Mon Feb 18 11:02:48 2008
 DateCreated: Sat Feb 16 12:02:15 2008
 DefaultGatewayMac: 00-0F-66-58-41-ED
ender
 Key LastWrite : Mon Dec 22 04:09:17 2008 UTC
 DateLastConnected: Sun Dec 21 23:09:17 2008
 DateCreated: Tue Sep 11 10:33:39 2007
 DefaultGatewayMac: 00-16-B6-2F-5B-14
ender2
 Key LastWrite : Sat Aug 16 14:53:18 2008 UTC
 DateLastConnected: Sat Aug 16 10:53:16 2008
 DateCreated: Mon Sep 24 22:16:10 2007
 DefaultGatewayMac: 00-15-E9-EA-39-D2
ender2 2
 Key LastWrite : Mon Jan 12 12:42:49 2009 UTC
 DateLastConnected: Mon Jan 12 07:42:49 2009
 DateCreated: Mon Aug 25 19:19:39 2008
 DefaultGatewayMac: 00-21-29-77-D0-2D
```

The above excerpt from the networklist.pl plugin output provides an interesting view into activities on the system with respect to recording and managing the wireless connections. First, take a look at the "Key LastWrite" and "DateLastConnected" values for each profile listed; depending upon the time of year, and the time zone settings (time zone, if daylight savings is enabled), these times are exactly either 4 or 5 hours off. Remember, Registry key LastWrite times are 64-bit FILETIME objects based on GMT/UTC, and the DateLastConnected values are 128-bit SYSTEMTIME objects. What this tells us is that when the dates and times are recorded for DateCreated and DateLastConnected values, the time zone and daylight savings settings are included in the computation. This is a very important piece of information for analysts, as assuming that these values are actually UTC will be incorrect and can make a mess of your analysis, a timeline, etc.

Second, in the above excerpt we see two profiles that include the name "ender2", and the second one has an additional "2" as well as a different MAC address. In this case, I know what happened…the original WAP with the SSID "ender2" had "died" and was replaced. Rather than replacing the original information,

Windows (in this case, Vista) created a second profile and left the original profile information intact. Understanding or just being aware of this can be very helpful to an analyst.

AutoStart

Much like the System hive, the Software hive contains a number (albeit a limited one) of locations from which applications and programs can be started with little to no interaction from the user beyond simply booting the system and logging in. Many of these locations are used by application authors for completely legitimate purposes; unfortunately, as we've mentioned with respect to Browser Helper Objects (BHOs), they're also used by malware authors.

The Run Key

Perhaps the most well known of all of the autostart locations is the ubiquitous "Run" key (*Microsoft\Windows\CurrentVersion\Run*) described in MS KB article 314866 (found online at https://support.microsoft.com/en-us/kb/314866). This key has long been used by both malware and legitimate applications alike as a refuge for persistence, and continues to be used, even today. I regularly see malware that creates a value beneath this key, whether I'm performing analysis during a breach response or researching new malware via such sites as the Microsoft Malware Protection Center (MMPC, found online at https://www.microsoft.com/security/portal/threat/threats.aspx). Further, I've seen systems infected by one variant of malware that were later infected by another variant of the same malware (as determined by reviewing the write-ups on the malware), so the Run key contained multiple values that pointed to the malware variants.

I'm not sure what it is about this particular key, but a good deal of malware persists via this location. And what's really interesting about this is that often times, the malware that persists via this key goes undetected for weeks or months when conventional, antivirus techniques (file examination) are used.

In October 2015, I ran across a description of malware that had been dubbed "Moker"; the write-up on this malware is available from the EnSilo website, found online at http://blog.ensilo.com/moker-a-new-apt-discovered-within-a-sensitive-network. The description of the malware went into significant detail regarding techniques the author(s) had used to ensure that the malware would avoid detection, and if found, be difficult to analyze. However, as determined later in the comments section of the write-up, the persistence mechanism turned out to be the Run key (the

particular hive…Software or NTUSER.DAT…is dependent upon the privileges of the infected user, as is often the case); at that point, I had difficulty understanding how this malware was difficult to detect.

Note

Many times, malware write-ups get an important aspect of persistence incorrect. Specifically, if malware infects a system via a user account with normal user privileges, very often that malware will persist by creating a value beneath the user's Run key, in the "HKCU" hive for the currently logged in user. In such instances, the malware will not start again once the system reboots; rather, it won't start until the infected user logs in.

If the infected user account has Administrator-level privileges, the malware will often create a value beneath the Run key within the Software hive, which will allow the malware to start once the system has been rebooted, and without requiring any user to log into the system.

The *soft_run.pl* RegRipper plugin collects values from the Run key within the Software hive, as well as a number of other similar autostart keys. For example, the *Microsoft\Windows\Current Version\Policies\Explorer\Run* key can be used by group policies to specify autostart programs (as described online *at* https://technet.microsoft.com/en-us/magazine/ee851671.aspx). The plugin will also check for redirected keys on 64-bit systems (if you remember the discussion of the Wow6432Node key earlier in this chapter).

The Notify Key

Another location of interest within the Software hive is the *Microsoft\Windows NT\CurrentVersion\Winlogon\Notify* key. Entries beneath this key define packages (most often DLLs) that are to receive notifications from the WinLogon process. These notifications can include when the user logs in and the shell (ie, Windows Explorer) loads, when the screensaver starts, when the workstation is locked or unlocked, when the user logs out, etc. All of these actions cause an event to be generated, and packages can be designated to take some action when this occurs. Examples of malware that makes use of this key include Virtumonde (aka Vundu) and Contravirus, and a backdoor identified at the ThreatExpert site as "Eterok.C" actually deletes entries from the *Winlogon\Notify* key.

When I say "location of interest," I know that sounds kind of hoity-toity, but one of the things I've found time and again over the years, especially with respect to autostart locations in the Registry, is that once you stop looking at the ones you know about,

and you stop looking for new ones, they start being used more often. Over the years, I've heard malware authors say that some autostart locations are no longer of use (the same has been said of NTFS alternate data streams but that's outside the scope of this book) but the fact of the matter is that there are a great deal of system administrators out there (as well as forensic analysts) who simply aren't aware of these locations and how they can be used. Add to that instances where antivirus applications do not detect the malware that's actually loaded (or the antivirus applications are disabled during the malware installation process) from these locations, and what ends up happening is that systems get and remain infected for a considerable period of time.

Tip

Different malware families will employ different persistence mechanisms using the Registry. For example, one of the hallmarks of a ZBot infection is the presence of a reference to the malware within the UserInit value in the *Microsoft\Windows NT\CurrentVersion\Winlogon* key within the Software hive.

Other malware will leave various artifacts within the Registry; while not used to maintain persistence, these artifacts can be used as indicators to determine if (and possibly when) the system was infected. For example, some variants of Ilomo/Clampi have been found to create the Microsoft\9593275321 key within the Software hive. Virut is a file infector, but some variants have been found to add a value named "UpdateHost" to the *Microsoft\Windows\CurrentVersion\Explorer* key in the Software hive, as well as adding an exception for themselves to the firewall policy.

Image File Execution Options

Yet another autostart location (I told you there were a lot of these!) can be found in the *Image File Execution Options* key. Now, Microsoft provides documentation (found online at http://support.microsoft.com/kb/824344) on the use of this key, which is intended to provide debugging capability and can also be used to turn off the Windows Update feature in Windows XP (found online at http://support.microsoft.com/kb/892894). Like many other tools and techniques that are useful to administrators, this technique can also be used for malicious purposes, and malware authors have been seen using this technique to maintain persistence of their applications. In fact, the Microsoft Threat Encyclopedia discusses the malware known as Win32/Bebloh.A (found online at https://www.microsoft.com/security/portal/threat/encyclopedia/entry.aspx?Name=TrojanSpy:Win32/Bebloh.A#tab=2), which uses this functionality to force Internet Explorer to be launched whenever any other browser (Opera,

Safari, Firefox, etc.) is launched. And this is nothing new…Dana Epp wrote a blog post (found online at http://silverstr.ufies.org/blog/archives/000809.html) on this issue in March 2005.

In short, by adding a "Debugger" value to the application subkeys beneath the *Image File Execution Options* key, you can force a debugger or another application to be loaded instead. You can demonstrate this easily by adding a key called "notepad.exe", and within that key, add a string value named "Debugger." Then, add the string "sol.exe" (or any other application that launches as a GUI) to the value. Now, use any method to launch Notepad. Pretty neat, huh? And this works with any application. If you were running a tool like Process Monitor while launching Notepad and monitoring for Registry accesses, you'd notice that the operating system accesses this key and attempts to locate a subkey for the application being loaded. So, this is functionality that, while included in Registry value, is implemented as a function of how the operating system operates. Interestingly, I have seen this autostart location during engagements and, as such, wrote the RegRipper *imagefile.pl* plugin to query the *Image File Execution Options* subkeys, looking for and displaying Debugger values.

Note

In August 2010, Microsoft released KB article 2264107 (found online at http://support.microsoft.com/kb/2264107) in order to address issues related to the DLL Search Order vulnerability, specifically as it relates to remote resources (ie, folders) accessible via SMB and WebDAV. Specific applications can be protected by adding the "CWDIllegalInDllSearch" value, with the appropriate data setting, to the Image File Execution Options key. The RegRipper *imagefile.pl* plugin was updated to check for both the Debugger and CWDIllegalInDllSearch values.

To see an example of how this functionality can be used for malicious purposes, consider the "Sticky Keys" accessibility functionality provided by Windows systems, particularly those running Terminal Services. With prior access to the system, an intruder can add the "sethc.exe" subkey to the *Image File Execution Options* key ("utilman.exe" also works) and then add the "Debugger" value, pointing it to the command prompt, cmd.exe. Once a connection is established from a remote system, the command to create the key looks like the following:

```
C:\>reg add "\\192.168.1.110\HKLM\Software\Microsoft\
Windows NT\CurrentVersion\Image File Execution Options\
sethc.exe" /v "Debugger" /t REG_SZ /d "cmd.exe" /f
```

Returning later using the Terminal Services Client (and with network level authentication disabled, which can also be done

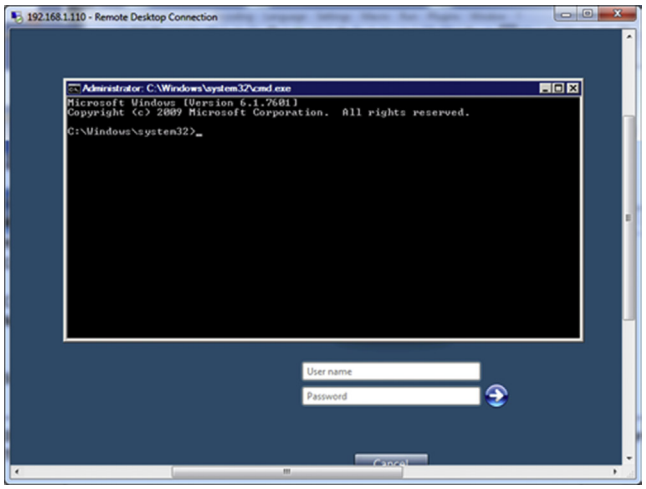


Figure 3.24 System-level command prompt, courtesy of "Sticky Keys".

remotely), the intruder does not need to authenticate to the system at all (that is, enter a username and password) but instead just hit the Shift key five times (or the "Win + U" key combination for utilman.exe) to get a System-level command prompt, as illustrated in Fig. 3.24.

Again, this requires no authentication whatsoever, which means that it will still work even if the IT staff has changed passwords throughout the enterprise. With a command prompt with System-level privileges, the intruder can run commands to…well, pretty much anything they want. For example, they can add a user account to the system, modify system settings, etc.

Using the functionality provided by this key is not something limited solely to the bad guys; the good guys can use it as well. Does your organization use *at.exe* to create Scheduled Tasks within the infrastructure, particularly on remote systems? Do your system administrators use *net.exe* to manage (create accounts, etc.) user accounts within the infrastructure? If not, why not create a subkey for the executables and point it to another executable or a batch file that will alert you to its use? That way, the first time an intruder tries to use either of these programs, they will find that they don't work, and you will be alerted. They may try again (and you'll be alerted again), and they may even go to the file itself, see if it exists, and even try to determine if it's the real one. It will take them some time and effort to figure out what's going on. In fact, they'll have to change their approach, perhaps enough so that you're provided some "breathing room" to respond. Regardless, you'll be alerted to the attempted use of the executable.

AppInit_DLLs

The AppInit_DLLs value is interesting, as according to MS KB article 197571 (found online at https://support.microsoft.com/en-us/kb/197571) the DLLs listed in this value (comma or space delimited) are loaded by each "Windows-based application." This means that the value will be read and the DLLs loaded with each GUI application that's launched on the system. Within Windows 7 and [Windows2008] R2, Microsoft modified the infrastructure for this value and its capabilities, as described online at https://msdn.microsoft.com/en-us/library/windows/desktop/dd744762(v=vs.85).aspx.

The value is located beneath the *Microsoft\Windows NT\CurrentVersion\Windows* key; you can use the *appinitdlls.pl* RegRipper plugin to retrieve any data from the value. I will say that in 15+ years (at the time of this writing) of performing incident response and digital forensic analysis, I have rarely seen DLLs listed in this value. Again…rarely. Not "never."

Shell Extensions

Shell extensions can also provide an autostart mechanism. In 2010, our team observed what was an apparently novel approach to persistence based on the use of approved shell extensions. There are a considerable number of articles available at the Microsoft website that address topics such as writing shell extensions and also shell extension security. However, where shell extensions come into play as a persistence mechanism is that they are loaded when the Windows Explorer "shell" loads and provide some sort of functionality extension beyond the basic shell. Many of the approved shell extensions that are loaded by the shell have explicit paths that lead directly to the DLL to be loaded, and in many cases, these are located in the Windows\system32 directory. However, some of the approved shell extensions (in the Software hive, as well as in the user's NTUSER.DAT hive) do not have explicit paths. Therefore, when Explorer.exe attempts to load the shell extension, it must first locate it, and in doing so, it begins searching in its own directory (C:\Windows) first. This DLL search order behavior is documented at the Microsoft website; the documentation can be found online at http://msdn.microsoft.com/en-us/library/ms682586.

During the malware reverse engineering panel at the "SANS What Works In Incident Response and Forensics" conference in July 2010, Nick Harbour (Nick worked at Mandiant at the time) briefly described this persistence mechanism, as well, based on what his team had seen, and how they approached the issue (Nick is well known for his malware reverse engineering skills). Nick's blog post addressed the DLL search order issue from a much wider

scope and appeared to refer to DLLs that are loaded based on their presence in an executable file's import table. To read more about how he described the issue, take a look at what Nick had to say about this persistence mechanism in an M-unition blog post, found online at http://blog.mandiant.com/archives/1207. Nick also mentions how to use the KnownDLLs (*ControlSet00n\Control\Session Manager\KnownDLLs*, described at the Microsoft website, at http://support.microsoft.com/kb/102985) key to protect a system from this sort of attack.

From the perspective of the shell extensions, in short, by using the same name as a legitimate approved shell extension (albeit one that was located in the *C:\Windows\system32* directory) and placing that DLL in the *C:\Windows* directory alongside explorer.exe, the malware was able to ensure that it was loaded each time a user logged in; however, this persistence mechanism required NO modifications to any files on the system (outside of the creation of one new one), nor did it require any modifications to the Registry. From the Microsoft site, we can see that the *SafeDllSearchMode* functionality is enabled by default (and can be disabled). However, close examination of the article reveals that regardless of whether the functionality is enabled or disabled, the DLL search order begins in "the directory from which the application loaded." As such, whenever a user logs into the system, the Windows Explorer shell (explorer.exe) is loaded, and then the approved shell extensions are automatically loaded.

In order to assist in investigations where this functionality may have been used as a persistence mechanism, I wrote the *shellext.pl* plugin for RegRipper. This plugin parses through the values of the *Microsoft\Windows\CurrentVersion\Shell Extensions\Approved* key in the Software hive, collects the names (GUIDs) and data (description of the shell extension) for each value, and then navigates to the *Classes\CLSID* key to map the GUID to a DLL path. An excerpt of the output of this plugin is provided as follows:

```
  {6756A641-DE71-11d0-831B-00AA005B4383} MRU AutoComplete
List
   DLL: %SystemRoot%\system32\browseui.dll
   Timestamp: Mon Apr 4 17:43:08 2005 Z
  {7BD29E00-76C1-11CF-9DD0-00A0C9034933} Temporary Internet
Files
   DLL: %SystemRoot%\system32\shdocvw.dll
   Timestamp: Mon Apr 4 17:43:09 2005 Z
  {f81e9010-6ea4-11ce-a7ff-00aa003ca9f6} Shell extensions
for sharing
   DLL: ntshrui.dll
   Timestamp: Mon Apr 4 18:37:13 2005 Z
```

Due to the amount of data available, this plugin can take several seconds to run; as such, I tend to run it via rip.exe rather than via a profile. However, from the output excerpt, you can see that two approved shell extensions (browseui.dll and shdocvw.dll) have explicit paths, whereas the third (ntshrui.dll) does not. In this case, in order to load the DLL, the Explorer.exe process must search for it in accordance with DLL search order; therefore, the search begins in *C:\Windows*, where Explorer.exe is located.

A very quick way to use this information during an examination is to collect all of the lines of the output that start with "DLL:" to a file, and then to parse the file looking at directory paths. For example, start with a command that appears as follows:

```
C:\tools>rip.exe -r D:\case\software -p shellext | find
"DLL:" >
D:\case\file\shellext.txt
```

The result of the above command will be a file containing only the lines that start with "DLL:", and from there, you can strip out the entries that do not contain path information such as "%SystemRoot%\system32" or something else. Of the remaining files, run a search for those files that appear in the C:\Windows directory. If they only appear in the C:\Windows directory, depending on the DLL in question, that may be expected; however, if files with that name appear in both the C:\Windows and the C:\Windows\system32 directory, you may have found something of value.

Note

The use of approved shell extensions as a persistence mechanism is very insidious, due to its very simplicity. This mechanism requires only that a DLL file of a specific name be created in a specific directory and does NOT require any modifications to the Registry. As long as the "subverted" shell extension does not remove regularly accessed functionality and the capability provided by the shell extension is not missed, then the malware may be run without any complaints from the user.

To protect a system against the sort of attack that takes advantage of the DLL search order, there are two options available. One is to locate all of the shell extensions in the Registry that use implicit paths and give each of them the appropriate explicit path. Another method is to add an entry for the DLL (ntshrui.dll) to the *ControlSet00x\Control\Session Manager\KnownDLLs* Registry key (found online at http://support.microsoft.com/kb/164501).

Overall, however, this is simply one example of a much larger issue that was originally identified as far back as the year 2000, but became more visible in August and September 2010, and was referred to as "DLL hijacking." In short, the use of shell extensions is but one example of a mechanism to get an executable to search for a DLL that it needs to load in order to perform some function. Readers interested in learning more about this issue should search for "DLL hijacking" via Google.

Using this technique, I mounted an acquired image as a read only drive letter on my analysis system and ran the above command. I located a shell extension named "slayerXP.dll", and when running the search, I found instances of the DLL in a ServicePack directory, as well as in the C:\Windows\system32 directory. Both instances had the same size, as well as the same MD5 hash. Further examination of the DLL indicated that it was a legitimate Microsoft file.

Browser Helper Objects

If the web browser in use on a system is Internet Explorer (IE), then another area that can be examined for indications of malware (usually, some sort of adware or a browser toolbar, although not always…) is the BHOs listing, which is found in the following key in the Software hive:

```
Microsoft\Windows\CurrentVersion\Explorer\Browser Helper
Objects
```

BHOs are DLLs that IE can load to provide additional functionality and allow for customization to the browser, much like plugins used for Firefox, or shell extensions for Windows Explorer (discussed in the "Shell Extensions" section later in this chapter). Examples of common BHOs include those for the Adobe Acrobat Reader, and the Google, Alexa, and Ask.com toolbars. Again, these are DLLs that are loaded by IE and not when the system is booted or a user logs into the system. If IE is not launched, the DLLs will not be loaded. However, if IE is used, then the DLLs will be loaded without any interaction by the user.

The use of BHOs to load malicious software is nothing new. In 2002, I was working in a full-time employment (as opposed to consulting) position at a company where someone had found something a bit unusual on their system. It turns out that the employee was in the marketing department, so what she found was indeed concerning. She was viewing the online content for our company website, and she noticed that in each instance where our company name was in the web page, the name was now a hyperlink…which was not the behavior for which the web page was designed. Out of curiosity, she clicked on the hyperlink (yeah, bad idea, I know…) and was taken to a competitor's web page! It turned out that her system had been infected with a BHO that would look for specific words and names in web pages and modify the contents of the web page to create hyperlinks to competitor's websites. I use the RegRipper bho.pl plugin to extract information about BHOs installed on the system for every examination, particularly those that involve malware of some kind.

Scheduled Tasks

Scheduled Tasks are likely not often thought about by most users (or administrators), but they do provide an excellent facility to ensure applications are run on a regular basis. In addition, some Scheduled Tasks can be configured to run based on various triggers, such as when a user logs in or logs out, for example. Scheduled Tasks can be created via a wizard or through the use of command line tools such as *schtasks.exe* or *at.exe*. In fact, dedicated adversaries are known to use Scheduled Tasks for both persistence (providing an autostart mechanism) and lateral movement with a network infrastructure.

Another important point with respect to Scheduled Tasks is that a good number of processes that were installed as Windows services on Windows XP and 2003 systems were moved to Scheduled Tasks as of Windows Vista, and this trend continued with Windows 7 and beyond. Want an easy way to see how many Scheduled Tasks are on a default installation of Windows 7 or even Windows 10? Simply open a command prompt and type "schtasks", and hit enter. If you want an accurate count, pipe the output of the command to a file and start counting.

When a Scheduled Task is created, a number of artifacts are created, including files within the file system, Windows Event Log records, and (to the focus of this book) a Registry key. For example, we can use the following command to create a Scheduled Task:

```
C:\>schtasks /create /sc daily /tn blahblah /tr "C:\program
files\microsoft games\solitaire\solitaire.exe" /st 22:16:00
```

The above code creates a task named "blahblah", which appears in the Registry beneath the *HKLM\Software\Microsoft\Windows NT\Schedule\TaskCache\Tree* key, as illustrated in Fig. 3.25.

Scheduled Tasks created using the *schtasks.exe* command can be set to run at a specific time (as with the above command), when a user logs on, when the system starts, or when the system is idle. If no other changes are made to the Scheduled Task, then the key LastWrite time can be incorporated into a timeline in order to further analysis.

Similar to *schtasks.exe*, using the *at.exe* command to create a Scheduled Task will create a Registry key in the same path; however, the *at.exe* command does not provide an option to name the tasks. Instead, it uses its own naming convention. The first task created is named "At1", the second is named "At2", and so on. The RegRipper *at.pl* plugin will display all Scheduled Tasks created using the *at.exe* command still visible in an extracted Software hive, and the *at_tln.pl* plugin will output the same information in a format suitable for including in a timeline.

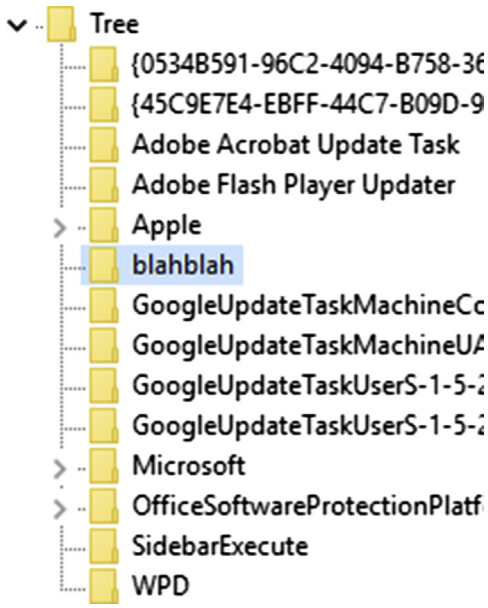


Figure 3.25 Scheduled Tasks viewed via the Registry Editor on Windows 10.

Again, dedicated adversaries use the Scheduled Task facility to great effect to not only remain persistent on systems, but to also move laterally within an infrastructure. Scheduled Tasks can be created on remote systems.

Warning

One aspect that an intruder can rely on is that many live incident response toolkits that I've seen will include commands to run either *schtasks.exe* or *at.exe*. These toolkits are intended to allow an analyst to run a number of commands, usually via a batch file, in order to quickly collect information from a live system. Each of the commands will show only those tasks created with that command; that is to say that commands created with *at.exe* will not appear when the list of Scheduled Tasks is queried using *schtasks.exe*. As such, both commands will need to be used in these toolkits.

AppCompatFlags

The AppCompatFlags key is usually thought of as providing information within the program execution category, but several of the subkeys may provide information within the autostart category; specifically, the *Custom* and *InstalledSDB* subkeys beneath the *Microsoft\Windows NT\CurrentVersion\AppCompatFlags* key.

These subkeys pertain to the Application Compatibility Database, described by Microsoft (found online at https://msdn.microsoft.com/en-us/library/bb432182(v=vs.85).aspx). Jon Erikson provided an excellent explanation of the Microsoft "Fix-It" capability that provides users with a solution to running an application that has compatibility issues with newer versions of Windows (Jon's BlackHat [Asia2014] paper can be found online at https://www.blackhat.com/docs/asia-14/materials/Erickson/WP-Asia-14-Erickson-Persist-It-Using-And-Abusing-Microsofts-Fix-It-Patches.pdf); this capability utilizes these two subkeys as well. And that's not all…a blog post from 2010 (found online at http://0xdabbad00.com/2010/09/12/how-emet-works) describes how Microsoft's Enhanced Mitigation Experience Toolkit (or "EMET") works and thoroughly describes the use of these two subkeys.

In short, during the process of installing a "Fix-It" database, a subkey is created beneath the *Custom* key, and this subkey's name is the application to be "fixed." The names of the values beneath this key will look like GUIDs (there may be more than one). In this way, this persistence mechanism is very similar to the *Image File Execution Options* key described earlier in this chapter. Then, beneath the *InstalledSDB* key, there will be one subkey for each of the value names beneath the *Custom\{AppName}* subkey, and each key will have several values that pertain the particular database file to which the entry refers. One value, named "DatabasePath", points to the specific database (or *.sdb file) on disk; another value, named "DatabaseDescription", gives a short description of the database.

Microsoft uses this capability to provide legitimate "fixes" for applications. For example, a patch for the Microsoft XML Core Services included the installation of an *.sdb file intended to "fix" Internet Explorer (a description can be found online at https://isc.sans.edu/forums/diary/Microsoft+Security+Advisory+2719615+MSXML+CVE20121889/13459). The "DatabaseDescription" value beneath the *InstalledSDB\{GUID}* keys read "CVE-2012-1889", indicating that the patch was intended to address a known vulnerability. This same capability can be (and has been) abused by malware authors who wish to have their application remain persistent, as well. Symantec's write-up on Trojan.Gootkit (found online at http://ae.norton.com/security_response/print_writeup.jsp?docid=2010-051118-0604-99), first published in 2010 and updated in 2015, describes malware that attempted to "patch" a number of applications, including OutLook, a number of different web browsers, and even the Windows Explorer shell.

The *appcompatflags.pl* plugin can be used to extract and correlate information from the *Custom* and *InstalledSDB* keys in order

to illustrate the contents of the persistence mechanism to the analyst; however, the caveat is that not all of the entries may be malicious, as some may be completely legitimate.

Program Execution

As stated previously in this chapter, very often the program execution category overlaps a good deal with the autostart category. With respect to locations within the Software hive that indicate program execution beyond those already discussed in the autostart category earlier in this chapter, I can't say that I've found many. You may see some applications recorded beneath that *AppCompatFlags* subkeys, but I tend to find indicators more often within the NTUSER.DAT hive than the Software hive; the RegRipper *appcompatflags.pl* plugin can be run against both hives.

LANDesk

Back in 2009, I was working with Don Weber, and he sent me something interesting from an engagement he was working on; specifically, the client he was working with was using the LANDesk product for software and system management within their infrastructure. What he found was the LANDesk application was recording information about program executions, and this information was being maintained in the following key:

```
LANDesk\ManagementSuite\WinClient\SoftwareMonitoring\
MonitorLog
```

I should note that if the operating system is 64-bit and the application is 32-bit, the path will be prepended with "Wow6432Node".

Beneath the MonitorLog key is a series of subkeys, one for each application executed, and in many cases, the key name contains the full path to executable (ie, "C:/Program Files (x86)/Internet Explorer/iexplore.exe"). Each of these keys has several values that appear to be self-explanatory, including "Current User" (the user account under which the application was most recently run), "First Started", "Last Started", "Total Duration", and "Total Runs". An example of these values, and their data, can be seen in Fig. 3.26.

This information can be extremely valuable to an analyst, providing information not only about when programs were first and most recently executed but also by which user account. From the few systems I've seen that have the LANDesk application installed, this information persists and provides a record of program execution much longer than the AppCompatCache data in the System hive, which we discussed earlier in this chapter.

(Default)	REG_SZ	(value not set)
Current Duration	REG_BINARY	00 00 00 00 00 00 00 00
Current User	REG_SZ	SYSTEM
First Started	REG_BINARY	c0 3c c1 9d d6 eb d0 01
Last Duration	REG_BINARY	30 ab 6e 01 00 00 00 00
Last Started	REG_BINARY	a0 26 87 2c a8 ec d0 01
Total Duration	REG_BINARY	d0 6f 2a 04 00 00 00 00
Total Runs	REG_DWORD	0x00000004 (4)

Figure 3.26 LANDesk values seen via the Registry Editor.

The *landesk.pl* plugin, last updated in March 2013, can parse and display this information for the analyst, and the *landesk_tln.pl* plugin will retrieve the same information in a format suitable for inclusion in a timeline of system activity.

Malware

There are a number of malware variants that have been seen to create Registry keys or values that are not associated with persistence. In some cases, the entries created are used to maintain configuration information or some value that the malware uses to identify the infected system. Some time ago, I began documenting many of these keys and values in RegRipper plugins; for example, I wrote the *renocide.pl* plugin based on what Microsoft published about the malware in February 2011 (the write-up can be found online at http://www.microsoft.com/security/portal/threat/encyclopedia/entry.aspx?Name=Win32/Renocide#tab=2). Microsoft's findings indicate that the malware created the *HKLM\Software\Microsoft\DRM\amty* key and then created a number of values beneath that key that held configuration information for the malware.

More recently, I've started documenting many of the Registry keys and values that I find that are used by malware, but not for persistence, within the *malware.pl* plugin. This particular plugin is broken down into sections, depending upon the hive in question, and all of the keys and values I'm interested in checking for are included in this one plugin. So, yes, it will fail to find something in the Software hive if it's specifically looking for something in the System hive. However, the plugin is designed to keep going and not throw

an exemption or do something else to cause RegRipper to have issues. Using it tends to go pretty smooth, and I know I'm going to be really glad when it finds something that I'd forgotten about.

Audio Devices

In November 2014, Brian Baskin posted an article to his blog (found online at http://www.ghettoforensics.com/2014/11/dj-forensics-analysis-of-sound-mixer.html) that discussed analyzing sound mixer settings or artifacts on systems for indications of applications (possibly malware, or more specifically, RATs) that were registered with an audio device. From the article, "Whenever an application requests the use of the Windows audio drivers, Windows will automatically register this application in the registry." In Brian's case, he found a reference to an alternate browser that had been run from a TrueCrypt-encrypted volume.

This is yet another example of artifacts (potentially, indicators of malicious activity) that are left on a system, not due to the direct action of a malicious user or malware (ie, something specifically included in the code of the malware), but rather due to the malware interacting within its ecosystem. As such, as Brian stated in his article, these artifacts tend to persist on a system and survive even an application uninstall.

The RegRipper *audiodev.pl* plugin will enumerate information from the appropriate data sources within the Software hive.

AmCache Hive

The "AmCache" hive is a file that has an identical structure to other Registry files but does not appear to be part of the Registry, at least not as we think of it or view it, such as via the Registry Editor. However, the file can be opened in a hive viewer application (see chapter: Processes and Tools), other tools have been written to parse the information in this file, and I wrote a RegRipper plugin specifically for parsing the file. The file is named "AmCache.hve", and is located in the *C:\Windows\AppCompat\Programs* folder. This file was first seen on Windows 8 and 8.1 systems and then on Windows 10 systems. I'd also seen it on the Windows 10 Technical Preview (by those who downloaded and installed it; I installed it as a virtual machine in Oracle's VirtualBox). In December 2014 and January 2015, folks started noticing that this file was appearing on Windows 7 systems as well, possibly as a result of an update. At the time of this writing, several folks have reported attempting to track down the Windows update that resulted in the creation of the file, but so far, no results have been shared.

Note

While the information within the AmCache.hve file does not appear in the Registry Editor view while the system is running, according to information available on the Microsoft Developer Network (MSDN), the debugger command "!reg hivelist" will result in the AmCache.hve file being listed as part of the response. This information can be found online at https://msdn.microsoft.com/en-us/library/windows/hardware/ff564790(v=vs.85).aspx.

The first time I heard about this file was when I read a blog post from Yogesh Khatri (the post can be found online at http://www.swiftforensics.com/2013/12/amcachehve-in-windows-8-goldmine-for.html) from December 2013. The file appears to be part of the Application Compatibility functionality on Windows systems and reportedly replaces the RecentFileCache.bcf file.

The AmCache.hve file appears to contain two keys that are of primary interest; Files and Programs. Fig. 3.27 illustrates these keys as seen in the AmCache.hve file extracted from a Windows 7 system.

Yogesh addresses and describes the contents of both of these keys (subkeys and values) in his blog post listed previously in this chapter, and in a subsequent blog post, found online at http://www.swiftforensics.com/2013/12/amcachehve-part-2.html.

As you can see in Fig. 3.27, beneath the *File* key are subkeys that Yogesh has identified as volume GUIDs; each of the subkeys identifies a different volume. This volume GUID information can be tracked back to the MountedDevices key in the System hive in order to determine to which drive letter that volume was mounted on the system. As seen in Fig. 3.27, the two volume GUIDs (b75801db-1456-11e0-9309-806e6f6e6963, and b75801dc-1456-11e0-9309-806e6f6e6963) map to the C:\ and D:\ volumes, respectively, on the Windows 7 system.

Beneath each of the volume GUID subkeys, there are additional subkeys, as illustrated in Fig. 3.28.

Yogesh has identified each of these subkeys as the MFT file reference number for a particular file. The first 2 bytes are the sequence number, and the remaining bytes of the key name are the MFT record number.

Let's take a look at an example using the AmCache.hve file first illustrated in Fig. 3.27. Beneath the volume GUID key for the D:\ volume, there is a subkey named "a00001d2", and per Yogesh's research, the key name would indicate that the MFT record number for the file is "1d2" in hex, or 466, in decimal, and the sequence number is "a", or 10 in decimal. Parsing the MFT for the D:\ volume from this system, we see that this is, in fact, the case; the file associated with file record number 466 has a sequence number of 10, or "a" in hex.

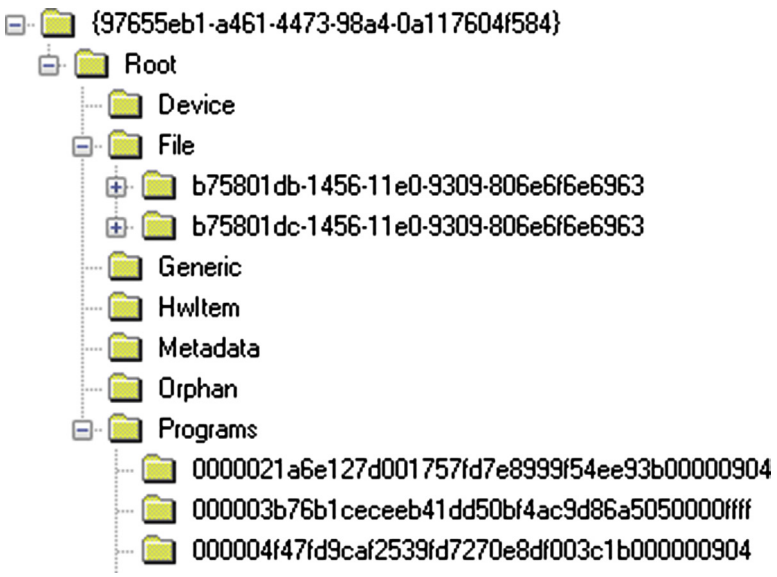


Figure 3.27 Windows 7 AmCache.hve file (opened in WRR).

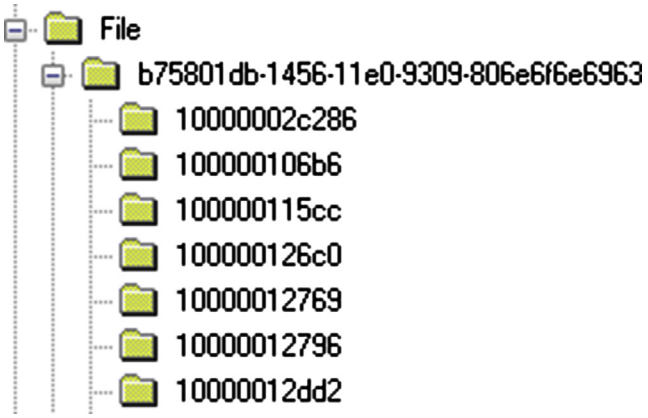
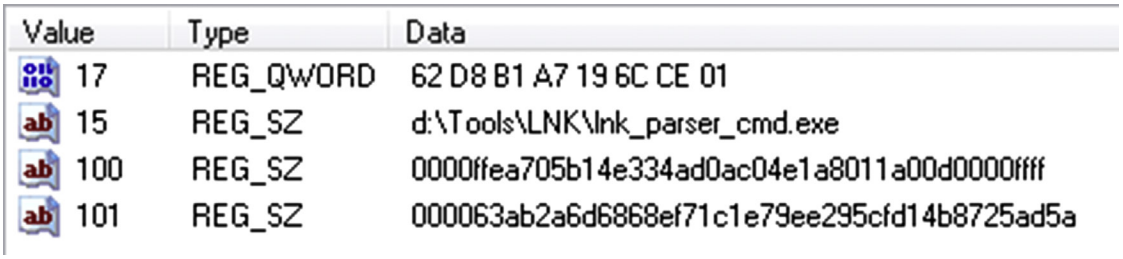


Figure 3.28 Volume GUID key contents (via WRR).

The contents of the "a00001d2" key appear in Fig. 3.29.

As you can see in Fig. 3.29, the key contents refer to a file named "lnk_parser_cmd.exe", which is found on the D:\ volume of this Windows 7 system. Per the MFT from this system, the MFT record in question points to ".\tools\lnk\lnk_parser_cmd.exe".

Yogesh's blog post from December 2013 contains a reference table for the various value names and their data that appear beneath the "file reference" keys. According to that table, value 15 is the full path to the program, value 17 is a last modified time stamp for the



Value	Type	Data
17	REG_QWORD	62 D8 B1 A7 19 6C CE 01
15	REG_SZ	d:\Tools\LNK\lnk_parser_cmd.exe
100	REG_SZ	0000ffea705b14e334ad0ac04e1a8011a00d0000ffff
101	REG_SZ	000063ab2a6d6868ef71c1e79ee295cfd14b8725ad5a

Figure 3.29 "a00001ds" key contents (via WRR).

file, and value 101 is the SHA-1 hash of the file. Yogesh's table (listed in his blog post) contains a greater listing of what various potential values refer to, so be sure to reference the post (in case there are updates), particularly if you need to identify other values.

Tip

The RegRipper plugin *amcache.pl* can be used to parse and display the contents of the AmCache.hve file.

Running a hashing tool (in this case, Jesse Kornblum's sha1deep64.exe, found online at http://md5deep.sourceforge.net) against the file, we can obtain the SHA-1 hash, as follows:

```
D:\Tools>sha1deep64 d:\tools\lnk\lnk_parser_cmd.exe
63ab2a6d6868ef71c1e79ee295cfd14b8725ad5a d:\tools\lnk\
lnk_parser_cmd.exe
```

As you can see, the SHA-1 hash of the file matches the value that appears in the AmCache.hve file seen in Fig. 3.29.

Tip

Analysts can take advantage of the fact that the SHA-1 hash of a file is maintained in the AmCache.hve file. For example, let's say that you're looking at an AmCache.hve file from a compromised system, and you find a key that refers to a file that no longer exists on the system. The key name for the file is "a00001d2", and the path to the file is suspicious (perhaps something like "C:\Windows\Temp\g.exe"). After parsing the MFT, you determine that the sequence number for the MFT record number 466 ("1d2" in hex) is now 12, or "c" in hex. This supports the finding that the file "C:\Windows\Temp\g.exe" was deleted from the system, and as the MFT record was reused, the sequence number was updated accordingly. However, if you suspect that the file was malware, you can use the SHA-1 hash of the file to search a site like VirusTotal.com to see if a file with that hash had been uploaded to the site and identified as malware.

Value	Type	Data
13	REG_DWORD	0x00000000
0	REG_SZ	Microsoft Security Client
1	REG_SZ	4.6.0305.0
2	REG_SZ	Microsoft Corporation
3	REG_SZ	1033
5	REG_DWORD	0x00000101
6	REG_SZ	AddRemoveProgram
b	REG_QWORD	00 00 00 00 00 00 00 00
a	REG_QWORD	E7 F4 8A 54 00 00 00 00
Files	REG_MULTI_SZ	b75801db-1456-11e0-9309-806e6f6e6963@4000031!
7	REG_MULTI_SZ	HKEY_LOCAL_MACHINE\Software\Microsoft\Windo
11	REG_MULTI_SZ	{23f2c78c-e131-4ca0-8f84-3473fb7728ba}
12	REG_MULTI_SZ	{266F7358-F5CB-4BD1-A279-A6992423A9A4}
f	REG_SZ	{23f2c78c-e131-4ca0-8f84-3473fb7728ba}
10	REG_SZ	{266F7358-F5CB-4BD1-A279-A6992423A9A4}
14	REG_DWORD	0x00000000
15	REG_DWORD	0x00000000
16	REG_BINARY	46 41 44 44 00 00 00 00 00 00 00 00 00 00 01 00 30

Figure 3.30 ProgramID key contents.

According to Yogesh's further research into the AmCache.hve file (his findings can be found online at http://www.swiftforensics.com/2013/12/amcachehve-part-2.html), the Programs key contains references to programs installed on Windows 8 (and Windows 7) systems, specifically information that is similar to what can be found in the "Programs and Features" applet in the Control Panel. Most often, this information appears to refer to programs or applications installed via a Microsoft Installer (MSI) file.

Beneath the Programs key (see Fig. 3.27), the subkeys appear to refer to program identifiers (IDs), which are assigned when an MSI file is compiled. Each of these keys contains information similar to what is illustrated in Fig. 3.30.

As with the contents of the Files key in the AmCache.hve file, Yogesh's blog post contains a table that illustrates what each of values visible in Fig. 3.30 references. For example, value 0 is the

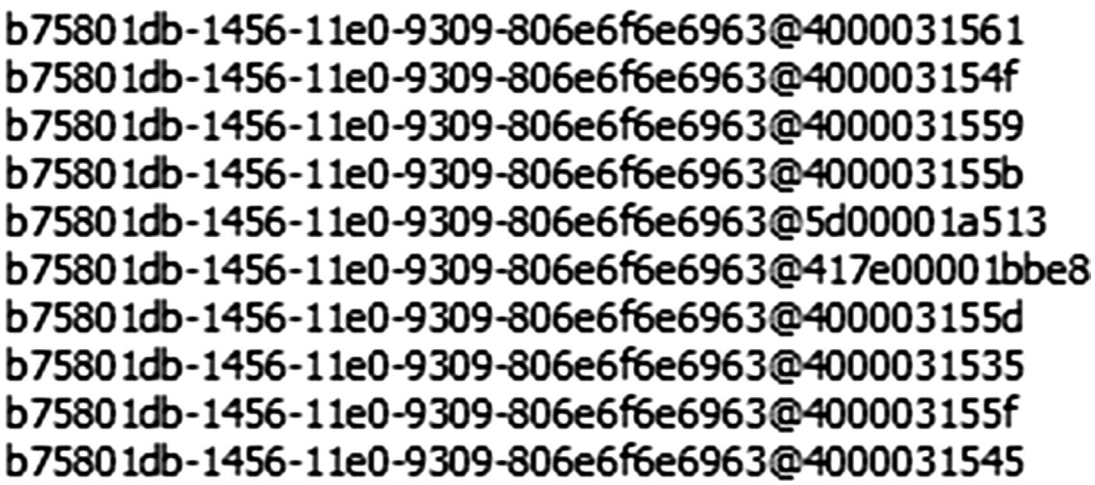


Figure 3.31 Files value data opened in WRR.

program name ("Microsoft Security Client"), and value 1 is the program version ("4.6.0305.0"). Value 6 refers to the Entry Type, or category, under which this program was installed. In many cases, the data for this value appears to be "AddRemoveProgram", but I have seen other data for this value. In this same AmCache.hve file, the ProgramID key that refers to Google Chrome includes a value "6" for which the data is "AddRemoveProgramPerUser".

The "Files" value is interesting, as the data is reportedly a list of files included in the MSI package, but not by file name. Instead the files are listed by volume GUID and the MFT file reference number, separated by the "@" symbol. The data appears as illustrated in Fig. 3.31.

As you can see in Fig. 3.31, the strings listed in the "Files" value point to the volume GUID for the D:\ volume on this system. The portion of each string following the "@" symbol reportedly is the MFT file reference number, but you should not expect to find these beneath the "Files" key within the AmCache.hve file, as these file reference numbers are associated with files that were part of the installer package.

The information beneath the Programs key can be useful to an analyst, depending upon the nature and goals of the investigation. In reviewing the data extracted from the AmCache.hve file from this Windows 7 system, I found that the program "pidgin-otr 4.0.0-1", from "Cypherpunks CA", was installed on the system. Admittedly, malware does not generally get installed on a system through the use of an MSI file, but information such as this can be used to determine programs that were installed on the system and may be useful in demonstrating a violation of corporate acceptable use policies.

Note

I should note that Yogesh's research regarding the AmCache.hve file is based on Windows 8, and the example we used in this chapter validates Yogesh's findings based on Windows 7 systems. However, while the AmCache.hve file has also been found to exist on Windows 10 Technical Preview systems (I have such a system installed as a virtual machine via Oracle's VirtualBox application), this research has yet to be extend to and validated with respect to the Windows 10 platform. This is an important distinction, as Microsoft has a long history of making subtle changes between versions of Windows, changes that can significantly impact research findings from folks like Yogesh.

Summary

In this chapter, we've discussed the Registry hives that pertain to the configuration and operation of the system as a whole and seen how there is a good deal of information available that can be extremely valuable to an analyst during an examination. What I've also tried to do is to break down the various keys and values of interest into artifact categories so that maybe they'd be a bit more manageable.

I've attempted to provide a quick snapshot of information available in the Security, SAM, System, and Software hives, as well as some information regarding a file that uses the same format, but is not part of the Registry; no volume or tome will ever be able to encapsulate every possible key, value, and setting that could potentially be of interest to an analyst. Rather, I've tried to give you, the reader, an overview of what's available, including some of the more common keys and values, and hopefully a few that you haven't seen before, but still said, "wow" when you read about them. I've mentioned some of the Registry keys and values that I, and other analysts, look for most often during examinations; however, that does not mean that these are all the keys and values that contain pertinent information. In fact, this will never be the case; there will always be a new application or new version, or some new malware or technique to compromise a Windows system that leaves a footprint in the Registry. Research into the Windows 7 Registry was still on-going while Windows 8, 8.1, and then 10 were released, and we haven't scratched the surface of what may be "new" in Windows 10. What I hope I have done, and continue to do, is provide you with an appreciation for how powerful a technique Registry analysis can be, particularly when used in combination with other analysis techniques, such as timeline analysis. The Windows Registry truly is a veritable treasure chest of data that can be used effectively by a knowledgeable analyst to add a great deal of context to an examination.