Using PowerShell to Capture and Compare Windows Registry and Live Memory Artifacts with Online Databases to Identify Suspect Files

Kenneth Dale Cook*, Narasimha Shashidhar*[§]
* Department of Computer Science, Sam Houston State University,
Huntsville, TX 77341, USA.
{kdc057, karpoor}@shsu.edu

Abstract - System administrators and forensic investigators alike face a multitude of challenges when seeking to identify sources of pertinent data while in the course of their work. The inconsistent identification and acquisition of significant registry keys is frustrating, second only to the common practice of overlooking unique data stored in system memory. Also challenging, is the practice of identifying suspect file signatures from the resulting data. Many tools are available for scanning and identifying suspect files, and as such it makes sense to utilize them where possible. In this paper, we present a ***PowerShell*** tool and the accompanying method to acquire, parse, and display not only significant registry data, but also perform live memory acquisition of the application compatibility cache where key registry attributes are stored before being later written to the registry. These keys, stored in memory, are of particular interest since they can be an indicator of executed processes that are not yet recorded in the registry, and therefore potentially helpful to system administrators and investigators. This tool identifies the contents of the Application Compatibility Cache stored in volatile memory, and compares them to the same dataset recorded to disk in the Windows Registry. The items that exist in memory, but are absent from the registry on disk, are hashed and submitted to the VirusTotal.com database where the results are returned and presented in the form of a report. This paper contains not only positive VirusTotal.com results, but also other significant data from the registry that may be of interest to the administrator and investigator.

Keywords – *Registry; Memory; PowerShell; Appcompatcache; ShimCache.*

1. Introduction

The windows registry, together with the system's volatile memory, contains information that is not only critical to the functioning of the system, but also of irreplaceable benefit to the systems administrator performing troubleshooting and diagnostic work, or the investigator searching for clues/artifacts. From critical system settings, executed and accessed files, to user preferences, the registry and volatile memory offer both a recorded history of what has happened on the system, as well as a road map to what will happen in the future.

In order to add to the existing toolsets that are available, this project expands on prior research and work involving the registry, volatile memory, and a handful of unique applications that enable analysis with the PowerShell scripting environment, consolidating all of these aspects into a single tool. Prior research shows that certain registry

attributes stored in memory have been found to be unique in that they may not be written to the registry hives on disk until a successful shutdown has been initiated. This project is a PowerShell tool that, with the integration of existing tools specific to accessing elements in volatile memory, seeks to address the following topics:

1) Create a consolidated report of significant registry keys.

2) Retrieve the Application Compatibility Cache entries in both the registry and memory.

3) Submit the hashes of uncommitted Application Compatibility Cache entries from Volatile memory to an online malware scanning database in order to check for suspect files and report results.

This paper is organized as follows. In Section 2, we provide a background review of the Windows Registry, Volatile Memory, and PowerShell, as well as the additional resources and executables. Their importance, role, and method of utilization will also be discussed. In Section 3, the methodology and further details of the script's functional areas are described. Section 4 is devoted to the final report and exported files as compiled by the tool, while Section 5 will provide a conclusion and briefly discuss future work.

2. Background

Whereas the focus of this project is to capture and present data present in the Windows Registry and Volatile Memory (RAM), followed by presenting a subset of this data to an online database for identification of suspect files, some background is necessary to describe the Windows Registry, Volatile Memory, and the primary tool – the PowerShell scripting environment. Additionally, there are external executables utilized for their specialization and those are described here as well.

A. Windows registry

The Windows Registry is a central repository in the form of a hierarchical database of configuration data for the Windows operating system and many of its programs [1]. First introduced into the Windows operating system with the release of Windows 95, it continues this role as a configuration database in Microsoft's latest releases. It has consolidated and organized the tasks of installing software device drivers, defining environment variables, and running startup programs. In addition, previously important files such as *config.sys, autoexec.bat, win.ini and system.ini*, previously used to store user settings and system parameters, have been consolidated into the registry [2].

The Windows registry is an important resource to the forensic investigator and system administrator due to its role within the operating system. For the investigator familiar with the type of information available and its location, a virtual treasure trove of valuable artifacts can be unearthed. As an investigative resource and troubleshooting tool, the registry may provide information on users, groups, hardware, software, networking, recently installed peripherals, and other areas such as programs executed and files accessed. This collection of information, taken in part or as a whole, may provide irreplaceable information and history concerning an incident.

Traditionally, the Windows Registry is accessed via a standard graphical user interface (GUI) that has largely remained unchanged across multiple versions of the parent operating system. The registry editor is accessed by referencing "regedit.exe" from the command line. It is a no-frills interface for casual browsing and even editing of the database that many refer to as the "brain" or "heart" of the operating system. While it does

permit backup and restore of the data within, it lacks in areas of automation, bulk commands, and especially the inability to view encrypted or binary data. It is in these areas where our tool excels, as discussed later in this paper.

Organizationally, the registry is divided into sections titled hives, containing a logical group of keys, sub keys, and values. Reviewing the work previously conducted by Farmer [2], and cross-referencing with the MSDN.Microsoft.com [3] documentation for the registry, we can define the primary hives along with their keys and values. Of particular note is that the first two hives are in fact aliases of other physical hives.

HKEY_CLASSES_ROOT (HKCR) – Contains file-name extension associations and COM class registrations, in order to define which application is executed when a file is opened. Also contains drag-and-drop rules, shortcuts, and user interface information. An Alias for: HKEY_LOCAL_MACHINE\SOFTWARE\Cl asses.

HKEY_CURRENT_USER (HKCU) - Defines preferences for the currently logged-on user, including folder, display, and control panel settings. An alias for the current user's branch in: HKEY_USERS\User SID

HKEY_USERS (HKU) – Defines default and new user configuration including application configurations and visual settings.

HKEY_CURRENT_CONFIG (HKCC) - Contains information about the current hardware profile or system configuration.

B. Volatile Memory

Volatile memory describes computer storage that maintains its digital contents only while the device is powered on, which in common usage, refers to the RAM (Random Access Memory) in personal computers and servers. This is in contrast to non-volatile storage such as hard disks, solid state drives, magnetic tape, and other similar media types that do not rely on constant power to retain information.

The Windows Operating System utilizes volatile memory (RAM) to store the operating system's code and data needed by the CPU. This process provides faster read and write access to the data by the CPU than traditional hard disks are able to provide. The information is stored and accessed, while also swapped in and out of Virtual memory, paging, and hibernation files as the need and settings require. For the purposes of our research project, we will focus on live memory acquisition and the data that is actively stored in volatile memory

The forensic values of the RAM contents are an important reason to focus on the data that resides within, as well as the methods used to obtain it. Through acquisition and analysis, we are able to see a history of executed commands, network connections, processes, and other items that only exist in memory. Disk encryption keys, application shims (injected code), chat messages, internet history, and others exist for the investigator to obtain through diligence and proper execution of memory imaging tools [6]. In addition to the aforementioned sensitive information, data such as usernames and passwords may be unintentionally exposed in the system memory during use and subsequent analysis.

In "Comparative Analysis of Volatile Memory Forensics, Live Response vs. Memory Imaging" Aljaedi et al., prominently state the significance of volatile memory analysis in digital investigations due to certain data residing only in physical memory (RAM),

and not existing in any state on the physical media. Examples such as Code Red, Witty, and the SQL Slammer worms function in memory without writing themselves to disk, and therefore support the importance of acquiring volatile data as one of the beginning steps in incident handling [7]. Additionally, these points are further supported in well-known incident handling guides [8].

The structure of physical RAM requires the use of external tools in order to easily extract the contents for analysis. Our project will utilize multiple tools in order to simplify access to the information held in RAM. This will involve creating an image of the memory contents as they exist at a point in time, and utilizing separate tools to parse the resulting image file for the data contents. These methods will be explained in detail later.

C. PowerShell

Windows PowerShell is the native scripting environment for Windows environments, consisting of a command line shell built on the .NET framework, enabling it to interface with .Net objects. The ability to accept and return .Net objects while most scripting shells accept and return text provides for a significant change in the way administrators are able to configure and manage Windows environments [9].

PowerShell version 2.0 was released in 2009 and included in Windows 7 and Windows Server 2008 R2. Versions for XP, Vista, and Server 2003 were subsequently released and PowerShell has been included in all versions of workstation and server operating systems since 2009. The latest version (x86 or x64) may be downloaded and installed from Microsoft's website [10].

The base command in PowerShell is comprised of a cmdlet, which is in the format of Verb-Nouns, such as Get-File hash (the cmdlet for obtaining the file hash of a file, which we will use later in our project). The self-descriptive nature of the name persists throughout the shell, providing searchable help. The results are output not at text, but rather as objects that persist through the pipeline [11].

The PowerShell pipeline supports the piping of output from one cmdlet to another, whereas either all the object properties or selected properties may be piped and thereby used as input in subsequent tasks. PowerShell provides methods to format data, and thus customization of the output objects is possible. Additionally, external executables are also supported, whereby PowerShell receives a text-stream of the output from the executable in order to make the output available to the PowerShell system [12, 13].

From a forensic viewpoint, PowerShell can perform an important role in gathering data. Its ability to configure and manage Windows system changes both locally and remotely may be leveraged to also query those same systems and system settings. The resulting data sets can be used for investigations, auditing and reporting. Of particular note is PowerShell's ability to natively read the Windows Registry and parse the data as a normal file system. The ability to use the common Get-Item and Get-ChildItem PowerShell cmdlets simplify the task of querying the Windows registry and compiling reports. This is in contrast to previous methods involving manually looking at regedit.exe or other scripting tools. Additionally, where other tools are necessary or preferred, the ability to work with the external tool's returned data stream within a single PowerShell script ensures that a single tool can be used and greatly reduces the multiple steps that would otherwise be necessary. The next section will describe

external tools that we will use with PowerShell to acquire the information.

D. WinPMem Version 1.6.2

This stand-alone tool, which enables us to acquire the system memory, while creating a single image file, is part of the Rekall Forensic and incident response framework, which bills itself as the most advanced open source memory analysis framework available [14, 15]. Like many forensic tools, the platform requires Python [16] to be installed. However, we have chosen to use a version of WinPMem that has been compiled into a windows executable in order bypass the requirement for python to be installed on the target system. The particular version on WinPMem used here, Version 1.6.2, is a prior version that natively supports the RAW image file creation, which is more broadly supported by our other chosen tools. During prior research, we discovered that the latest version, Winpmem, 2.1.post4.exe, supports a native format of AFF4. While it is possible to convert memory dumps from AFF4 to RAW format, we find this step to be unnecessary while adding unnecessary complexity to the process. In addition, it appears that Winpmem 1.6.2 uses less memory when running than subsequent versions. This is an important distinction since our acquisition tools should have the smallest footprint possible; especially if our investigations include processes which seek to examine unallocated structures. In those cases, the extra memory usage will overwrite unallocated space in the memory, potentially destroying valuable information [17].

E. Volatility Version 2.6 Windows Standalone Executable

The Volatility framework is one of the most widely used memory analysis and forensic toolsets available, as represented by years of published academic research. Based on Python, it is a cross-platform, modular, and extensible platform that enables collaboration, innovation, and accessibility to knowledge and tools within the forensic and offensive software/security communities. Used extensively by commercial investigators, academia, military, and law enforcement organizations, its reputation is widely known and prior success with it has resulted in its use within this project [18]. This stand-alone executable, like the previously mentioned WinPMem.exe, allows us to take advantage of this tool without installing the prerequisite Python distribution. Additionally, it retains the ability to fully support the vast catalog of plug-ins available to extend the native feature set. Our process leverages the plugins available for Volatility such as dumpregistry, and shimcachemem to not only extract the cached registry files from the image file we created, but to also extract and decipher the Application Compatibility Cache information that is stored within RAM. Their use will be further explained later in this paper.

F. Registry Ripper V1.0

Harlan Carvey is a name synonymous with practitioners of Digital Forensics, as not only an author of popular books on the subject, but also as principal author of the tool commonly referred to as RegRipper. An open source tool written in Perl, it serves as a registry data extraction and analysis tool. Offered in two versions, a GUI and a Command line tool, its strength lies in the exhaustive plugin library that enables the user to extract keys, sub keys, and their data [19].

From a forensics standpoint, RegRipper's significance is unquestionable. It has long been a staple in many investigator's toolbox, and its reputation and ease of use in other projects lends itself to this project without challenges. The ability for RegRipper to utilize its plugins and extract targeted data from the extracted hive could not have as easily be obtained otherwise. In this project we utilize plugins such as comdlg32, recentdocs, mp2, userassist_tln, and usbsotr2 in order to access the data from the extracted registry hives.

G. AppCompatCacheParser V9.8.0

The AppCompatCacheParser.exe by Eric Zimmerman is a standalone Windows executable for Windows 7, 8, and 10. It is used to dump shimcache entries directly from the registry stored on disk [20]. Contrary to the feature set of the aforementioned tools, the AppCompatCacheParser.exe has a single goal with limited options. When executed, it extracts the binary data in the AppCompatCache Key in the registry and exports it to a readable tab separated value (.tsv) file which can be viewed in a text editor or imported into a spreadsheet viewer. Each record in the resulting *.tsv file contains the following fields: ControlSet, CacheEntryPosition, LastModifiedTimeUTC Path, and Executed. This data is significant to the project as we will compare it to the same entries stored in volatile memory (RAM) in order to isolate any AppCompatCache entries that are in memory, but which have not yet been committed to the registry stored on disk.

H. VirusTotal.com API

The final external tool that this project utilizes is the virustotal.com API. Virus Total utilizes over 70 antivirus scanners and other tools such as URL and domain blacklisting services to analyze submitted executables and record the data. The website provides a method for uploading files for analysis, or the user may upload the hash of a file and Virus Total will check their database to see if they have a record for that file with that hash value.

This service is one of many providing cloud-based malware scanning to the Internet community. While the web interface allows for uploading one file at a time, the HTTP-accessible API provides a platform for a more automated and scriptable solution (such as our project) to be developed, whereby multiple items can be scanned from one instance. A free API key, necessary to use the API, is available to anyone that registers, although it has limited usage and restricts the number of inquiries to four submissions per minute. Additional calls may be made with an alternate, paid account. Since the other tools we are using are open source with no cost, this project will utilize the free API and limit the requests to no more than four per minute [21]. While prior work has been completed by others with each of these tools, this project extends the individual functionality by leveraging each application's strengths and combines their functionality into a new tool.

3. Tool Description

In simplest terms, the tool utilizes existing cmdlets within PowerShell as a basis for data acquisition while leveraging mature projects described in the prior section such as Winpmem and the Volatility framework to create and analyze volatile memory from a Windows machine. In the following tool description, we will discuss the methodology used, followed by a description of the actual tool script in detail. The major processes are divided into the actionable areas as follows.

A. Methodology

1) Integrate Winpmem with PowerShell to obtain the Memory dump and store for analysis.
2) Utilize standard PowerShell cmdlets to access the Registry on disk and record data for later use.
3) Integrate PowerShell and the Volatility framework in order to obtain the registry from the memory dump and store for analysis.
4) Use multiple tools to obtain the Application Compatibility Cache data from both the committed registry and uncommitted data in volatile memory.
5) Submit unique AppCompatCache entries from memory that do not exist in the registry to an online malware database (say, virustotal.com) for identification of suspect files.
6) Record data to an easy to read HTML file.

B. Tool Script

The tool itself is divided into 18 commented sections where specific actions occur in each. A brief summary of these areas is presented here to better understand the subsequent report in section 4, which is the output of the tool. Throughout the script, a common output goal is repeated, and that is obtain the data, create a hash of the system library or executable referenced when possible, create output for HTML report, and finally, export data to CSV file(s). The CSV files typically obtain the root RegKey, ValueName, ValueData, and generated hash.

1) Sections 1-7 of the script use the PowerShell Get-ItemProperty to access different keys and Sub keys in the HKLM, HKCR, and HKCU Registry hives as they currently exist on the system's hard drive. The different sections, containing different functions, are necessary due to the different structure of the hives, keys, and sub keys, along with the data they contain. The data targeted in these sections pertained to the RUN, RUNONCE, EXTENSION ASSOCIATION, COMMAND PROCESSOR, APPINIT_DLLS, and DEBUGGER options and settings within the registry. Additionally, The SERVICES sub keys are returned. Together, these datasets provide a detailed look at what has executed, as well as what will execute in the future in certain specific circumstances.

2) Pending completion of the above Sections 1-7, a function comprised of the Volatility stand-alone executable and assorted registry ripper plugins, are invoked in order to access the previously obtained image file, extract the registry hives to disk, and then parse those hives for specific data. The registry ripper plugins used are COMDLG32, RECENTDOCS, MP2, USERASSIST_TLN, and USBSTORE. Multiple variables store the data for each of the plugins and are used in the following sections.

3) Sections 8-13 of the tool use the data collected in the prior paragraph. This data, representing Registry keys as stored in the volatile memory, contain information on LAST FILE/FOLDER VISISTED, RECENT DOCUMENT LISTS, MOUNTED DRIVES/NETWORK ACCESSES, ACCESSED OBJECTS, and USB DEVICES as accessed recently by the host. These sections are unique in that while the prior sections used native PowerShell cmdlets to access the data, thereby resulting in similar and consistently formatted results, this is not the case with the Volatility command and its plugins. Each of the plugins resulted in a

unique report of the data, and while on its own would serve as a helpful report, in our use, it needs to be parsed and the data extracted to be utilized in a consistent and similarly formatted report. Thus, in each of the sections (8-13), unique functions and REGEX syntax are used to isolate the desired data and further conform to our reporting style. As with prior data, this information is written to both a CSV file and an HTML report to be discussed later in this paper (Section 4).

4) Sections 14-16 focus on the Application Compatibility Cache areas of the registry, both as stored on disk and the uncommitted registry information that is stored in volatile memory until the next shutdown sequence is completed, at which time the registry on disk is updated with those entries in RAM. The method used in these three sections, begins by obtaining the APPCOMPATCACHE entries from Disk using the AppCompatCacheParser.exe, which exports its data to a *.tsv file. The subsequent step seeks to obtain the APPCOMPATCACHE entries from volatile memory using the volatility executable with the shimcachemem plugin and output that data to a *.csv file. Finally, in Section 16, these two datasets are compared and newer, uncommitted entries that only exist in memory are extracted and saved to their own *.csv file. This resultant dataset is used exclusively in the next and last working section of the tool, namely section 17.

5) Section 17 is the last working section of the tool. The function in this section takes each entry from the prior section, which represents uncommitted appcompatcache data in volatile memory, and submits the hash of the executable to the API provided by VirusTotal.com. The results are returned to the script in the form of a positive or negative result. Negative results are discarded, and positive result sets are collected with their dataset converted to HTML for the final report.

4. Report

The final report is presented as an HTML-formatted output of the script's results. Generally, the results are presented with the following Headers: REGISTRY KEY, VALUE NAME, and VALUE DATA. Also obtained, but not always displayed in the report for space reasons, is the SHA256 HASH of the executable named in the VALUE DATA field. In addition, there are several CSV files that catalogue the output and the working data that are made available for further examination if desired.

A. HTML report

The HTML report is the final output of this tool. The report serves as an example of how the tool's data may be viewed to provide a dashboard into a host's registry settings, as well as any VirusTotal alerts pointing to possible suspicious files. The report contains a clickable table of contents with descriptions of each section. The table of contents contains the following areas, and figures are provided to demonstrate the formatting of the data as collected and presented by the tool.

TABLE OF CONTENTS

1. ALERTS

Figure 1 displays the first section of the report, the VirusTotal.com results, assuming that positive results were returned. If no positive results are present, this section states this fact in the report. In the example, ORDER is the position of record in the appcompatcache in

volatile memory, the LAST MODIFIED entry is the last time the executable itself was modified, the EXEC flag indicates whether or not the file was executed on the system or merely accessed, The FILE PATH is self-explanatory, the HASH is the SHA256 hash of the File, the HITS are the number of positive results from VirusTotal.com, and the LINK TO RESULTS field contains a hyperlink to the details page on VirusTotal.com where the detailed returns and comments for this particular hash are recorded. In the example provided in Figure 1, three of the engines used by VirusTotal flagged ACCParser.exe as a suspicious file. The hyperlink, if followed, will display the details to include which three engines, and more.

Order:	11
Last Modified:	2018-02-10 23:38:48
Exec:	True
FilePath:	C:\DATA\Tool\ACCParser.exe
Hash:	CB8CA3DC4B00ADF61C8A1A51038970913A
Hits:	3
Link to Results:	Link To VirusTotal.com Details for this Hash.

Fig 1. VirusTotal.com result example.

The next section of the Table of Contents:

2. *APPLICATION COMPATIBILTY CACHE*
   - *2.1 Live Registry…*
   - *2.2 Volatile Memory…*
   - *2.3 AppCompatCache keys in memory…*

Figure 2 displays the fields available in the HTML report for Section 2, of ORDER, LAST MODIFIED, EXEC, and FILEPATH. These fields contain the same type of data as those described in Figure 1.

Order:	6
Last Modified:	2015-07-09 17:57:57
Exec:	True
FilePath:	C:\Windows\system32\notepad.exe

Fig 2. HTML Report, section 2 example.

The next section of the Table of Contents:

3. *REGISTRY KETS FROM LIVE REGISTRY*
   - *3.1 AutoStart / AutoRun*
   - *3.2 Application Initialization and Default Executable*
   - *3.3 Services*

Figure 3 displays an example of the type of data available in this area of the HTML report. In these datasets, certain attributes are not available, such as ORDER or EXEC.

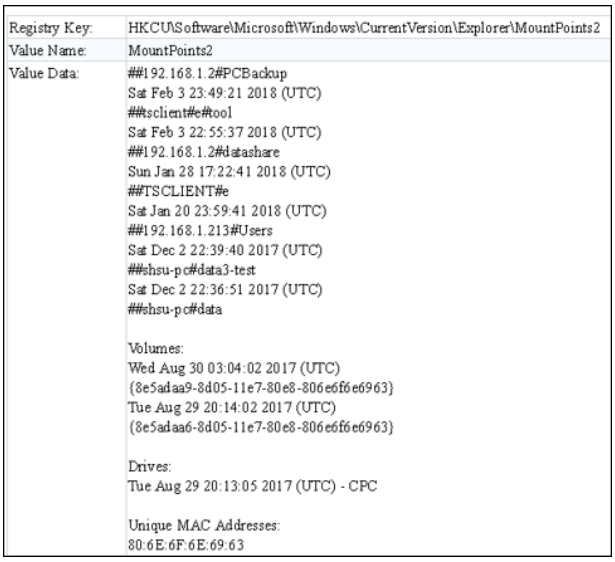
Registry Key:	HKEY_CLASSES_ROOT\CABFolder\shell\open\command
Value Name:	Default
Value Data:	C:\Windows\Explorer.exe

Fig 3. HTML Report, Section 3 example.

The next section of the Table of Contents:

4. *REGISTRY KEYS FROM VOLATILE MEMORY*
   - *4.1 Most Recently Used Keys*
   - *4.2 Network Drive / Mount Point*
   - *4.3 User Assist*
   - *4.4 USBStor*

This is the final section of the Table of Contents. The data contained is formatted similarly to the prior section as displayed in Figure 3. The exception is section 4.2 and 4.4. When data is available, the source tool presents it differently, and therefore those datasets are presented in one field as displayed in Figure 4. In the example, we see that the prior mount points for the operating system include several local SMB shares and volumes. In spite of the formatting, the same three fields of REGISTRY KEY, VALUE NAME, and VALUE DATA are represented to maintain consistency.



Registry Key:	HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\MountPoints2
Value Name:	MountPoints2
Value Data:	##192.168.1.2#PCBackup Sat Feb 3 23:49:21 2018 (UTC) ##tsclient#e#tool Sat Feb 3 22:55:37 2018 (UTC) ##192.168.1.2#datashare Sun Jan 28 17:22:41 2018 (UTC) ##TSCLIENT#e Sat Jan 20 23:59:41 2018 (UTC) ##192.168.1.213#Users Sat Dec 2 22:39:40 2017 (UTC) ##shsu-pc#data3-test Sat Dec 2 22:36:51 2017 (UTC) ##shsu-pc#data Volumes: Wed Aug 30 03:04:02 2017 (UTC) {8e5adaa9-8d05-11e7-80e8-806e6f6e6963} Tue Aug 29 20:14:02 2017 (UTC) {8e5adaa6-8d05-11e7-80e8-806e6f6e6963} Drives: Tue Aug 29 20:13:05 2017 (UTC) - CPC Unique MAC Addresses: 80:6E:6F:6E:69:63

Figure 4. HTML Report, Section 4 example.

Depending on the age of the operating system, or time since last reload, as well as how the system is used, the size of the registry and therefore the HTML report can vary widely. It was not uncommon during testing to see printed reports over 100 pages in length.

B. Exported Files

The files that result from executing the script are saved for reference, in the event that the investigator would like to look at additional information left out of the report, such as additional hashes, or to run their own queries against the CSV file for sorting and comparing.

1) Collected-data.csv – This csv file contains the output from the majority of the script, sections 1-13.

2) Out-Cache.csv – Contains data from the live Registry Application Compatibility Cache data from sections 14-16.

3) RegCompatCache.csv – Contains results of the AppCompatCacheParser application in section 14.

4) RamShimCache.csv – Contains the sorted and ordered results of the Volatility application and the shimcachemem plugin in section 15.

5) ShimCacheMem.csv – Contains the raw output of the Volatility application and the shimcachemem plugin in section 15.

6) <%VariableName%>_AppCompatCache.tsv – Where the first part of this file name represents the profile used in Volatility, combined with the Host computer name, this is a tab-separated file created by the Application "AppCompatCacheParser".

7) Final-HTML-Report.html - This is the Final report as created by the tool to display the results as collected.

While the exported files are not part of the final report, they are left available in the event that the investigator might wish to parse them differently or otherwise benefit from their contents.

5. Conclusion

This project discussed the significance of the registry and how certain aspects of it can not only be useful in investigations, but also how certain contents can vary depending on if they are stored in the memory or on disk, as items may exist in memory that have not yet been committed to disk. Exploring these differences and developing a tool for reporting the findings was the primary focus and result of this project. While developing the tool, the three primary goals were met; these being: Creating a consolidated report of significant registry keys, retrieving the Application Compatibility Cache entries from both the registry and memory, and lastly, submitting the hashes of uncommitted Application Compatibility Cache entries from Volatile memory to an online malware scanning database in order to check for suspect files and report results.

The main challenges incurred during this project resulted from developing the tool in PowerShell, which proved to be a challenging task due to the integration of many distinct tools involved in our research project. Once captured, parsing and manipulating the data from different sources into a common format sometimes proved difficult, but was almost universally accomplished.

This project, while serving as an example, or Version 1.0 of a tool, provides many opportunities for enhancement and modification to benefit others. Some areas or ideas to improve upon in the future are:

1) Obtain an enhanced API key from VirusTotal.com. The key in use in this version of the tool utilizes the free key and has limits on how many inquiries can be made per minute. This can be a significant issue in an OS with many records.

2) Provide a function that may allow for remote execution and capturing of the image to a remote host, thereby not writing anything to the target host.

3) Provide an interactive option to eliminate duplicate executables from the report.

4) Provide for Operating System detection and therefore automatic profile setting with the volatility binary.

5) Provide for Windows Hiberfil.sys and Windows Crash dump file support.

It is possible that given the amount of work that has been completed with this version, and the opportunities for future development and benefit, that this project may be viewed as a beneficial contribution in the areas of registry and memory acquisition and analysis from within the PowerShell environment.

References

[1] T. Roy and A. Jain, "Windows registry forensics: an imperative step in tracking data theft via USB devices", International Journal of Computer Science and Information Technologies (IJCSIT), 3(3), 4427-33, 2012.

[2] D.J. Farmer. *A forensic analysis of the Windows registry*. Champlain College Burlington, Vermont, 2007.

[3] https://msdn. microsoft.com/en-us/library/windows/desktop/ms724877(v=vs.85).aspx, MSDN. Microsoft.Com. "Structure of the Registry". Latest Access Time for the website is 2 July 2018.

[4] https://technet.microsoft.com/en-us/library/ee176771.aspx, Microsoft, "Registry Overview". Latest Access Time for the website is March 25, 2018.

[5] S. Zhang, L. Wang, R. Zhang, and Q. Guo, "Exploratory study on memory analysis of windows 7 operating system", In Advanced Computer Theory and Engineering (ICACTE), 3rd International Conference on (Vol. 6, pp. V6-373). IEEE, August 2010.

[6] M. H. Ligh, A. Case, J. Levy, and A. Walters. *The art of memory forensics: detecting malware and threats in windows, linux, and Mac memory*. John Wiley & Sons, 2010.

[7] A. Aljaedi, D. Lindskog, P. Zavarsky, R. Ruhl, and F. Almari, "Comparative analysis of volatile memory forensics: live response vs. memory imaging", Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE Third International Conference on Social Computing (SocialCom), 2011 IEEE Third International Conference on. IEEE, 2011.

[8] http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-61r2.pdf, NIST Special Publication 800-86, Guide to Integrating Forensic Techniques into Incident Response, Latest Access Time is July 2, 2018.

[9] https://docs.microsoft.com/en-us/powershell/scripting/getting-started/getting-started-with-windows-powershell?view=powershell-6, Microsoft, "Getting Started with Windows PowerShell".

Latest Access Time for the report is July 2, 2018.

[10] https://www.microsoft.com/en-us/download/details.aspx?id=50395, "Windows PowerShell". Latest Access Time for the website is July 2, 2018.

[11] https://msdn.microsoft.com/en-us/library/ms714395%28v=vs.85%29.aspx, "Cmdlet Overview", Latest Access Time for the website is July 2, 2018.

[12] https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_pipelines?view=powershell-6, "About Pipelines", Latest Access Time for the website is July 2, 2018.

[13] https://docs.microsoft.com/en-us/powershell/module/, "About Objects", Latest Access Time for the website is July 2, 2018.

[14] microsoft.powershell.core/about/about_objects?view=powershell-6, "About Objects", Latest Access Time for the website is July 2, 2018.

[15] https://github.com/google/rekall, "Rekall Framework", Latest Access Time for the website is July 2, 2018.

[16] Python. Python.org. 2016.

[17] J. Williams, and B. McCrillis, Memory Forensiscs; Always Test Your Forensics Tools. White Paper, Available: https://www.renditioninfosec.com/whitepapers/ Rendition%20InfoSec%20-%20Memory%20Forensics %20Tool %20Testing.pdf. 2018.

[18] http://www.volatilityfoundation.org/26. 2016, "Volatility Framework", Latest Access Time for the website is July 2, 2018.

[19] https://github.com/keydet89/RegRipper2.8. 2017. "Registry Ripper", Latest Access Time for the website is July 2, 2018.

[20] https://binaryforay.blogspot.com/2015/05/introducing-appcompatcacheparser.html, "AppCompatCacheParser". Eric Zimmerman. Latest Access Time is July 2, 2018.

[21] https://support.virustotal.com/hc/en-us/articles/115002126889-How-it-works. "VirusTotal API", VirusTotal.com. Latest Access Time for the website is July 2, 2018.

[22] https://technet.microsoft.com/en-us/library/cc978714.aspx?f=255&MSPPError=-2147217396, "Command Processor," Latest Access Time for the website is July 2, 2018.

[23] https://technet.microsoft.com/en-us/library/cc939696.aspx, "AppInit_DLLs," Latest Access Time is July 2, 2018.

[24] https://blogs.msdn.microsoft.com/mithuns/2010/03/24/image-file-execution-options-ifeo/, "Image File Execution Options (IFEO)," Latest Access Time is July 2, 2018.

[25] https://docs.microsoft.com/en-us/windows-hardware/drivers/install/hklm-system-currentcontrolset-services-registry-tree, Microsoft Corp., "HKLM\SYSTEM\ CurrentControlSet\Services Registry Tree," Latest Access Time for the website is July 2, 2018.

[26] https://docs.microsoft.com/en-us/dotnet/framework/winforms/controls/known-folder-guids-for-file-dialog-custom-places, "Known Folder GUIDs for File Dialog Custom Places," Latest Access Time for the website is July 2, 2018.