2

PROCESSES AND TOOLS

INFORMATION IN THIS CHAPTER

- Forensic Analysis

Introduction

When I sat down to address and update/rewrite this chapter for the second edition of the book, I wanted to do two things. First, I removed the "Live Analysis" section of the chapter. The reason for this was that for digital forensic analysis, we're not accessing live systems; access to live systems most often occurs during enterprise incident response, and many organizations already have a capability for accessing the Windows Registry on live systems during such incidents, at an enterprise level. As such, our assumed means for interacting with the Windows Registry throughout the rest of this chapter will be through postmortem or "dead box" analysis. After all, this is the "use case" many of us encounter during day-to-day analysis operations; like many, I either access images acquired from systems or I'll be asked to take a look at specific Registry files that were extracted from images.

I should also point out that F-Response (more information on F-Response is available online at http://www.f-response.com) provides remote access to live systems, but does so in manner such that tools mentioned in this chapter can be used very effectively in conjunction with F-Response to achieve the desired analysis.

Second, I wanted to focus more on analysis processes than tools, focusing on the process but understanding that tools can play an important role in analysis, and that analysis processes are often implemented through the use of tools. Over the years, I've been asked time and again what tools I use for Registry analysis, and what it really comes down to is that the analysis process I'm using defines which tools I use. There are a number of tools available for "Registry analysis" but the tool used depends directly on the analysis process being employed.

Analysts faced with extracting and analyzing data from the Windows Registry may be required to do so in a number of different scenarios. During troubleshooting or incident response scenarios, administrators may want to query multiple systems for Registry data, or an analyst may want to examine Registry hives

Windows Registry Forensics. http://dx.doi.org/10.1016/B978-0-12-803291-6.00002-4

extracted from an acquired image for indications of an intrusion or violations of acceptable use policies. Regardless of the data to be extracted and reviewed, an analyst is going to use some sort of tool to collect that data, and to some extent analyze it. In this chapter, we'll address some of the options that an analyst has available and present some tools that may be used in those situations. With this foundation, my hope is that analysts will then be able to make decisions on the best option to use for their particular situation.

In this chapter we will be focusing on the use of open source and freely available tools. There are a couple of reasons for this, the first being that such tools are generally accessible to a much wider audience than commercial forensic analysis applications. Second, I feel that it's important for analysts to understand the mechanics of what they're trying to achieve, to understand what's going on "under the hood" before using the commercial forensic analysis applications. Third, there are a number of open source and freely available tools available that provide functionality, either in and of themselves or as part of a process, that commercial forensic analysis applications do not provide. Finally, as an author, I simply cannot afford to purchase all of the forensic analysis applications, and while writing this book, had access to only one of the commercial forensic analysis applications available on the market (ie, ProDiscover).

The processes and tools presented and discussed in this chapter should not be considered an exhaustive list. These are simply the tools I have used or encountered (mostly used) myself and do not indicate a preference either way. Are there other, better tools? Possibly. However, the point I'm trying to make isn't which is the best tool, but to demonstrate what we're trying to accomplish so that you, the reader, will be able to make a decision as to which is the best tool for you. There may be tools available for Linux or Mac platforms, but I will be sticking to the Windows platform; the tools discussed all run on Windows systems. Some of the tools discussed in the chapter will, in fact, work on platforms other than Windows, which does not restrict an analyst to a particular analysis platform.

KNOW YOUR TOOLS

I took a question once from an analyst who was preparing to go to court over an issue with a former employee having been terminated. The tool this analyst was using to parse the Registry, and specifically the user's TypedURLs key, had produced some confusing output; the tool had listed all of the values with the key's LastWrite time, but without each value's name. The question was, how could all of the values have the same time? After all, the defense would make the claim that no

person could type over a dozen URLs into the browser's address bar at the same time.

I explained the nature of most recently used values, key versus value structures, how the time stamps applied, and hoped that the explanation helped the analyst with their issue. The fact is that if you're going to use a tool, it's important to be aware of what data it's collecting and how it's displaying that data.

Forensic Analysis

The tool you use for a task depends upon the task itself, doesn't it? When you have a task in front of you, there is a process to completing that task, and for the task or perhaps for various steps within that process, there is a tool that will help you complete that task. Some processes may require a single tool; for some laptop systems, the entire process for disassembling and then reassembling the laptop requires a single screwdriver. Other more complicated processes (building a house) may require a completely different tool—a saw, a hammer, etc.—for various steps of the overall process.

Warning

Tools are fine, processes are great, but one of the biggest issues I've seen when it comes to the analysis of data is analysts understanding what they're looking at and correctly interpreting the data. Tools and processes provide an analyst with a means for extracting and displaying various data from the Registry, be it keys, values, or value data, but it's still incumbent upon the analyst to correctly understand and interpret the meaning and context of that data. Particularly in chapters "Analyzing the System Hives and Case Studies: User Hives" of this book, we will address some of the data that are often misinterpreted, as this can have a significant negative impact on the analyst's findings.

Viewing Registry Hives

One of the first things an analyst may want to do is simply view a Registry hive file; that is, load it into a tool that is capable of interpreting the format and displaying the data within the hive in an easy-to-understand manner. Even with the information about key and value cell formats from chapter "Registry Analysis," most analysts are unlikely to open a hive file in a hex editor to view the contents of the file and will most often use a viewer written to display hive files.

Tip

Opening a hive file in a hex editor is not always a bad thing. In fact, it can be a very valuable troubleshooting step. For example, if you're using any of the tools described in this chapter and not getting the information back that you were expecting to see, or any information at all, it's usually a good idea to try opening the hive file in a hex editor to see if it's really a hive file, or if perhaps something went awry in the extraction process. Take a look at the first 4 bytes of the file; do you see "regf"? Then go 4 KB into the file (4096 bytes, or offset 0x1000 in hexadecimal format); do you see "hbin"? There have been instances when analysts have been unable to extract information from Registry hive files, only to finally open the file in a hex editor and see that it's full of zeros. Jamie Levy has seen this same thing time and again when assisting Volatility users who have reported that various plugins don't work when run against a memory dump… sometimes part of the process goes wrong and you don't get the data that you were expecting. Before reaching out to get assistance, take a moment to check the data itself, as invariably, that's one of the questions that someone attempting to assist you is going to ask.

RegEdit

Perhaps the commonly available tool for viewing the contents of the Windows Registry is the Registry Editor (aka RegEdit). Many administrators and analysts are likely familiar with interacting with the Windows Registry via the RegEdit tool, as illustrated in Fig. 2.1, which is a tool native to Windows systems.

You can load a Registry hive exported from an image by clicking on the HKEY_LOCAL_MACHINE hive in the RegEdit user interface, then selecting File from the menu bar, and choosing "Load Hive." When the "Load Hive" dialog appears, give the hive you're loading a unique name (such as "Software_Test") and click "Ok." The hive will be added to RegEdit, as illustrated in Fig. 2.2.

Once analysis is complete, simply unload the hive. Select the hive, as illustrated in Fig. 2.2, click File in the menu bar, choose "Unload Hive." Click "Yes" on the "Confirm Unload Hive" dialog that appears, and you're done.

RegEdit offers some limited search functionality but doesn't provide an easy means for looking at key LastWrite times and doesn't offer any functionality for translating binary data into something useful to the analyst. After all, RegEdit is a tool that Microsoft developed for administrators to access the Registry, not specifically for forensic examiners and incident responders. However, this application can be useful in providing an analyst with an initial look at a hive file.

Tip

An artifact of the use of RegEdit is that when the application is closed, the last key that was in focus at the time is recorded in a value in the user's NTUSER.DAT hive file. We'll go into this in more detail in chapter Case Studies: User Hives, but suffice to say at this point that this has been an extremely valuable forensic resource on more than one occasion.

Windows Registry Recovery

The Windows Registry Recovery (WRR) from MiTeC is a tool that I like to use if I simply want to view the contents of a Registry hive file. One of the things I like about WRR is that the interface is very similar to that of RegEdit, and as such, it's nice to be able to operate in a familiar environment.

When you first launch WRR and open a hive file, you'll be presented with an interface similar to what is shown in Fig. 2.3.

On the left-most side of Fig. 2.3, you'll notice that there are several "Explorer Tasks" available, which can be very useful

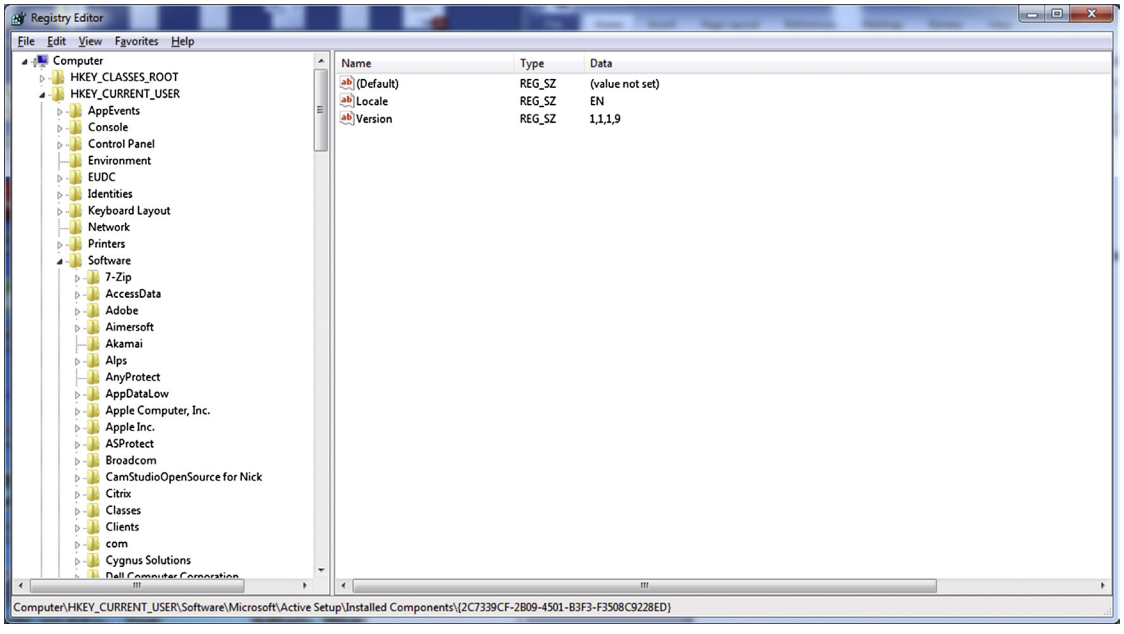

Figure 2.1 The Windows Registry Editor.

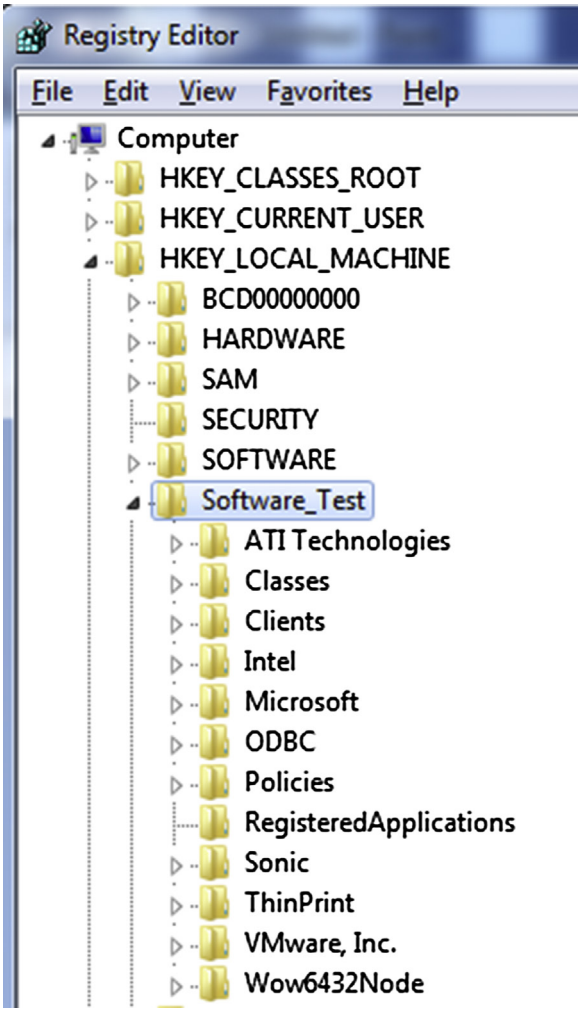

Figure 2.2 Exported hive loaded in RegEdit.

for collecting and parsing specific data from specific hive files. Clicking on any of the available buttons will populate a view with the data retrieved from the hive. Something to be aware of is that the buttons do not pertain to actions that work for all hives. For example, with a Software hive loaded, clicking on the "Windows Installation" button will populate a view similar to what is seen in Fig. 2.4, because the information is available in that hive. Clicking on the "Hardware" button, however, will populate a view that simply states "<no information found>" because the necessary information is found in the System hive, not the Software hive.

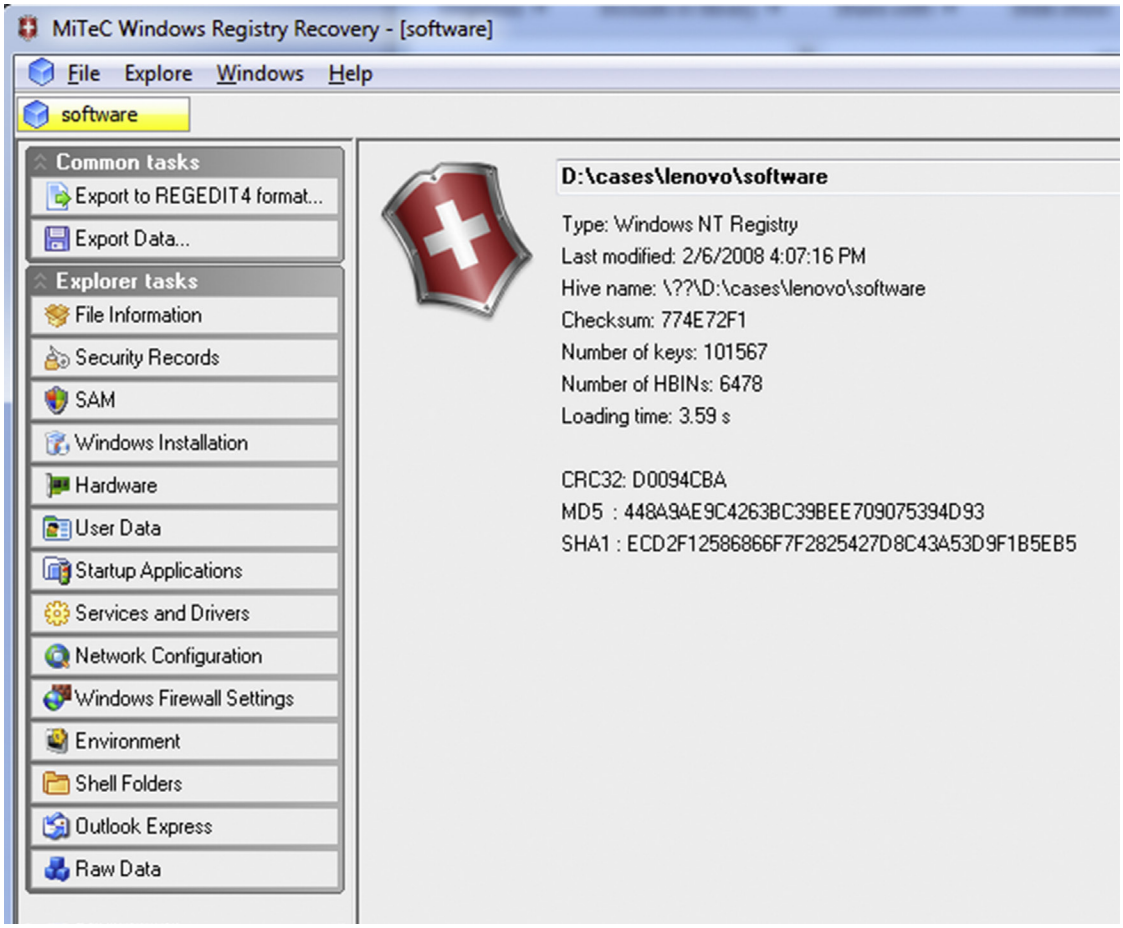

Figure 2.3 Partial WRR user interface.

The Explorer Task functions act as parsers (discussed later in this chapter), albeit without any specific indicators as to the hives to which they apply. Even so, these functions can still be very revealing to an analyst, providing insight into components and information available from various hive files. The information these functions can retrieve is not extensive but it can be informative.

WRR also has a pretty good search capability, or "Find" function that can be very useful when looking for indications of specific artifacts or indicators within a hive file. With a hive opened in WRR, and the "Raw Data" view opened, click on the button with the magnifying glass icon to open the Find dialog, as illustrated in Fig. 2.5.

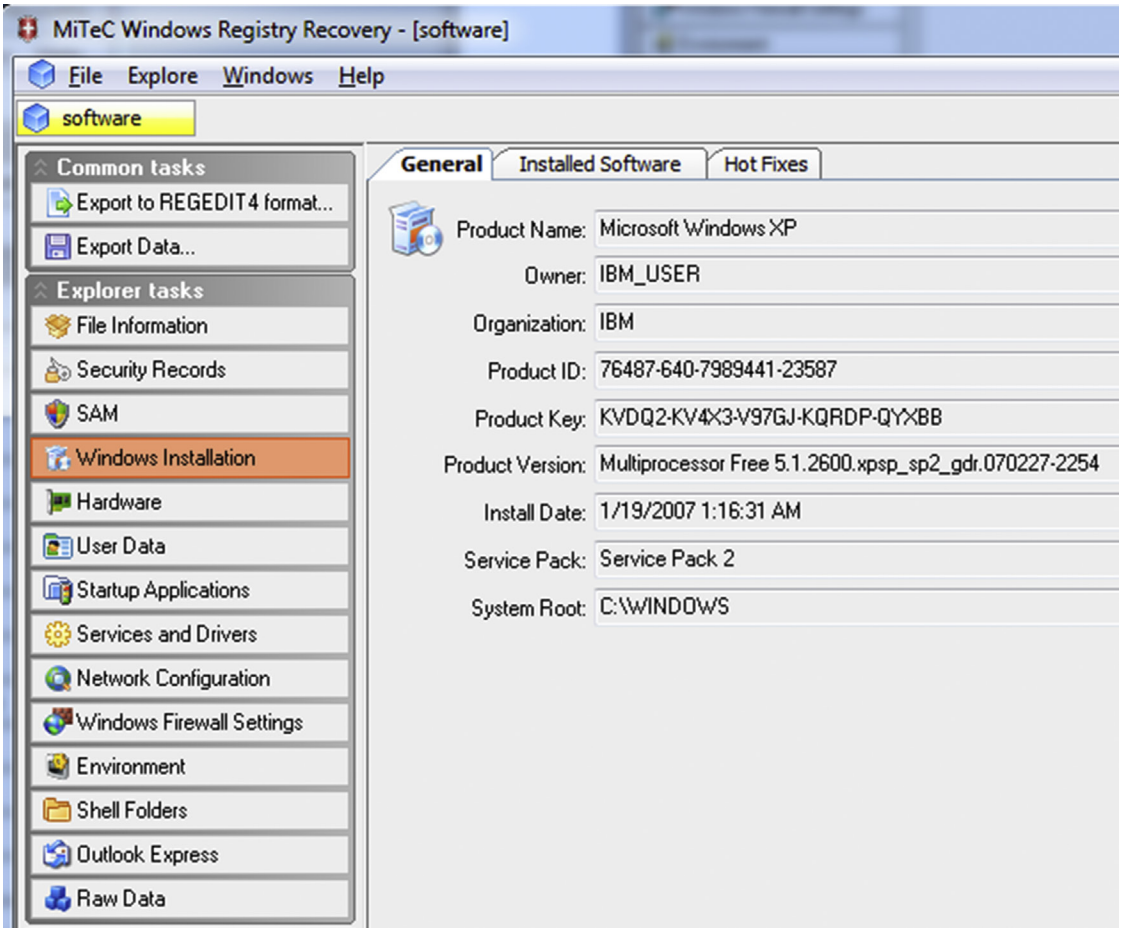

Figure 2.4 Windows installation info available via WRR.

Once the dialog is open, enter your search term, select what structures you want searched (keys, values, data), and click "Find Next". Depending upon how large the hive file is, the search can take several minutes. When the search is complete, any hits will be displayed in the bottom-most pane in the WRR user interface, and double-clicking on any of the hits will cause that location to be opened for viewing. I've used this search functionality to look for globally unique identifiers (GUIDs), key and value names, as well as portions of text that may occur within value data.

Something else that's very useful about WRR is that with the "Raw Data" view open, you can right-click on a key, choose "Properties," and view information about the key, such as the index, the relative offset of the key structure within the hive file, and the Last-Write (or "Date Modified") time.

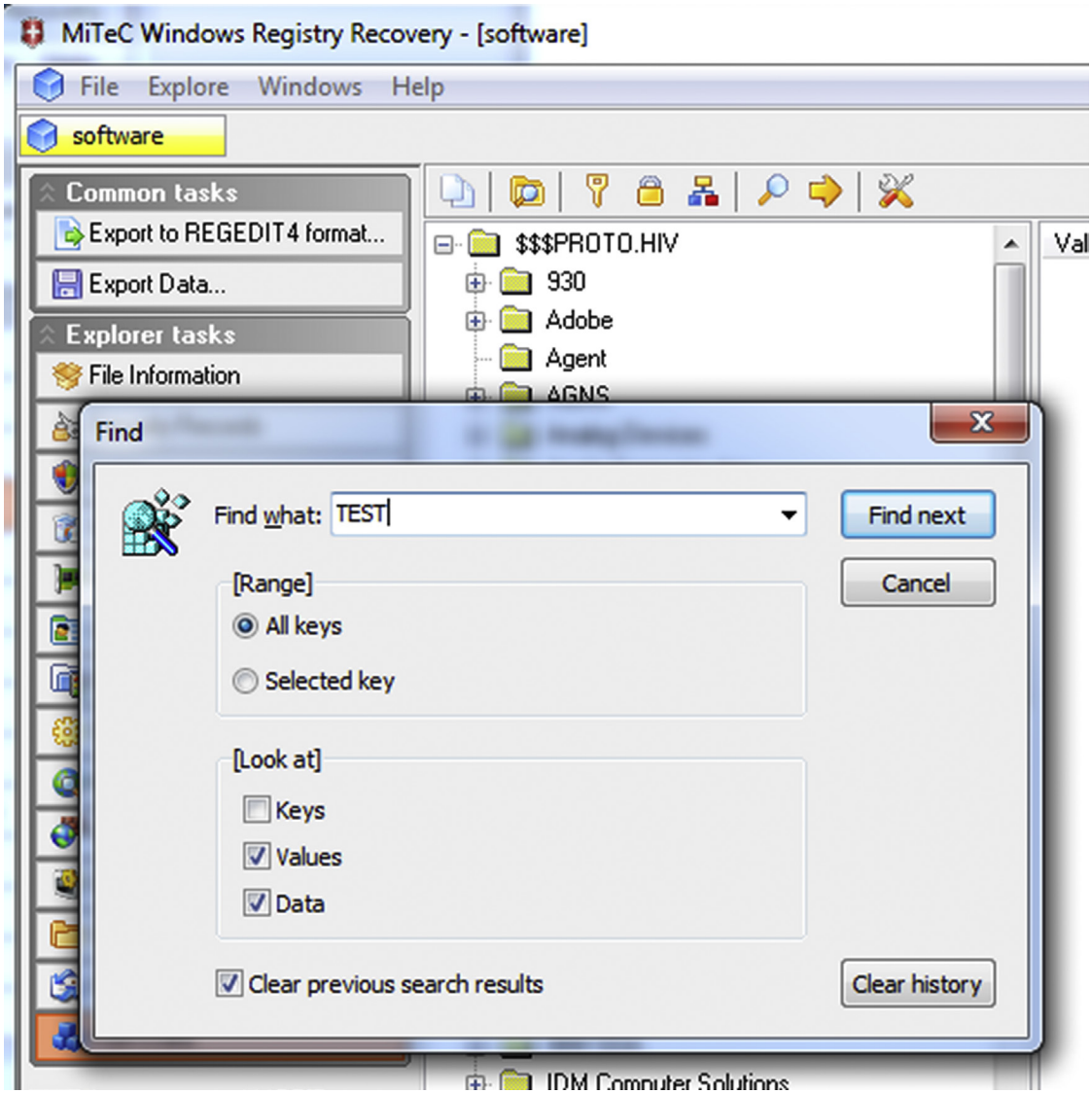

Figure 2.5 "Find" dialog in WRR.

Perhaps my most prevalent use of WRR is to use it in conjunction with other analysis processes, such as to view the values and data within a specific key of interest during timeline analysis (more information regarding timeline analysis will be presented later in the chapter). Knowing that a key was modified at a specific time is very helpful, but it can be even more helpful to understand either what values and data are beneath that key, or what

was actually modified. I've also used WRR to browse through a hive file after other analysis processes have completed, looking for data that may be of use. This is usually a less specific approach, but often results in interesting findings that I can incorporate into other, future analysis. For example, in one instance, I found that specific information about a particular model of cell phone had data stored within the Software hive of the system to which it had been connected, and that information included the electronic serial number, among other things. There have also been times where I've discovered information about other Registry keys and values that were unrelated to the case at hand but may be useful during future analysis.

As mentioned, a drawback of WRR is that there is nothing that identifies to which hives the specific data extraction applies. What I mean by that is that if you open a Software hive in WRR and click the "Services and Drivers" button, you will be presented with a "Services" and a "Drivers" tab, both of which will be empty. Some of the buttons will display "no information found" if the hive file does not contain the information that the function is attempting to retrieve. This functionality can be very useful, if you are aware of what data is being retrieved, and from which hive file.

Another drawback of WRR is that it doesn't handle "big data" at all. What I mean by "big data" is binary value data types that are larger than 2 or 3 KB. Now, there aren't many values that have "big data"; there is one that many forensic analysts look to (the ShimCache or "AppCompatCache" data, which we will discuss in greater detail in chapter Analyzing the System Hives) for clues, and it's clear that WRR doesn't handle that data. Not only does it not parse it and display it in a more readable manner, but it doesn't properly read the data within the hive so that it can be exported from the hive and parsed with another tool.

Registry Explorer

Eric Zimmerman has spent considerable time and effort developing a tool (using .NET) called Registry Explorer. At the time of this writing, Eric has graciously made version 0.7.1.0 of Registry Viewer available. A portion of the Registry Viewer interface appears in Fig. 2.6.

From just the portion of the interface available in Fig. 2.6, which illustrates a Software hive loaded into the view, you can already see that it has some pretty interesting features. For example, when the hive is loaded, you can see the "live" hive, or the keys and values within the hive that you would be able to see in the Registry Editor. You can also see deleted keys and values,

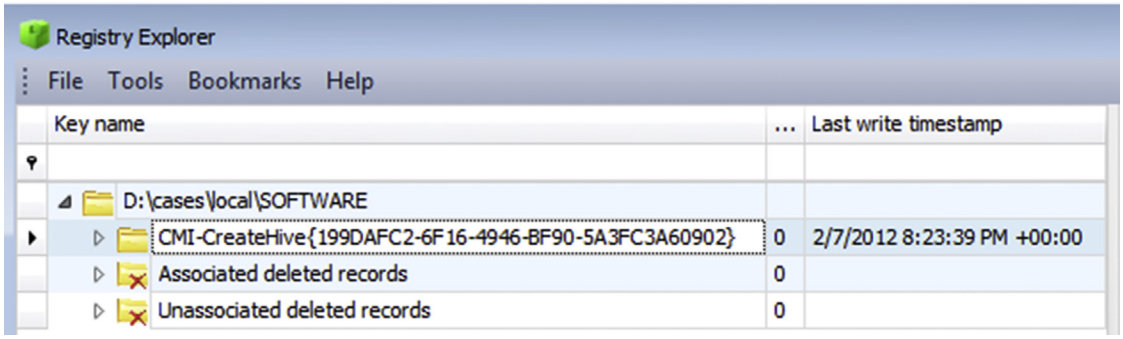

Figure 2.6 Portion of the Registry Viewer interface.

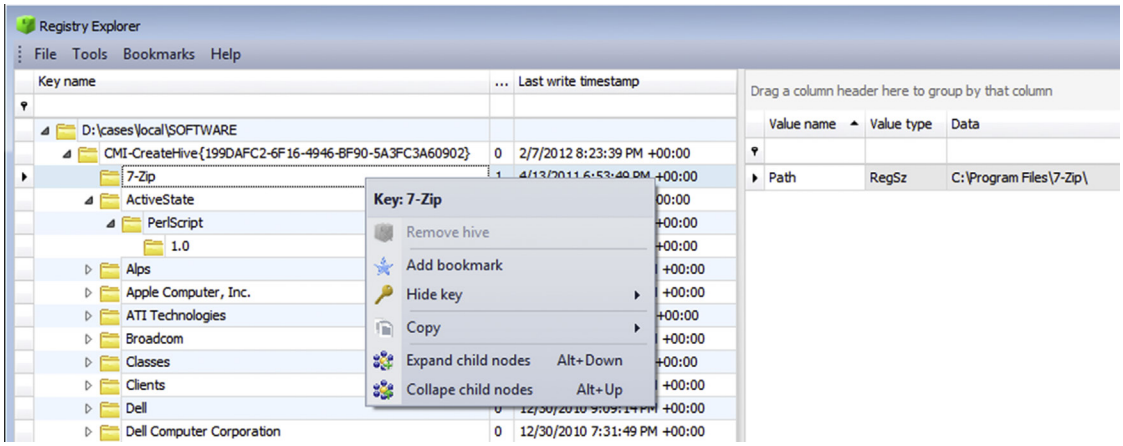

Figure 2.7 Registry Explorer user interface.

which have been broken out into "associated" and "unassociated" deleted records. The "associated" deleted records are those that can be traced all the way back to the root key of the hive, much like a file path that is traced all the way back to the root of the volume, such as "C:\". As such, the "unassociated" deleted records are those that are free standing and cannot be traced back to the root key of the hive.

Fig. 2.7 illustrates a Software hive opened in Registry Explorer.

As you can see in Fig. 2.7, the left-hand pane of the user interface displays Registry keys in the familiar folder view, with the key LastWrite times visible just to the right of the key. Right-clicking on a key brings up a context menu. Values beneath the key are displayed in the right-hand pane.

Something that isn't shown in Fig. 2.7 is that the Registry Explorer will display value slack, which is illustrated in Fig. 2.8.

Value name	Value type	Data	Value slack
Apoint	RegSz	C:\Program Files\DellTPad\A...	00-00
Broadcom Wireless Manager UI	RegSz	C:\Program Files\Dell\DW WL...	
MSC	RegSz	"C:\Program Files\Microsoft S...	73-00-20-00-28-00
NvCplDaemon	RegSz	RUNDLL32.EXE C:\Windows\...	00-00
NVHotkey	RegSz	rundll32.exe C:\Windows\sy...	00-00-00-00
nwiz	RegSz	nwiz.exe /installquiet	00-00-00-00
SysTrayApp	RegExpandSz	C:\Program Files\IDT\WDM\s...	

Figure 2.8 Registry Viewer displaying value slack.

Value slack is space left over when the actual value data does not occupy all of the space allocated to the data.

Pros and Cons

As with any other tools, Registry viewers in general have their own pros and cons, strengths and weaknesses. For example, the Registry viewer applications we've discussed here are all GUI tools and make the information within hive files very visible and accessible. In fact, as you've seen, several of the tools can make a great deal of information available to the analyst.

A "con" of viewers is that the data visible in the tools very often needs to be translated by some other process or tool before it can be incorporated into a more comprehensive analysis process, such as timeline analysis. While several of the viewer applications give you access to the key LastWrite times, if this information is to be incorporated into timeline analysis, it must be entered manually into the timeline. This is not a big deal for a single key, but quickly becomes an issue, as this manual process does not scale well. Also, not all of the GUI tools provide the means to parse and translate the binary data values that are often of interest to an investigator.

That being said, please do not misunderstand me…viewers do have a place in analysis processes. Viewers are excellent when you need to develop some awareness of the type of hive file that you're dealing with, or if you want to browse around the various data within the hive file. When I first encountered one of the new hive files available on Windows 8 systems (more about that in chapter Analyzing the System Hives), I opened the hive file in a viewer just to see what it contained. As the hive file followed the same structure used by other hive files, it was easily opened for viewing.

Tip

Registry viewers are great not only for exploring hive files you've extracted from images for new information but also for exploring hives from other systems. A few friends have shared hive files extracted from Windows Phone 8 smart phones and other systems to which they have had access, and because those hives follow the same structural format as most of the hive files we're familiar with, they've opened for easy viewing and exploration.

Parsers

Tools that parse specific Registry data are different from viewers in that parsers allow analysts to retrieve and decode very specific data sources from within the Registry itself, rather than simply opening the entire Registry for viewing. As we've seen so far, viewers can be very useful, offering the analyst search functionality and providing some capacity for parsing data (albeit limited).

Parsers are also very useful, in that most viewers do not allow for parsing of specific data, and there is a good deal of data within the Registry that requires some sort of parsing or decoding in order for it to be of value to the forensic analyst. In order to get to specific data via a viewer application, analysts need to remember paths and traverse entire structures, which can be very cumbersome, particularly if you're dealing with multiple hives or multiple values within a hive file. Further, a viewer isn't the best tool to use when you're trying to correlate data from multiple keys or values, even within the same hive file, or if you need to parse binary value data that includes things like time stamps, flags and other data within the structure. Many of the available parsers tend to be much better suited to this sort of thing, with the only shortcoming being that it can be difficult to incorporate the output of different parsing applications into other analysis processes (without some sort of transition to and from an intermediate format, that is).

Note

A very useful parsing tool is Microsoft's own Autoruns (found online at https://technet.microsoft.com/en-us/sysinternals/bb963902) application, which includes both a GUI and command line version (autorunsc.exe). This tool is advertised as being able to "analyze offline systems," which I have taken to mean an acquired image mounted as a (read-only) volume. However, when attempting to do so using Autoruns version 13 on several mounted (Windows XP, Windows 7) images, the application would crash. I have found several locations online where the use of this application has been discussed and successfully used, and I have tried using similar parameters but as yet have been unable to get the application to work. However, the application does provide quite a comprehensive listing of locations within the Registry from where applications (including malware) can be automatically started.

There are a number of parsing tools available, and the one (or ones) you may choose to use depends upon which data sources within the Registry you'd like to parse. In this section, I'll present several of the available tools, and this should not be considered to be an exhaustive list, as I am sure that there are tools available that I haven't seen yet. As such, this list should be viewed as simply an example of what is available.

Didier Stevens has a tool named simply "UserAssist" (found online at http://blog.didierstevens.com/programs/userassist/) which parses the UserAssist data from within a user's NTUSER.DAT hive file for versions of Windows ranging from Windows 2000 to Windows 8. This tool can help an analyst get a view as to what applications had been run on a system; in the case of the UserAssist data, the tool provides information about applications a user executed, most often by interacting with the system via the Windows Explorer interface. For example, information recorded in the UserAssist subkeys indicates applications that the user launched by double-clicking an icon on the Desktop, or one that was visible in a folder opened in Windows Explorer, or by traversing the Program Menu on the Taskbar.

The folks at Mandiant (now part of FireEye) released the Python-based ShimCacheParser (found online at https://github.com/mandiant/ShimCacheParser) for parsing the AppCompatCache value data from Windows systems up through Windows 8. According to research conducted by the Mandiant team, this data is part of the Application Compatibility functionality within Windows and can provide indications of applications executed on a system. For a more detailed explanation of their findings, see the *Leveraging the Application Compatibility Cache in Forensic Investigations* white paper found online at https://dl.mandiant.com/EE/library/Whitepaper_ShimCacheParser.pdf. I've used the information from this value data to great effect in my analysis, as it has indicated the use of various applications well after those applications have been removed from the system. This value data will be discussed in greater detail in chapter Analyzing the System Hives.

Tip

Something I really appreciate about a lot of the Registry keys and values during investigations involving intrusions and/or malware is that many of them are not created directly by the intruder or the malware, but instead by the intruder or malware's interaction with the "ecosystem." A valuable side effect of this is that efforts to remove indications of malicious activity often leave these indicators behind, and they tend to persist well beyond these "antiforensics" efforts.

In December 2014, Eric Zimmerman released version 0.5.0.4 of his ShellBag Explorer tool (at the time of this writing, found online at https://www.dropbox.com/s/lw9d0zrzqcrccy4/ShellBagsExplorer.zip?dl=0), a GUI tool for parsing a user's shellbag entries in an Explorer-style format. Shellbags are an artifact associated with folders accessed by a user through the Windows Explorer interface (or "shell"). I've used the "shellbag" artifacts to develop an understanding of resources accessed by the user, particularly during incident response involving targeted threat actors (commonly referred to as "advanced persistent threats" or "APT") that have foregone the use of malware and have started accessing the compromised infrastructure through Terminal Services. These artifacts will be discussed in greater detail in chapter Case Studies: User Hives.

Eric also provides other parser tools for download via the web page found online at http://binaryforay.blogspot.com/p/software.html.

In January 2015, Willi Ballenthin released his amcache.py Python script (found online at https://gist.github.com/williballenthin/ee512eacb672320f2df5) for parsing the AmCache.hve Registry hive file introduced in Windows 8. Research conducted by Yogesh Khatri, and provided through his blog (found online at http://www.swiftforensics.com/2013/12/amcachehve-in-windows-8-goldmine-for.html), indicates that this file, while not specifically part of what we tend to think of as the "Windows Registry," is formatted in the same manner as Registry hive files and contains information about recently executed applications. As with other data, this file can assist analysts in finding indications of application executed on the system.

Tip

During January 2015, I downloaded the Windows 10 Technical Preview and installed it into a virtual machine running in Virtual Box. After interacting with it a bit, I checked and saw that Windows 10 also has an AmCache.hve file. Later, after updating my Windows 7 host system and guest virtual machines (within Virtual Box), I found that these system also now have an AmCache.hve file. So while the big question from forensic analysts when a new Windows operating system is developed is, "what's new?", it appears that what was once "new" (as of Windows 8) is not only included in Windows 10 ("new–new"), but Windows 7 has also been "retrofitted" with this functionality.

This list of parsing tools is simply an example and should not be considered to be all-inclusive. There are other tools available that will retrieve specific data from Registry hive files and display that

data in a manner that is perhaps more meaningful for the analyst. All of the listed tools, as well as many of those that aren't listed here, are excellent tools that an analyst can use to gain insight into various data sources available within the Registry, and further their investigation.

Pros and Cons

While parsers are very useful tools and can quickly provide an analyst with insight into specific keys and values, the one drawback is that with many of them, it's not terribly easy to incorporate the retrieve data into other, more comprehensive analysis processes. So, while an analyst may not need to memorize Registry key paths and spend a great deal of time traversing tree structures in a viewer, they may need to utilize an intermediate data format and some sort of "glue" application (maybe a Perl or Python script) to incorporate that data into other analysis processes.

RegRipper

RegRipper started out as a series of separate scripts I'd written to parse out specific Registry keys, values and data, and after a while, I had more scripts than I could reasonably keep track of and maintain separately. Also, I found that over time, I was running the scripts over and over again, in the same (or similar) sequence. So I sat down and looked at what was common across not only the scripts, and also how I was using them, and from there I created a framework where I could easily reuse what was common and repetitive and focus just on writing the code to parse and translate various keys, values, and value data. I guess you could say that as a whole, RegRipper is something of a "superparser"; rather than parsing one specific subset of data from a hive, such as a user's UserAssist data or the AppCompatCache data, RegRipper allows an analyst to parse a wide range of data from hives, even combining and correlating data from within a hive file.

One of the things I find most useful about RegRipper (and yes, I am more than a little biased, I admit) is its flexibility, and how quickly a plugin can be developed, modified, and deployed. Several years ago, I was working on my own incident response case when another analyst on our team said that he'd found portable executable files stored in Registry values in the case he was working on. Within a few minutes, I wrote a plugin that iterated through the Registry values in a hive file, looked for binary data types that were 1024 bytes or greater in size, and then looked for any that started with "MZ" (the first 2 bytes of a Windows portable executable file). More recently, when I became aware that the right-to-left

override (RLO) Unicode control character was being used in both the file system and the Registry to "hide" the existence of malware, I wrote a plugin to detect the use of this character in Registry key and value names within hive files.

Tip

The use of the RLO character to obfuscate the existence of malware on a system was discussed in the "How to Hide Malware in Unicode" article on the Dell SecureWorks Research blog, found online at http://www.secureworks.com/resources/blog/how-to-hide-malware-in-unicode/.

When the Poweliks malware was identified and seemed to be blasting its way across the Internet in the latter half of 2014, I was able to quickly write and utilize a plugin to help determine if a system was infected with the malware. The design of RegRipper, as a framework, makes it easy to update without having to recompile and redistribute significant code bases; all that is required is that a plugin be written, tested, and distributed. As the plugins are really nothing more than text files, they can be quickly and easily shared, and even reviewed and modified just as quickly. RegRipper will be covered in greater detail in chapter "RegRipper" of this book. We will also discuss in that chapter how analysts can derive greater use from RegRipper.

Timeline Analysis

Timeline analysis is an extremely powerful analysis process, as it not only allows an analyst to see when various events occurred, but it also allows them to see when those events occurred within the context of other events. Creating a timeline of system activity from multiple data sources found on Windows systems is covered in detail in Chapter 7 of *Windows Forensic Analysis*, which can be found on Amazon at http://www.amazon.com/Windows-Forensic-Analysis-Toolkit-Edition/dp/0124171575.

From chapter Registry Analysis, we know that Registry keys contain metadata within their structure that is called the Last-Write time, which is analogous to the last modification time of that key. This time can pertain to when the key was created, or when a value or subkey was added or deleted. In short, the time pertains to when the key was modified in some way. We also know that the structure of a Registry value does not contain similar metadata;

there is no element of the structure that records when the value was last modified. Something we will see more in the remaining chapters of this book is that there are a number of values throughout the Registry whose data (be it binary or string data) contain time stamps that can be used in timeline analysis. So, in short, there is a great deal of potentially valuable data in the Registry that can be added to a timeline, and much of this we will discuss in much more depth in the subsequent chapters of this book. However, when creating a timeline, one of the first data sources I tend to add to a timeline is the key LastWrite times from applicable hive files, very often starting with (but not stopping with) the Software and System hives. The tool I use to this is regtime.exe, available online as both a Windows .exe file and a Perl script from https://github.com/keydet89/Tools. Using the tool is fairly straightforward; simply type a command similar to the following at the command prompt:

```
C:\tools>regtime -m HKLM/Software/ -r d:\cases\test\
software >>
d:\cases\test\events.txt
```

The regtime.exe tool runs through the hive file (in this case, a Software hive), extracting the key names and paths, and LastWrite times, prepending "HKLM/Software/" to the key path, and printing this information to the console (ie, STDOUT) in the appropriate format. As the output is sent to the console, we want to be sure to redirect it to the appropriate file; in this case, we've opted to append it (rather than creating an entirely new file) to the events file that will be the basis for our timeline.

Tip

I tend to have the tools I write send their output to the console so that they're more flexible. For example, let's say you're only really interested in Registry keys beneath the Classes subkey, because you suspect that one of the subkeys may have been added or modified as a malware persistence mechanism. Rather than incorporating all of the keys within the Software hive into your timeline (or using them all to create a nanotimeline of just the Registry key LastWrite times), you can pipe the output of the tool through a "find" command in a manner similar to the following:

```
C:\tools>regtime -m HKLM/Software/ -r d:\cases\test\software| find
"Classes" /i >> d:\cases\test\events.txt
```

The above command will only redirect those key paths that include the word "Classes" (the "/i" switch means that case is irrelevant) to the output file.

The use of regtime.exe is pretty flexible and doesn't lock you into just one analysis process or mechanism. For example, if you have a regular, consistent process that you use, you can easily automate this process. If one of the steps in your analysis process is to extract the hive files of interest from an acquired image or to mount the acquired image as a read-only volume, it can be easy to automate the use of this tool using a batch file.

The current version of RegRipper (version 2.8, available online from https://github.com/keydet89/RegRipper2.8) available as of this writing includes a number of plugins that end in "_tln". These plugins provide output in the five-field TLN format, similar to regtime.exe, but tend to primarily focus on extracting time-stamped information from various Registry value data rather than key LastWrite times. As such, it's often advantageous to run both regtime.exe and various RegRipper plugins (depending upon the nature of your investigation) as this approach may provide indications of Registry keys that were modified, which may then become pivot points in your analysis. I've done this time and again; in fact, not long ago, this technique provided indications of a Registry artifact associated with specific malware that seems to provide indications not only that the malware is/was installed on the system but also provides a possible indication of the version of the malware. Additional analysis and research is required to validate this finding, but this finding may have been missed if a comprehensive, inclusive analysis approach hadn't been employed from the beginning.

Not all RegRipper plugins that produce TLN output focus solely on parsing value data for time stamps. The secrets_tln.pl plugin, for example, retrieves the LastWrite time for the "Policy\Secrets" key within the Security hive. I wrote this plugin because the work that I do involves responding to incidents where intruders have stolen credentials from an organization, and one of the tell-tale indicators of a specific use of the tool gsecdump.exe, as described in *The Art of Memory Forensics* (information about the book can be found online at http://www.memoryanalysis.net/#!amf/cmg5), is that the LastWrite time for this key is modified. Once I heard of this, I wrote the plugin and began incorporating the information in timelines. However, rather than running regtime.exe against the entire Security hive and incorporating a lot of extraneous data in my timeline (the increased volume simply makes the timeline more cumbersome to analyze) I only included the output of the plugin. Over time, I saw the finding shared by the Volatility team validated time and time again.

Warning

When incorporating time-stamped information from any data source to a timeline of system activity, analysts must keep the context of that time stamp in mind at all times. Misunderstanding the context of the time value can lead to significant misinterpretation of the data in the timeline. We'll address this again in much greater detail in chapter Analyzing the System Hives (it's important enough to mention several times) but a great example of this is the time stamps within the ShimCache, or AppCompatCache value, data from the System hive file. For all but one version of Windows, there is only a single time stamp for each entry in the AppCompatCache data, and that time stamp is last modification time from file system metadata (ie, the $STANDARD_INFORMATION attribute for NTFS file systems). This time stamp is often misunderstood to be the last time the listed application was executed, and this misinterpretation has led to analysts misidentifying the window of compromise on the system. This misinterpretation has been seen in conference presentations that have been made publicly available.

Differencing

Determining the difference between two Registry hive files can be a very valuable analysis technique, particularly when conducting malware analysis, or comparing historical data to "current" data and attempting to determine changes that may have occurred between two points in time. Tools such as RegShot (found online at http://sourceforge.net/projects/regshot/) allow malware analysts to create a snapshot of the Registry at one point in time, take some action (infect a system with malware), and then compare a subsequent Registry snapshot to the first one in order to determine what changes may have occurred as a result of the action that had been taken.

"Diffing" Registry hive files is not something that is solely reserved for use by malware analysts. Comparisons of Registry hive files can be very useful during digital forensic analysis, by comparing the currently available hives to those from Volume Shadow Copies (VSCs) or those available within the *C:\Windows\system32\config\RegBack* folder.

Tip

I've used this technique a number of times to determine the nature of Registry data that I was analyzing; specifically, was the data I was looking at newly created, or had it been modified by some action? In most cases, I had found that a Registry key had been modified during a particular time period, and the other artifacts that I observed "near" that time indicated that a malware infection had occurred. By comparing the contents of the hive file to a previous iteration of the hive, either from within the RegBack folder or from a selected VSC, I was able to determine that the key had been created, rather than modified, at that time. This has helped me to identify artifacts that can then be used as indicators to determine if other systems had been infected with the same, or similar malware.

The Perl module on which RegRipper is based, Parse::Win32 Registry, ships with several scripts, one being regdiff.pl. When you install the module (if you need to do so), the script is automatically added to your system in the \Perl\site\bin folder. Using this script is very simple; just type the name of the script and include the two Registry hive files that you want to compare, similar to the following:

```
C:\Perl\site\bin>regdiff.pl d:\cases\test\software d:\
cases\test\regback\software
```

As with other command line tools, the output of the script is displayed in the command prompt window. As and as such, the output can be filtered through a "find" command, or it can be redirected to a file for preservation and later analysis.

There is also a command line tool named regdiff.exe, available online from http://p-nand-q.com/download/regdiff.html, which will allow you to compare two Registry files in much the same way as the regdiff.pl Perl script.

Deleted Keys and Values

In the spring of 2008, Jolanta Thomassen contacted me about providing an idea (and being a sponsor) for her dissertation for work at the University of Liverpool. I pointed her to an old (c.2001) post on the Internet, asking about unallocated space within a Registry hive file. Not long after, Jolanta produced regslack, a Perl script that combs through a hive file and locates deleted keys and unallocated space. If you remember from chapter Registry Analysis, when a key is deleted, the first 4 bytes (DWORD) of the key, which is the length of the key, is changed from a negative value (as a signed integer) to a positive value. For example, if the "live" key had a length of –120 as decimal value, then the deleted key length is 120.

Regslack is a command line tool, and very easy to use. Simply open a command prompt and pass the path to the Registry file in question to the program:

```
C:\tools>regslack.pl d:\cases\test\software
```

Regslack sends its output to the console (ie, STDOUT), so be sure to redirect it to a file (ie, "> file"), as in some cases, there can be quite a lot of information. Regslack has proven quite useful during a number of examinations. For example, if you find indications of a user account being active on a system but can't find that account listed in the SAM hive, try running regslack against the hive file. In one instance, I found indications of a user account with the name "Owner" and an RID of 1003 in the Event Logs on the system, but no indication of such an

account within the SAM hive. Running regslack, I found the following:

```
SAM\SAM\Domains\Account\Users\Names\Owner
Offset: 0x3c70 [Fri Jun 18 17:03:22 2004]
SAM\SAM\Domains\Account\Users\000003EB
Offset: 0x3d08 [Fri Jun 18 18:59:27 2004]
```

The second key (Users\000003EB) had two values (F and V) associated with it, just as you'd expect for a local user account. The V value included the name "Owner." Thanks to regslack, I'd found the user account, as well as the time when the account had been deleted, indicated by the time stamp "[Fri Jun 18 17:03:22 2004]." With a little more work, using Perl code that I've already written (as part of RegRipper), I could extract and translate that binary data from those values into something a bit more understandable.

I have also used regslack to great effect to recover deleted keys and values from a user's hive file, in particular after the user had run an application called Window Washer on their system. I researched the version of the application and found that it reportedly did delete certain keys when run. Sure enough, the key was not visible in the allocated (or "live") space within the hive file, but it was fairly easy to recover using regslack. There were indications that Window Washer had been run several times, so I suggested to the customer that we extract the user hive files from the System Restore Points and see if we could find anything of value within them.

Tip

Jolanta's dissertation is available online at the SentinelChicken website at http://sentinelchicken.com/data/JolantaThomassenDISSERTATION.pdf.

As of version 0.51, the Parse::Win32Registry module also has the ability to extract deleted keys and values from within a hive file. One of the scripts that James provided with the distribution of the module, regscan.pl, includes code that references checking whether or not a found entry is allocated (ie, *$entry->is_allocated*). Modifying the code slightly to skip over and not display allocated entries allows us to see just the deleted keys and values. The documentation that James has provided for the module includes a section on lower level methods for processing hive files and refers to the entry object methods that allow for a lower level of access to entries within the hive file. This can allow us to walk through a hive file and locate deleted keys and values.

Tip

I wrote a RegRipper plugin (del.pl) for extracting information about deleted keys and values found within a hive file. There is also a plugin named del_tln.pl that will output the information about deleted keys in the appropriate five-field format for use in timelines.

As we've already discussed earlier in this chapter, there are a number of other tools that will also retrieve deleted Registry keys and values as well as display unallocated space within the hive file. As of this writing, Eric Zimmerman has put a great deal of effort into ensuring that his tool, Registry Explorer, correctly locates and displays information about deleted keys and values.

Memory

I know I said that this chapter focused specifically on "dead box" analysis and that remains true…even though we're now going to discuss accessing Registry information in memory. The fact is that I've conducted analysis of a good number of images acquired from laptops where the hibernation file has proved to be extremely valuable. In one particular instance, I was able to determine that a particular remote access Trojan (RAT) was running on a system when the hibernation file was created, allowing me to see that it was running at one point. This was important, because analysis of the rest of the acquired image indicated that not only had the user installed the RAT from a thumb drive, but that same user had also removed the RAT files and its persistence mechanism as well.

There are a number of Volatility plugins available for parsing memory dumps for Registry data. The command reference for those plugins can be found online at https://code.google.com/p/volatility/wiki/CommandReference#Registry. Using these plugins, you can list available hives, dump individual hives, print the contents of arbitrary keys, or dump a user's UserAssist key entries (the UserAssist key entries will be discussed in more detail in chapter Case Studies: User Hives). Even if there isn't a plugin available to pull specific information that you're interested in from within a memory dump or hibernation file, you can always extract the hive file (or files) you're interested in and use other tools to view the file or parse out specific keys, values, or data. On September 25, 2012, the Volatility team posted an article on their blog that described, in part, a shellbag Volatlity plugin that would parse shellbag

entries from memory, comparing that to the output of the printkey plugin. That blog post can be found online at http://volatility-labs.blogspot.com/2012/09/movp-32-shellbags-in-memory-setregtime.html. This topic is also covered on pages 299 and 300 of the book, *The Art of Memory Forensics* (information about the book can be found online at http://www.memoryanalysis.net/#!amf/cmg5).

Summary

There are a number of very useful tools and techniques available for extracting data from Registry hive files, during both "live" (interacting with a live system) and "forensic" (interacting with hive files extracted from a system or acquired image) analysis. The tools or techniques you employ depend on how you engage and interact with the Registry, as well as the goals of your interaction and analysis. You can opt to use a viewer application, something like RegRipper that extracts and parses specific keys and values based on plugins or like regslack which parses unallocated space within a hive file. In my opinion, tools such as those discussed in this chapter have the advantages of not only being freely available, but the open source tools I've written and provided were written by someone actively involved in a wide range of analysis; I've not only been engaged in data breach investigations (most commonly associated with the theft or exposure of credit card data), but I've analyzed malware outbreaks, intrusions (including those associated with the APT), and I've assisted law enforcement in dealing with potential "Trojan defense" issues. As I mentioned in this chapter, the RegRipper suite of tools (which includes rip.pl and the Plugin Browser) was developed to meet and service my needs and the needs of my analysis. These tools were not developed in a manner that resulted in my having to modify my analysis to meet the needs or limitations of the tools. Ultimately, the goal has always been to provide my customers with timely, accurate results, and the tools discussed in this chapter have helped me deliver on this.

Regardless of which approach is taken, as described in chapter Registry Analysis, your actions and analysis should be thoroughly documented in a clear, concise manner.