University of New Orleans

ScholarWorks@UNO

University of New Orleans Theses and Dissertations

Dissertations and Theses

Fall 12-20-2013

Analysis of Windows 8 Registry Artifacts

Jeremy M. Stormo
JeremyStormo@gmail.com

Analysis of Windows 8 Registry Artifacts

A Thesis

Submitted to the Graduate Faculty of the
University of New Orleans
in partial fulfillment of the
requirements for the degree of

Master of Science
in
Computer Science
Information Assurance

by

Jeremy M. Stormo

B.S. Louisiana State University, 2007

December, 2013

Table of Contents

List of Figures

ABSTRACT

Microsoft's series of Windows operating systems represents some of the most commonly encountered technologies in the field of digital forensics. It is then fair to say that Microsoft's design decisions greatly affect forensic efforts. Because of this, it is exceptionally important for the forensics community to keep abreast of new developments in the Windows product line. With each new release, the Windows operating system may present investigators with significant new artifacts to explore. Described by some as the heart of the Windows operating system, the Windows registry has been proven to contain many of these forensically interesting artifacts. Given the weight of Microsoft's influence on digital forensics and the role of the registry within Windows operating systems, this thesis delves into the Windows 8 registry in the hopes of developing new Windows forensics utilities.

Keywords: Windows 8, Registry, Forensics, Digital Forensics

1. Introduction

The Windows series of operating systems collectively accounts for the majority of all installed operating systems in the world. It is shown in a survey from w3schools for September 2013 that Win8, Win7, Vista, WinXP and Server variants of Windows collectively account for 82.5% of operating systems being recorded in their internet usage log files. That is compared to the combined Linux, Mac and Mobile share of 17.4% [1]. This family of operating systems is found powering everything from personal computers and cellular devices to medical equipment. Therefore, as this technology is encountered so frequently, each new release of the operating system must be met with vigorous analysis because of the overwhelming potential benefits forensics experts may realize. Such benefits might include more information on a user's network activity, identifying data exfiltration or indicators of security compromise. The opportunity for discovering these new sources of information is too great to be ignored.

Of all the forensically relevant constructs found in a typical Windows operating system, the Windows registry is of particular interest to an investigator. The registry exists as an extensive hierarchical database for use by system utilities as well as third-party applications. Its purpose is to provide a common platform for the storage of configuration settings and preferences. As such, the registry's usefulness is dependent on the information stored within it.

The Windows registry can be used to reveal much information of the course of a typical investigation. There are many keys that are well known to the forensics community for the data they provide. TypedURLs will reveal what specific URLs a person has typed into their Internet Explorer address bar. Most recently used lists are sources of information showing documents that have been opened or applications that have run. ShellBags are a set of keys that represent user settings for File Explorer. They can show that a user has opened particular directories on the system or network shares. Malware trying to achieve persistence after a system restart may have a presence within a registry. This is since there are several keys related to automated application or service launching during system boot. Application uninstall information can give an investigator an idea of what programs have been installed on the system. Particularly useful is the ability to identify removable USB drives that have been used on the by their unique serial number. These are only a few examples of useful artifacts that can be utilized during an investigation. Though the registry is a source of valuable information, it poses a problem common to forensic analysis: one of scale.

The registry contains a deluge of data, much of which may not be deemed valuable to a specific investigation. As an example, the final dataset I created had a total of 849,713 keys and values. The total number of keys and values on the Win7 system I used for analysis is 1,022,621. In order to avoid being overwhelmed by the scale of data, common investigatory techniques avoid directly sifting through the information in favor of utilizing tools and scripts. These are designed to focus investigators' analytical efforts on artifacts of known significance. Such tools have become invaluable to investigators as a way to improve case turnaround and achieve a more fruitful investigation.

One of these tools is the open source application *Registry Decoder* [2]. A key feature of this tool is the plugin framework. It allows researchers to quickly develop scripts in order to exploit newly found registry artifacts. In this thesis, I will analyze the Windows 8 registry using the principles of differential analysis common to forensics research with the goal of extending the *Registry Decoder* tool for the benefit of the forensics community.

2. Windows Registry Overview

The Windows registry is a hierarchical database split between multiple binary files known as hives. One of these hives, the hardware hive, exists only in volatile memory on the system and is recreated every time the system is started. The "Windows\System32\config" directory contains the hive files SAM, Security, Software, System and Default. For each user on the system, there is an additional ntuser hive "ntuser.dat" located at %UserProfile%. For example, "C:\Users\Jerry\ntuser.dat".

Each of these registry files is allocated on disk in 4KB blocks. The first block is the registry header which contains the signature "regf" as well as modification timestamps and a checksum. This base block is followed by a number of hive bin (HBIN) blocks. Each HBIN also has a

32byte header storing the signature “hbin”, the offset of the block within the file and the size of the block. This organization results in the structures being chained together in a well-defined way.

Each HBIN then stores a series of cells allocated in multiples of 8 bytes. The first four bytes indicate the total length of the cell. The remainder of the cell contains either value data or one of several possible types of records. The possible record types include key records, value records, subkey-lists, value-lists and other types not covered here.

The key record is responsible for maintaining a collection of offsets to locations of other related data structures that may exist in any HBIN within the hive. It can have a reference to another record of whatever type is needed. In addition to pointing to another record, it also maintains a pointer back to its parent key record.

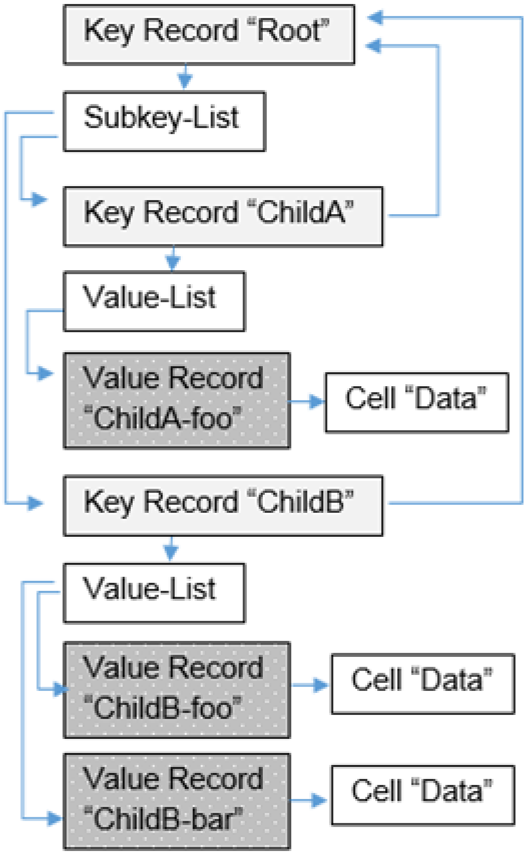


Figure 1: Physical layout of registry. Shows how structures are broken up and referenced.

A value record contains metadata information about the value being stored and then a pointer to the cell where the actual data is stored.

Both subkey-lists and value-lists act as intermediary structures between the parent key record and a collection of child records. Subkey-lists organize additional key records while value-lists contain value records. Another difference is how the list itself is maintained. Subkey-lists are a sorted pair of the child record offset with a hash of the child record’s name. Value-lists are not sorted and do not get hashed.

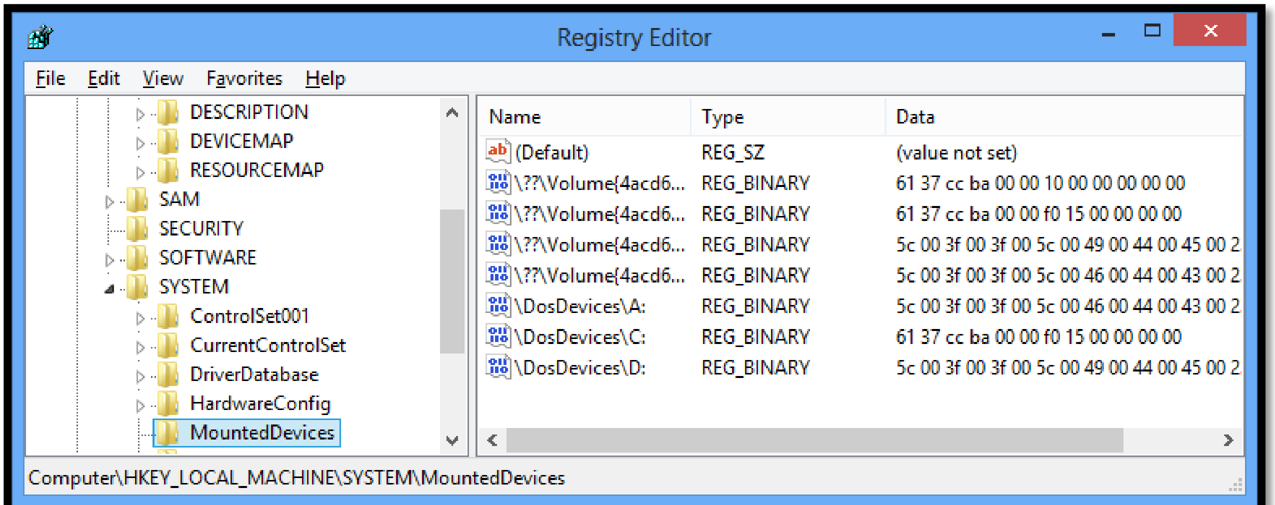
Figure 1 provides a visualization of the overall physical layout of these structures within the file. The key record “Root” points to a subkey-list. This subkey-list tracks the two children of “Root”. Each child of root is pointing to a value-list which tracks all the value records for that child. The value record stores the name of the value and a pointer to the data itself. The children key records also point back to their parent.

Figure 2 shows the logical view of the registry from within the Windows system tool *RegEdit*. Here it can be seen that though the registry is physically separated into several distinct hive files, the operating system treats the structure as one uniform hierarchical database.

3. Related Work

Extensive efforts have been made to analyze the Windows registry as evidenced by the availability of useful investigative tools. In this chapter, I discuss some of the specific contributing efforts to research on the registry. I will also elaborate on developed tools we will utilize in our investigation.

During my research, Amanda Thomson published the *Windows 8 Forensic Guide* [3]. It serves as an excellent guide to many of the artifacts discovered within the Windows 8 file system. It has also served to corroborate some of my own research into the Windows 8 registry. While this



Name	Type	Data
(Default)	REG_SZ	(value not set)
\??\Volume{4acd6...	REG_BINARY	61 37 cc ba 00 00 10 00 00 00 00 00
\??\Volume{4acd6...	REG_BINARY	61 37 cc ba 00 00 f0 15 00 00 00 00
\??\Volume{4acd6...	REG_BINARY	5c 00 3f 00 3f 00 5c 00 49 00 44 00 45 00 2
\??\Volume{4acd6...	REG_BINARY	5c 00 3f 00 3f 00 5c 00 46 00 44 00 43 00 2
\DosDevices\A:	REG_BINARY	5c 00 3f 00 3f 00 5c 00 46 00 44 00 43 00 2
\DosDevices\C:	REG_BINARY	61 37 cc ba 00 00 f0 15 00 00 00 00
\DosDevices\D:	REG_BINARY	5c 00 3f 00 3f 00 5c 00 49 00 44 00 45 00 2

Computer\HKEY_LOCAL_MACHINE\SYSTEM\MountedDevices

Figure 2: Logical view of registry in *RegEdit*. It appears much like a typical directory tree.

work is useful for the investigation of known artifacts, it lacks in describing techniques useful for the discovery of new artifacts.

Simson Garfinkel's work A *general strategy for differential forensic analysis* helps to formalize a common approach to forensic research and analysis [4]. His work describes the cornerstone of the process I employ for this research, differential analysis. The comparison of two sets of related data from different points in time. One of the sets acts as a baseline against which a researcher can compare the other derivative sets. The results of this comparison can be analyzed to garner a better understanding of the data or provide a more filtered dataset for a more concise starting point for analysis. This strategy can help identify the effect of specific system activity on the state of the registry.

Much of the work presented in Garfinkel's paper is focused on establishing the series of operations required to transition the state of one dataset to that of another. In my work of distinguishing features unique to Win8, there are no meaningful operations to be had when diffing registry hives from Win7 to those of Win8. Instead, it proves to be more useful to examine the set-theoretic difference of Win8 and Win7. That is, the set of keys and values found to be exclusive to the Win8 registry hives.

Another researcher active in the field of Windows registry analysis is Harlan Carvey. Carvey's book *Windows Registry Forensics* [5] has been a helpful aid in understanding the process of analyzing the registry and the meaningful interpretation of artifacts. He discusses the forensic value of the Windows registry and gives an overview of how to access and navigate the data stored within it. Similar to Thomson's work, this book has a focus on artifacts and their meaning while lacking discussion of how an investigator should go about discovering new artifacts of value in the registry.

3.1 Registry Decoder

Registry Decoder[2] is an extensible Windows registry acquisition and analysis tool. Acquisition is handled by the *Registry Decoder Live* component which is capable of acquiring all registry hives from a machine. This includes volatile hives which are located in the system's memory and historic hives preserved by Windows backup functionality. The *Registry Decoder Offline* component is then able to parse the collected data and operate on it with a collection of powerful plugins. These plugins are useful for automating the discovery and reporting of many relevant registry artifacts.

3.2 *RegShot*

RegShot[6] is an open source analysis tool that allows an examiner to capture and compare the state of the registry at two different points in time. These snapshots are then diffed against one another with the result being an extensive report on the changes made in the time between the two snapshots.

3.3 *ProcessMonitor*

ProcessMonitor[7] is a comprehensive tool suite that provides monitoring capabilities for a wide range of system activities. Proper configuration of activity filters allows an investigator to narrow the tool's focus to subsets of system activity such as registry reads or modifications.

3.4 *find_times.py*

The Python script *find_times.py*[8] is a tool in active development by Andrew Case. It is designed to discover the presence of potential Windows timestamps within the registry. This tool is designed to aid in the discovery of previously unknown artifacts. These discovered artifacts should prove particularly useful for the development of timelines.

3.5 *Yet Another Registry Utility*

Yet Another Registry Utility (YARU) [9] is a registry viewer application. Its advanced features include the ability to show the contents of unused space within the registry, known as slack space. It can also recover deleted registry keys and access hives from a live machine.

4. Experiment Designs

In my research, I focused on finding new artifacts of interest in the Windows 8 registry. To do so, I utilized the registry analysis tools listed previously in Chapter 2 and the differential analysis process as described by Garfinkel [4]. Utilizing virtualization software, I began by creating a virtual machine using the Windows 8 consumer preview as provided by Microsoft in Feb 2012. The process described was again later performed with the commercial release of Windows 8 professional edition.

After the operating system's installation, I used the snapshot functionality of the virtual machine to create a clean restore point to return to if needed. I then configured a single user for the system and installed each of my forensics tools. I acquired the active registry hives for the system using *Registry Decoder Live* tool and again took a snapshot of the VM. This snapshot and registry data would serve as the primary baseline for subsequent system activity as I investigated artifacts indicative of usage.

I then generated activity on the machine typical of general usage. I installed and utilized a collection of applications including Metro UI ones. These included the popular gaming software Steam, the video service Youtube, the archive manager 7-zip, the game Angry Birds, the media application IHeartRadio and Microsoft Office. During my activity, I would regularly take new snapshots describing the activity I had performed in the elapsed time since the previous entry. Activity included browsing articles on news.ycombinator.com via Internet Explorer, The download and play of games by Steam, watching videos by Youtube and communications by Skype. After generating usage history for these applications, I uninstalled the applications Steam and IHeartRadio with the intent to investigate the deletion of registry keys. I now had a collection of snapshots and registry acquisitions taken after series of any major events such as internet usage or application uninstalls.

With this data I ran *Registry Decoder* on the acquired registry hives. This allowed me to examine the state of the Windows 7 compatible plugins that ship with *Registry Decoder*. Each plugin performed well resulting in generated reports containing the data akin to that as if the plugins were run on a Windows 7 machine. None appeared to fail due to not having been designed for Windows 8. From here I began to explore the differences between the registry hives collected throughout the data generation.

I began to investigate methods to reduce the number of keys and values in my dataset to something that could be manually parsed for interesting artifacts. I conceived that I could likely eliminate a portion of data from consideration if those key and value paths were already present within Windows 7. I found it beneficial to author python scripts to perform diffing between matching pairs of registry hives. The purpose was to quickly utilize registry hives for Windows 7 and compare the key record and value record paths to those found in the acquired Windows 8 hives. My *diffReg.py* script produced a list of keys and values that only existed in the Windows 8 registry files and not in the Windows 7 ones. While I did not expect perfect results which would reduce the examined data to only entries within Windows 8, I did succeed in greatly filtering the data.

Using my script, I was able to reduce the total number of key and value paths in my acquisition, by a notable amount in some cases. Of 110,462 entries in the System hive, I was able to eliminate 11.7%. Most impressively, an original set of 696,311 entries in the Software hive was reduced by 41.9%. The highest percent reduction was 79.5% from a set of 229 entries in the Security hive. **Figure 3** shows he results for the hives analyzed as the total of the set-theoretic difference and intersection of the Win8 and Win7 data. While these initial attempts at reducing

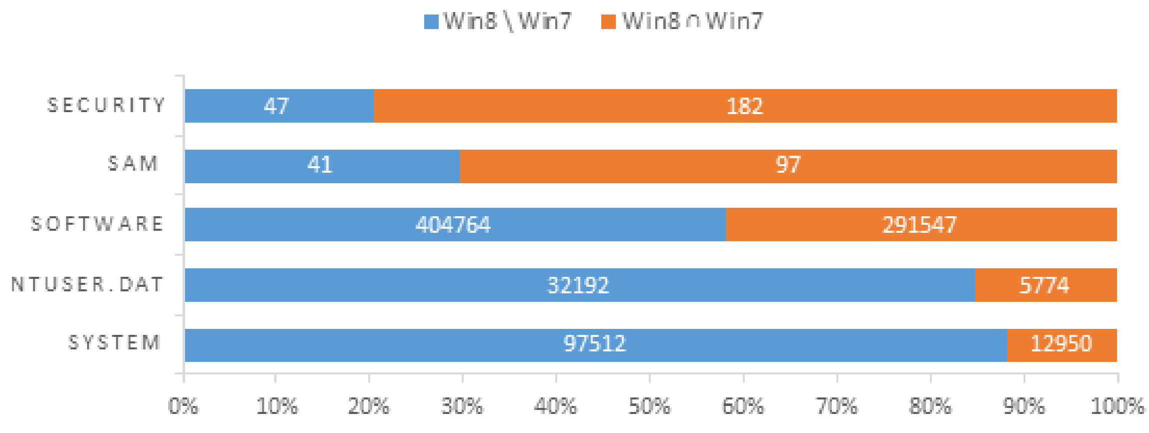
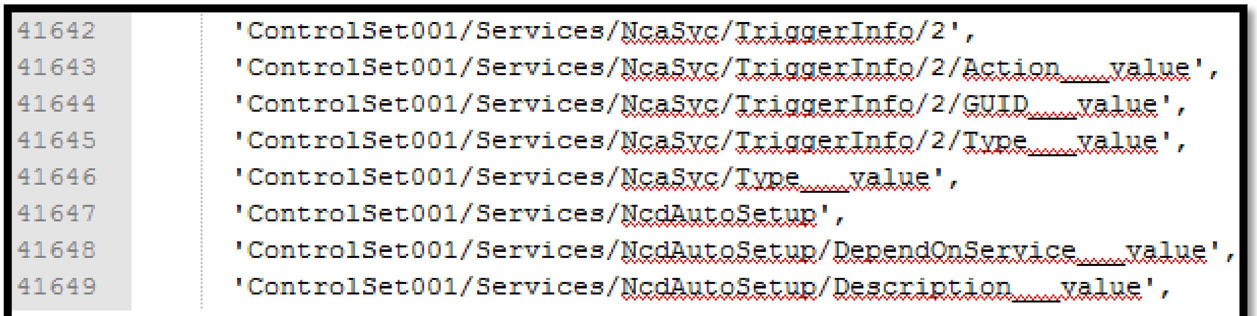


Figure 3: Win8 dataset divided into set difference and set intersection with Win7 data.

the common entries between Windows 7 and Windows 8 machines demonstrated success, further work on this approach with a larger sample of data could prove to be a fruitful way to quickly identify artifacts exclusive to a given version of the Windows operating system.

The script *diffReg.py* used the python module *pyregf* to recursively iterate through all the entries in the registry. It then generated a unique path value for each entry based on the name of the entry and its parent entry. Next it performed the difference set operation with the set of entries from the second registry file. The result was a set of entries exclusive to the first registry hive. The contents of the set were printed in a sorted order so that visual inspection could easily differentiate between path branches within the registry. Any potentially interesting entries were then researched using the *YARU* tool to navigate the corresponding registry hives. This was particularly effective at identifying where an entire branch of keys and values were unique to the Windows 8 registry. **Figure 4** shows sample results from the *diffReg.py* script. NcaSvc and NcdAutoSetup are both services that were introduced in Windows 8.

In order to track activity on a finer scale I utilized the tools *RegShot* and *ProcessMonitor*. I restored the virtual machine to various states of interest and used *RegShot* to more quickly iterate through the process of acquiring data and diffing it with a previous state. As I worked, *ProcessMonitor* was configured to focus on registry activity occurring from specific processes. This allowed me to monitor live registry activity as I interacted with Windows 8. This approach was particularly useful when experimenting with new features to the Windows 8 operating system such as new alternative methods of user logon authentication. After these results I turned my efforts to discovering previously unknown keys in Windows 8 using additional python tools.



```
'ControlSet001/Services/NcaSvc/TriggerInfo/2',
'ControlSet001/Services/NcaSvc/TriggerInfo/2/Action___value',
'ControlSet001/Services/NcaSvc/TriggerInfo/2/GUID___value',
'ControlSet001/Services/NcaSvc/TriggerInfo/2/Type___value',
'ControlSet001/Services/NcaSvc/Type___value',
'ControlSet001/Services/NcdAutoSetup',
'ControlSet001/Services/NcdAutoSetup/DependOnService___value',
'ControlSet001/Services/NcdAutoSetup/Description___value',
```

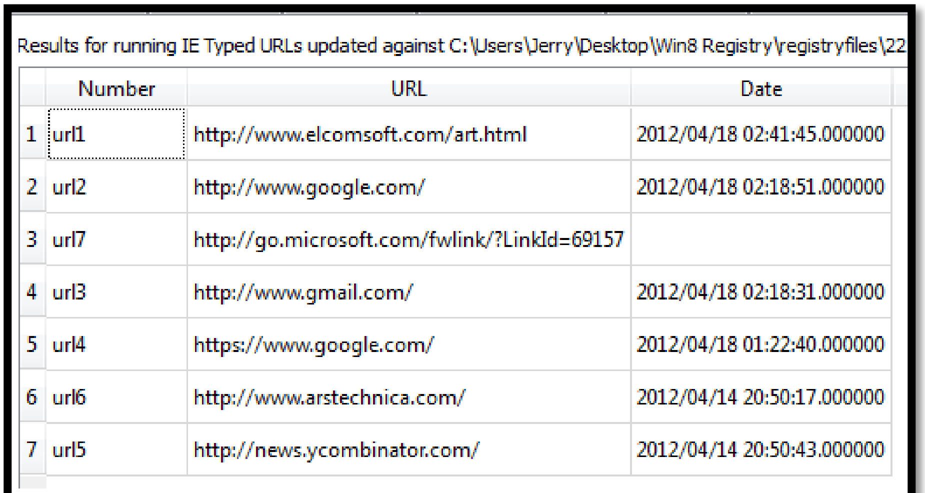
Figure 4: Sample *diffReg.py* results. These keys and values exist only in the Win8 data.

I utilized a new analysis utility in development by Andrew Case currently known as *find_times.py*. This tool identified all potential timestamps within each of my hive files, generating several promising leads to investigate further. Special attention was given to results generated with valid timestamps falling within the duration of the data generation.

Each of the methods I employed proved useful to my investigation in some way. The *diffReg.py* script reduced the amount of data that I had to analyze. The *find_times.py* script drew attention to particular registry keys that might aid in timelining efforts. The use of *ProcessMonitor* helped develop my understanding of how specific registry keys were being used by the operating system.

5. Experimental Results

The first relevant artifact I discovered was the presence of a new key, TypedURLsTime. This key is located alongside the already popular TypedURLs key in the ntuser hive of each user on the system. The path to the key is “ntuser.dat\Software\Microsoft\Internet Explorer\TypedURLsTime.” The key TypedURLs is an artifact which contains a listing of the 25 most recent URLs that have been typed into the Internet Explorer address bar. It can be an exceptionally useful key in an investigation since it shows that a website has been visited by a specific user account on the computer. More so, the existence of a URL in this key shows intent



Results for running IE Typed URLs updated against C:\Users\Jerry\Desktop\Win8 Registry\registryfiles\22

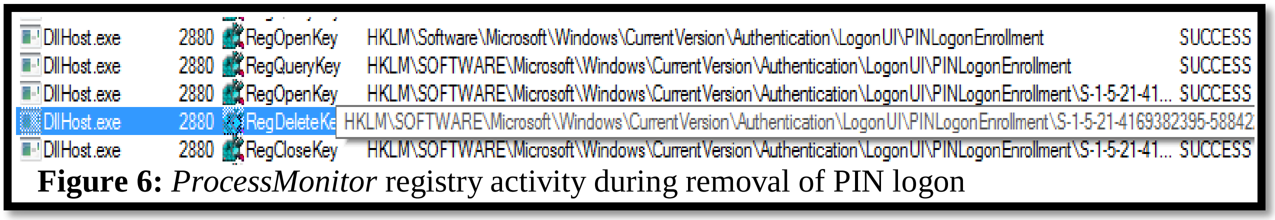
	Number	URL	Date
1	url1	http://www.elcomsoft.com/art.html	2012/04/18 02:41:45.000000
2	url2	http://www.google.com/	2012/04/18 02:18:51.000000
3	url7	http://go.microsoft.com/fwlink/?LinkId=69157	
4	url3	http://www.gmail.com/	2012/04/18 02:18:31.000000
5	url4	https://www.google.com/	2012/04/18 01:22:40.000000
6	url6	http://www.arstechnica.com/	2012/04/14 20:50:17.000000
7	url5	http://news.ycombinator.com/	2012/04/14 20:50:43.000000

Figure 5: *RegistryDecoder* TypedURLs plugin report

to visit a website as the address was actually typed into the navigation bar of Internet Explorer. The new TypedURLsTime key provides additional information to an investigator about the Internet usage of a user.

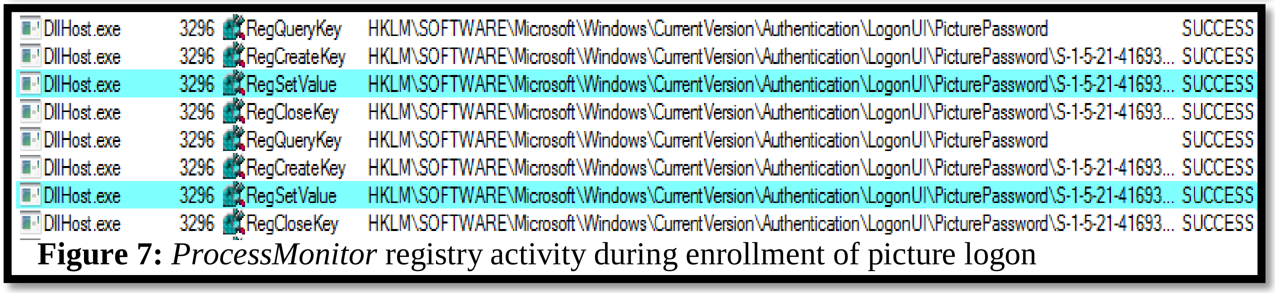
The presence of the TypedURLsTime key is of great forensic use as it can be used as an aid in timelining the activity of a user. This timestamp adds one more possible indicator of evidence tampering as well since the timestamp values here can be corroborated with other gathered data. **Figure 5** shows the report generated by the *Registry Decoder* plugin I developed. It is also worth noting that the number of saved URL entries has increased from 25 TypedURLs in Windows 7 to 50 in Windows 8.

Another set of artifacts I discovered pertains to the new authentication features of Windows 8. These are the PIN and Picture alternative password features. Windows 8 provides users with the opportunity to enroll in alternative login methods. A user can enroll in the PIN authentication feature in order to quickly log in to their account from the start screen. With PIN login, the user creates a four digit PIN number that can be used as an alternative to signing into their account. This enrollment is recorded in "Software\Microsoft\Windows\CurrentVersion\Authentication \LogonUI\PINLogonEnrollment." A key entry with the user's Security Identifier (SID) will exist for each user enrolled. **Figure 6** shows how *ProcessMonitor* recorded the system activity when a user unenrolled from the PIN login feature. In this case, their key is being removed from the registry by the process DllHost.exe. Similarly, the user may enroll in the Picture Password feature of Windows 8.



```
DllHost.exe   2880  RegOpenKey    HKLM\Software\Microsoft\Windows\CurrentVersion\Authentication\LogonUI\PINLogonEnrollment                 SUCCESS
DllHost.exe   2880  RegQueryKey   HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI\PINLogonEnrollment                 SUCCESS
DllHost.exe   2880  RegOpenKey    HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI\PINLogonEnrollment\S-1-5-21-41...  SUCCESS
DllHost.exe   2880  RegDeleteKe HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI\PINLogonEnrollment\S-1-5-21-4169382395-58842
DllHost.exe   2880  RegCloseKey   HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI\PINLogonEnrollment\S-1-5-21-41...  SUCCESS
```

Figure 6: *ProcessMonitor* registry activity during removal of PIN logon



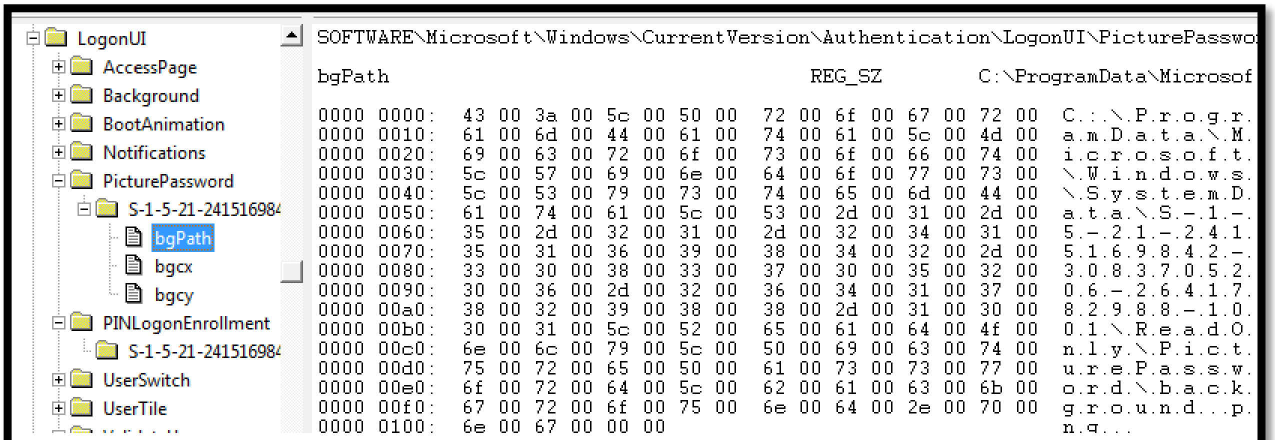
```
DllHost.exe   3296  RegQueryKey   HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI\PicturePassword                  SUCCESS
DllHost.exe   3296  RegCreateKey  HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI\PicturePassword\S-1-5-21-41693...  SUCCESS
DllHost.exe   3296  RegSetValue   HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI\PicturePassword\S-1-5-21-41693...  SUCCESS
DllHost.exe   3296  RegCloseKey   HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI\PicturePassword\S-1-5-21-41693...  SUCCESS
DllHost.exe   3296  RegQueryKey   HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI\PicturePassword                  SUCCESS
DllHost.exe   3296  RegCreateKey  HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI\PicturePassword\S-1-5-21-41693...  SUCCESS
DllHost.exe   3296  RegSetValue   HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI\PicturePassword\S-1-5-21-41693...  SUCCESS
DllHost.exe   3296  RegCloseKey   HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI\PicturePassword\S-1-5-21-41693...  SUCCESS
```

Figure 7: *ProcessMonitor* registry activity during enrollment of picture logon

Enrollment in the Picture Password feature of Windows 8 works in very much the same way. The difference is that in order to login with a picture, the user first selects an image. This image is saved to a Windows system folder associated with the user's account. In addition, the user is asked to make three gestures on the image with the mouse. Examples would be drawing shapes on the picture or simply clicking on it. Now when the user requests to sign in with the Picture Password, they are presented with the image they selected. They then have to interact with it in the designed way in order to authenticate. **Figure 7** shows the registry activity through *ProcessMonitor* as the user signs up for the Picture Password feature.

Both of these are useful to a forensics investigator in a few ways. Each of these artifacts shows an alternative method to access the system which might be exploitable. However, the implications of the Picture Password are more interesting. Consider the case that a contraband image is chosen as the Picture Password. This case shows intent to interact with the image chosen since the image had to have been intentionally selected during the enrollment procedure. Additionally, the image had to have been interacted with in order to define the gestures that would allow for a successful logon. **Figure 8** shows both artifacts within the registry and the values associated with the Picture Password.

Metro UI Apps are a new type of application within Windows 8. The Metro UI itself is the title given to the unique redesign of the Windows interface. The prominent feature in the Metro UI is the reliance on a home screen where the user may access frequently used applications organized as tiles. This feature tries to focus the operating system away from the classic desktop interface



```
SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI\PicturePasswo

bgPath                                  REG_SZ       C:\ProgramData\Microsof

0000 0000:   43 00 3a 00 5c 00 50 00   72 00 6f 00 67 00 72 00   C.:.\.P.r.o.g.r.
0000 0010:   61 00 6d 00 44 00 61 00   74 00 61 00 5c 00 4d 00   a.m.D.a.t.a.\.M.
0000 0020:   69 00 63 00 72 00 6f 00   73 00 6f 00 66 00 74 00   i.c.r.o.s.o.f.t.
0000 0030:   5c 00 57 00 69 00 6e 00   64 00 6f 00 77 00 73 00   \.W.i.n.d.o.w.s.
0000 0040:   5c 00 53 00 79 00 73 00   74 00 65 00 6d 00 44 00   \.S.y.s.t.e.m.D.
0000 0050:   61 00 74 00 61 00 5c 00   53 00 2d 00 31 00 2d 00   a.t.a.\.S.-.1.-.
0000 0060:   35 00 2d 00 32 00 31 00   2d 00 32 00 34 00 31 00   5.-.2.1.-.2.4.1.
0000 0070:   35 00 31 00 36 00 39 00   38 00 34 00 32 00 2d 00   5.1.6.9.8.4.2.-.
0000 0080:   33 00 30 00 38 00 33 00   37 00 30 00 35 00 32 00   3.0.8.3.7.0.5.2.
0000 0090:   30 00 36 00 2d 00 32 00   36 00 34 00 31 00 37 00   0.6.-.2.6.4.1.7.
0000 00a0:   38 00 32 00 39 00 38 00   38 00 2d 00 31 00 30 00   8.2.9.8.8.-.1.0.
0000 00b0:   30 00 31 00 5c 00 52 00   65 00 61 00 64 00 4f 00   0.1.\.R.e.a.d.O.
0000 00c0:   6e 00 6c 00 79 00 5c 00   50 00 69 00 63 00 74 00   n.l.y.\.P.i.c.t.
0000 00d0:   75 00 72 00 65 00 50 00   61 00 73 00 73 00 77 00   u.r.e.P.a.s.s.w.
0000 00e0:   6f 00 72 00 64 00 5c 00   62 00 61 00 63 00 6b 00   o.r.d.\.b.a.c.k.
0000 00f0:   67 00 72 00 6f 00 75 00   6e 00 64 00 2e 00 70 00   g.r.o.u.n.d...p.
0000 0100:   6e 00 67 00 00 00                                   n.g...
```

Figure 8: *YARU* view of picture login keys. Value is system path to the image used.

and instead focus on the programs themselves. These programs that make use of the interface redesign are known as Metro UI applications. They are installed on a per user basis with those installations appearing in the "Software\Microsoft\Windows\ CurrentVersion\Appx\AppxAllUserStore" key. Here a list of all installed MetroUI applications can be found under the key "Applications." They are also listed on a by user basis under a key using the user's SID. This information is potentially useful to an investigator, as installed applications are commonly important to a forensics investigation. It is important that an investigator not miss this additional information on the programs installed and used by each user.

An interesting feature of Windows 8 is its integration with the Microsoft Live platform and Windows Cloud services. Due to these features, a user account is now intimately linked with a Microsoft Live account. Several of the features of the Windows 8 operating system, such as the Windows Store, require a Live account to function. There are now additional artifacts of interest within a user's keys in the Security Accounts Manager (SAM) hive. This hive stores the login information for a user. The user's passwords are stored in a hashed format within this hive. New artifacts here include the Microsoft Live account tied to the user account as well as the user's cached, encrypted account credentials for their Live account. This is because Windows 8 can be configured to explicitly use the Microsoft Live credentials to login to the system. In the case where there is no Internet connectivity, the cached credentials are used for authentication.

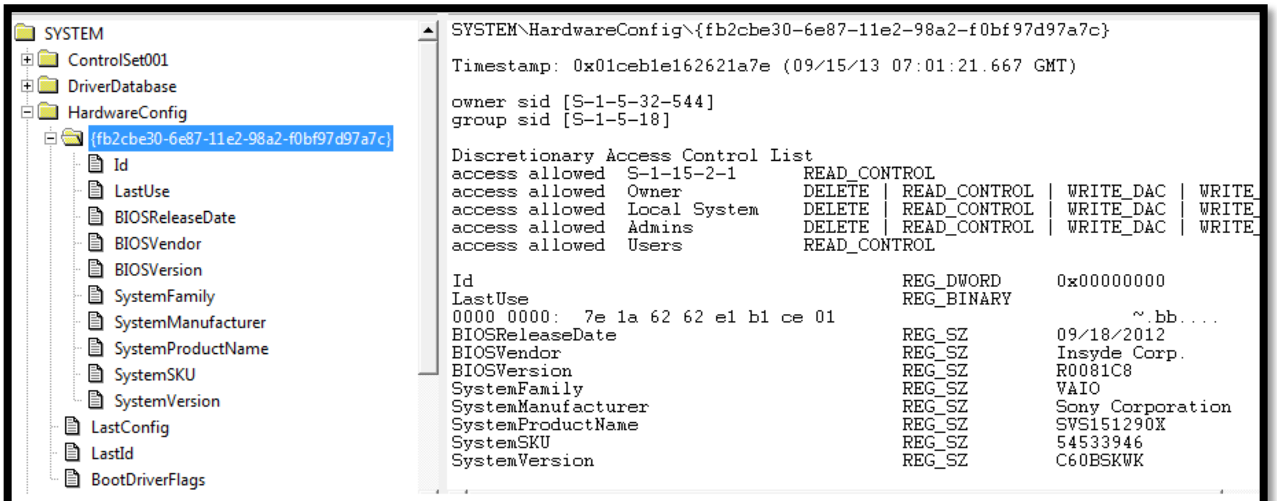
Next is the distinct result of the *diffReg* python script I developed to compare the registry keys within a typical Windows 7 install to those in a Windows 8 system. I was able to more easily recognize new branches of keys within the registry. This called my attention to one I had previously overlooked. I found that Windows 8 now stores data about the BIOS installed on the system.

This information is stored in the System hive at the key path "System\HardwareConfig." Using this, an investigator can determine the version information of the BIOS installed and the last time the computer was turned on. **Figure 9** shows this information as viewed in the *Yaru* registry tool.

An interesting artifact discovered within the DEFAULT hive is the presence of detailed information on access points the system has been connected to. “DEFAULT\Software\Intel\Wireless\ConnectionHistory” contains several keys entitled “NDP0” and “NDP1”. Each of these keys contain detailed information on a wireless network the machine had connected to. The SSID, channel information and authentication method serve to collectively help identify the specific access point. . There is an additional timestamp associated with this key which reveals the most recent time the machine was connected to each wireless network. This information could prove exceptionally useful to an investigator in order to determine what network particular activity occurred on or physical presence within a range of the access point at a time.

With the aid of the new *find_times* utility in development by Andrew Case, I was able to discover quite a few potentially interesting artifacts within the registry. Many of these artifacts proved to be well known sources of information within the registry. A few others may prove fruitful after further analysis. Some of the notable ones were the presence of additional keys that seemed to be providing information on most recently used applications.

Several of these such keys related to an install of Microsoft Office 2013. The root key for this information is located at “ntuser.dat\Software\Microsoft\Office\15.0”. For each Office application, there is a key here with its very own set of most recently used documents and



```
SYSTEM\HardwareConfig\{fb2cbe30-6e87-11e2-98a2-f0bf97d97a7c}

Timestamp: 0x01ceb1e162621a7e (09/15/13 07:01:21.667 GMT)

owner sid [S-1-5-32-544]
group sid [S-1-5-18]

Discretionary Access Control List
access allowed  S-1-15-2-1      READ_CONTROL
access allowed  Owner           DELETE | READ_CONTROL | WRITE_DAC | WRITE_
access allowed  Local System    DELETE | READ_CONTROL | WRITE_DAC | WRITE_
access allowed  Admins          DELETE | READ_CONTROL | WRITE_DAC | WRITE_
access allowed  Users           READ_CONTROL

Id                                          REG_DWORD     0x00000000
LastUse                                     REG_BINARY
0000 0000:  7e 1a 62 62 e1 b1 ce 01                         ~.bb....
BIOSReleaseDate                             REG_SZ        09/18/2012
BIOSVendor                                  REG_SZ        Insyde Corp.
BIOSVersion                                 REG_SZ        R0081C8
SystemFamily                                REG_SZ        VAIO
SystemManufacturer                          REG_SZ        Sony Corporation
SystemProductName                           REG_SZ        SVS151290X
SystemSKU                                   REG_SZ        54533946
SystemVersion                               REG_SZ        C60BSKWK
```

Figure 9: *YARU* view of BIOS information

folders. As an example, “Word\User MRU\LiveId_...\ File MRU” shows that the user worked with the document “C:\Users\Jake\Downloads\dogflier.doc”. In addition, the entries under “Word\Security\Trusted Documents\ TrustRecords” provide a record of files that the user agreed to a Microsoft Office security warning in order to open. These trust records can help track user activity which could have enabled a malicious file to run.

Other recently used documents were being listed by the 3D modeling application *SketchUp.* The usefulness of this technique surely deserves more analysis and I will be certain to continue my efforts with Andrew Case.

In an effort to immediately benefit the forensics community, I developed plugins for each of my findings for *Registry Decoder*. Developing a plugin means the creation of a python module with particular values defined. In addition to defining the plugin name, description and targeted hive, the method run_me( ) provides the entry point for the plugin. This method will commonly start with a call to the *Registry Decoder* API call, reg_get_required_key(string). This function takes a string for its parameter which specifies a registry key path that is required for the use of the plugin. An example may be "\HardwareConfig” for the plugin *biosinfo.py*. The function then either fails to find the key in which case the plugin stops running or the key record object is returned. The key record object can then have its sub-keys and values traversed and used for analysis. Hence, the required key is typical going to be the parent to the sub-keys and values to be parsed. Many other useful methods exist within the API defined in the *util.py* module of the *Registry Decoder* source code. For example, the API method reg_set_report_header(tuple) which will give the investigator a more complete and better defined report. Using these methods provided in the API provides a researcher with direct access to navigate the registry structure. This allows for easy development of plugins to parse, correlate and present data in a clean way. To present the data, a call to the report(tuple) function will print the results of any analysis within the tabbed report generated in the GUI. In order to load custom plugins into *Registry Decoder,* it can be run with the command line argument “-d <PATH>” where PATH is an additional plugin directory to load.

6. Conclusion

The vast majority of the Windows 8 registry remains unchanged. The major forensic artifacts of note from the Windows 7 operating system still exist within Windows 8. Additionally, the design decisions made with Windows 8 introduced new artifacts relevant to forensic efforts.

This work expanded on research into Microsoft's newest flagship operating system, Windows 8. My efforts resulted in the discovery of several new artifacts including ones useful for the fingerprinting of the system's BIOS, alternative authentication services, timelining of Internet Explorer activity, wireless network usage and the discovery of additional sources of recently used history.

In order to aid in analysis of registry keys existing exclusively on one system, I created the *diffReg.py* script. Working from the process of differential analysis, this tool proved exceptionally useful in filtering through an abundance of data to focus efforts on a workload more manageable for manual analysis. It is my hope that further development work on this script may even serve as a beginning to a new registry analysis utility.

For each of the artifacts I discovered, I developed new plugins for the *Registry Decoder* utility which make my research immediately useful to investigators. These plugins perform automated analysis on Windows registry hives and present information on each of the artifacts I've discussed in the form of reports.

TypedURLS.py presents the timestamping information now available within Windows 8 alongside the analysis of Internet Explorer typedURLs. This aids an investigator in determining Internet usage history.

BIOS.py provides all the fingerprinting information for the system BIOS including the last boot time.

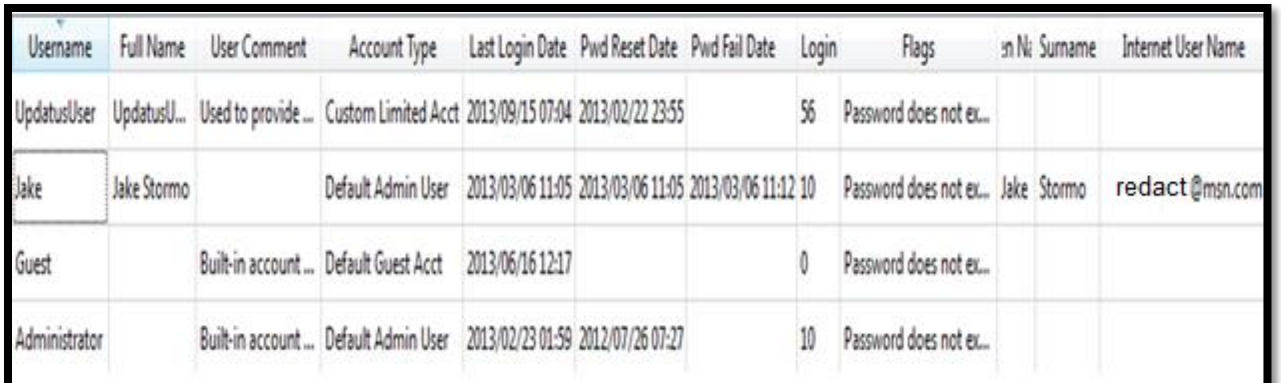
AlternativeLogin.py reports on user enrollment in the Windows 8 photograph and PIN login authentication features. Further details such as the path to the image used for authentication is also provided.

Reporting on the email address used to sign up for a Microsoft Live account and a user's full name is performed in the *UserInfo.py* plugin. Additionally this plugin provides *RegistryDecoder* the ability to parse many other well-known artifacts from the SAM hive. These artifacts include login timestamps, number of times a user has logged in and flags associated with their account. **Figure 10** shows a report generated by the *UserInfo* plugin.

WirelessAP.py reports the information available for wireless access points that have been connected to. This information includes the last time the device was authenticated with, the SSID of the device and information about its operation including the wireless channel and security protocols used for authentication.

Office2013.py parses all of the MRU lists from each of the Microsoft Office products. Each product provides the investigator with its own unique list of documents recently opened by the application.

It is my hope that the utilities I've developed and the artifacts discovered will aid the forensics community as a whole.



Username	Full Name	User Comment	Account Type	Last Login Date	Pwd Reset Date	Pwd Fail Date	Login	Flags	n N:	Surname	Internet User Name
UpdatusUser	UpdatusU...	Used to provide ...	Custom Limited Acct	2013/09/15 07:04	2013/02/22 23:55		56	Password does not ex...			
Jake	Jake Stormo		Default Admin User	2013/03/06 11:05	2013/03/06 11:05	2013/03/06 11:12	10	Password does not ex...	Jake	Stormo	redact@msn.com
Guest		Built-in account ...	Default Guest Acct	2013/06/16 12:17			0	Password does not ex...			
Administrator		Built-in account ...	Default Admin User	2013/02/23 01:59	2012/07/26 07:27		10	Password does not ex...			

Figure 10: *RegistryDecoder* UserInfo plugin report

7. References

[1] w3schools. (2013). OS Platform Statistics and Trends [online] http://www.w3schools.com/browsers/browsers_os.asp

[2] Case, A., & Marziale, V. (2011). Registry Decoder [online] https://code.google.com/p/registrydecoder/

[3] Thomson, A. (2012). Windows 8 Forensic Guide [online] http://propellerheadforensics.files.wordpress.com/2012/05/thomson_windows-8-forensic-guide2.pdf

[4] Garfinkel, S., et al. (2012). A general strategy for differential forensic analysis [online] http://dfrws.org/2012/proceedings/DFRWS2012-6.pdf

[5] Carvey, H. (2011). *Windows Registry Forensics.* Syngress, Massachusetts

[6] Buecher, M, et al. (2008). RegShot [online] http://sourceforge.net/projects/regshot/

[7] Russinovich, M., & Cogswell, B. (2013). Process Monitor v3.05 [online] http://technet.microsoft.com/en-us/sysinternals/bb896645

[8] Case, A. (2013). find_times.py

[9] TZWorks LLC, (2011) YARU [online] https://www.tzworks.net/prototype_page.php?proto_id=3

[10]Morgan, T. (2009). The Windows NT Registry File Format [online] http://sentinelchicken.com/data/TheWindowsNTRegistryFileFormat.pdf

[11]Alghafli, K., et al. (2010). Forensic Analysis of the Windows 7 Registry [online] http://ro.ecu.edu.au/cgi/viewcontent.cgi?article=1071&context=adf

[12]Carvey, H. (2008). Registry Ripper [online] http://regripper.wordpress.com/

[13]Thurrot, P. (2011). Windows 8 and the Registry. [online] http://winsupersite.com/blog/supersite-blog-39/windows8/windows-8-registry-140910

[14]Python 2.7 reference manual [online] http://docs.python.org/2/library/

VITA

Jeremy Stormo is a native to Metairie, LA having been born and raised there. In 2007, he graduated from Louisiana State University in Baton Rouge with a Bachelor of Science in Computer Science. In 2011, Jeremy entered the University of New Orleans Computer Science graduate program in pursuit of a M.S. with a concentration in Information Assurance.