1

REGISTRY ANALYSIS

INFORMATION IN THIS CHAPTER

- Core Analysis Concepts
- What is the Window Registry?
- Registry Structure

Introduction

The Windows Registry is a core component of the Windows operating systems, and yet when it comes to digital analysis of Windows systems, is perhaps the least understood component of a Windows system. This may be due to how little information seems to have been written on the subject; however, if you spend just a little time looking around, you'll find that there has actually been quite a bit of information regarding the Windows Registry documented. This apparent disparity may be due to the fact that most of the commercial forensic analysis applications do little more than open the Windows Registry in a viewer-type application and do not provide for the application of previously developed (from the analyst's most recent case, or provided by other analysts) intelligence to the available data. Whatever the reason, my purpose for writing this book is to illustrate the vital importance of the Windows Registry to digital forensic analysis. This is not to say that the Windows Registry is the *only* aspect of the system that requires attention; nothing could be further from the truth. However, the Windows Registry can provide a great deal of valuable information and context to a digital examination, and as such, there is a particular value in addressing this topic in a book such as this one.

The Windows Registry maintains a great deal of configuration information about the system, maintaining settings for various functionality within the system (ie, may be enabled or disabled). In addition, the Registry maintains historical information about user activity; in order to provide the user with a "better" overall experience, details about applications installed and accessed, as well as window positions and sizes, are maintained in a manner similar to a log file. All of this information

Windows Registry Forensics. http://dx.doi.org/10.1016/B978-0-12-803291-6.00001-2

can be extremely valuable to a forensic examiner, particularly when attempting to establish a timeline of system and/or user activity. A wide range of cases would benefit greatly from information derived from the Registry, if the analyst were aware of the information and how to best exploit it for the purposes of their examination.

WHAT'S IN THE REGISTRY?

The first thing to keep in mind when conducting Registry analysis is that not everything can be found there. Believe it or not, one particular question that I still see asked is, "Where are file copies recorded in the Registry?" Windows systems do not record file copy operations, and if such things were recorded, I'd think that some other resource (Windows Event Log, maybe) would be far more suitable.

Not everything is recorded in the Registry, but the Windows Registry is still an incredibly valuable forensic resource.

Information in the Registry can have a much greater effect on an examination than I think most analysts really realize. There are many Registry values that can have a significant impact on how the various components of the system behave; for example, there is a Registry value that tells the operating system to stop updating file last access times, so that whenever a file is opened (albeit nothing changed) for viewing or searching, the time stamp is not updated accordingly. And oh, yeah…this is enabled by default beginning with Windows Vista and is still enabled by default on Windows 7 and Windows 8 systems. Given this, how do examiners then determine when a file was accessed? Well, there are other resources, both within the Registry and without (Jump Lists, for example) that can provide this information, particularly depending upon the type of file accessed and the application used to access the file.

A few examples of Registry values that can impact an examination include (but are not limited to) the following:

- Alter file system tunneling (specifics of file system tunneling can be found online at http://support.microsoft.com/kb/172190) behavior, or the updating of last accessed times on files and folders
- Have files that the user deletes automatically bypass the Recycle Bin
- Modify system crash dump, Prefetcher, and System Restore Point behavior
- Clear the pagefile when the system is shut down
- Enable or disable Event Log auditing
- Enable or disable the Windows firewall

- Redirect the Internet Explorer web browser to a particular start page or proxy
- Automatically launch applications with little to no input from the user beyond booting the system and logging in

All of these Registry settings can significantly impact the direction of an investigation. In a number of instances, I have found valuable data in the pagefile (such as responses from web server queries) that would not have been there had the pagefile been cleared on shut down. When examining a Windows system that was part of a legal hold (an order was given to not delete any data), it can be very important to determine if the user may have cleared the Recycle Bin, or if the system was set to have deleted files automatically bypass the Recycle Bin. The use of application prefetching, which is enabled by default on workstation versions of Windows (but not server versions, such as Windows 2008 R2), can provide valuable clues during intrusion and malware discovery cases.

These are just a few examples; there are a number of other Registry keys and values that can have a significant impact (possibly even detrimental) on what an analyst sees during disk and file system analysis. Some of these values do not actually exist within the Registry by default and have to be added (usually in accordance with a Microsoft (MS) Knowledge Base (KB) article) in order to affect the system. At the very least, understanding these settings and how they affect the overall system can add context to what the analyst observes in other areas of their examination.

REGISTRY VALUES AND SYSTEM BEHAVIOR

The Windows Registry contains a number of values that significantly impact system behavior. For example, an analyst may receive an image for analysis and determine that the Prefetch directory contains no Prefetch (*.pf) files. Registry values of interest in such a case would include those that identify the operating system and version; by default, Windows XP, Vista, and Windows 7 will perform application prefetching (and generate *.pf files); however, Windows 2003 does not perform application prefetching (although it can be configured to do so) by default. The Prefetcher itself can also be disabled, per MS KB article 307498 (found online at http://support.microsoft.com/kb/307498). This same value can be used to enable or disable application prefetching.

The purposes of this book are to draw back the veil of mystery that has been laid over the Registry, and to illustrate just how valuable a forensic resource, the Registry, can really be during malware, intrusion, or data breach examinations, to name just a few. The Windows Registry contains a great deal of information that can provide significant context to a wide range of investigations. Not

only that, but there are also a number of keys and values, as we'll discuss later in this book, in which information persists beyond that deletion or removal of applications and files. That's right…if a user accesses a file or installs and runs an application, the indications of these actions (and others) will remain long after the file or application has been removed and is no longer available. This is due to the fact that much of the "tracking" that occurs on Windows systems is a function of the operating system, of the environment, or ecosystem in which the application or user functions. As such, much of this activity occurs without the express knowledge of the user or application…it just happens. Understanding this, as well as understanding its limitations, can open up new vistas (no pun intended) of data to an analyst.

Core Analysis Concepts

Before we begin discussing the Windows Registry analysis specifically, there are several core analysis concepts that need to be addressed as they are pertinent to examinations as a whole. Keeping these concepts in mind can be extremely beneficial when performing digital analysis in general.

Locard's Exchange Principle

Dr. Edmund Locard was a French scientist who formulated the basic forensic principle that *every contact leaves a trace*. This means that in the physical world, when two objects come into contact, some material is transferred from one to the other, and vice versa. We can see this demonstrated all around us, every day…let's say you get a little too close to a concrete stanchion while trying to parallel park your car. As the car scrapes along the stanchion, paint from the car body is left on the stanchion and concrete, and paint from the stanchion becomes embedded in the scrapes on the car.

Interestingly enough, the same holds true in the digital world. When malware infects a system, there is usually some means by which it arrives on the system, such as a browser "drive-by" infection, via a network share, USB thumb drive, or an e-mail attachment. When an intruder accesses a system, there is some artifact such as a network connection or activity on the target system and the target system will contain some information about the system from which the intruder originated. Some of this information may be extremely volatile, meaning that it only remains visible to the operating system (and hence, an analyst) for a short period of time. However, remnants of that artifact may persist for a considerable amount of time.

EVERYTHING LEAVES A TRACE

Any interaction with a Windows system, particularly through the Windows Explorer graphical interface, will leave a trace. These indications are not always in the Registry, and they may not persist for very long, but there will be something, somewhere. It's simply a matter of knowing what to look for and where, and having the right tools to gain access to and correctly interpret the information.

The quote, "absence of evidence is not evidence of absence" is attributed to the astrophysicist Dr. Carl Sagan and can be applied to digital forensics as well. Essentially, if an analyst understands the nature of a user's interaction with a Windows system, then the lack or absence of an artifact where one is expected to be is itself an artifact. During a recent examination, I was trying to determine a user's access to files on the system and could not find the RecentDocs (this key will be discussed in greater detail in chapter Case Studies: User Hives) key within the user's NTUSER.DAT hive file; RegRipper did not find it, and I could not locate the key manually. As it turns out, the user had run the "Window Washer" application, which reportedly "clears the list of recently accessed documents." The time associated with the user launching the application (derived from the user's UserAssist key) corresponded to the LastWrite time on the RecentDocs parent key.

While examining a system that was part of a larger incident, our team had determined that there was a malware file on the system (a dynamic link library, or DLL) but could not determine the method used to load and launch the malware. A timeline consisting of file system and Event Log events clearly showed the user logging in, the process being launched, the DLL file being accessed, and then the known file system artifacts being created. Our first thought was that there was some autostart location or trigger within the user's NTUSER.DAT hive file, but we could not find anything. It turned out that the DLL in question was loaded as a result of some Windows shell extensions not having explicit paths listed in the Registry, and the operating system following its designated search order to locate a DLL by that name.

In both instances, the absence or lack of an expected artifact was itself an artifact, and spurred additional, in-depth analysis.

So how does this apply to Registry analysis? When a user, even an intruder who has gained access to the system, interacts with the system, and particularly with the Windows Explorer user interface (aka the "shell"), some rather persistent artifacts are created. If a malicious user logs into the system and plugs in a USB thumb drive, there is an exchange of information that occurs, and some of those artifacts persist in the Registry (the same is true when connecting to devices via Bluetooth). If the malicious user then launches applications, there will be additional artifacts created. When a user connects their system to a wireless access point (WAP), information about the WAP persists on the system. I was told by another analyst several years ago that he'd been involved

in an investigation where a former employee was thought to have provided sensitive data to a competitor and then accepted employment with the competitor. An examination of the system revealed that while the employee was traveling, he'd connected his corporate laptop to WiFi at a Starbucks near the competitor's building and then to the WiFi in the competitor's offices about an hour later. A combination of the WiFi access point names and a geolocation lookup of the medium access control (MAC) addresses of the access points, along with date and time stamps, revealed important clues about the employee's activities.

Analysts need to keep Locard's exchange principle in mind during an examination because it can not only tell them that there *are* artifacts, but can also point them to where those artifacts may be located. Knowing likely sources of artifacts significantly reduces the time it takes to conduct an investigation; instead of combing through all of the data, the examiner can target those specific areas to quickly determine whether they will be fruitful or not. This sort of data collection and triage can greatly assist incident response activities as well as increase the throughput of investigations, as a whole, without sacrificing quality or completeness.

Least Frequency of Occurrence

I first heard the term "least frequency of occurrence" mentioned in the context of digital forensics at the SANS Forensic Summit during the summer of 2009. Peter Silberman (an analyst with the consulting firm Mandiant) used the term to describe malware infections on systems. His point was that in the old days, malware (and in particular worms) would spread rapidly, infecting and reinfecting systems. In short order, a system would be so heavily infected that it would become completely unusable by anyone, let alone the attacker. The result was that not only were infected systems unusable to the attacker, but the failing systems provided a clear indication to the "victim" organization that they were infected. In order to address this, malware authors began using a unique "mutex," a software programming object that allows for mutual exclusion, within their malware in order to prevent the system from becoming reinfected. Once the system was infected, the mutex would be present in memory; upon reinfection, the malware would check for the mutex and, if found, not proceed with the infection.

The offshoot of this is that the mutex is very often random (although sometimes not so random) and always unique. This became an excellent indicator of a malware infection; in fact, Kris Harms (at the time an analyst with Mandiant, who later became

an analyst with Cylance) discussed during a presentation the use of the Microsoft SysInternals tool *handle.exe* to list all the mutexes available in memory for all of the running processes on the system and then sorting the output by the unique mutexes. Kris demonstrated that a quick look at those mutexes that only occurred once or infrequently across processes very often resulted in rapid and accurate detection of malware, even if the mutex name itself had been changed.

Demonstrating Kris' use of handle.exe is outside the scope of this book, but it does serve as an example of how the concept of *least frequency of occurrence* (LFO) can be used, not only for malware but also for intrusions, and therefore can also be very important to our analysis.

The point of LFO is that during the lifetime of a system, malware infections and intrusions constitute what occurs least frequently on that system. Operating system and application updates are extremely "noisy," generating a great deal of file system (file creations, modifications, and deletions) and Registry (keys being created, values updated, etc.) activity, and occurring fairly frequently. Windows XP, by default, will create a System Restore Point every 24 h (as well as under other conditions) and will also launch its Disk Defragmenter utility every three calendar days to perform a limited defrag. Windows XP also generates or updates Prefetch files whenever an application is launched. Beginning with Windows Vista, the operating systems began maintaining Volume Shadow Copies (as opposed to the Windows XP System Restore Points) in order to provide a recovery mechanism. When a user installs software from Apple (such as QuickTime, iTunes, etc.), a Scheduled Task is *created* on the system to look for updates to those applications once a week, and the user can choose to install those updates, creating and modifying files within the file system. Microsoft releases operating system and application updates monthly, and sometimes does so "out of band," or out of the regular update release schedule. What this means is that there is a *lot* of normal file system and Registry activity that occurs on a system, but in contrast, when malware infects a system, a few files (and maybe Registry keys/values) are created, and there may also be some network connections as the malware communicates off of the system. When an intruder accesses a system via Remote Desktop due to an easily guessed password, there may be several Event Log records generated (we will discuss how to determine the audit configuration on a system in detail in chapter Analyzing the System Hives), some Registry keys created or modified, and depending upon the actions they take, maybe some files created, modified, or deleted on that system. Again, with the exception

of turning the compromised system into a repository for pirated movies or music files, a malware infection or intrusion will very often constitute the least frequent activity on the system. In fact, many intrusions go undetected for long periods of time, as the intruder will use very simple techniques to minimize as much as possible the artifacts left on a system. This can also be true for other types of issues, such as viewing illegal images. A file (or a few files) is added to the system, the files are viewed (as we'll see in chapter Case Studies: User Hives, some Registry keys will be updated), and then the files may be shared or deleted. All in all, adding, viewing, and deleting these files really do not constitute a considerable amount of activity, particularly when compared to operating system and application updates.

What this often means to our analysis is that during intrusions or malware infections, we wouldn't usually be looking for large numbers of files being added to the system, or of massive numbers of Registry keys or values being created, or regular or significant spikes in activity of any kind. Most often, spikes in file system and Registry activity will indicate an operating system or application software update (or much to the chagrin of the analyst, a system administrator running antivirus application scans) not a malware infection or system intrusion.

Windows Isn't Just "Windows"

What I mean by this is that the version of Windows being examined matters; in fact, it matters a great deal. From a user's perspective, the change from Windows 2000 to Windows XP was pretty significant, because what they saw on the screen and the way in which they interacted with Windows, through the user interface, changed significantly. The same can be said for the change from Windows 7 to Windows 8 and 8.1. From a forensic analyst's perspective, the changes have been even more significant; with each new release of Windows, it seems that more artifacts are available, with many of these artifacts being maintained in different formats and locations, not just within the Registry but within the file system as a whole.

Consider user searches; with Windows XP, the terms that a user searched for via the shell were maintained in subkeys beneath the "ACMru" Registry key in the user's NTUSER.DAT hive file. With Windows Vista, the search terms were saved to an XML file, and then with Windows 7, the terms were saved beneath a key named "WordWheelQuery." With Windows 8, user search terms were saved in the Registry and associated with "Search Charm" in the user interface, and apparently, with Windows 8.1, those terms are again saved in a file.

There are other significant changes between versions of Windows. With Windows XP and 2003, system events were recorded in the Event Logs, files with the ".evt" extension in the *Windows\system32\config* folder. As of Windows Vista, the logs were referred to as Windows Event Logs and had a ".evtx" extension, a new location, and an entirely different binary format, requiring a completely different set of tools to parse them. Oh, and even more events were (and still are) being recorded by default.

There are many more such differences between the versions of Windows, and we'll go into many of those differences in this book as they apply directly to the Registry for each version.

Remnants

One of the things I really like about digging into the Registry is the amount of information that is available, often times even after a user or intruder has taken "antiforensics" steps in order to hide their activities. Often users and intruders will take steps to cover their tracks and remove indications of their activities without realizing that their interactions with the operating system (and often times, with applications) are being "recorded" automatically.

For example, I was examining a system about a year ago which had been found to be infected with a particular variant of a remote access Trojan (RAT). This particular RAT variant is usually installed as a Windows service, allowing the intruder to access the system with privileges greater than that of the system administrator. Further, this particular bit of malware is most often assumed to be installed via a "spearphishing" e-mail, in which the user is enticed to click on a link or malicious document, resulting in the installation of the RAT. In this particular case, Registry artifacts revealed that the RAT had been installed as a result of someone with physical access to the system plugging a USB thumb drive into the system (it was mounted as the E:\ volume) and launching an installer application. When the user's employer requested that they turn in the system for examination, the user attempted to remove the RAT…in fact, artifacts in the Registry revealed that the last key in focus in the Registry Editor before it was closed by the user was the key alphabetically following the name with which the RAT was installed. Additional information was extracted from the hibernation file through the use of the Volatility Framework, but the preponderance of artifacts extracted from the Registry clearly indicated that the RAT was installed and running on the system with the full knowledge (and involvement) of the user.

As I'm writing this section of the book, I'm working (as part of my day job) on an examination in which an intruder had access to an infrastructure via domain administrator credentials and

Terminal Services. I'd tried to backtrack the intruder's connections as they hopped from system to system, but they had a penchant for clearing the Windows Event Logs on some (albeit not all) systems. Even though I could not see all of the data that I wanted to, there was more than enough data in the Registry to tell me when they were logged into the system and active (days and times), as well as other systems to which they connected. Even without the Windows Event Log records, I was able to build a time-based map of the intruder's activities, showing what time of day they were active, files and resources they'd accessed, etc. I was also able to illustrate their connection to a file transfer protocol server.

Data available in the Registry can be very revealing on a number of fronts. Although the intruder had cleared the Windows Event Log and we were not able to see where they were when they logged into the system we were examining, but we could see when they were active on the system (which we could then correlate with other sources, such as domain controller and VPN logs), what they'd done, and other systems they'd connected to through the use of the Terminal Services Client as well as via Microsoft networking. All of this data remained even following the intruder's "antiforensics" steps.

Goals

Before starting any analysis at all, every analyst should carefully consider and document their goals. What are you looking for? What questions are you trying to answer? What do you hope to ultimately achieve through your analysis? We do this because this helps us understand what it is we should be doing, what data we should extract, where we should go to look forartifacts and clues, and what data can be correlated to address the issue. Too often, analysts get caught up in the "find all bad stuff" mindset (or allow customers to hem them into it) and in doing so spend hours upon hours "doing analysis," yet never actually answer the questions before them. Believe me, I understand how you'll be looking for one thing but find something else that, while interesting, may not have anything to do with your immediate analysis. Pursuing these kinds of things is called having "shiny object syndrome"; like a fish or a kitten, you're easily distracted by shiny objects. An example of this is locating all of the malware and spyware on a system, when the customer just wanted to know if a user on the system had accessed or copied a file (as in a fraud or exposure of intellectual property issue).

Your goals may vary depending upon your employer and the type of work you generally do. If you're a consultant, your goals may vary from case to case; during one examination, you may

have to determine if a system was infected with malware, and if so, the capabilities of that malware (ie, what data did it extract, where was the data sent, was the malware specifically targeted at the organization, etc.). In another examination, you may have to determine if there was sensitive information (ie, personally identifiable information, credit card data, classified data, etc.) stored on the system, while another examination may pertain to violations of corporate acceptable use policies. If you're a law enforcement officer, you may be faced with a possible issue of fraud, or you may need to demonstrate that a computer owner had knowledge of and viewed contraband images.

Regardless of the type of examination, your goals are where everything starts and ends. For consultants, not answering a customer's questions can lead to serious issues, such as spending far more time on your "analysis" than your contract allows, or attempting to bill a customer when you haven't answered their questions. Our analysis goals give us direction and focus and allow us to provide those answers in a timely and efficient manner.

Documentation

Perhaps the most important aspect of any analysis, after the goals, is documentation. Forensic analysts and incident responders should document all aspects of what they do, from the acquisition of hard drives and the transfer and management of acquired images, to their analysis plan and actual case notes. Many organizations have their own acquisition methodology and chain of custody documentation, usually in some sort of form or checklist. This is a good start, but documenting case work should not stop there.

What can sometimes be missed is documentation of the overall analysis process. Before conducting analysis, do you sit down and ensure that you understand the goals of the analysis, or the questions that you're trying to answer? Whether you're a consultant working for a customer or an examiner performing work in support of law enforcement, there's usually some reason why you're sitting there with a hard drive or an acquired image. What is that reason? Most likely, it's that someone has questions that need to be answered. So start your analysis plan by documenting the goals that you're trying to achieve. From there, you can begin framing out your steps going forward and noting where you need to look, and those tasks that you need to achieve. For example, if the goal is to determine the existence of specific e-mails, you'll likely want to check for .pst/.ost files, or maybe check the Registry and determine which e-mail client was used, determine if web-based e-mail was used, etc.

Note

Many times when beginning an examination involving the use of a web browser on a Windows system, I'll see analysts start off by saying, "I'd check the contents of the user's TypedURLs key." That key, located in the NTUSER.DAT file within the user profile, contains a list of the URLs typed into the Internet Explorer address bar. But is that really a good place to start? What if there are no entries? What does that tell you? Perhaps a better place to start would be determine which web browser the user was using, or at least which web browsers were installed on the system, before targeting browser-specific artifacts.

The analysis plan can lead the analyst directly into documenting the analysis process itself. So why would we do this? What happens if at some point during the analysis process, you get sick or become injured? What happens if the analysis needs to be handed off to someone else? Another very real possibility is what happens if 6 months or a year after you complete your analysis, you have to answer questions about it? I know several analysts to whom this has happened recently. For myself, I've worked with customers who've come back with questions six or more months after accepting the final report and paying their bill…had I not had clear, concise documentation, I would have had trouble answering their questions in an intelligible manner. We've all been busy to the point where we can't remember what we had for breakfast, let alone the specifics of an examination from 6 months ago. Your case notes and documentation can be extremely important at that point, and it's best not to have to figure that out after the fact.

Another important aspect of documenting your analysis is that it allows you to go back and look at what you did, and improve the process. Documentation is the basis for improvement, and you can't improve a process if you don't have one. Your documentation provides that process. If you didn't document what you did, it didn't happen. By listing out the steps you followed in your analysis, you can see which ones were perhaps less fruitful and can be skipped or improved upon the next time, and which ones provided greater value. This also allows for other, less experienced analysts to learn from what you have done, what worked and what didn't, so that more analysts are able to achieve a similar, greater level of analysis.

Challenges of Registry Analysis

While often fruitful, Registry analysis isn't always easy, and there are two primary challenges when it comes to Registry analysis. Depending on your particular experiences, there may be other challenges, but these are the two big ones as I see them.

The first challenge to Registry analysis is that the Registry itself isn't all that well understood by responders and analysts. To be honest, I'm not even sure that there's really *anyone* who completely understands the Windows Registry! The Registry is a critical, core component of the Windows operating systems and records a considerable amount of information about the system configuration and usage, as well as user activity, particularly when the user is interacting with the system through the Windows Explorer shell. With just the operating system itself, I don't think that there's really anyone who completely understands why some keys and values have the paths and contain the structures that they do, or what activities lead to the keys or values being created or modified, let alone the structure of various binary value data. This lack of understanding by the vendor obviates any thorough knowledge and understanding by analysts and leaves the analyst to perform considerable testing to determine and illustrate how various artifacts originated on the system.

While considerable work has been performed and documented in this area, the awareness that this work is possibly incomplete persists. As new versions of the operating system are developed, locations and formats for storing data in the Registry change, as well, and some keys or values may be added, moved, modified, or simply removed. Very little is known and documented about what actions cause various keys to be modified; while some testing has been done for a very small number of keys, new questions are being posed all the time that would, quite honestly, require access to the source code to the operating system in order to completely answer. Being closed source the way Windows is, having complete access to the source code isn't likely to happen anytime soon.

Several years ago, Cory Altheide (whom I used to work with, and is now a responder for Google) and I conducted some research into tracking the use of USB devices across Windows systems. After we were done, we published our findings, confident that we'd figured out a way to determine when a USB device was last connected to a system. More recently, Rob Lee (of SANS fame) conducted additional testing and determined that what Cory and I had determined was really the first time that the device had been connected during the current (or most recent) boot session, meaning that if the system was running for several days and the USB device connected and disconnected several times, the best we could hope to show (with just the data we'd found) was when the device had first been connected during that boot session. Additional information is available in Windows Vista and Windows 7, but there simply is no comprehensive listing of actions, by a user or within the operating system, that would affect particular Registry keys.

MALWARE AND THE WINDOWS REGISTRY

Most of the time when looking for indications of malware remaining persistent on a system, I'll go right to the Registry. Not only is this a popular location for malware to use to maintain persistence, but very often new persistence locations in the Registry are discovered by analyzing a new bit of malware that's been found. The reason is that many malware authors will become aware of these locations and how to use them well before anyone else, including antivirus vendors and malware analysts.

Analyzing the Registry for new bits of malware can often be a game of catch-up, as some new means of persistence may have been discovered by the bad guys and not yet commonly known by responders and incident analysts.

To make matters worse, not only do malware authors make extensive use of the Registry so that their creations will remain persistent on systems across reboots and logins, but some have even gone so far as to place entire Windows executable files into binary value data!

The other challenge of Registry analysis is the fact that while the binary structure of the Registry remains the same across versions of Windows (ie, the core binary structure of the Registry is very much the same between Windows 2000 and Windows 7, inclusive), important keys and values change between versions, often very drastically. In many cases, this applies to the base operating system as well as to new and even existing applications. This can make it very difficult for an analyst who figures out and documents some specific Registry keys and values based on a particular version of an application and operating system, only to find those settings null and void when an updated version of the application or the operating system is released.

One example of these changes is how user search terms are maintained within the Registry. With Windows XP, you could find various search terms under a key named "ACMru." Subkeys beneath this key pertained to particular form fields that a user could submit terms to when performing searches. With Windows Vista, search terms were recorded in a file, but not in the Registry. With Windows 7, search terms are again stored in the Registry, but under an entirely different path, beneath a key named "WordWheelQuery." These keys are discussed in greater detail in chapter Case Studies: User Hives.

It is not the goal of this chapter or even this book to provide a comprehensive listing of all similar changes that occur between various versions of the Windows operating system; rather, it is enough to understand that these changes can and do occur, and it is incumbent upon analysts to keep up-to-date on analysis techniques and procedures, particularly as they pertain to the Windows Registry.

Tip

Something that is very important to keep in mind when considering whether to engage in live response activities (as opposed to acquiring an image of the hard drive and conducting postmortem analysis) is that while your actions do have an effect on the system (processes loaded into memory, files created on the system as a result of your actions, etc.), so does your *inaction*. Think about it. A live system is running, with things going on all the time. Even while a system just sits there, processes are running and actions are occurring on the system. With Windows XP, simply wait 24 h and a System Restore Point will (by default) be automatically created. Wait 3 days and the system will conduct a limited defragmentation. Windows can reach out for and install updates. All of these can overwrite data that could possibly be recovered. Also consider the fact that if someone is exfiltrating data from your systems, then while you wait and do nothing, they continue to take more data. So the question of live response really comes down to (1) do I do nothing or (2) do I take the correct actions to protect my organization as best I can under the circumstances?

What Is the Windows Registry?

So far we've talked about Registry analysis, but what is the Windows Registry? According to Microsoft KB article 256986 (found online at http://support.microsoft.com/kb/256986) the Windows Registry is a "central hierarchal database," intended "to store information that is necessary to configure the system for one or more users, applications, and hardware devices." In short, the Windows Registry is a binary data structure meant to replace the configuration and initialization (*.ini*) files used by previous versions of Windows (okay, Windows 3.1). For your normal Windows user and for most administrators, this is pretty transparent and means very little to them. Most users and administrators do not interact directly with the Registry, instead interacting with it through some sort of graphical user interface (GUI), such as the Registry Editor that is distributed with most Windows installations. Fig. 1.1 illustrates the Registry Editor on Windows XP.

As you can see in Fig. 1.1, the Registry Editor provides a user or administrator with an easy means to navigate the Registry by providing a layer of abstraction. There may be times when even an administrator doesn't go as far as using the Registry Editor, as most interaction with the Registry may be through application installation (ie, launching the installation process, which then adds and modifies Registry entries) or removal.

Many of the instructions and KB articles available from Microsoft that deal with interacting with the Registry do so by having the reader interact with a GUI component of the Windows Explorer shell, or through another application. For example, a user wouldn't

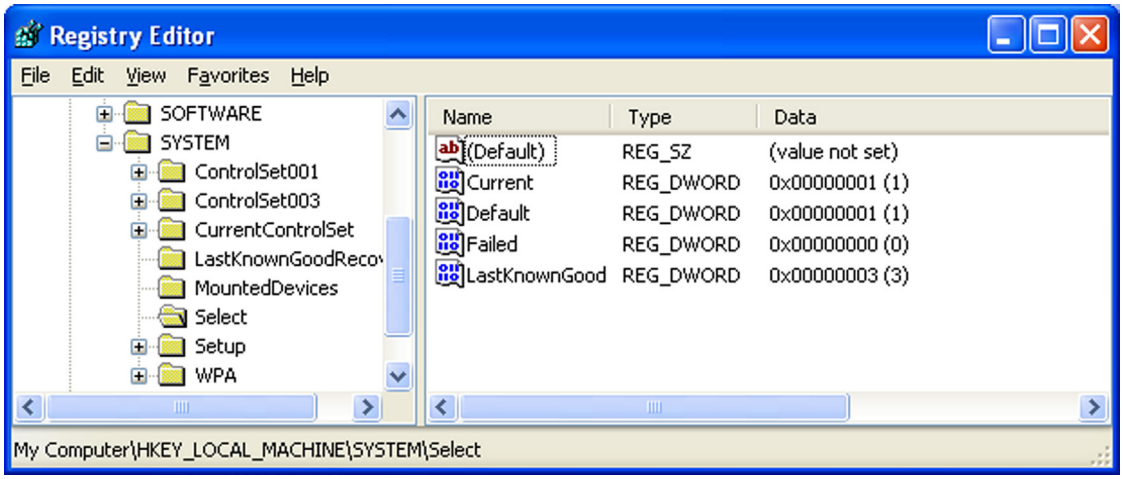

Figure 1.1 Registry Editor.

directly access the Registry to delete keys and values created when an application is installed; instead, they would likely use the Add/Remove Programs Control Panel applet. In those instances where Microsoft does identify specific Registry keys, there is always a stern warning against directly modifying the Registry, as to do so might leave the system inoperable.

Note

Graphical tools are primarily intended to make a task easier for the user but also protect the users from themselves. The GUI prevents the user from seeing what happens "under the hood." However, that's exactly where malware authors and attackers go…under the hood. The best source of information regarding autostart locations in the Registry is the antivirus vendors; as they receive new malware samples to analyze, they begin to see what methods and autostart locations (and persistence mechanisms) these folks are using. Neither Microsoft nor application vendors provide such a breadth of information. Further, relying on antivirus vendors to let us know what they're seeing is reactive, not proactive.

Purpose of the Windows Registry

Microsoft tells us that the Registry maintains configuration information about the system, but what does this really mean? It's one thing to say that the Registry replaces the text-based .ini files of old and is a database that maintains configuration information about the system and applications that run on it, but what does that really mean to the incident responder and forensic analyst? We're not so much interested in what this means to a user or to an administrator;

instead, what we'd like to know is, what does that mean to those of us who would need to delve into this resource? Well, what it means is that there's a lot of information in the Registry that tells the operating system and applications what to do, where to put things, and how to react to certain stimulus. There are a lot of little nuances that can have a significant effect on incident response and forensic analysis that are all managed through the Registry. For example, one Registry value tells the operating system to clear the pagefile when the system is shut down, and another setting tells the operating system whether or not to enable the use of a hibernation file, while yet another value disables the updating of last access times within the file system. When you think about it, all of these values can have a significant impact on a wide range of incident response activities and digital forensic analysis.

Devices that have been connected to the system are tracked through the Registry. Information about devices is maintained in the Registry so that the devices are recognized and presented as they were previously when they're reconnected to the system; as such, this information can be extremely valuable to a forensic analyst when attempting to track the use of an iPod, digital camera, or thumb drive on a system or across several systems.

The Registry also tracks a great deal of information about a user's activities. This can be very beneficial to a forensic analyst. Let's say you sit down to play a game of Solitaire on your Windows system, and the first time you run the application, you get the default settings, with respect to how many cards are dealt and how the game is timed and scored. You change most of these settings to something else and then resize and reposition the game window. When you're done playing, you close the window and shut down the system. The next day, you come back and launch the game again, and all of your settings are still there, having persisted across a log out and reboot. This is due to the fact that the settings are recorded in the Registry, so that the next time you launch the application or game, your most recent and preferred settings are read, and the application window is presented in the location, size, and shape that you left it.

Warning

Not all applications create a presence in the Registry. For example, some peer-to-peer (P2P) sharing applications are cross-platform and Java-based, and as such don't rely on the Windows Registry to store information. Instead, they use configuration files in order to make cross-platform coding easier.

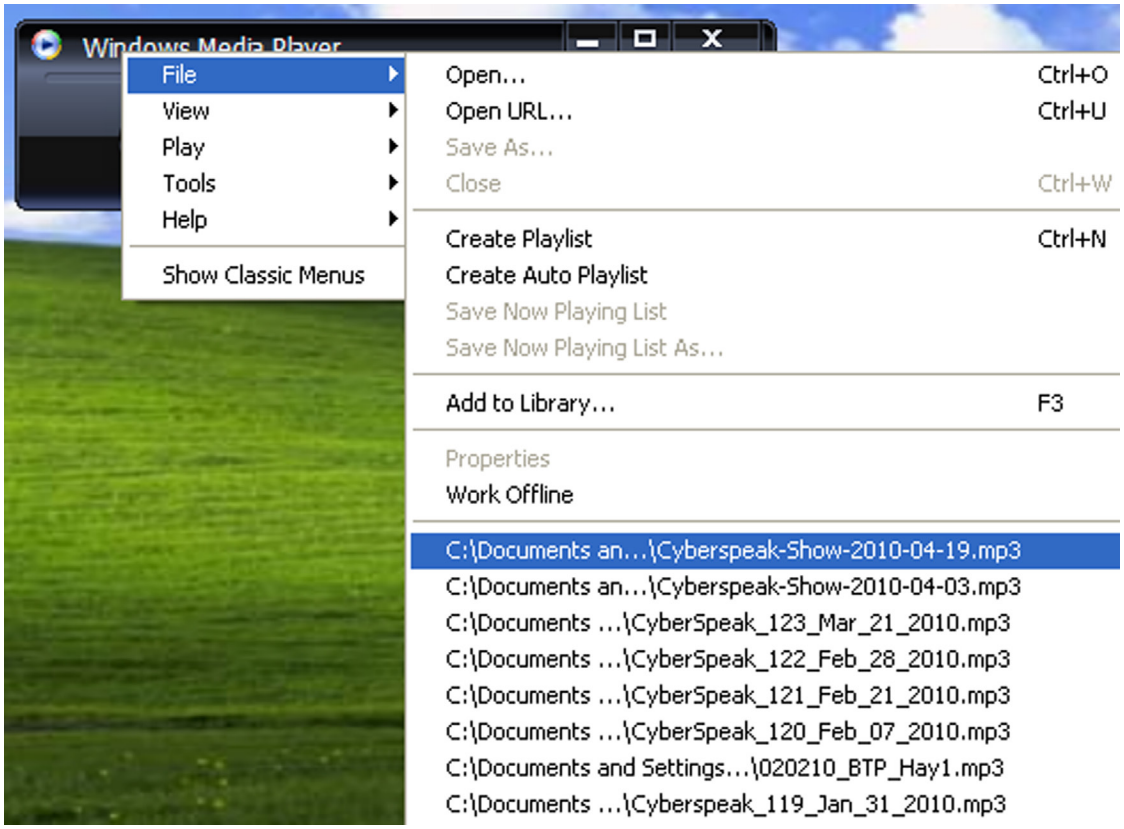

Figure 1.2 Windows Media Player File menu item showing recently accessed files.

The Registry also tracks a number of other user actions, such as clicking through the Program menu to start an application, as well as keeping track of recently accessed files that are associated with various applications, such as MS Word, Excel, Windows Media Player, etc. The user will generally see these files on the Recent Documents portion of the Program menu, or as part of a drop-down menu specific to the application, as illustrated in Fig. 1.2.

Much of the information tracked in the Registry can be associated with a time value of some sort, and as such, the Registry becomes something of a log file. As will be addressed later in this chapter, all Registry keys maintain a property called their "Last-Write time." Whenever a Registry key is modified…created, values or subkeys are created or deleted, or a value is modified…the key's LastWrite time is updated to reflect that change. This value is analogous to a file's last modification time (although, as of yet, I have been unable to locate an accessible application programming interface, or "API" that allows for the arbitrary modification

of LastWrite times, as there is with file MAC times). However, this is not the only place that time stamps are maintained in the Registry. Many values contain time and date information, and often in different formats. In this way, the Registry can be considered in many respects to be a log file.

Tip

While analyzing a system to determine if a user had looked at images or videos (as opposed to a virus or worm putting those files on the system), I ran across the use of the Window Washer application, which is intended to "clean up" behind a user. In this case, the application maintained the last date and time that it had been run in its own Registry values, which I was able to correlate to other, similar data. There were two separate values, one for date and one for time, maintained as strings.

Location of the Windows Registry on Disk

From a forensic analysis perspective, an analyst does not generally interact with the Registry through the Registry Editor. An analyst will most likely interact with Registry hives files directly, through a commercial forensic analysis application, or as a result of extracting them from a file system or from an acquired image. There are a number of such tools available, several of which will be discussed in chapter Processes and Tools. However, it is important for the analyst to know where these files exist on disk so that they can be retrieved and analyzed.

The main, core system Registry hive files (specifically, SAM, Security, Software, and System) can be found in the *Windows\system32\config* folder, as illustrated in Fig. 1.3.

The files themselves illustrated in Fig. 1.3 are referred to as "hive" files, as the files contain the binary database structures or "hives." These are the hive files that maintain configuration information about the system, such as operating system version and

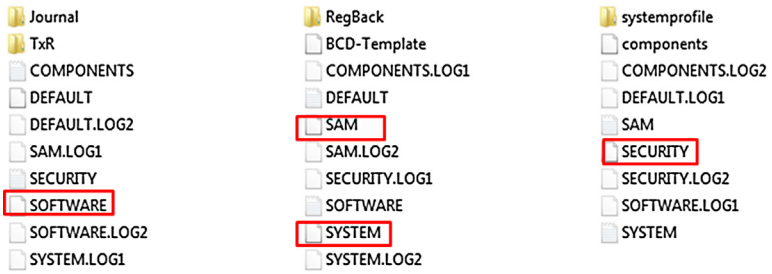

Figure 1.3 Registry hive files in the *Windows\system32\config* folder (Windows 7).

settings, local user account information, installed software and components, etc.

On Windows Vista and above systems, there is another hive file in the *Windows\system32\config* folder named "Components." While there are a number of keys and values listed in this hive file, as of this writing, I have yet to find anything significant from a forensics or incident response standpoint; however, this may change in the future as more attention is focused on this file.

With Windows 8, we found that there were several new Registry hives files in the *Windows\system32\config* folder: BBI, BCD, Components, Drivers, and ELAM. Not a great deal of information is available on the purpose of these files, but they can be opened in a Registry viewer application. However, a search on the Microsoft website reveals that "ELAM" stands for "early launch anti-malware," which is intended to protect Windows 8 systems from rootkits. From a reference to ELAM at the Microsoft Dev Center (found online at http://msdn.microsoft.com/en-us/library/windows/desktop/hh848061(v=vs.85).aspx):

> *Windows 8 introduces a new feature called Secure Boot, which protects the Windows boot configuration and components, and loads an Early Launch Anti-malware (ELAM) driver. This driver starts before other boot-start drivers and enables the evaluation of those drivers and helps the Windows kernel decide whether they should be initialized.*

My testing system was conducted on a Windows 8.1 laptop, with just MS Office 2013 installed, and little else. As such, I did not have an "ELAM driver" installed.

AMCACHE.HVE

When I first began working with a Windows 8 (and later Windows 8.1) system, one of the items about this system was "new," was the existence of a file named "AmCache.hve" in the *C:\Windows\AppCompat\Programs* folder. I initially opened this file in a hex editor and almost immediately noticed that it appeared to have a structure similar to a Registry hive file. I then opened the file in a Registry viewer application (several viewers are discussed in chapter Processes and Tools) and saw that the viewer application did, indeed, parse and display the contents of the file. After I created a Windows 10 Technical Preview virtual machine (VM) in VirtualBox, I saw that the Windows 10 system also had this file. Then, in January 2015, I noticed that at some point near the end of 2014, update installed on my Windows 7 systems (host systems, as well as testing VMs) had resulted in the AmCache.hve file being created on those systems, as well. This file is not part of what we refer to as the Windows Registry, but it does seem to contain data pertinent to forensic analysts and follows the same structure as the Registry hive files. As such, this file will be discussed in greater detail in chapter Analyzing the System Hives.

Information specific to individual users is maintained in the NTUSER.DAT hive file that is located in the user profile. For Windows 2000, XP, and 2003, the user profiles are found in the "Documents and Settings" directory at the root of the system drive, whereas for Vista and above, the user profiles are found in the "Users" directory. There is also another user hive that is merged with the NTUSER.DAT hive file on a live system when a user logs in, allowing for a unified presentation of the information from both hives. This is the USRCLASS.DAT hive, located in the user's profile, in the *Local Settings\Application Data\Microsoft\Windows* folder on Windows XP and 2003, and in the *AppData\Local\Microsoft\Windows* folder on Windows 7 and above. The information maintained in this hive file can vary between operating system versions; some information found in the NTUSER.DAT hive on Windows XP has been moved to the USRCLASS.DAT hive on Windows Vista and above. Many of these differences will be addressed later in this book.

REGISTRY REDIRECTION AND VIRTUALIZATION

With more modern versions of Windows, Microsoft has implemented redirection and virtualization with respect to the Registry. Registry redirection (description found at the Microsoft website online at http://msdn.microsoft.com/en-us/library/aa384232(VA.85).aspx) essentially means that on 64-bit versions of Windows, some Registry calls by 32-bit applications are redirected to another portion of the Software hive. What this means to an analyst is that some 32-bit application information (ie, those keys that are not identified as being shared between 64- and 32-bit applications) will appear in the HKEY_LOCAL_MACHINE\Software\Wow6432Node key path, rather than in the HKEY_LOCAL_MACHINE\Software key path. Similar redirection does not occur within the Software key in the user's hive. The Microsoft Technet site provides a list of shared keys, found online at https://msdn.microsoft.com/en-us/library/windows/desktop/aa384253(v=vs.85).aspx. Note that Registry reflection for synchronization has been disabled as of Windows 2008 and Windows 7.

Registry virtualization is a bit different, and impacts an examiner's analysis differently. Microsoft describes Registry virtualization (description found online at http://msdn.microsoft.com/en-us/library/aa965884(VS.85).aspx) as, beginning with Windows Vista, "an application compatibility technology that enables registry write operations that have global impact to be redirected to per-user locations." What this means is that Registry modifications (writes, anything to create keys or values) that have a global impact on the system will be written instead to a "virtual store" (*HKEY_USERS\< SID>_Classes\VirtualStore\Machine\Software* key path), which translates to the USRCLASS.DAT hive file mentioned above.

Portions of the Windows Registry visible through the Registry Editor are "volatile," meaning that they are populated when the system is booted or when a user logs in and do not exist on disk

when the system is shut off. This is extremely important for first responders and forensic analysts to understand, as there may be valuable data that does not exist within an acquired image and *must* be collected while the system is still running.

One example of volatile data is the *HKEY_CURRENT_USER* hive. When viewed through the Registry Editor, you can clearly see this hive, and after a little exploration, you'll find that the information in this portion of the Registry pertains specifically to the logged on user. However, when you shut the system down and analyze an acquired image, you won't find an *HKEY_CURRENT_USER* hive or any file by that name. That's because this hive is populated by using the hive for the user that's logged into the system.

For the currently logged in user, the *HKEY_CURRENT_USER\SessionInformation* key contains a value named ProgramCount that keeps track of the number of programs you have running on your desktop. This is the count you see when you lock your workstation. However, this value doesn't exist in the user's NTUSER.DAT file when the system is shut down.

Another example of volatile Registry keys and values is the *HKEY_LOCAL_MACHINE/Hardware* key and its subkeys. This key stores information regarding the devices connected to the system (CPU, keyboard, mouse, hard drive, etc.) and their assigned resources and is populated when the system boots up.

If you open the Registry Editor and navigate to the *HKEY_LOCAL_MACHINE\System* hive, you'll see a key named "CurrentControlSet," and most likely two others whose names begin with "ControlSet00" and end in a number. The CurrentControlSet doesn't exist when the system is shut down and is populated at boot time from one of the available ControlSets.

Note

When performing postmortem analysis of the Registry, it is a straightforward process to determine which ControlSet had been mounted as the CurrentControlSet on the live system. Simply open the System hive in a viewer and locate that Select key. Beneath that key, you will find a value named "Current," whose data is a number. If the data is "0x0001," the ControlSet mounted as the CurrentControlSet was ControlSet001 (found online at http://support.microsoft.com/kb/100010).

Yet another example of a volatile portion of the Registry is the HKEY_CLASSES_ROOT key. When the system is booted, this key is populated with the contents of the HKEY_LOCAL_MACHINE\Software\Classes key, and when a user logs in, the

HKEY_CURRENT_USER\Software\Classes key contents are added and, according to Microsoft, take precedence of the entries from HKEY_LOCAL_MACHINE entries (found online at https://msdn.microsoft.com/en-us/library/cc144148(VS.85).aspx).

What's important to keep in mind is that there are portions of the Windows Registry that only exist in memory. Thanks to folks like Aaron Walters and Brendan Dolan-Gavitt (both of Volatility memory analysis fame), this information can be accessed, retrieved, and analyzed; the necessary tools for collecting this data will be discussed later in this book.

Tip

Understanding the version of Windows that you're analyzing can have a significant impact on your examination. For example, Windows XP maintains System Restore Points by default, which means that depending on the system being used, you may have access to a great deal of historical data. Portions of the Registry are maintained in System Restore Points (ie, not all portions of the hives are stored, as it wouldn't do well to reset a user's password to an older one when restoring a system to a previous state) and can be easily accessed during analysis. Also, keep in mind that System Restore Points are created for a number of reasons, such as driver installations, as well as simply being created every 24 h. More recent versions of Windows use Volume Shadow Copies to maintain backups of files, and accessing those Volume Shadow Copies can give you a view into the Registry in an earlier state. Understanding System Restore Points and Volume Shadow Copies can provide a view into Registry data that isn't accessible through any other means.

Finally, Windows 7 includes the ability to run XPMode, a specific Windows XP installation intended to provide backward compatibility to run older applications. Users can install applications that have trouble running in Windows 7 into the XPMode Virtual PC installation and access them via the Windows 7 desktop. This also means that on any Windows 7 system with XPMode installed, there is a second source of potentially valuable Registry hive files!

Where Else Can We Find Registry Data?

Another Registry hive file that has gained a great deal of attention since Windows 8 was released is the AmCache.hve file, found in the *Windows\AppCompat\Programs* folder. Yogesh Khatri published some interesting research to his blog (found online at http://www.swiftforensics.com/2013/12/amcachehve-in-windows-8-goldmine-for.html) regarding the AmCache.hve file in December 2013. In his initial post, he indicated that the information he was sharing was a "logical continuation" of Corey Harrell's blog post regarding the Application Experience and Compatibility feature of Windows (Corey's post can be found online at http://journeyintoir.blogspot.in/2013/12/revealing-recentfilecachebcf-file.html). What this file, formatted in the same

manner as Registry hive files, appears to contain is information regarding applications that have been installed and possibly executed on the system. There is still more research to be done, but Yogesh has documented a great deal of information from this hive file, including Registry key names that appear to be MFT file reference numbers (for those systems running the NTFS file system), allowing the information to be tied back to the file system. This hive file will be discussed in greater later in the book.

On Windows 7 systems (I don't have an image of Windows Vista system to check for this), there's a file "System Volume Information" folder named Syscache.hve that follows the same format as Registry hive files, and in fact, you can extract this file from an image and open it in a Registry viewer application. At the time of this writing, there is not a great deal of public information available about this file. There is some information available at the "Windows Registry Knowledge Base" (which can be found online at https://code.google.com/p/winreg-kb/wiki/SysCache), but it largely seems incomplete. This file is likely associated with Volume Shadow Copies in some way.

Nomenclature

When working in the incident response and digital forensics field, as with many other fields, it is necessary to have and observe specificity of terminology. Basically, this is just a fancy way of saying that we all need to agree on what different things are called, and then call them that. When I took one of my first vendor-specific training courses for a commercial forensic analysis application, the instructor spent the first hour or more explaining what a "CPU," "hard drive" or "disk," a "computer system" really were. As someone with an electrical engineering degree, if you ask me to go into a room with a computer and retrieve a "CPU," I'm going to open the computer, go to the motherboard and extract that little black square thing with all of the pins coming out of it, so I really hope that you aren't expecting the entire computer.

In short, it's important that when talking about parts of the Registry, we all have to have a consistent understanding of what it is we're referring to, so that we can communicate clearly and avoid (as much as possible) confusion and misunderstandings. Fig. 1.4 illustrates the various components of the Registry, specifically keys, subkeys, values, and data. We'll go into more depth regarding the details of the binary structure of these components.

From the Registry Editor view illustrated in Fig. 1.4, "keys" and "subkeys" are the folders displayed in the left-hand pane of the

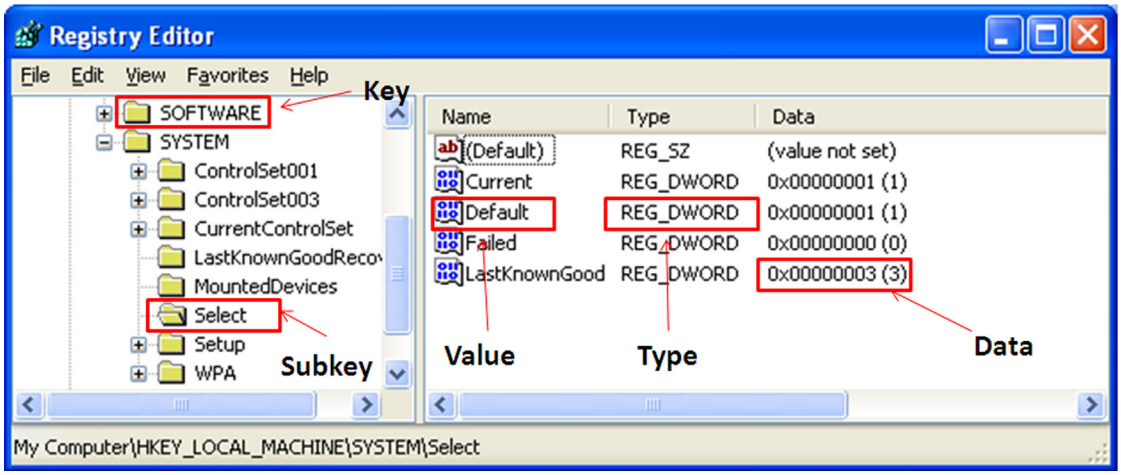

Figure 1.4 Registry nomenclature.

editor. This is an apt metaphor, in that keys can contain or point to other keys (ie, subkeys) as well as values. Keys also contain very valuable information from a forensic perspective (their LastWrite time) within their binary structure. Values, in the right-hand pane in Fig. 1.4, are much simpler and contain data of a specific type, be it a string value, multiple string values, binary, or DWORD, which is just a 32-bit binary value.

More importantly, we now have a frame of reference for discussing the Registry and Registry analysis throughout the rest of this book, and a common understanding of what a "key" is and what a "value" is, and how they relate to each other. Many times in such discussions, consistent terminology may be reversed or simply not used, and confusion ensues.

Registry Structure

Now that we've seen where the Registry "lives" within a live system, and subsequently within an acquired image, it's important to take that one step further and understand the structure of the Registry itself, as we may find vitally important information in places other than within Registry hive files. For example, we may find Registry data within unallocated space from an acquired image, or within the hive file itself (yes, Registry hive files do contain "unallocated space"!). We may also find Registry data and indeed entire hives within a memory dump from a live, running system, or within the Windows pagefile.

Tip

Brendan Dolan-Gavitt has done considerable work with respect to locating and accessing Registry information within Windows memory dumps and has contributed plugins to the Volatility project for accessing this data.

Regardless of where Registry data (keys, values) are found, it is important to understand the binary structure of the Registry, so that we can understand what Registry viewing applications are showing us. Whether we're viewing a Registry hive file via a commercial forensic analysis application or a hive file viewer, understanding the structure of the Registry helps us understand what we're seeing, as well as what we aren't seeing. Remember that the viewer provides a layer of abstraction, representing to the analyst what the data should look like; as such, some data may not be apparent or easily read and understood, due to shortcomings in the viewer, the nature of the data, etc.

REGISTRY HIVES AND SEARCHES

When performing PCI data breach investigations, one of the things I needed to do was search across the entire hard drive for what could be credit card data, including both the numbers themselves, as well as track data. In one particular instance, my search revealed a number of hits within Registry hive files, specifically an NTUSER.DAT hive in one user profile, and within the Software hive file. Viewing the data around the search hits within the hive files, I did not see anything that resembled a Registry key or value; likewise, opening the hive files in a viewer and searching for the search hits provided no indications that the hits were key or value names, or in Registry data. As it turned out, the search hits were actually located in file slack, something that we were able to determine through an understanding of the binary structure of the Registry.

Thankfully, the binary structure of the Registry itself has remained fairly consistent across the various versions of the Windows operating system, from Windows NT all the way through to Windows 7. This means that a viewer application that understands the structure of the Registry will, for the most part, work equally well on all versions. What's changed, however, are the names and locations of various keys and values…where data is stored and what format it is in will differ between versions of the Windows operating system. Windows XP, for example, maintained information about WAPs that had been connected to (connections that were managed by Windows, rather than a third-party utility) in a binary data structure within values beneath a specific Registry

key. Vista and Windows 7 employ an entirely different format for similar information and add some additional information…all of which is located beneath a different Registry key.

A great place to start in developing an understanding of the hive file structure is Mark Russinovich's "Inside the Registry" article in Windows NT Magazine (available online at http://technet.microsoft.com/en-us/library/cc750583.aspx). This article provides an excellent overview of the structure of the Registry, identifying the various cell types (key, value, subkey list, value list, etc.), bins, and the cell map relationships between them.

When I initially began looking into the structure of the Registry from a programming perspective, I relied heavily on Peter Nordahl's work with his offline NT Password and Registry Editor (found online at http://pogostick.net/~pnh/ntpasswd) in order to understand the binary structures that comprise a Registry hive file. Peter's utility allows you to boot a Windows system (originally from a diskette, there's now a version that runs on a boot CD) and, for one, modify any password. When you reboot the system, you can then log into the system using the user account you select and the new password you created. I used an early version of this utility to access Windows XP systems turned in by departing users in a corporate environment, and I have used the boot CD version more recently when booting an acquired image through VMWare. While the utility itself has been extremely useful, what I was looking for was the source code, which Peter provides. Within the source distribution archive is a file called "ntreg.h," which contains constant values and definitions for various structures within the Registry. Within the source archive you will also find a file named "WinReg.txt," which has a bit of a summary of what's in the ntreg.h file, including descriptions of some of the structures without as much detail as the header file. Using this information, along with a hex editor, I was able to start writing my own binary Registry hive file parser in Perl, allowing me access the information stored within the files and obtain as much detail as I wanted. As I began developing this hive file parser, I ran across the Parse::Win32Registry Perl module (available online at https://code.google.com/p/parse-win32registry/) written by James Macfarlane. This module provides an easy-to-use object-oriented (OO) interface for accessing various structures within the hive files. I should point out that this is an entirely different module from the Win32::TieRegistry module that ships with ActiveState's Perl distribution, in that the Win32::TieRegistry module allows a Perl programmer to interact with a *live* Registry (on a running system, as may be the case during incident response), not directly with the hive files, as is the case with James' module.

In the spring of 2008, Jolanta Thomassen asked me if I would act as her sponsor for her graduate thesis, which involved understanding the structure of the Windows Registry with a specific focus on locating deleted keys and values within the hive file itself. This topic had intrigued me for quite some time (as a reference for her, I provided a link to a UseNet post I'd made in 2001 asking about unallocated space in hive files), and Jolanta did a fantastic job not only in understanding what deleted keys and values "look like" but also how to recover them and present them in an easy to understand format. The result of her work is a utility called "regslack," the Windows portable executable (PE) version of which I use quite regularly, and I have to say, effectively.

In February 2009, Peter Norris posted his master's thesis regarding *The Internal Structure of the Windows Registry* online at http://amnesia.gtisc.gatech.edu/~moyix/suzibandit.ltd.uk/MSc/. Peter's work goes into considerable detail regarding the binary structure of the Windows Registry and also referenced Jolanta's work. It is beyond the scope and focus of this book to review Peter's work in detail, and such a review is left as an exercise to the reader.

Mark Russinovich's "Inside the Registry" article, mentioned earlier in this chapter, describes a number of Registry cell or "record" types. Of those, we are primarily interested in and will be focusing on the key and value cells/records, as these provide the vast majority of information of interest to forensic analysts. Other cell types (subkey list, value list, etc.), while significant, are beyond the scope of this book, and a detailed examination of those cell types is left as an exercise to the reader. These cell types are simply pointers to lists of subkeys or values, and are not key or value structures themselves.

Note

In January 2015, FBI Special Agent Eric Zimmerman began his own blog called "Binary Foray," describing it as a "blog about programming, digital forensics, and other such things." Eric had already developed several tools, two of which are specifically designed to assist analysts in engaging in Registry analysis: ShellBag Explorer and Registry Explorer. Both of the tools will be discussed at greater length in the subsequent chapters of this book. Eric has put a great deal of effort into developing a detailed understanding of the structure of the Windows Registry, including developing not only an extremely detailed understanding of various cell types within the Registry, but also developing tools that expose those structures to analysts. At the time of this writing, I would definitely consider his blog posts on the various Registry structures to be required reading for anyone interested in developing more than the most basic understanding of the Windows Registry.

Registry hive files are made up of 4-KB sections or "bins." These bins are meant to make allocation of new space (as the hive file grows), as well as the maintenance of the hive file itself, easier. The first 4 bytes of a normal hive file starts with "regf" (or 0x66676572). From there, as you traverse through the hive file on a binary level, as with a hex editor, every 4096 bytes you should see "hbin" (0x6E696268). Per Peter Norris' thesis work, various cells within the hive files do not cross hbin sections; that is, a key cell will not be split between two adjacent hbin sections, overlapping the border between them. As such, the hbin sections can be considered self-contained.

Tip

As with other types of files, allocation of new space for hive files, as the Registry grows, can pose something interesting challenges for a forensic analyst. When a new hbin section is required, that 4-KB section is, in many cases, allocated from previously used space within the file system, space that at one time may have contained valid data. During one examination in particular, I ran a search for credit card numbers and received several hits "in" Registry hive files. Closer examination of the data indicated that the discovered credit card numbers were not part of the "live" Registry (not contained in key or value names, nor in value data), and that the most likely explanation was that the numbers had resided in sectors that had previously comprised another file (possibly a database) which had been deleted.

The first hbin marker is very important, as this is the base location for offset values listed with the key and value cells throughout the rest of the hive file. What this means is that when you're reading values within a key cell structure (which we'll be looking at shortly) and you read an offset, that value is the offset from the first hbin marker. For example, as we'll see shortly, each key cell contains a value for the offset to its parent key, which essentially points back to that key. That offset, in bytes, is measured from the beginning of the first hbin marker, which itself is 4096 bytes from the beginning of the hive file. On the surface of this, you may be wondering how this information is useful. Several open source tools that assist the analyst with locating and extracting (ie, "carving") data from unallocated space within an image allow the analyst to designate a header and footer for locating data or to designate a header or marker (also known as a "magic number") and then read in a set number of bytes. These data carving tools can be used to search unallocated space or similar unstructured data such as the Windows pagefile or a hibernation file for Registry "hbin" sections.

Registry Key Cells

The Registry "hbin" sections are made up of several types of cells, but for our purposes, we're going to focus on the key and value cells. Key cells (or "keys") are very important to forensic analysts as they contain time-based information within their structure, in the form of their LastWrite time. The LastWrite time is a 64-bit FILETIME structure, marking the number of 100 ns intervals since midnight of January 1, 1601 (a description of the FILETIME structure can be found online at http://support.microsoft.com/kb/188768). A key cell (without the name) is 80 bytes long and starts with a 4-byte (in Microsoft parlance, a "DWORD") value indicating its size, followed by the node identifier, node type, the offset to the key's parent, the number of subkeys, the offset to the subkey list, the number of values, the offset to the value list, the offset to the security identifier, and the length of the key name (begins immediately after the key structure). Note that this is not a comprehensive list of the values within the key cell structure but rather an overview of the values that are of greatest interest. Fig. 1.5 illustrates the binary structure of a Registry key (viewed in a hex editor) with the node identifier (ID) and LastWrite time values of the structure highlighted.

As illustrated in Fig. 1.5, the node ID is "6E 6B" (0x6B6E in little endian format), or "nk," and is followed by the node type of 0x2C, which indicates a root node (0x20 indicates a "normal" key node). Immediately following the node type is the LastWrite time, which is a 64-bit FILETIME object.

Warning

It was once thought that a key LastWrite time was only updated when some action was taken through the API: a key was created, or a subkey or value was added or deleted, for example. In 2012, Joakim Schicht created a small utility named "SetRegTime" that allows a user to modify key LastWrite times to arbitrary values through the API. The utility was first available on Google Code (found online at https://code.google.com/p/mft2csv/wiki/SetRegTime), which includes some explanation of the utility as well as examples of how to use it. The utility was later found on GitHub (found online at https://github.com/jschicht/SetRegTime), which includes a link back to the Google Code page.

Table 1.1 lists the key cell structure details, illustrating the elements of that structure that are of primary interest to forensic analysts.

Table 1.1 should not be considered all-inclusive, as it details those structure elements that are most important to forensic analysts. Again, the size of the structure detailed in Table 1.1 is 80 bytes, and the first 4 bytes of the structure contain the size of the

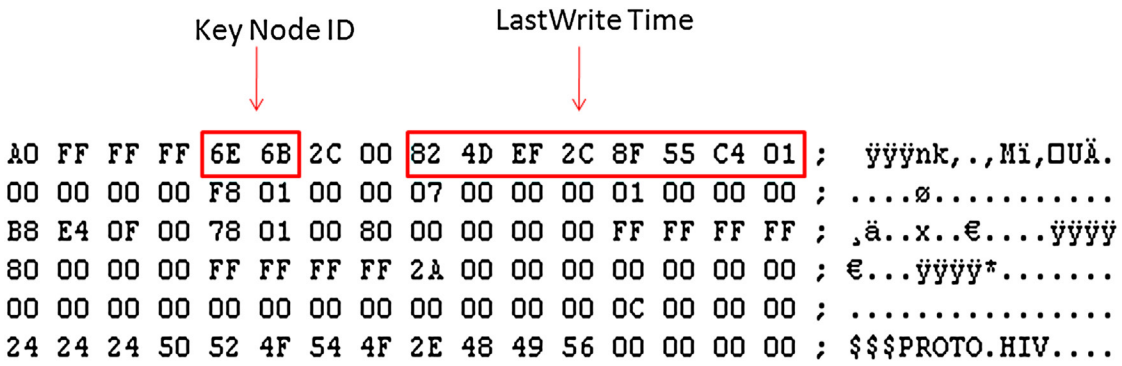

Figure 1.5 Registry key structure with node ID and LastWrite time.

Table 1.1 Registry Key Cell Structure Details

Offset (bytes)	Size (bytes)	Description
0	4	Size
4	2	Node ID ("nk," or 0x6B6E)
6	2	Node type (0x2C or 0x20)
8	8	LastWrite time
20	4	Offset to this key's parent
24	4	Number of subkeys
32	4	Offset to the list of subkey records
36	4	Number of values
44	4	Offset to the value list
48	4	Offset to security identifier record
76	2	Length of the key name

key cell, which includes the key name and any necessary padding. Therefore, the total size of a Registry key is the 80 byte header, the name, and padding; for the key illustrated in Fig. 1.5, that is 96 bytes.

The size value (the first 4 bytes or "DWORD") is an important aspect of the key structure of which to take notice. When read as an unsigned integer, the size is "4294967200," and we know that a single key would not usually be expected to be on the order of 4 GB in size. However, when read as a signed integer value, those 4 bytes equal "-96." Again, the key "header" itself is 80 bytes, and the actual name of the key begins immediately after the key structure. The name of the key illustrated in Fig. 1.5, "$$$PROTO.HIV," is 12 bytes and there are an additional 4 bytes of padding, rounding

out 16 bytes. That makes the total size of the key itself 96 bytes. This is important, as Jolanta (and others) had determined that for normal, allocated Registry keys, the size is a negative value when read as a signed integer value. However, when a key is "deleted," the size value is made positive. If the key in Fig. 1.5 were deleted, the size would be changed to "60 00 00 00," or 0x60. This, along with some other checks, is how deleted keys can be located within unallocated space within the hive file.

Note

Time-based information is maintained in the Registry (and on Windows systems, in general) in a number of formats. There are values whose data consists of (in part or entirely) a 32-bit Unix epoch time format, while the LastWrite times of keys, as well as data of some values, consist of 64-bit FILETIME objects. Still other time-based data is maintained as 128-bit SYSTEMTIME objects (a description of the SYSTEMTIME structure can be found online at https://msdn.microsoft.com/en-us/library/ms724950(VA.85).aspx) and others are simply maintained as strings (for example, the Skype application has a value named "LastUpdatedDate" in the user's NTUSER.DAT file with string data of "01/10/2009").

Registry Value Cells

The other type of cell that we want to take a close look at is the value cell. Remember, Registry keys can "contain" subkeys and values; actually, as we've seen, a key doesn't actually contain this information, as it instead has offsets to pointers to subkey and value lists. Value cells, on the other hand, are much simpler, as they don't contain pointers to any other cells. They are important as they do contain value names and point to the data that, in many cases, we're interested in knowing and understanding. Fig. 1.6 illustrates the binary structure of a value cell, with the value node identifier and value type highlighted.

Table 1.2 provides the relevant value cell structure details. As with the key cell, the first 4 bytes of a value cell (as illustrated in Fig. 1.6) contain the size of the cell.

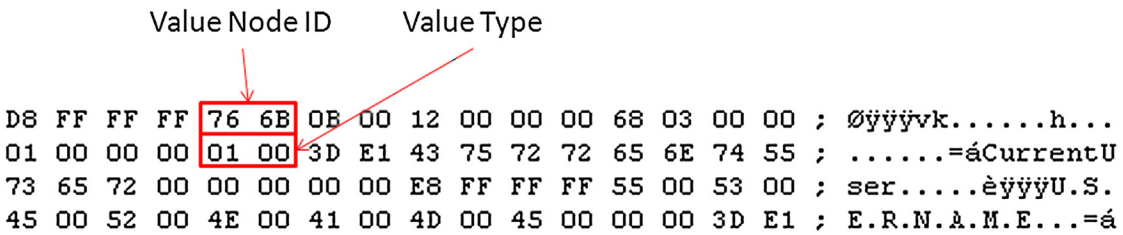

Figure 1.6 Registry value structure with node ID and value type.

Table 1.2 Registry Value Cell Structure Details

Offset (bytes)	Size (bytes)	Description
0	4	Size (as a negative number)
4	2	Node ID ("vk," or 0x6B76)
6	2	Value name length
8	4	Data length
12	4	Offset to data
16	4	Value type

Notice that while value cells contain some specific information, something that they do not contain is a FILETIME object, nor any other reference to a time stamp of any kind. Again, as with the key cell, not all of the value cell structure elements are listed, and Table 1.2 should not be viewed as all-inclusive. For example, immediately after the "value type" element is a 2-byte element called "Flags," and as of this writing, I have not been able to locate an available description of this element, nor of its use.

DATA STRUCTURES FOR THE WIN!

Most analysts will probably look at this information on the structure of Registry keys and values, and wonder to themselves, "when would I ever use something like this?" I'll tell you, I use this information *all* the time. Seriously. The first edition of this book was published in 2011, and even in the early months of 2014, I was using this information to great effect on several examinations. In one particular case, I used the binary structure information for Registry values to identify the use of a Unicode character in the value name that effectively hid the presence of malware from the user, as well as analysts, based on how graphical Registry viewer tools handle the Unicode character. In short, the Unicode character "\xA0" was used in a value name, as well as within the data (a file system path), and that character appeared as a space in viewer tools, so that the value and data appeared to point to a legitimate Microsoft application. Using this information, a hex editor, and some coding skills, I was able to see past this technique and locate the malware.

Registry values can point to data of a variety of types. Table 1.3 lists the available Registry value types, along with their names and descriptions. This information is also available from Microsoft website at http://msdn.microsoft.com/en-us/library/ms724884.aspx.

After walking through all of this information on the structure of individual key and value structures, and even though we haven't

Table 1.3 Registry Value Types

Type	Name	Description
0	REG_NONE	No value type
1	REG_SZ	Unicode null-terminated string; can be Unicode or ASCII
2	REG_EXPAND_SZ	Unicode null-terminated string with environment variables/ references
3	REG_BINARY	Binary data (no set length or structure)
4	REG_DWORD	32-bit number
5	REG_DWORD_BIG_ENDIAN	32-bit number
6	REG_LINK	Unicode symbolic link
7	REG_MULTI_SZ	Multiple Unicode strings, each "\00" terminated
8	REG_RESOURCE_LIST	Resource list (resource map)
9	REG_FULL_RESOURCE_DESCRIPTOR	Resource list (hardware description)
10	REG_RESOURCE_REQUIREMENTS _LIST	A series of nested arrays that store information about device drivers
11	REG_QWORD	64-bit number

really gone into how the keys and values are linked together, it should be evident at this point that each hive file is essentially something of a small file system. There's a root key that points to other keys (subkeys) and values. Keys can "contain" or point to other keys and values, and values point to data. The links do not go the other way; however, values do not point back to keys nor do values "contain" keys. So in a way, the keys can be considered to be analogous to folders, with only last modification (LastWrite) time stamps, and values can be considered analogous to files. The Registry hive file can have unallocated space, and keys and values that are deleted become part of the unallocated space within the hive file and can be extracted until they are overwritten (much like files within the file system). It's also possible for value data to contain slack space; if the space allocated for a value data is more than what is actually being used (sound familiar?), then there may be slack space containing data that can be extracted.

Summary

In this chapter, we've taken a look at what the Windows Registry is, on a variety of levels. By now, you should have a basic understanding of not only what the Registry is and its purpose but also where the Registry "lives" on disk and where to look for Registry files within an acquired image. This is extremely

important from a forensic analysis perspective, as it allows the analyst to understand issues that may develop through the use of commercial forensic analysis applications. Also, we've addressed more detailed information, going so far as to outline the binary structure of key and value cells. This information allows the analyst to search for and recognize these structures, not only within Registry hive files but also within other data sources, such as the Windows pagefile, memory dumps, and hibernation files, as well as unallocated space on disk.