Malware memory analysis for non-specialists

Investigating publicly available memory image 0zapftis (R2D2)

R. Carbone
Certified Forensic Hacking Investigator (EC-Council)
Certified Incident Handler (SANS)
DRDC Valcartier

Defence Research and Development Canada – Valcartier

Technical Memorandum
DRDC Valcartier TM 2013-177
October 2013

Abstract

This technical memorandum examines how an investigator can analyse an infected Windows memory dump. The author investigates how to carry out such an analysis using Volatility and other investigative tools, including data carving utilities and anti-virus scanners. Volatility is a popular and evolving open source-based memory analysis framework upon which the author has proposed a memory-specific methodology for aiding fellow novice memory analysts. The author examines how Volatility can be used to find evidence and indicators of infection. This technical memorandum is the third in a series concerning Windows malware-based memory analysis. This current work examines the 0zapftis (R2D2) infected memory image.

Résumé

Ce mémorandum technique examine comment un investigateur peut analyser une image mémoire d'une machine Windows infectée. L'auteur investigue les techniques d'analyse utilisant Volatility et d'autres outils tels que les utilitaires de récupération de données et les scanneurs anti-virus. Volatility est un cadre populaire d'analyse de mémoire en source libre sur lequel l'auteur s'appuie pour proposer une méthodologie spécifique à la mémoire pour aider ses collègues analystes novices. L'auteur examine comment Volatility peut être utilisé pour trouver des preuves ou des indicateurs d'infection. Ce mémorandum technique est le troisième d'une série visant la découverte de maliciel par le biais d'une analyse de la mémoire. Le présent travail examine l'image de mémoire infectée par 0zapftis (R2D2).

This page intentionally left blank.

Executive summary

Malware memory analysis for non-specialists: Investigating publicly available memory image 0zapftis (R2D2)

Carbone, R.; DRDC Valcartier TM 2013-177; Defence Research and Development Canada – Valcartier; October 2013.

While memory analysis has largely been carried out by software reverse engineers and malware analysts, the advent of memory analysis-based forensic frameworks such as Volatility has made it possible for non-memory specialists to engage in the forensic analysis of malware-infected memory images. By combining Volatility, data carving utilities and anti-virus scanners, novice analysts have all the necessary tools required for conducting memory-based investigations.

The author's primary objective is to demonstrate through tutorials how investigators can conduct meaningful memory-based investigations on their own.

This technical memorandum examines the 0zapftis (R2D2) Trojan horse, in order to build a compendium of tutorials that can be used by the Canadian Armed Forces and our partners as a basis for conducting their own investigations. This work is the third in a series that examines various Windows-based malware infected memory images. The two previous reports in this series, TM 2013-018 and TM 2013-155, examined the Zeus Trojan horse (the former) while the latter examined the Prolaco worm and SpyEye Trojan horse. It is hoped that these documents will serve as a learning guide.

Although others have engaged in the analysis of some of these publicly available memory images, the author is of the opinion that these analyses are insufficient for use as a learning guide. Specifically, these analyses are either too limited in their investigative scope or report too little information to be of use to budding memory analysts. Moreover, many of the analyses leave the reader asking more questions than when he began, due to their overall in using a comprehensive investigative context. Thus, the author has strived to ensure that his investigative actions and lines of inquiry were well documented, even if some portions of a given investigation were unsuccessful, in order to ensure that the investigative context used was coherent.

This work was carried out over a period of several months as part of the Live Computer Forensics project, an agreement between DRDC Valcartier and the RCMP (SRE-09-015, 31XF20).

The results of this project will also be of great interest to the Canadian Forces Network Operations Centre (CFNOC), the RCMP's Integrated Technological Crime Unit (ITCU), the Sûreté du Québec and other cyber investigation teams.

Sommaire

Malware memory analysis for non-specialists: Investigating publicly available memory image 0zapftis (R2D2)

Carbone, R. ; DRDC Valcartier TM 2013-177 ; Recherche et développement pour la défense Canada – Valcartier; octobre 2013.

Bien que l'analyse de la mémoire ait été principalement effectuée jusqu'à présent par les rétro-ingénieurs logiciels et les analystes de maliciel, les avancées des cadres d'analyse de la mémoire, tel que Volatility, permettent maintenant aux non-spécialistes de la mémoire d'effectuer des analyses d'image mémoire de machines infectées par des maliciels. En combinant Volatility, les outils de récupération de données et les scanneurs anti-virus, les analystes novices possèdent tous les outils requis pour investiguer une image mémoire.

L'objectif premier de l'auteur est de démontrer, par le biais d'un tutoriel, comment un investigateur peut réaliser une analyse de la mémoire par lui-même.

Ce mémorandum technique examine l'image de mémoire de 0zapftis (R2D2), pour monter un ensemble de tutoriels qui pourront être utilisés par les Forces Armées canadiennes et nos partenaires pour faire leurs propres investigations. Ce travail est le troisième d'une série visant la découverte de maliciel par le biais d'une analyse de la mémoire d'une machine Windows infectée. Les deux premiers rapports de cette série, TM 2013-018 et le TM 2013-155, examinait le cheval de Troie Zeus (premier rapport) et le verre de Prolaco et le cheval de Troie SpyEye (deuxième rapport). Nous espérons que ces documents serviront de guide d'apprentissage.

Bien que d'autres aient mentionné avoir effectué l'analyse de ces images mémoires publiques, l'auteur croit que ces analyses ne sont pas assez détaillées pour servir de guide d'apprentissage. Spécifiquement, ces analyses sont soit trop limitées dans ce qu'elle investigue ou ne donnent pas assez de détails pour être complètement utiles. De plus, plusieurs de ces analyses font que le lecteur a, en bout de ligne, plus de questions que de réponses étant donné le peu de détails approfondis sur le contexte de l'investigation. L'auteur a donc déployé tous les efforts pour s'assurer que toutes les actions et les champs d'enquête sont bien documentés et cohérents dans le contexte, même si certains essais étaient infructueux.

Ce travail fut réalisé sur une période de plusieurs mois dans le cadre du projet "Live Computer Forensics", qui est une entente entre RDDC Valcartier et la GRC (SRE-09-015, 31XF20).

Les résultats de ce projet seront également d'un grand intérêt pour le Centre d'opérations des réseaux des Forces canadiennes (CORFC), le Groupe intégré de la criminalité technologique (GICT) de la GRC, la Sûreté du Québec, ainsi que d'autres équipes d'enquêtes cybernétiques.

Table of contents

List of tables

Acknowledgements

The author would like to thank Mr. Francois Rheaume, Defence Scientist, for peer reviewing this text and providing helpful comments to improve it. Moreover, the author would also like to extend his thanks to Mr. Martin Salois, Defence Scientist, for translating portions of this text.

Disclaimer policy

It must be understood from the outset that this technical memorandum examines computer malware and that handling virulent software is not without risk. As such, the reader should ensure that he has taken all the necessary precautions to avoid infecting his own computer system and those around him, whether on a corporate network or isolated system.

The reader should neither construe nor interpret the work described herein by the author as an endorsement of the aforementioned techniques and capacities as suitable for any specific purpose, construed, implied or otherwise. Moreover, the author does not endorse the specific use of any specific anti-virus product, the use of Volatility or any data carving technology. Many similar software tools, utilities and scanners exist beyond those used herein. They may be commercial or free and open source in nature and as such, the onus is on the reader to determine which software best suits his specific needs. While the author felt most comfortable working from within a Linux environment, the author does not specifically recommend the use of such a system for the reader. Instead, the reader should use the environment in which he is most comfortable.

Furthermore, the author of this technical memorandum absolves himself in all ways conceivable with respect to how the reader may use, interpret or construe this technical memorandum. The author assumes absolutely no liability or responsibility, implied or explicit. Moreover, the onus is on the reader to be appropriately equipped and knowledgeable in the application of digital forensics. Due to the offensive nature of computer malware, the author is no way responsible for the reader's use of any malware, whether examined herein or otherwise, in any offensive or defensive nature against any entity, even against the reader himself, for any purposes whatsoever, for any construed reasons.

Finally, the author and the Government of Canada are henceforth absolved of all wrongdoing, whether intentional, unintentional, construed or misunderstood on the part of the reader. If the reader does not agree to these terms, then his copy of this technical memorandum should be destroyed. Only if the reader agrees to these terms should he or she continue in reading it beyond this point. It is further assumed by all participants that if the reader has not read said Disclaimer upon reading this technical memorandum and has acted upon its contents, then the reader assumes all responsibility for any repercussions that may result from the information and data contained herein.

Requirements, assumptions and exclusions

The author assumes that the reader is altogether familiar with digital forensics and the various techniques and methodologies associated therein. This technical memorandum is not an introduction to digital forensics or to said techniques and methodologies. However, the author will endeavour to ensure that the reader can carry out his own forensic analysis of a computer memory image suspected of malware infection.

The experimentation conducted throughout this technical memorandum was carried out atop a Fedora Core 19 64-bit Linux operating system. Six different anti-virus scanners were used throughout this investigation. They include, in alphabetical order, AVG, Avast, BitDefender, Comodo, FRISK F-Prot and McAfee command line scanners. As for data carving tools and utilities, the author used Photorec version 6.14, part of the Testdisk (version 6.14) suite of data recovery tools.

The reader is required to have permission to use these tools on his computer system or network. Use of these tools and the analysis of virulent software always carry some inherent risk that must be securely managed and adequately mitigated.

An in-depth study of memory analysis techniques is outside the scope of this work, as it requires a comprehensive study of Windows operating system internals and software reverse engineering techniques, both of which are difficult subjects to approach. Instead, this work should be considered as a guide to using the Volatility memory analysis framework with respect to malware infection.

When working with or examining files dumped using various Volatility plugins, the use of the terms processes, memory sample files and memory dump files are used interchangeably.

Finally, the use of masculine is employed throughout this text to simplify it.

Target audience

The targeted audience for this technical memorandum is the computer forensic investigator who assesses suspect computer memory images for evidence of infection. Although computer memory analysis is a new field within the realm of digital forensics, there are those who have been conducting malware analysis and software reverse engineering for years, long before it came to the attention of most practitioners. Thus, those seasoned veterans are aptly skilled, taking years to develop their abilities. As such, the Volatility framework, while capable of providing insight to novices, is all the more capable in expert hands.

The author has written this technical memorandum for others who, like himself, are required from time to time to conduct memory malware assessments and investigations. However, the author, like many others, is not seasoned enough to take full advantage of Volatility's capabilities. As such, this technical memorandum combines both traditional forensic investigative techniques, coupled with Volatility's non-expert (non-reverse engineering) plugins, in order to develop an investigative how-to for non-memory experts.

1 Background

1.1 Objective

The objective of this technical memorandum is to examine how a computer forensic investigator, without specialised computer memory or software reverse engineering knowledge, can successfully investigate a memory image suspected of infection. More specifically, this document provides a methodological approach novice memory analysts can use to investigate suspected memory images.

The work carried out herein is based on the publicly available memory image 0zapftis. This malware is also commonly known as the R2D2 Trojan and for the remainder of this document will be referred to as such. This document, the third in a series of many, examines the investigative techniques necessary for a novice to conduct such memory analyses on his own. The first report in this series written by the author examined the Zeus Trojan Horse, found in TM 2013-018 [1] while the second examined Prolaco and SpyEye, found in TM 2013-155 [2].

Ultimately, these reports will provide a methodological and foundational framework that novice memory analysts and experienced investigators alike can rely on for guidance.

1.2 Why write new tutorials?

The purpose of writing new tutorials was addressed in the first report of this series. [1]

1.3 Infected memory image information

The infected memory image for R2D2 was procured from the following location: http://code.google.com/p/volatility/wiki/PublicMemoryImages. Its SHA1 hash, in uncompressed, form is as follows:

Table 1: Infected memory image metadata.

Memory image	Size (MiB)	SHA1 hash value
0zapftis.vmem	256 (exactly)	e4d4f4d1c304919ed51e17593a56d24b37c5acd9

1.4 Data carving

An in-depth examination of data carving can be found in two memorandums written by the author, specifically [1][3].

1.5 Malware and anti-virus scanners

1.5.1 Specifics

An examination of malware and anti-virus scanner specifics can be found in [1].

1.5.2 Caveat

An analysis concerning the caveats of using malware and anti-virus scanners was conducted in [1].

1.6 Detailed list of software tools used

1.6.1 Anti-virus scanners

This memorandum makes use of six anti-virus scanners, five of which are the same as those used in [1][2]. The only difference is that ClamAV is no longer used and has been replaced by Comodo Antivirus. These six anti-virus scanners continue to represent a diverse cross-section of various detection mechanisms necessary for identifying numerous malware. Each scanner was updated September 17, 2013, the date upon which the analysis was carried out herein. Scanner specifics are listed in the following table:

Table 2: List of anti-virus scanners and their command line parameters.

Anti-virus scanner	Command line parameters
AVG 2013 command line scanner version 13.0.3114	avgscan -H -P -p
Avast v.1.3.0 command line scanner	avast -c
BitDefender for Unices v7.90123 Linux-amd64 scanner command line	bdscan (no parameters used)
Comodo Antivirus Product Version 1.1.268025.1 / Virus Signature Database Version 16954	cmdscan -v -s
FRISK F-Prot version 6.3.3.5015 command line scanner	fpscan -u 4 -s 4 -z 10 --adware --applications --nospin
McAfee VirusScan for Linux64 Version 6.0.3.356 command line scanner	uvscan --RECURSIVE --ANALYZE --MANALYZE --MIME --PANALYZE --UNZIP --VERBOSE

1.6.2 Data carving

Photorec was used for data carving. The specifics concerning program settings were examined in [1].

1.6.3 Volatility

An examination of Volatility, its capabilities, main authors and contributors is found in [1].

A list of Windows-specific plugins currently supported by this version of Volatility can be found in Annex A.

1.7 Investigative methodology

The original infected memory image-based investigative methodology was put forward in [1] and lightly amended in [2] to deal with *strings*-based textual extraction and analyses.

2 Memory investigation and analysis of R2D2

2.1 Background

2.1.1 Mise-en-scène

This analysis examines a memory image suspected of harbouring the R2D2 Trojan horse, as based on the methodology put forward in Section 1.7. Much information could be found on the web concerning this particular infection. What was found, [4][5][6][7][8][9][10][11][12][13][14][23][24] and [25] provided a wealth of information. Specifically, R2D2 is a communications interception C&C based malware capable of spying on VoIP and a myriad of web-based communications.

However, in contrast to previous reports by the author that examined malware memory analysis [1][2], the various malware reports, articles and journals cited above will not be used directly in this investigation. Instead, this information is cited for the reader so that he can better understand the malware's capabilities rather than rely on them for conducting this investigation. Moreover, as with previous analyses [1][2], no use was made of existing analyses of this memory image.

Specifically, in order to gain practical experience analysing memory images, the author is of the opinion that there is no substitute for applying keen attention to detail and spotting the "needle in the haystack." This approach, while intuitive in nature, is second to none when attempting to spot out of the ordinary minutiae. Thus, this specific investigation, while applying the methodology outlined in Section 1.7, will also point out detected anomalies that may indicate potential indicators of compromise or other infection-based evidence.

2.1.2 Context

The Chaos Computer Club (CCC), Europe's largest hacker association [18], first broke the story concerning the R2D2 botnet in October 2011. Moreover, they successfully reverse engineered the botnet and then made it publically available for analysis by others [8]. However, their malware samples were modified from the original as they no longer point to the same hardcoded command and control (C&C) server. The original server used IP address 207.158.22.134 and network port 443[1] for its covert channel [8]. The CCC changed the IP address and port to 172.16.98.1/port 6666 [19]. The CCC refers to this malware as *Staatstrojaner* [18][19] although the media refers to it as *Bundestrojaner*.

As already stated, this analysis makes no direct use of the publicly available literature and reports concerning this infection. Only certain information such as the change of IP address just mentioned are made note of throughout this memorandum.

[1] This port is usually reserved for SSL encrypted traffic and is generally found open on firewalls. Many malware use this port specifically because it is open. In this way, encrypted data can be sent through the firewall without arousing suspicious, unless the firewall uses deep packet inspection, in which case nefarious traffic might be seen, blocked and reported.

Allegations of this malware having been developed by a European federal government abound. However, as with all allegations, it is not possible to determine with any certainty how this malware came into being. [10][11][20][21][22] and [23]

2.2 Preliminary investigative steps

The steps examined in this subsection should be considered as necessary preliminary steps for examining a potentially infected memory image.

2.2.1 Safeguard the memory image

The memory image *0zapftis.vmem* was set to immutable atop an Ext4-based filesystem. The command used to perform this, carried out as the root user, was:

```
$ sudo chattr +i 0zapftis.vmem
```

This results in a memory image that can no longer be modified, even by the root user. This is to prevent accidental modifications from occurring to this file.

2.2.2 Preliminary anti-virus scanning results

Scanning only the memory image itself with the six scanners outlined in Section 1.6.1, the only scanner that identified memory image *0zapftis.vmem* as infected was Avast. Its output is as follows:

```
./R2D2_Report/0zapftis.vmem   [infected by: Win32:R2D2-F [Trj]]
```

Preliminary anti-virus scanner examination indicates that this memory image is likely infected with the R2D2 Trojan horse. It appears that Avast was the only scanner capable of examining, at least partially, the memory image's internal structures. All anti-virus results were recorded and saved.

2.2.3 Data carving and file hashing

Photorec succeeded in recovering 636 files carved from the memory image as per the recommended Photorec settings put forward in Section 1.6.2. Ten duplicate files were found, thereby leaving 626 unique files recovered. Of these 636 recovered files, 471 were identified as PE-based files. Of those, 307 were identified as Windows 32-bit DLLs, while 164 were identified as standard Windows 32-bit PEs and device drivers. No 64-bit PE-based files were found. Only one file was identified as UPX-based. Finally, 16 files were detected as MS-DOS 16-bit executables for Windows 3.x.

Other file types were detected but were of no immediate use. However, their types were recorded and saved for possible future use within this analysis.

All recovered files were SHA1-hashed and then validated against NSRL hash-set 2.41 (June 2013). Results were stored for future use. Twenty-eight unique SHA1 hashes were confirmed as matching against the NSRL hash-set. NSRL SHA1-filename matches can be found in Annex C.

Finally, CTPH-based hashing (fuzzy hashing) was conducted using the *ssdeep* tool against the carved data files and stored for future use.

2.2.4 Anti-virus scanning results for carved data files

Using the six scanners and combining their output through UNIX command line processing tools (e.g. *cat*, *sort*, *find*, *tr*, *strings*, *awk*, *grep*, *uniq*, etc.), two matches were established. The first match involved the AVG and BitDefender scanners while the second match involved the Avast, Comodo and McAfee scanners. These matches are shown in the table below.

Specific logs for each scanner can be found in Annex B and matches are indicated accordingly therein. Moreover, all six scanners succeeded in detecting one or more possible infections from the carved data files.

Table 3: Matching of potentially infected carved data file vs. scanner.

Potentially infected file	Detecting Scanner
f0140472.exe	AVG
	BitDefender
f0181456.dll	Avast
	Comodo
	McAfee

2.3 Volatility analysis

In order to investigate this specific memory image, the use and output of various Volatility plugins are examined.

2.3.1 Step 1: Background information, process listings and analysis

This step examines the various Volatility plugins used to provide background information and context to the memory image. Process-based plugins are often able to provide confirmation of computer memory infection or compromise. However, they are not particularly helpful for determining if a computer system has been used inappropriately.

2.3.1.1 Imageinfo plugin

This Volatility plugin is used to provide basic contextual information about a suspect memory image. This should always be the first Volatility plugin used by an investigator.

Consider the following output from this plugin, using command "*volatility -f 0zapftis.vmem imageinfo*":

```
Determining profile based on KDBG search...

          Suggested   Profile(s)   :   WinXPSP2x86,   WinXPSP3x86
(Instantiated with WinXPSP2x86)
                      AS Layer1 : JKIA32PagedMemoryPae (Kernel AS)
                      AS       Layer2       :       FileAddressSpace
(/media/scratch/R2D2_Report/0zapftis.vmem)
                       PAE type : PAE
                            DTB : 0x319000L
                           KDBG : 0x80544ce0
          Number of Processors : 1
     Image Type (Service Pack) : 2
                KPCR for CPU 0 : 0xffdff000
            KUSER_SHARED_DATA : 0xffdf0000
          Image date and time : 2011-10-10 17:06:54 UTC+0000
     Image local date and time : 2011-10-10 13:06:54 -0400
```

This memory image appears to be running atop a 32-bit Windows XP computer system with Service Pack 2. It is equipped with one PAE-based processor and the memory image is 256 MiB in size (based on the memory image's size determined using *ls -l*). The memory image was acquired October 10, 2011 at 13:06:54 EDT.

2.3.1.2 Pslist plugin

The next step is to determine which processes are running within the memory image in order to determine if anything out of the ordinary is immediately visible. The *pslist* plugin provides a detailed process listing. It makes use of virtual memory addressing and offsets. This should always be the first process listing plugin used from Volatility.

Consider this plugin's output, using command "*volatility -f 0zapftis.vmem pslist*":

Table 4: Volatility Pslist plugin output sorted by PID.

Offset(V)	Name	PID	PPID	Thds	Hnds	Sess	Wow64	Start	Exit
0x819cc830	System	4	0	55	162	------	0		
0x816d63d0	VMwareTray.exe	184	1956	1	28	0	0	2011-10-10 17:04:41	
0x8180b478	VMwareUser.exe	192	1956	6	83	0	0	2011-10-10 17:04:41	
0x818233c8	reader_sl.exe	228	1956	2	26	0	0	2011-10-10 17:04:41	
0x815e7be0	wuauclt.exe	400	964	8	173	0	0	2011-10-10 17:04:46	
0x81945020	smss.exe	536	4	3	21	------	0	2011-10-10 17:03:56	
0x817a34b0	cmd.exe	544	1956	1	30	0	0	2011-10-10 17:06:42	
0x816c6020	csrss.exe	608	536	11	355	0	0	2011-10-10 17:03:58	
0x813a9020	winlogon.exe	632	536	24	533	0	0	2011-10-10 17:03:58	
0x816da020	services.exe	676	632	16	261	0	0	2011-10-10 17:03:58	
0x813c4020	lsass.exe	688	632	23	336	0	0	2011-10-10 17:03:58	
0x81772ca8	vmacthlp.exe	832	676	1	24	0	0	2011-10-10 17:03:59	
0x8167e9d0	svchost.exe	848	676	20	194	0	0	2011-10-10 17:03:59	
0x817757f0	svchost.exe	916	676	9	217	0	0	2011-10-10 17:03:59	
0x816c6da0	svchost.exe	964	676	63	1058	0	0	2011-10-10 17:03:59	
0x815daca8	svchost.exe	1020	676	5	58	0	0	2011-10-10 17:03:59	
0x813aeda0	svchost.exe	1148	676	12	187	0	0	2011-10-10 17:04:00	
0x817937e0	spoolsv.exe	1260	676	13	140	0	0	2011-10-10 17:04:00	
0x81754990	VMwareService.e	1444	676	3	145	0	0	2011-10-10 17:04:00	
0x8136c5a0	alg.exe	1616	676	7	99	0	0	2011-10-10 17:04:01	
0x815c4da0	wscntfy.exe	1920	964	1	27	0	0	2011-10-10 17:04:39	
0x813bcda0	explorer.exe	1956	1884	18	322	0	0	2011-10-10 17:04:39	

Looking at the above listing, nothing appears out of the ordinary. Although process *alg.exe* is present and can sometimes be used to indicate the presence of malware, as a lone indicator, it is not sufficient to warrant further investigation at this point as it is typically considered a legitimate Windows XP process.

Perhaps the next plugin, *psscan*, will reveal more information.

2.3.1.3 Psscan plugin

The *psscan* plugin uses physical memory addressing and scans memory images for _EPROCESS pool allocations, in contrast to the *pslist* plugin that uses virtual memory addressing and scans for EPROCESS lists. The benefit of using this plugin is that sometimes it succeeds in listing processes that cannot be found using other process listing plugins (i.e., *pslist* and *pstree*).

Consider this plugin's output, using command "*volatility -f 0zapftis.vmem psscan*":

Table 5: Volatility Psscan plugin output sorted by PID.

Offset(P)	Name	PID	PPID	PDB	Time created	Time exited
0x01bcc830	System	4	0	0x00319000		
0x018d63d0	VMwareTray.exe	184	1956	0x05e00160	2011-10-10 17:04:41	
0x01a0b478	VMwareUser.exe	192	1956	0x05e00260	2011-10-10 17:04:41	
0x01a233c8	reader_sl.exe	228	1956	0x05e00280	2011-10-10 17:04:41	
0x017e7be0	wuauclt.exe	400	964	0x05e002c0	2011-10-10 17:04:46	
0x01b45020	smss.exe	536	4	0x05e00020	2011-10-10 17:03:56	
0x019a34b0	cmd.exe	544	1956	0x05e00200	2011-10-10 17:06:42	
0x018c6020	csrss.exe	608	536	0x05e00040	2011-10-10 17:03:58	
0x015a9020	winlogon.exe	632	536	0x05e00060	2011-10-10 17:03:58	
0x018da020	services.exe	676	632	0x05e00080	2011-10-10 17:03:58	
0x015c4020	lsass.exe	688	632	0x05e000a0	2011-10-10 17:03:58	
0x01972ca8	vmacthlp.exe	832	676	0x05e000c0	2011-10-10 17:03:59	
0x0187e9d0	svchost.exe	848	676	0x05e000e0	2011-10-10 17:03:59	
0x019757f0	svchost.exe	916	676	0x05e00100	2011-10-10 17:03:59	
0x018c6da0	svchost.exe	964	676	0x05e00120	2011-10-10 17:03:59	
0x017daca8	svchost.exe	1020	676	0x05e00140	2011-10-10 17:03:59	
0x015aeda0	svchost.exe	1148	676	0x05e00180	2011-10-10 17:04:00	
0x019937e0	spoolsv.exe	1260	676	0x05e001a0	2011-10-10 17:04:00	
0x01954990	VMwareService.e	1444	676	0x05e001c0	2011-10-10 17:04:00	
0x0156c5a0	alg.exe	1616	676	0x05e001e0	2011-10-10 17:04:01	
0x017c4da0	wscntfy.exe	1920	964	0x05e00240	2011-10-10 17:04:39	
0x015bcda0	explorer.exe	1956	1884	0x05e00220	2011-10-10 17:04:39	

Again, nothing appears particularly conspicuous. Moreover, this output looks very similar to the output of the *pslist* plugin. However, in order to be certain, the subsequent step will examine the differences in their output.

2.3.1.4 Differentiating the output between the pslist and psscan plugins

Distinguishing between the output of the *pslist* and *psscan* plugins may not be obvious at first glance. For this task, shell-based text processing is of significant use. By using the following command, it is readily possible to differentiate the output between the two plugins:

```
$ cat pslist.txt psscan.txt | awk '{print $2"\t"$3}' | sort |
uniq -c | grep -v "    2"
```

This command results in the following output:

```
1 --------------------  ------
1 ----------------      ------
```

Thus, by using these commands, it was determined that there was no discernible difference in their output. Perhaps the next plugin, *psxview*, will be of more assistance.

2.3.1.5 Psxview plugin

Volatility provides an additional capability for detecting hidden running processes. The *psxview* plugin provides a detailed listing of processes in a memory image by using five specific process detection methods. These include *pslist*, *psscan*, *thrdproc*, *pspcdid* and *csrss*. Moreover, the plugin makes use of physical memory addressing.

For a process to be considered hidden, it should be invisible to, at a minimum, any non-*csrss* detection mechanism but may also be undetectable by subsequent process detection methods. However, if a process is not seen by the *pslist* mechanism then the process is without a doubt hidden.

Consider the following output from this plugin, using command "*volatility -f 0zapftis.vmem psxview*":

Table 6: Volatility Psxview plugin output sorted by PID.

Offset(P)	Name	PID	pslist	psscan	thrdproc	pspcdid	csrss
0x01bcc830	System	4	TRUE	TRUE	TRUE	TRUE	FALSE
0x018d63d0	VMwareTray.exe	184	TRUE	TRUE	TRUE	TRUE	TRUE
0x01a0b478	VMwareUser.exe	192	TRUE	TRUE	TRUE	TRUE	TRUE
0x01a233c8	reader_sl.exe	228	TRUE	TRUE	TRUE	TRUE	TRUE
0x017e7be0	wuauclt.exe	400	TRUE	TRUE	TRUE	TRUE	TRUE
0x01b45020	smss.exe	536	TRUE	TRUE	TRUE	TRUE	FALSE
0x019a34b0	cmd.exe	544	TRUE	TRUE	TRUE	TRUE	TRUE
0x018c6020	csrss.exe	608	TRUE	TRUE	TRUE	TRUE	FALSE
0x015a9020	winlogon.exe	632	TRUE	TRUE	TRUE	TRUE	TRUE
0x018da020	services.exe	676	TRUE	TRUE	TRUE	TRUE	TRUE
0x015c4020	lsass.exe	688	TRUE	TRUE	TRUE	TRUE	TRUE
0x01972ca8	vmacthlp.exe	832	TRUE	TRUE	TRUE	TRUE	TRUE
0x0187e9d0	svchost.exe	848	TRUE	TRUE	TRUE	TRUE	TRUE
0x019757f0	svchost.exe	916	TRUE	TRUE	TRUE	TRUE	TRUE
0x018c6da0	svchost.exe	964	TRUE	TRUE	TRUE	TRUE	TRUE
0x017daca8	svchost.exe	1020	TRUE	TRUE	TRUE	TRUE	TRUE
0x015aeda0	svchost.exe	1148	TRUE	TRUE	TRUE	TRUE	TRUE
0x019937e0	spoolsv.exe	1260	TRUE	TRUE	TRUE	TRUE	TRUE
0x01954990	VMwareService.e	1444	TRUE	TRUE	TRUE	TRUE	TRUE
0x0156c5a0	alg.exe	1616	TRUE	TRUE	TRUE	TRUE	TRUE
0x017c4da0	wscntfy.exe	1920	TRUE	TRUE	TRUE	TRUE	TRUE
0x015bcda0	explorer.exe	1956	TRUE	TRUE	TRUE	TRUE	TRUE

Based on the plugin's output, no hidden processes were found for this memory image.

Although some processes may be listed as hidden by the *csrss* method, they generally are not hidden. Therefore any process marked as hidden (FALSE) by this method requires that another method (*pslist*, *psscan*, *thrdproc* and *pspcdid*) confirm the suspicion. For Windows 7 and Vista systems, the list of internal processes is not available, and in some cases where Windows XP required memory pages might have been swapped out, the outcome of *csrss* may be affected. [15]

2.3.1.6 Summary and analysis

The Volatility plugins used in this step have not succeeded in finding any indicators of compromise. Thus, subsequent plugins, specifically state-based plugins, may reveal evidence of an infection.

2.3.2 Step 2: State-based information and analysis

This step examines state-based plugins that can be used to establish evidence of an infection. These plugins often provide information that process listing-based plugins cannot.

2.3.2.1 Cmdscan and consoles plugins

The *cmdscan* and *consoles* plugins may reveal additional information about commands typed into a command shell.

The *cmdscan* plugin is used to query the process memory of *csrss.exe* or *conhost.exe* for possible commands that may have been entered into the system shell (*cmd.exe*; i.e. PID 544) or through a backdoor or RDP session by an attacker. Specifically, it looks for COMMAND_HISTORY based structures left behind in memory. The scanning of *csrss.exe* applies to Windows XP, 2003, Vista and Server 2008 while the use of *conhost.exe* applies to Windows 7. The effect of this plugin against Windows 2000, 8 and Server 2012 is not currently known and has not been attempted by the author. [16]

The *consoles* plugin is similar to *cmdscan* except that it searches for CONSOLE_INFORMATION based data structures instead. More specifically, it provides the command history of commands fed to the system shell (*cmd.exe*; i.e. PID 544) or through backdoors and this data structure keeps both the input and output buffers for commands found using this plugin. [16]

To query a memory image using these two plugins, the following commands are issued:

```
$ volatility -f 0zapftis.vmem cmdscan

$ volatility -f 0zapftis.vmem consoles
```

The *cmdscan* plugin revealed the following, where key information has been highlighted and bolded:

```
CommandProcess: csrss.exe Pid: 608
CommandHistory: 0x11132d8 Application: cmd.exe Flags: Allocated,
Reset
CommandCount: 2 LastAdded: 1 LastDisplayed: 1
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x4c4
Cmd #0 @ 0x4e1eb8: sc query malwar
Cmd #1 @ 0x11135e8: sc query malware
```

The *consoles* plugin revealed the following, where key information has been highlighted and bolded:

```
ConsoleProcess: csrss.exe Pid: 608
Console: 0x4e2370 CommandHistorySize: 50
HistoryBufferCount: 2 HistoryBufferMax: 4
OriginalTitle: %SystemRoot%\system32\cmd.exe
Title: C:\WINDOWS\system32\cmd.exe
AttachedProcess: cmd.exe Pid: 544 Handle: 0x4c4
----
CommandHistory: 0x1113498 Application: sc.exe Flags:
CommandCount: 0 LastAdded: -1 LastDisplayed: -1
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x0
----
CommandHistory: 0x11132d8 Application: cmd.exe Flags: Allocated,
Reset
CommandCount: 2 LastAdded: 1 LastDisplayed: 1
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x4c4
Cmd #0 at 0x4e1eb8: sc query malwar
Cmd #1 at 0x11135e8: sc query malware
----
Screen 0x4e2a70 X:80 Y:300
Dump:
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp

C:\Documents and Settings\Administrator>sc query malwar
[SC] EnumQueryServicesStatus:OpenService FAILED 1060:

The specified service does not exist as an installed service.


C:\Documents and Settings\Administrator>sc query malware

SERVICE_NAME: malware
        TYPE               : 1  KERNEL_DRIVER
        STATE              : 4  RUNNING

(STOPPABLE,NOT_PAUSABLE,IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0
```

Based on the output of these two plugins, some individual, either locally or remotely, queried the system for some service named *malware*. This service was found to be running and was found to be a kernel-based driver.

This information is a very important indicator of compromise as it provides several important clues. The first is that there appears to be a malicious driver on the system providing some unknown service, which is currently active. Moreover, any process initiated by this driver is not visible to Volatility's process listing plugins (i.e. *pslist*, *psscan* and *psxview*). Thirdly, the service is known as *malware*. Taken together, these clues will help the investigator track down the malware.

2.3.2.2 Connscan plugin

The first network-based Volatility plugin that should be used is *connscan*. It is used to verify the existence of ongoing network connections and scans a memory image for current or recently terminated connections. This plugin makes uses of physical memory addressing.

Consider plugin's output, using command "*volatility -f 0zapftis.vmem connscan*":

Table 7: Volatility Connscan plugin output.

Offset(P)	Local Address	Remote Address	PID
0x01a25a50	0.0.0.0:1026	172.16.98.1:6666	1956

Based on this information, PID 1956 (*explorer.exe*) has established a connection with remote system 172.16.98.1 using port 6666. This port is a well-known malware based port [17]. Recall that the IP address and network port are not the original ones (see Section 2.1.2 for details). The original IP address was 207.158.22.134 and was found communicating on port 443.

The *Whois* information for these two IP addresses is examined in the following subsections.

2.3.2.2.1 Whois for first suspicious address

The false remote IP address, 172.16.98.1, has been determined to belong to a private web address, as based on the following *Whois* information:

```
NetRange:       172.16.0.0 - 172.31.255.255
CIDR:           172.16.0.0/12
OriginAS:
NetName:        PRIVATE-ADDRESS-BBLK-RFC1918-IANA-RESERVED
NetHandle:      NET-172-16-0-0-1
Parent:         NET-172-0-0-0-0
NetType:        IANA Special Use
Comment:        These addresses are in use by many millions of
independently operated networks, which might be as small as a
single computer connected to a home gateway, and are
automatically configured in hundreds of millions of devices.
They are only intended for use within a private context  and
traffic that needs to cross the Internet will need to use a
different, unique address.
Comment:
Comment:        These addresses can be used by anyone without
any need to coordinate with IANA or an Internet registry.  The
traffic from these addresses does not come from ICANN or IANA.
```

```
We are not the source of activity you may see on logs or in e-
mail records.  Please refer to http://www.iana.org/abuse/answers
Comment:
Comment:          These addresses were assigned by the IETF, the
organization  that  develops  Internet  protocols,  in  the  Best
Current Practice document, RFC 1918 which can be found at:
Comment:        http://datatracker.ietf.org/doc/rfc1918
RegDate:        1994-03-15
Updated:        2013-08-30
Ref:           http://whois.arin.net/rest/net/NET-172-16-0-0-1

OrgName:          Internet Assigned Numbers Authority
OrgId:            IANA
Address:          12025 Waterfront Drive
Address:          Suite 300
City:             Los Angeles
StateProv:        CA
PostalCode: 90292
Country:          US
RegDate:
Updated:          2012-08-31
Ref:              http://whois.arin.net/rest/org/IANA

OrgAbuseHandle:   IANA-IP-ARIN
OrgAbuseName:     Internet  Corporation  for  Assigned  Names  and
Number
OrgAbusePhone:    +1-310-301-5820
OrgAbuseEmail:    abuse@iana.org
OrgAbuseRef:      http://whois.arin.net/rest/poc/IANA-IP-ARIN

OrgTechHandle:    IANA-IP-ARIN
OrgTechName:      Internet  Corporation  for  Assigned  Names  and
Number
OrgTechPhone:     +1-310-301-5820
OrgTechEmail:     abuse@iana.org
OrgTechRef: http://whois.arin.net/rest/poc/IANA-IP-ARIN
```

2.3.2.2.2 Whois for second suspicious address

The original botnet C&C remote IP address, 207.158.22.134, has been determined to belong to American Internet Services, a California-based ISP and web-hosting company, as per the following *Whois* information:

```
NetRange: 207.158.0.0 - 207.158.63.255
CIDR: 207.158.0.0/18
OriginAS: AS6130
NetName: AIS-WEST2
NetHandle: NET-207-158-0-0-1
Parent: NET-207-0-0-0-0
NetType: Direct Allocation
Comment: ADDRESSES WITHIN THIS BLOCK ARE NON-PORTABLE.
RegDate: 1996-06-22
Updated: 2012-03-02
Ref: http://whois.arin.net/rest/net/NET-207-158-0-0-1

OrgName: American Internet Services, LLC.
OrgId: AMERI-504
Address: 9305 Lightwave Ave.
```

```
City: San Diego
StateProv: CA
PostalCode: 92123
Country: US
RegDate: 2008-11-26
Updated: 2012-06-05
Ref: http://whois.arin.net/rest/org/AMERI-504

ReferralServer: rwhois://rwhois.americanis.net:4321

OrgAbuseHandle: ABUSE1714-ARIN
OrgAbuseName: Abuse
OrgAbusePhone: +1-858-576-4272
OrgAbuseEmail: abuse@americanis.net
OrgAbuseRef: http://whois.arin.net/rest/poc/ABUSE1714-ARIN

OrgTechHandle: AIS10-ARIN
OrgTechName: AIS
OrgTechPhone: +1-858-576-4272
OrgTechEmail: routing@americanis.net
OrgTechRef: http://whois.arin.net/rest/poc/AIS10-ARIN

OrgNOCHandle: NOC2657-ARIN
OrgNOCName: NOC
OrgNOCPhone: +1-858-576-4272
OrgNOCEmail: noc@americanis.net
OrgNOCRef: http://whois.arin.net/rest/poc/NOC2657-ARIN

NetRange: 207.158.37.0 - 207.158.37.255
CIDR: 207.158.37.0/24
OriginAS: AS6130
NetName: M5-SECURITY-NETBLK-10
NetHandle: NET-207-158-37-0-1
Parent: NET-207-158-0-0-1
NetType: Reassigned
RegDate: 2008-12-17
Updated: 2008-12-17
Ref: http://whois.arin.net/rest/net/NET-207-158-37-0-1

OrgName: M5 Computer Security
OrgId: MCS-227
Address: 3368 Governor Drive #F-124
City: San Diego
StateProv: CA
PostalCode: 92122
Country: US
RegDate: 2005-11-09
Updated: 2011-11-03
Ref: http://whois.arin.net/rest/org/MCS-227

OrgTechHandle: MMC112-ARIN
OrgTechName: McCafferty, Michael
OrgTechPhone: +1-619-985-2547
OrgTechEmail: mike@m5computersecurity.com
OrgTechRef: http://whois.arin.net/rest/poc/MMC112-ARIN

OrgTechHandle: RENWI-ARIN
OrgTechName: Renwick, James
OrgTechPhone: +1-619-800-2055
OrgTechEmail: joe@gonetforward.com
OrgTechRef: http://whois.arin.net/rest/poc/RENWI-ARIN
```

```
OrgAbuseHandle: ABUSE1898-ARIN
OrgAbuseName: Abuse
OrgAbusePhone: +1-877-344-4678
OrgAbuseEmail: abuse@m5hosting.com
OrgAbuseRef: http://whois.arin.net/rest/poc/ABUSE1898-ARIN

RTechHandle: MMC112-ARIN
RTechName: McCafferty, Michael
RTechPhone: +1-877-344-4678
RTechEmail: mike@m5computersecurity.com
RTechRef: http://whois.arin.net/rest/poc/MMC112-ARIN
```

2.3.2.2.3 Summary

This plugin has clearly found evidence of a covert communication. Moreover, both IP addresses were found to be in the continental U.S.

2.3.2.3 Connections plugin

The *connections* plugin can be used to find evidence of both recently terminated and ongoing communications. It therefore makes sense to use this plugin as it may reveal additional network-based information. Moreover, this plugin supports both physical and virtual memory addresses.

However, using command "*volatility -f 0zapftis.vmem connections*" yielded no output.

2.3.2.4 Sockets and sockscan plugins

Volatility offers two additional network-based plugins, *sockets* and *sockscan*. The *sockets* plugin lists open sockets and may provide additional information about covert network channels, while the *sockscan* plugin scans a suspect memory image for all TCP sockets. Generally, the output is the same for both plugins with the exception of memory addresses, where the *sockets* plugin uses virtual memory addressing while the *sockscan* plugin uses physical memory addressing.

Thus, using the following commands it will be possible to determine which processes are ready for a connection:

```
$ volatility -f 0zapftis.vmem sockets > sockets.txt

$ volatility -f 0zapftis.vmem sockscan > sockscan.txt

$ cat sockets.txt sockscan.txt | awk '{$1="";print}' | sort -n |
uniq > sockets_sockscan.txt
```

The output of file *sockets_sockscan.txt* appears as shown in the following table:

Table 8: Volatility Sockets and Sockscan plugins output sorted by PID.

PID	Port	Proto	Protocol	Address	Create Time
4	445	17	UDP	0.0.0.0	10/10/2011 17:03:55
4	445	6	TCP	0.0.0.0	10/10/2011 17:03:55
688	0	255	Reserved	0.0.0.0	10/10/2011 17:04:00
688	4500	17	UDP	0.0.0.0	10/10/2011 17:04:00
688	500	17	UDP	0.0.0.0	10/10/2011 17:04:00
916	135	6	TCP	0.0.0.0	10/10/2011 17:03:59
964	1029	17	UDP	127.0.0.1	10/10/2011 17:04:42
964	123	17	UDP	127.0.0.1	10/10/2011 17:04:00
1148	1900	17	UDP	127.0.0.1	10/10/2011 17:04:41
1616	1025	6	TCP	127.0.0.1	10/10/2011 17:04:01
1956	1026	6	TCP	0.0.0.0	10/10/2011 17:04:39

Examining this data, the covert communication found emanating from *explorer.exe* (see Section 2.3.2.2 for details) is not in this output. Thus, somewhere behind *explorer.exe* there is clearly a hidden communication channel in use.

2.3.2.5 Filescan plugin

If an infection is active and does not show itself via the network then the *filescan* plugin may be of assistance as it may be able to find open file handles in memory. Unfortunately, no direct link to these files is possible as the physical disk image is not available for analysis. Firstly, this plugin makes use of physical address offsets.

The preferred method for detecting indicators of compromise is twofold. First, using keywords (e.g. 0zapftis, infection, rootkit, worm, etc.) it may be possible to find the infection, as malware programmers do not often using innocuous looking filenames. Of course, this is at best a hit and miss approach. Secondly, an investigator can attempt to detect suspicious files based on their names and locations. However, this requires that the investigator has a very good working knowledge of the underlying operating system. Just looking blindly at filenames[2] and locations will not produce meaningful results, unless something really sticks out.

For this specific investigation, since emphasis is placed on detecting indicators of compromise without the use of external documentation, the investigator must studiously examine this plugin's output. Thus, running command "*volatility -f 0zapftis.vmem filescan*," after extensive verification against a list of known Windows XP filenames, resulted in the following highly suspicious file:

[2] Recall that a reliable source of filenames is the NSRL hash-set. It can be broken down manually (using command line text processing tools) by software product and operating system.

Table 9: Volatility Filescan plugin output for suspicious Windows DLL.

Offset (P)	#Ptr	#Hnd	Access	Name
0x015b8128	1	0	R--r-d	\Device\HarddiskVolume1\WINDOWS\system32\mfc42ul.dll

This file does not belong in the Windows System32 directory. While it looks valid, because many *mfc*-based files can be found in a valid Windows installation, this file does not match any known list of files (NSRL hash-set 2.41). However, file *mfc42u.dll* is a very close match to this suspicious filename and is a known Windows file. This suspicious DLL has been found in memory at address *0x015b8128* and it may have been used to carry out DLL injection.

Unfortunately, examining the output generated from this plugin can be both time-consuming and painstaking.

2.3.2.6 Mutantscan plugin

The Volatility *mutantscan* can sometimes reveal interesting information about Windows thread-based mutexes in memory. This plugin makes use of physical offset addressing.

Using command "*volatility -f 0zapftis.vmem mutantscan*" yielded the following pertinent information, after pruning the output of non-pertinent mutexes:

Table 10: Volatility Mutantscan plugin output of suspicious mutexes.

Offset (P)	#Ptr	#Hnd	Signal	Thread	CID	Name
0x017d6f60	2	1	0	0x813b7230		1956:2000 SYS!ICP!94062
0x018d8180	2	1	1	0x00000000		SYS!IPC!79025
0x0197ef38	3	2	1	0x00000000		SYS!ICP!393-1M
0x0197efe0	2	1	-1	0x813bea80	1956:1980	SYS!IPC!79027
0x01a2eac0	3	2	1	0x00000000		SYS!ICP!393-1MR

This output indicates that at least two processes or threads labelled as PID 1956 (*explorer.exe*) are using suspicious looking mutexes, *SYS!IPC!*. These have been highlighted in red in the above table. Moreover, other non-PID 1956 mutexes have been isolated because they look like they are from the same source, specifically some suspicious process or thread related to the PID 1956 mutexes listed above.

It appears the above-listed mutexes are using IPC-based synchronization and communication.

Thus, based on this information and the above table, it can be inferred that these suspicious mutexes are working together by some process or thread related to PID 1956 to carry out the covert communication (see Section 2.3.2.2 for details).

2.3.2.7 Handles plugin

The Volatility *handles* plugin can reveal interesting information about processes and the resources attached or associated to them that might not be found using previously examined plugins. The *handles* plugin makes use of virtual memory addressing.

Using command "*volatility -f 0zapftis.vmem handles*," the following pruned output is of interest to the investigation and is as follows:

Table 11: Volatility Handles plugin output for suspicious handles.

Offset (V)	PID	Handle	Access	Type	Details
0x81489a40	1956	0xa8	0x1f0003	Event	DUMMY!DUMMY
0x81489a40	1956	0xbc	0x1f0003	Event	DUMMY!DUMMY
0x815d6f60	1956	0xc0	0x1f0001	Mutant	SYS!ICP!94062
0x816d8180	1956	0x164	0x1f0001	Mutant	SYS!IPC!79025
0x8177ef38	1956	0x124	0x1f0001	Mutant	SYS!ICP!393-1M
0x8177ef38	1956	0xac	0x1f0001	Mutant	SYS!ICP!393-1M
0x8177efe0	1956	0xa0	0x1f0001	Mutant	SYS!IPC!79027
0x8182eac0	1956	0x114	0x1f0001	Mutant	SYS!ICP!393-1MR
0x8182eac0	1956	0xb0	0x1f0001	Mutant	SYS!ICP!393-1MR
0xe1a84680	1956	0xa4	0xf0007	Section	SYS!ICP!3949-1
0xe1cc0e78	1956	0x13c	0xf0007	Section	SYS!ICP!393-1
0xe1cc0e78	1956	0xb4	0xf0007	Section	SYS!ICP!393-1

It is likely that other suspicious handles were present but were not flagged due to the lack of appropriate context in which to evaluate them.

Although hundreds of entries were generated by the *handles* plugin, going through it was a time-consuming process.

These specific handles were flagged because they do not appear to be legitimate for *explorer.exe*. While many processes and threads communicate with other processes and threads, *explorer.exe* is not a program that typically does it in this fashion. Moreover, events such as *DUMMY!DUMMY* are highly suspicious, as is the number of mutexes in use by *explorer.exe*. Furthermore, it was suspicious that out of all the processes on the system that only *explorer.exe* was found using IPC thread-based communications. Finally, matches are readily obtained between the names of IPCs from tables 10 and 11. These matches have been highlighted as pink in both tables.

2.3.2.8 Threads and Thrdscan plugins

Two Volatility plugins will be used in this section, specifically the *threads* and *thrdscan* plugins. Armed with the information provided by the *handles* plugin, it is worthwhile investigating potential information that could be revealed using Volatility's *threads*-based plugins.

The *threads* plugin searches for _ETHREADS and _KTHREADS data structures while the *thrdscan* plugin searches for _ETHREADS data structures. The output from each plugin differs significantly. Moreover, the former plugin uses virtual memory addressing whereas the latter uses physical memory addressing.

Using these two plugins, the following information was obtained concerning PID 1956 (*explorer.exe*):

```
$ volatility -f 0zapftis.vmem threads | grep 1956

$ volatility -f 0zapftis.vmem thrdscan | grep 1956 | awk
'{$2="";$4="";print}'
```

The *threads* plugin command resulted in the following pruned output:

```
ETHREAD: 0x01984238  Pid: 1956  Tid: 132
ETHREAD: 0x01a2f8e8  Pid: 1956  Tid: 124
ETHREAD: 0x813b7230  Pid: 1956  Tid: 2000
ETHREAD: 0x813bc560  Pid: 1956  Tid: 396
ETHREAD: 0x813bea80  Pid: 1956  Tid: 1980
ETHREAD: 0x813c4988  Pid: 1956  Tid: 2024
ETHREAD: 0x813c4da8  Pid: 1956  Tid: 2020
ETHREAD: 0x8148cc28  Pid: 1956  Tid: 164
ETHREAD: 0x815c24c0  Pid: 1956  Tid: 1992
ETHREAD: 0x815cbda8  Pid: 1956  Tid: 1960
ETHREAD: 0x815cdda8  Pid: 1956  Tid: 2012
ETHREAD: 0x816cf230  Pid: 1956  Tid: 2004
ETHREAD: 0x816cf658  Pid: 1956  Tid: 2008
ETHREAD: 0x816d1a80  Pid: 1956  Tid: 1996
ETHREAD: 0x816d43d0  Pid: 1956  Tid: 160
ETHREAD: 0x816dd230  Pid: 1956  Tid: 2028
ETHREAD: 0x816dda80  Pid: 1956  Tid: 2016
ETHREAD: 0x8178b658  Pid: 1956  Tid: 2032
ETHREAD: 0x81883da8  Pid: 1956  Tid: 320
ETHREAD: 0x818e72a0  Pid: 1956  Tid: 292
ETHREAD: 0x81906368  Pid: 1956  Tid: 2040
```

Whereas the *thrdscan* plugin resulted in the following pruned output:

```
0x015b7230      2000   2011-10-10 17:04:39
0x015bc560       396   2011-10-10 17:04:46
0x015bea80      1980   2011-10-10 17:04:39
0x015c4988      2024   2011-10-10 17:04:40
0x015c4da8      2020   2011-10-10 17:04:40
0x0168cc28       164   2011-10-10 17:04:41
0x017c24c0      1992   2011-10-10 17:04:39
0x017cbda8      1960   2011-10-10 17:04:39
0x017cdda8      2012   2011-10-10 17:04:40
0x018cf230      2004   2011-10-10 17:04:39
0x018cf658      2008   2011-10-10 17:04:39   2011-10-10 17:04:39
0x018d1a80      1996   2011-10-10 17:04:39
0x018d43d0       160   2011-10-10 17:04:40
0x018dd230      2028   2011-10-10 17:04:40
0x018dda80      2016   2011-10-10 17:04:40
0x01984238       132   2011-10-10 17:04:40   2011-10-10 17:06:48
0x0198b658      2032   2011-10-10 17:04:40
0x01a2f8e8       124   2011-10-10 17:04:40   2011-10-10 17:06:47
0x01a83da8       320   2011-10-10 17:04:45
```

```
0x01ae72a0        292   2011-10-10 17:04:44
0x01b06368       2040   2011-10-10 17:04:40
```

From the plugins' output, TID 1980 and 2000, highlighted in red in the above output, can be correlated with the output from the mutantscan plugin (1956:1980 and 1956:2000) from Table 10. Whether the remaining threads have contributed to the infection is not currently known but there is reason to suspect that some of the additional non-exited threads may have contributed to this infection.

2.3.2.9 Driverscan and DriverIRP plugins

The *driverscan* plugin scans a memory image for driver objects and uses physical memory addressing while the *driverirp* plugin scans memory for driver IRP hooking. The latter plugin uses neither virtual nor physical memory addressing; instead, it accepts KDBG and KPCR addresses.

Through these plugins, it may be possible to find the specific driver alluded to by *cmdscan* and *consoles* plugins (see Section 2.3.2.1 for details). The following commands were issued to query the memory image for evidence about the malicious driver:

```
$ volatility -f 0zapftis.vmem driverscan

$ volatility -f 0zapftis.vmem driverirp
```

The output from these commands was pruned. Output from plugin *driverscan* was as follows:

Table 12: Volatility Driverscan plugin output of suspicious driver.

Offset (P)	#Ptr	#Hnd	Start	Size	Service Key	Name	Driver Name
0x01a498b8	3	0	0xf9eb4000	0x1500	malware	malware	\Driver\malware

Clearly, this is the malicious driver and it is located at physical memory address *0x01a498b8*.

Output from plugin *driverirp* was as follows:

```
DriverName: malware
DriverStart: 0xf9eb4000
DriverSize: 0x1500
DriverStartIo: 0x0
 0 IRP_MJ_CREATE                     0xf9eb4d76  winsys32.sys
 1 IRP_MJ_CREATE_NAMED_PIPE          0xf9eb4d76  winsys32.sys
 2 IRP_MJ_CLOSE                      0xf9eb4d76  winsys32.sys
 3 IRP_MJ_READ                       0xf9eb4e00  winsys32.sys
 4 IRP_MJ_WRITE                      0xf9eb4d76  winsys32.sys
 5 IRP_MJ_QUERY_INFORMATION          0xf9eb4d76  winsys32.sys
 6 IRP_MJ_SET_INFORMATION            0xf9eb4d76  winsys32.sys
 7 IRP_MJ_QUERY_EA                   0xf9eb4d76  winsys32.sys
 8 IRP_MJ_SET_EA                     0xf9eb4d76  winsys32.sys
 9 IRP_MJ_FLUSH_BUFFERS              0xf9eb4d76  winsys32.sys
10 IRP_MJ_QUERY_VOLUME_INFORMATION   0xf9eb4d76  winsys32.sys
11 IRP_MJ_SET_VOLUME_INFORMATION     0xf9eb4d76  winsys32.sys
12 IRP_MJ_DIRECTORY_CONTROL          0xf9eb4d76  winsys32.sys
```

```
13 IRP_MJ_FILE_SYSTEM_CONTROL         0xf9eb4d76  winsys32.sys
14 IRP_MJ_DEVICE_CONTROL              0xf9eb4e46  winsys32.sys
15 IRP_MJ_INTERNAL_DEVICE_CONTROL     0xf9eb4d76  winsys32.sys
16 IRP_MJ_SHUTDOWN                    0xf9eb4d76  winsys32.sys
17 IRP_MJ_LOCK_CONTROL                0xf9eb4d76  winsys32.sys
18 IRP_MJ_CLEANUP                     0xf9eb4d76  winsys32.sys
19 IRP_MJ_CREATE_MAILSLOT             0xf9eb4d76  winsys32.sys
20 IRP_MJ_QUERY_SECURITY              0xf9eb4d76  winsys32.sys
21 IRP_MJ_SET_SECURITY                0xf9eb4d76  winsys32.sys
22 IRP_MJ_POWER                       0xf9eb4e66  winsys32.sys
23 IRP_MJ_SYSTEM_CONTROL              0xf9eb4d76  winsys32.sys
24 IRP_MJ_DEVICE_CHANGE               0xf9eb4d76  winsys32.sys
25 IRP_MJ_QUERY_QUOTA                 0xf9eb4d76  winsys32.sys
26 IRP_MJ_SET_QUOTA                   0xf9eb4d76  winsys32.sys
27 IRP_MJ_PNP                         0x804f320e  ntoskrnl.exe
```

Examining the *driverirp* plugin's output, it is not readily possible for non-reverse engineers to determine which driver IRP function codes[3] are typically used for standard device drivers and which are used for malware. Unfortunately, such knowledge is not readily available in the form of a whitelist or blacklist.

2.3.2.10 Ldrmodules plugin

The *ldrmodules* Volatility plugin scans a memory image for signs of unlinked files (such as DLLs) in memory that may be indicative of a suspicious or malicious file lurking in memory. Since an already suspicious DLL was spotted, using the *filescan* plugin (*mfc42ul.dll*), others may be hiding. Running this plugin may help to find them. However, this plugin can also find other possibly hidden files in memory including executables and various types of libraries.

Using command "*volatility -f 0zapftis.vmem ldrmodules | grep False*" generated the following output:

Table 13: Volatility Ldrmodules plugin output sorted by PID.

PID	Process	Base	InLoad	InInit	InMem	MappedPath
4	System	0x7c900000	False	False	False	\WINDOWS\system32\ntdll.dll
184	VMwareTray.exe	0x00400000	True	False	True	\Program Files\VMware\VMware Tools\VMwareTray.exe
192	VMwareUser.exe	0x00400000	True	False	True	\Program Files\VMware\VMware Tools\VMwareUser.exe
228	reader_sl.exe	0x00400000	True	False	True	\Program Files\Adobe\Reader9.0\Reader\reader_sl.exe
400	wuauclt.exe	0x00400000	True	False	True	\WINDOWS\system32\wuauclt.exe
536	smss.exe	0x48580000	True	False	True	\WINDOWS\system32\smss.exe
544	cmd.exe	0x4ad00000	True	False	True	\WINDOWS\system32\cmd.exe

[3] An IRP function code is denoted by *IRP_MJ_*.

PID	Process	Base	InLoad	InInit	InMem	MappedPath
608	csrss.exe	0x00450000	False	False	False	\WINDOWS\Fonts\vgasys.fon
608	csrss.exe	0x4a680000	True	False	True	\WINDOWS\system32\csrss.exe
608	csrss.exe	0x01230000	False	False	False	\WINDOWS\Fonts\dosapp.fon
608	csrss.exe	0x01250000	False	False	False	\WINDOWS\Fonts\cga80woa.fon
608	csrss.exe	0x01260000	False	False	False	\WINDOWS\Fonts\cga40woa.fon
608	csrss.exe	0x010a0000	False	False	False	\WINDOWS\Fonts\vgaoem.fon
608	csrss.exe	0x01240000	False	False	False	\WINDOWS\Fonts\ega40woa.fon
632	winlogon.exe	0x01000000	True	False	True	\WINDOWS\system32\winlogon.exe
676	services.exe	0x01000000	True	False	True	\WINDOWS\system32\services.exe
688	lsass.exe	0x01000000	True	False	True	\WINDOWS\system32\lsass.exe
832	vmacthlp.exe	0x00400000	True	False	True	\Program Files\VMware\VMware Tools\vmacthlp.exe
848	svchost.exe	0x01000000	True	False	True	\WINDOWS\system32\svchost.exe
916	svchost.exe	0x01000000	True	False	True	\WINDOWS\system32\svchost.exe
964	svchost.exe	0x01000000	True	False	True	\WINDOWS\system32\svchost.exe
964	svchost.exe	0x02030000	False	False	False	\WINDOWS\system32\stdole2.tlb
1020	svchost.exe	0x01000000	True	False	True	\WINDOWS\system32\svchost.exe
1148	svchost.exe	0x01000000	True	False	True	\WINDOWS\system32\svchost.exe
1260	spoolsv.exe	0x01000000	True	False	True	\WINDOWS\system32\spoolsv.exe
1444	VMwareService.e	0x00400000	True	False	True	\Program Files\VMware\VMware Tools\VMwareService.exe
1616	alg.exe	0x01000000	True	False	True	\WINDOWS\system32\alg.exe
1920	wscntfy.exe	0x01000000	True	False	True	\WINDOWS\system32\wscntfy.exe
1956	explorer.exe	0x01000000	True	False	True	\WINDOWS\explorer.exe

Upon close examination of the table's contents, nothing was found to be out of the ordinary. In fact, due to the specific nature of the processes involved and the types of files listed as unlinked, nothing suspicious or malicious should be construed from this information.

2.3.2.11 Summary and analysis

The Volatility plugins used in this step of the analysis have revealed important clues concerning the infection. It is now known that a covert communication channel was in use by some process/thread hidden/injected under/into PID 1956 (*explorer.exe*). Moreover, it has been

discovered that a malicious driver has been loaded and that a suspicious DLL has been found in the Windows *System32* directory.

The next section will concentrate on isolating and dumping both the suspicious kernel driver *winsys32.sys* and DLL *mfc42ul.dll* from memory so that they can be further analysed.

2.3.3 Step 3: Memory dumping and analysis of DLL and driver

Once sufficient evidence has been established indicating that suspicious or possibly malicious processes, DLLs or drivers may be hiding in memory, they can be dumped from memory for further analysis. This step examines how to dump them from memory and corroborate them with the evidence thus far obtained.

The evidence thus far indicates that one malicious driver has been loaded and that a highly suspicious DLL has been found associated with PID 1956 (*explorer.exe*). Moreover, PID 1956 was found in the midst of a covert communication with some unknown remote system.

2.3.3.1 Create data directories

Create directories *malfind*, *dlldump* and *moddump* for storing memory samples that are to be dumped from the memory image using Volatility. This is done using the following commands:

```
$ mkdir malfind

$ mkdir dlldump

$ mkdir moddump
```

2.3.3.2 Malfind plugin

2.3.3.2.1 Running the plugin

Volatility's *malfind* plugin was specifically designed to search for malware hidden through code injection. If memory address offsets are specified then they must be physical memory addresses.

Using the following commands, it was attempted to find and dump injected code associated with PID 1956 (*explorer.exe*):

```
$ volatility -f 0zapftis.vmem malfind -p 1956 -o 0x015bcda0 --
dump-dir=malfind
```

This command found no indication of injected code as no output or dumped file resulted from this command.

The following command was then run at large against the entire memory image to detect if other processes had not been hijacked via code injection:

```
$ volatility -f 0zapftis.vmem malfind --dump-dir=malfind
```

This command succeeded in dumping 10 sample files from memory. However, looking only at the textual output generated by the *malfind* plugin, no indication of maliciously injected code had been found. Nevertheless, subsequent analyses will confirm or rule out the pertinence of these dumped files.

2.3.3.2.2 AV scanning

All 10 samples were scanned using the six aforementioned scanners. No indication of infection was found among them.

2.3.3.2.3 SHA1 and fuzzy hashes

All 10 dumped files were hashed using the *sha1sum* command to determine their SHA1 signatures. No identical SHA1 hashes were identified, indicating that each memory sample was unique. The files were then fuzzy hashed to determine if there were any similarities between them, however, none was found.

The SHA1 and fuzzy hashes were then compared against those of the carved data files. No identical or similar hashes were detected, respectively.

Finally, the SHA1 hashes were compared against the NSRL 2.41 hash-set but no matches were identified.

Thus, the *malfind*-dumped memory samples are independent of the previously established SHA1 and fuzzy hashes for the carved data files.

2.3.3.2.4 Summary

The *malfind* plugin did not succeed in dumping any maliciously injected code from any identified memory image process. Although 10 memory samples were dumped, they were all determined to be innocuous and independent of already established (carved data files) SHA1 and fuzzy hashes.

Other memory dumping approaches will be used in the following subsections.

2.3.3.3 Dlllist plugin

The *dlllist* plugin is primarily used to determine which DLLs are loaded for a given process. However, it can also be used to identify all DLLs loaded into memory. Running command "*volatility -f 0zapftis.vmem dlllist*" identified, in total, 847 DLLs loaded into memory.

Based on the *dlllist*-determined list of loaded DLLs, suspicious DLL *mfc42ul.dll* was found within the process space of PID 1956 (*explorer.exe*), as shown in the following table:

Table 14: Volatility Dlllist plugin output for PID 1956 against suspicious DLL mfc42ul.dll.

Base address	Size	Path
0x10000000	0x59000	C:\WINDOWS\system32\mfc42ul.dll

However, upon much closer inspection of the list of DLLs generated by this plugin 15 instances of this DLL were found in the memory space of other processes. Of these instances, only one was identified as belonging to PID 1956 (*explorer.exe*). The following table lists all processes identified with DLL *mfc42ul.dll*, including other pertinent information, as based on the *dlllist* plugin output:

Table 15: Volatility Dlllist plugin output for all detected instances of mfc42ul.dll (sorted by PID).

PID	Process Name	DLL Base Address	DLL Size (in bytes)	Disk Location
184	VMwareTray.exe	0x00390000	364,544	C:\WINDOWS\system32\mfc42ul.dll
192	VMwareUser.exe	0x00390000	364,544	C:\WINDOWS\system32\mfc42ul.dll
228	reader_sl.exe	0x10000000	364,544	C:\WINDOWS\system32\mfc42ul.dll
400	wuauctl.exe	0x10000000	364,544	C:\WINDOWS\system32\mfc42ul.dll
544	cmd.exe	0x10000000	364,544	C:\WINDOWS\system32\mfc42ul.dll
632	winlogin.exe	0x10000000	364,544	C:\WINDOWS\system32\mfc42ul.dll
676	services.exe	0x10000000	364,544	C:\WINDOWS\system32\mfc42ul.dll
688	lsass.exe	0x10000000	364,544	C:\WINDOWS\system32\mfc42ul.dll
832	vmacthlp.exe	0x10000000	364,544	C:\WINDOWS\system32\mfc42ul.dll
848	svchost.exe	0x10000000	364,544	C:\WINDOWS\system32\mfc42ul.dll
964	svchost.exe	0x10000000	364,544	C:\WINDOWS\system32\mfc42ul.dll
1260	spoolsv.exe	0x10000000	364,544	C:\WINDOWS\system32\mfc42ul.dll
1444	VMwareService.e	0x10000000	364,544	C:\WINDOWS\system32\mfc42ul.dll
1920	wscntfy.exe	0x10000000	364,544	C:\WINDOWS\system32\mfc42ul.dll
1956	explorer.exe	0x10000000	364,544	C:\WINDOWS\system32\mfc42ul.dll

Based on the above list, it can be inferred that this DLL is likely conducting DLL injection. However, the next step will be to dump these DLL instances from the memory image.

2.3.3.4 Dlldump plugin

2.3.3.4.1 Running the plugin

Volatility's *dlldump* plugin was specifically designed to dump DLLs from memory to disk. If memory address offsets are specified then they must be physical memory addresses.

Based on the information already established using the *dlllist* plugin, in order to dump all detected instances of DLL *mfc42ul.dll* from the memory image, the following commands were used:

```
$volatility -f 0zapftis.vmem dlldump -p 184 -b 0x00390000 --dump-
dir=dlldump
```

```
$volatility -f 0zapftis.vmem dlldump -p 192 -b 0x00390000 --dump-
dir=dlldump

$volatility -f 0zapftis.vmem dlldump -p 228 -b 0x10000000 --dump-
dir=dlldump

$volatility -f 0zapftis.vmem dlldump -p 400 -b 0x10000000 --dump-
dir=dlldump

$volatility -f 0zapftis.vmem dlldump -p 544 -b 0x10000000 --dump-
dir=dlldump

$volatility -f 0zapftis.vmem dlldump -p 632 -b 0x10000000 --dump-
dir=dlldump

$volatility -f 0zapftis.vmem dlldump -p 676 -b 0x10000000 --dump-
dir=dlldump

$volatility -f 0zapftis.vmem dlldump -p 688 -b 0x10000000 --dump-
dir=dlldump

$volatility -f 0zapftis.vmem dlldump -p 832 -b 0x10000000 --dump-
dir=dlldump

$volatility -f 0zapftis.vmem dlldump -p 848 -b 0x10000000 --dump-
dir=dlldump

$volatility -f 0zapftis.vmem dlldump -p 964 -b 0x10000000 --dump-
dir=dlldump

$volatility -f 0zapftis.vmem dlldump -p 1260 -b 0x10000000 --
dump-dir=dlldump

$volatility -f 0zapftis.vmem dlldump -p 1444 -b 0x10000000 --
dump-dir=dlldump

$volatility -f 0zapftis.vmem dlldump -p 1920 -b 0x10000000 --
dump-dir=dlldump

$volatility -f 0zapftis.vmem dlldump -p 1956 -b 0x10000000 --
dump-dir=dlldump
```

In all, 15 *dlldump* commands were issued but only 14 instances were successfully acquired. The instance of *mfc42ul.dll* from PID 632 (*winlogin.exe*) had been paged out.

The following is a list of the results obtained for the aforementioned *dlldump* commands:

Table 16: Specifics concerning dumped instances of mfc42ul.dll (sorted by PID).

Process (V)	Process Name	Dumped Data Filename	Size (in bytes)
0x813a9020	winlogon.exe	Error: DllBase is paged	360,448
0x813bcda0	explorer.exe	module.1956.15bcda0.10000000.dll	360,448
0x813c4020	lsass.exe	module.688.15c4020.10000000.dll	360,448

Process (V)	Process Name	Dumped Data Filename	Size (in bytes)
0x815c4da0	wscntfy.exe	module.1920.17c4da0.10000000.dll	360,448
0x815e7be0	wuauclt.exe	module.400.17e7be0.10000000.dll	360,448
0x8167e9d0	svchost.exe	module.848.187e9d0.10000000.dll	360,448
0x816c6da0	svchost.exe	module.964.18c6da0.10000000.dll	360,448
0x816d63d0	VMwareTray.exe	module.184.18d63d0.390000.dll	360,448
0x816da020	services.exe	module.676.18da020.10000000.dll	360,448
0x81754990	VMwareService.e	module.1444.1954990.10000000.dll	360,448
0x81772ca8	vmacthlp.exe	module.832.1972ca8.10000000.dll	360,448
0x817937e0	spoolsv.exe	module.1260.19937e0.10000000.dll	360,448
0x817a34b0	cmd.exe	module.544.19a34b0.10000000.dll	360,448
0x8180b478	VMwareUser.exe	module.192.1a0b478.390000.dll	360,448
0x818233c8	reader_sl.exe	module.228.1a233c8.10000000.dll	360,448

2.3.3.4.2 SHA1 and fuzzy hashes

Although 14 of the 15 instance of *mfc42ul.dll* were successfully dumped from the memory image, it was expected that they would be identical to one another. However, after having generated their SHA1 hashes using the command *sha1sum*, it was established that they were not at all the same. In fact, each instance was different from every other instance, as per the following table of their generated SHA1 hashes:

Table 17: SHA1 hashes for Dlldump-acquired instances of mfc42ul.dll (sorted by filename).

Dumped Data Filename	SHA1 Hash
module.184.18d63d0.390000.dll	b63af9f45fbe0e1380b8fd5143d46a468fc6e9c8
module.192.1a0b478.390000.dll	0d02b0ed777028ec352d1727927303da69508585
module.228.1a233c8.10000000.dll	81db53bf63354958ac752efab57314b9a8475493
module.400.17e7be0.10000000.dll	2e7f329664f6d207a96e0d3402c9e3412e8072de
module.544.19a34b0.10000000.dll	e09c7af8150630eaeedf5cb12054fdd84a4d84bc
module.676.18da020.10000000.dll	23b69ffa9fa58c1f4581eecb6dce633350123d76
module.688.15c4020.10000000.dll	80c2ecf7951c989d524f3681fa6b0a0c3e735923
module.832.1972ca8.10000000.dll	31a699364b70d0c7cf5486375e99dd6e9b3a9d4b
module.848.187e9d0.10000000.dll	c4d65618358516df348852b692f2ded6b586870a
module.964.18c6da0.10000000.dll	57f9b685009d3b52f5e2a53fe89c8b96c251a283
module.1260.19937e0.10000000.dll	0734a8e1b7773e91818bafdc0d4cef25661adf56
module.1444.1954990.10000000.dll	437e81e7e0d09750a920cdbeda8fb7cef67235af

Dumped Data Filename	SHA1 Hash
module.1920.17c4da0.10000000.dll	1a497bfb06f68611ddd7e8c6dd72a41f4f604531
module.1956.15bcda0.10000000.dll	dc1b40e584e6ab887bc04e39331e28c56a5c78a8

Using fuzzy hashes it will be possible to determine the extent of the similarities between the dumped memory samples, if any. To generate fuzzy-based hashes, the *ssdeep* command was used. The following table lists their similarities:

Table 18: Fuzzy hash matching of acquired mfc42ul.dll instances (sorted by %).

Matched File #1	Matched File #2	Match (in %)
module.192.1a0b478.390000.dll	module.184.18d63d0.390000.dll	(100)
module.676.18da020.10000000.dll	module.1260.19937e0.10000000.dll	(100)
module.832.1972ca8.10000000.dll	module.228.1a233c8.10000000.dll	(100)
module.848.187e9d0.10000000.dll	module.1260.19937e0.10000000.dll	(100)
module.848.187e9d0.10000000.dll	module.676.18da020.10000000.dll	(100)
module.964.18c6da0.10000000.dll	module.1260.19937e0.10000000.dll	(100)
module.964.18c6da0.10000000.dll	module.676.18da020.10000000.dll	(100)
module.964.18c6da0.10000000.dll	module.848.187e9d0.10000000.dll	(100)
module.228.1a233c8.10000000.dll	module.1920.17c4da0.10000000.dll	(97)
module.688.15c4020.10000000.dll	module.544.19a34b0.10000000.dll	(97)
module.832.1972ca8.10000000.dll	module.1260.19937e0.10000000.dll	(97)
module.832.1972ca8.10000000.dll	module.676.18da020.10000000.dll	(97)
module.848.187e9d0.10000000.dll	module.832.1972ca8.10000000.dll	(97)
module.964.18c6da0.10000000.dll	module.832.1972ca8.10000000.dll	(97)
module.1444.1954990.10000000.dll	module.1260.19937e0.10000000.dll	(96)
module.1920.17c4da0.10000000.dll	module.1260.19937e0.10000000.dll	(96)
module.1920.17c4da0.10000000.dll	module.1444.1954990.10000000.dll	(96)
module.228.1a233c8.10000000.dll	module.1260.19937e0.10000000.dll	(96)
module.228.1a233c8.10000000.dll	module.1444.1954990.10000000.dll	(96)
module.400.17e7be0.10000000.dll	module.1260.19937e0.10000000.dll	(96)
module.400.17e7be0.10000000.dll	module.1444.1954990.10000000.dll	(96)
module.400.17e7be0.10000000.dll	module.1920.17c4da0.10000000.dll	(96)
module.400.17e7be0.10000000.dll	module.228.1a233c8.10000000.dll	(96)
module.544.19a34b0.10000000.dll	module.1260.19937e0.10000000.dll	(96)
module.544.19a34b0.10000000.dll	module.1444.1954990.10000000.dll	(96)
module.544.19a34b0.10000000.dll	module.1920.17c4da0.10000000.dll	(96)

Matched File #1	Matched File #2	Match (in %)
module.544.19a34b0.10000000.dll	module.228.1a233c8.10000000.dll	(96)
module.544.19a34b0.10000000.dll	module.400.17e7be0.10000000.dll	(96)
module.676.18da020.10000000.dll	module.1444.1954990.10000000.dll	(96)
module.676.18da020.10000000.dll	module.1920.17c4da0.10000000.dll	(96)
module.676.18da020.10000000.dll	module.228.1a233c8.10000000.dll	(96)
module.676.18da020.10000000.dll	module.400.17e7be0.10000000.dll	(96)
module.676.18da020.10000000.dll	module.544.19a34b0.10000000.dll	(96)
module.688.15c4020.10000000.dll	module.1260.19937e0.10000000.dll	(96)
module.688.15c4020.10000000.dll	module.1444.1954990.10000000.dll	(96)
module.688.15c4020.10000000.dll	module.1920.17c4da0.10000000.dll	(96)
module.688.15c4020.10000000.dll	module.228.1a233c8.10000000.dll	(96)
module.688.15c4020.10000000.dll	module.400.17e7be0.10000000.dll	(96)
module.688.15c4020.10000000.dll	module.676.18da020.10000000.dll	(96)
module.832.1972ca8.10000000.dll	module.1444.1954990.10000000.dll	(96)
module.832.1972ca8.10000000.dll	module.1920.17c4da0.10000000.dll	(96)
module.832.1972ca8.10000000.dll	module.400.17e7be0.10000000.dll	(96)
module.832.1972ca8.10000000.dll	module.544.19a34b0.10000000.dll	(96)
module.832.1972ca8.10000000.dll	module.688.15c4020.10000000.dll	(96)
module.848.187e9d0.10000000.dll	module.1444.1954990.10000000.dll	(96)
module.848.187e9d0.10000000.dll	module.1920.17c4da0.10000000.dll	(96)
module.848.187e9d0.10000000.dll	module.228.1a233c8.10000000.dll	(96)
module.848.187e9d0.10000000.dll	module.400.17e7be0.10000000.dll	(96)
module.848.187e9d0.10000000.dll	module.544.19a34b0.10000000.dll	(96)
module.848.187e9d0.10000000.dll	module.688.15c4020.10000000.dll	(96)
module.964.18c6da0.10000000.dll	module.1444.1954990.10000000.dll	(96)
module.964.18c6da0.10000000.dll	module.1920.17c4da0.10000000.dll	(96)
module.964.18c6da0.10000000.dll	module.228.1a233c8.10000000.dll	(96)
module.964.18c6da0.10000000.dll	module.400.17e7be0.10000000.dll	(96)
module.964.18c6da0.10000000.dll	module.544.19a34b0.10000000.dll	(96)
module.964.18c6da0.10000000.dll	module.688.15c4020.10000000.dll	(96)
module.1956.15bcda0.10000000.dll	module.1260.19937e0.10000000.dll	(57)
module.1956.15bcda0.10000000.dll	module.1444.1954990.10000000.dll	(57)
module.1956.15bcda0.10000000.dll	module.1920.17c4da0.10000000.dll	(57)
module.228.1a233c8.10000000.dll	module.1956.15bcda0.10000000.dll	(57)

Matched File #1	Matched File #2	Match (in %)
module.400.17e7be0.10000000.dll	module.1956.15bcda0.10000000.dll	(57)
module.544.19a34b0.10000000.dll	module.1956.15bcda0.10000000.dll	(57)
module.676.18da020.10000000.dll	module.1956.15bcda0.10000000.dll	(57)
module.688.15c4020.10000000.dll	module.1956.15bcda0.10000000.dll	(57)
module.832.1972ca8.10000000.dll	module.1956.15bcda0.10000000.dll	(57)
module.848.187e9d0.10000000.dll	module.1956.15bcda0.10000000.dll	(57)
module.964.18c6da0.10000000.dll	module.1956.15bcda0.10000000.dll	(57)

Of the 14 acquired instances of *mfc42ul.dll*, each has been determined as partially matching at least one or more instances of the DLL, as per the above table. However, while some of the above-listed matches have been identified as 100% similar to other instances their SHA1 hashes tell a different story. Thus, those matches listed as 100%, while very similar (statistically close to 100%), were inevitably different by perhaps only a few bytes.

The reason none of the instances of *mfc42ul.dll* had the same SHA1 hash is because each DLL was partially fragmented in memory, due to the way they were loaded[4] into memory. When they were dumped to disk using the plugin, fragmentation was not taken into account; instead, they were dumped as they were found in memory.

Comparing the SHA1 hashes of the 14 dumped DLLs to those of the carved data files resulted in no matches. Moreover, comparing these files against the NSRL 2.41 hash-set revealed no matches.

While comparing the fuzzy hashes of the 14 dumped files to those of the carved data files, the following matches were identified:

Table 19: Fuzzy hashes for Dlldump-acquired mfc42ul.dll instances vs. carved data files.

Mfc42ul.dll Instance	Carved Data File	Match (in %)
module.1260.19937e0.10000000.dll	recup_dir.1/f0215376.dll	(30)
module.1444.1954990.10000000.dll	recup_dir.1/f0215376.dll	(30)
module.1920.17c4da0.10000000.dll	recup_dir.1/f0215376.dll	(30)
module.1956.15bcda0.10000000.dll	recup_dir.1/f0215376.dll	(25)
module.228.1a233c8.10000000.dll	recup_dir.1/f0215376.dll	(30)
module.400.17e7be0.10000000.dll	recup_dir.1/f0215376.dll	(30)
module.544.19a34b0.10000000.dll	recup_dir.1/f0215376.dll	(30)
module.676.18da020.10000000.dll	recup_dir.1/f0215376.dll	(30)

[4] Computer memory is based on pages. These pages are not only continuously swapped in and out of memory, as per the Windows virtual memory manager, thereby contributing to fragmentation, but as memory is used and released, free memory pages inevitably become fragmented as data is loaded/unloaded into them.

Mfc42ul.dll Instance	Carved Data File	Match (in %)
module.688.15c4020.10000000.dll	recup_dir.1/f0215376.dll	(30)
module.832.1972ca8.10000000.dll	recup_dir.1/f0215376.dll	(30)
module.848.187e9d0.10000000.dll	recup_dir.1/f0215376.dll	(30)
module.964.18c6da0.10000000.dll	recup_dir.1/f0215376.dll	(30)

Upon examining Table 19 and then comparing it against the scanner results obtained from the carved data files (see Section 2.2.4), it turns out that carved data file *recup_dir.1/f0215376.dll* was detected as infected by the Avast and F-Prot scanners (as found in Annex B). More specifically, this carved data file was detected as R2D2 by Avast.

Finally, the SHA1 and fuzzy hashes of the 14 dumped instances of *mfc42ul.dll* were compared against those of the *malfind*-dumped memory samples, but no matches were found.

2.3.3.4.3 AV scanning

After an exhaustive analysis using the aforementioned AV scanners, it can be definitively determined that this DLL and its instances are not only malicious but are responsible for the R2D2 infection of this computer system, as based on the following information:

Table 20: AV scanner results for mfc42ul.dll instances.

AV Scanner	Detection of infection
Avast	All 14 instances detected as infected
AVG	All 14 instances detected as infected
BitDefender	All 14 instances detected as infected
Comodo	All 14 instances detected as infected
F-Prot	Found no infections
McAfee	All 14 instances detected as infected

Scanner log specifics can be found in Annex D. Based on these results, it is almost certain that the botnet was emanating from these DLLs and that they were conducting DLL injection.

2.3.3.4.4 Summary

There is no doubt that the recovered DLL instances were responsible for the R2D2 infection of this system. This subsection has examined how the DLL instances were dumped and analysed through AV scanning and hashing (SHA1 and fuzzy) against the NSRL hash-set, carved data files and *malfind*-dumped memory samples.

Although some may have preferred to have dumped all memory resident DLLs and then validate them in the same manner, many additional hours of analysis would have been required.

2.3.3.5 Moddump plugin

2.3.3.5.1 Running the plugin

Volatility's *moddump* plugin was specifically designed to dump drivers from memory to disk. If memory address offsets are specified then the *Start* address found in Table 12 obtained from the *driverscan* plugin should be used.

To dump driver *malware/winsys32.sys* (see Section 2.3.2.9 for details) from the memory image to disk, the following command was used:

```
$ volatility -f 0zpaftis moddump -b 0xf9eb4000 --dump-dir=moddump
```

The dumped driver, *winsys32.sys* had the following metadata:

Table 21: Metadata concerning Moddump-specific driver winsys32.sys.

Filename	moddump/driver.f9eb4000.sys
Size	5,376 bytes
SHA1 hash	47628778e0cd821038e92ef83bd36979830f4871
Fuzzy hash	96:xu+J4szsciPxsYJDCs9OODexF+CLUpXfMg:Lrzkxsmt5CAEg

2.3.3.5.2 SHA1 and fuzzy hashes

The SHA1 and fuzzy hashes for driver *winsys32.sys* were compared against both the SHA1 and fuzzy hashes of the carved data files. A SHA1 and fuzzy hash match of 100% were obtained for carved data file *f0181456.exe*, indicating that the files are identical.

Comparing the dumped driver's SHA1 hash against the NSRL 2.41 hash-set resulted in no matches.

The SHA1 and fuzzy hashes of the dumped driver were compared against those of the *malfind*-dumped memory samples, but no matches were found. Finally, the dumped driver's SHA1 and fuzzy hashes were compared against those of the dumped DLLs but again no matches were identified.

2.3.3.5.3 AV scanning

When using the aforementioned AV scanners against the dumped driver, four of the six scanners detected it as infected. Specifics are listed in the following table:

Table 22: AV scanner detection of Moddump-based driver winsys32.sys.

Scanner	Detected as
Avast	Win32:R2D2-F [Trj]
AVG	Trojan horse BackDoor.Badbot.C
BitDefender	N/A
Comodo	Backdoor.Win32.R2D2.~C
F-Prot	N/A
Mcafee	BackDoor-FCA!sys trojan

Based on these scanner results, there is little doubt that this driver is related to the R2D2 infection and the aforementioned malicious DLL, *mfc42ul.dll*.

2.3.3.5.4 Summary

Although some may have preferred to dump all drivers from the memory image and then validate them through AV scanning and hash analysis, this would have introduced a great deal of analytical overhead.

There is no doubt that the recovered driver is related to the R2D2 infection. This subsection examined how the driver was dumped and analysed with respect to its relationship with the carved data files, NSRL hash-set, *malfind*-dumped memory samples and dumped DLL instances.

2.3.3.6 Summary and analysis

This step followed up on investigative clues determined through the application of various Volatility plugins used in Step 2 (see Section 2.3.2). These plugins provided sufficient indications that an attempt to dump, then subsequently scan and hash the suspicious DLL and driver would prove beneficial in determining the source of the infection.

It turns out that the infection was caused by both a malicious DLL and driver, specifically m*fc42ul.dll* and *winsys32.sys*. They are a part of the same underlying infection but each likely carries out different tasks.

In this step, three specific Volatility plugins were used. The first, the *malfind* plugin, did not succeed in detecting or dumping any maliciously injected code, thereby indicating that no such detectable code or infection mechanism was in use in this particular investigation.

The *dlllist* plugin was used to determine not only the memory address of PID 1956's instance of *mfc42ul.dll* but identified, in all, 15 such instances in the memory image. Then, using the *dlldump* plugin, 14 of these instances were successfully dumped to disk. It was determined that these dumped DLL instances were not identical, but were similar to one another to varying degrees. Moreover, some of them matched certain carved data files. Based on scanner analyses, there is no doubt that these DLLs are part of the R2D2 infection.

The *moddump* plugin was used to dump malicious driver *winsys32.sys* from memory. Analysis has revealed it to be a part of this infection and that it is a 100% match to carved data file *f0181456.exe*, indicating that data carving can be useful to help triage a memory image prior to conducting in-depth memory analysis.

There was no point in pursuing the use of the *memdump*, *procexedump* or *procmemdump* plugins, as was done in [1][2] as the clues and evidence found in Step 2 did not provide any indication that the processes were themselves directly infected.

2.3.4 Registry

The Windows registry serves to both complicate and facilitate the investigator's work. It is commonly used by malware to configure system settings for permanent infection. However, the difficulty in working with the registry lies in knowing where to look. The registry is spread out across many data files (commonly known as registry hives) in various locations and each serves a specific purpose with respect to system, application and user configurations. Annex E provides a listing of registry keys commonly used by malware. The list has had several entries added to it since report [2].

2.3.4.1 Hivelist plugin

The purpose of using the *hivelist* plugin is to determine which registry hives[5] are available in the memory image.

Consider the plugin's output, using command "*volatility -f 0zapftis.vmem hivelist*":

Table 23: Volatility Hivelist plugin output.

Virtual Address	Physical Address	Filename and Location
0x8066e904	0x0066e904	[no name]
0xe1bf6b60	0x0af3cb60	\Device\HarddiskVolume1\Documents and Settings\Administrator\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
0xe1bb2b60	0x0accab60	\Device\HarddiskVolume1\Documents and Settings\Administrator\NTUSER.DAT
0xe1a4db60	0x08b7cb60	\Device\HarddiskVolume1\Documents and Settings\LocalService\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
0xe1991b60	0x07d9ab60	\Device\HarddiskVolume1\Documents and Settings\LocalService\NTUSER.DAT
0xe1844458	0x07741458	\Device\HarddiskVolume1\Documents and Settings\NetworkService\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat

[5] A registry hive denotes the actual disk file and its location on disk.

Virtual Address	Physical Address	Filename and Location
0xe183e008	0x076b8008	\Device\HarddiskVolume1\Documents and Settings\NetworkService\NTUSER.DAT
0xe1544b60	0x05c63b60	\Device\HarddiskVolume1\WINDOWS\system32\config\software
0xe154db60	0x05c6fb60	\Device\HarddiskVolume1\WINDOWS\system32\config\SAM
0xe154d008	0x05c6f008	\Device\HarddiskVolume1\WINDOWS\system32\config\default
0xe1544008	0x05c63008	\Device\HarddiskVolume1\WINDOWS\system32\config\SECURITY
0xe13b5a40	0x02463a40	[no name]
0xe1018388	0x020bf388	\Device\HarddiskVolume1\WINDOWS\system32\config\system
0xe1008b60	0x020c3b60	[no name]

2.3.4.2 Printkey plugin

Using all proposed registry keys identified in Annex E, 1078 Volatility *printkey* commands were issued via a script to query the memory image for information pertaining to traces of this malware's activities. Building such a script takes only a few minutes. Based on the physical memory addresses listed in the above table, used in conjunction with various command line tools including *cat*, *awk* and *sed*, it is quickly assembled.

All output generated by the script was captured and stored to a text file for subsequent analysis.

After running the script, the following pertinent information concerning driver *winsys32.sys* was found:

```
Registry: User Specified
Key name: 0000 (S)
Last updated: 2011-10-10 17:03:55

Values:
REG_SZ        Service         : (S) malware
REG_SZ          ClassGUID         :  (S)  {8ECC055D-047F-11D1-A537-
0000F8753ED1}
REG_SZ        DeviceDesc      : (S) malware2
Legend: (S) = Stable   (V) = Volatile

----------------------------
Registry: User Specified
Key name: malware (S)
Last updated: 2011-10-10 17:03:55

Values:
REG_DWORD     Type            : (S) 1
REG_EXPAND_SZ    ImagePath                             :     (S)
\??\C:\WINDOWS\system32\drivers\winsys32.sys
REG_SZ        DisplayName     : (S) malware2
```

The malware driver *winsys32.sys* was found in registry keys *ControlSet001\Enum\Root\LEGACY_malware\0000* and *ControlSet001\services\malware* at memory address *0xe1018388*.

After running the script, the following pertinent information concerning DLL *mfc42ul.dll* was found:

```
Registry: User Specified
Key name: Windows (S)
Last updated: 2011-10-10 16:56:35

Subkeys:

Values:
REG_SZ          AppInit_DLLs    : (S) mfc42ul.dll
REG_SZ          DeviceNotSelectedTimeout : (S) 15
REG_DWORD       GDIProcessHandleQuota : (S) 10000
REG_SZ          Spooler         : (S) yes
REG_SZ          swapdisk        : (S)
REG_SZ          TransmissionRetryTimeout : (S) 90
REG_DWORD       USERProcessHandleQuota : (S) 10000
```

The various instances of DLL *mfc42ul.dll* were loaded from registry key *Microsoft\Windows NT\CurrentVersion\Windows* at memory address *0xe1544b60*.

Thus, the persistence of this infection was made possible through the Windows registry.

2.3.4.3 Userassist plugin

The final registry-based Volatility plugin run against the memory image was *userassist*. This plugin has the potential to provide, among other things, registry-based information pertaining to programs run and files opened by the user.

Unfortunately, this plugin did not result in any useful information concerning the infection.

2.3.5 Step 5: Miscellaneous

This final step examines two additional lines of inquiry not examined in the author-proposed methodology (see Section 1.7 for details) as they are optional and might only be of occasional use.

More specifically, it may be possible to determine, at least partially, the capabilities of the malicious driver and perhaps the encryption key used by the DLL to secure its covert communications.

2.3.5.1 Devicetree

The Volatility *devicetree* plugin is used to determine the relationship between drivers and their required Windows devices. In so doing, it may be possible to determine what device, and hence

purpose, of a malicious. Running command "*volatility -f 0zapftis.vmem devicetree*," after pruning, generated the following output:

```
DRV 0x01a498b8 \Driver\malware
---| DEV 0x816c8d80 KeyboardClassC FILE_DEVICE_KEYBOARD
```

Based on this output, the malicious driver *malware* requires a keyboard-based device. The only logical reason for this is that the driver is a keyboard logger and by having direct access to this device it will be able to record user keystrokes.

2.3.5.2 Extract encryption keys

It has been established that the R2D2 infection relies on not only a covert communication channel (see Section 2.3.2.2 for details) but that it also encrypts its communications using ECB-based AES encryption [8][24] and [25]. While references [8][24] and [25] list the actual encryption key used for AES encryption, an investigator should know how to find and extract these keys.

Two readily useable FOSS-based encryption detection and extraction-based software includes *aeskeyfind*[6] and *interrogate*[7]. Both tools are easy to use. Running either command will reveal that the AES encryption key in use by this infection is readily identifiable. The AES key has been identified as:

```
4903930819949694289383046828a8f50ab994024581931fbcd7f3ad93f53293
```

This key corroborates the information found in [8][24] and [25]

2.3.5.3 Summary and analysis

Although this step was brief, it was demonstrated that AES encryption can be detected and extracted from memory. In so doing, it has been confirmed that the malware infecting this memory image is in fact R2D2.

However, encryption key detection is not limited to only AES but can include additional forms including RSA, recovered using *aeskeyfind* and *interrogate*, while others such as BitLocker and PGP can be recovered using Passware and Elcomsoft.

Finally, through the Volatility *devicetree* plugin, it was possible to discern some of the capabilities of the device driver. It was determined that device driver *malware* was in fact a keylogger.

[6] Aeskeyfind can be found at https://citp.princeton.edu/research/memory/code/.
[7] Interrogate can be found at https://github.com/carmaa/interrogate.

3 Conclusion

What can be concluded from this work is that using sound investigative footwork, combined with the capabilities of the Volatility memory analysis framework, investigators can readily analyse and investigate suspected memory-based infections.

Trojan horse 0zapftis (R2D2) was the most difficult to investigate to date in this series of reports. It was well hidden and required a keen attention to details in order to isolate it. Although it was possible to dump the malware from memory using the *dlldump* and *moddump* plugins, getting to this stage was not obvious. Although documentation concerning the infection was sufficient to learn more about what was left behind in memory, this case was solved without it.

Upon having dumped both the driver and malicious DLL the cause of the infection has been isolated. The memory image was infected by a botnet that was controlling various aspects of the system. The DLL was loaded with various processes due to the use of registry key *Microsoft\Windows NT\CurrentVersion\Windows*. Once the DLL was loaded using this registry key, it could then begin injecting itself. It was later determined that the malicious device driver was in fact a keylogger.

Throughout this document, based on the proposed methodology as put forward in [2], the author has demonstrated the manner in which a forensic memory analysis can be conducted by non-memory specialists. Thus, even novice memory investigators can successfully conduct difficult memory analyses, when armed with straightforward tools, techniques and methodology.

Not all analyses will be able to rely on well-prepared malware reports. This is why this investigation did not make direct use of them during the analysis of the memory image. Moreover, as with previous analyses [1][2], no use was made of existing analyses of this memory image. The techniques and methodology presented herein will be of use, to varying extents, newer and more difficult to analyse malware.

This document, the third in a series of many, has guided the reader through a difficult to find malware composed of a driver and multiple instances of a malicious DLL. It is hoped that other similar reports will continue to be possible in order to continue building a sufficient compendium of knowledge for memory analysis for use by novice and expert memory analysts alike. While the degree of difficulty varies substantially from case to case, the Volatility framework, when combined with investigative knowhow, tools, techniques and methodology is a highly adept analysis-based framework.

This page intentionally left blank.

References

[1] Carbone, Richard. Malware memory analysis for non-specialists: Investigating a publicly available memory image of the Zeus Trojan horse. Technical Memorandum. Defence R&D Canada – Valcartier. TM 2013-018. April 2013.

[2] Carbone, Richard. Malware memory analysis for non-specialists: Investigating publicly available memory images for Prolaco and SpyEye. Technical Memorandum. Defence R&D Canada – Valcartier. TM 2013-155. October 2013.

[3] Carbone, Richard. File recovery and data extraction using automated data recovery tools: A balanced approach using Windows and Linux when working with an unknown disk image and filesystem. Technical memorandum. TM 2009-161. Defence R&D Canada - Valcartier. January 2013. http://cradpdf.drdc-rddc.gc.ca/PDFS/unc122/p531895_A1b.pdf.

[4] http://gbata.org/wp-content/uploads/2013/06/KeynoteSpeakers-2013.pdf.

[5] Elsevier. Network Security. Newsletter. ISSN 1353-4858. Elsevier. October 2011. http://www.secniche.org/released/NESE_FRAME_AKS_RJE.pdf.

[6] Cluley, Graham. 'Government' backdoor R2D2 Trojan discovered by Chaos Computer Club. Blog. NakedSecurity/Sophos. October 2011. http://nakedsecurity.sophos.com/2011/10/09/government-backdoor-trojan-chaos/.

[7] Gorman, Gavin O. Backdoor.R2D2: The Long Arm of the Law? Blog. Symantec. October 2011. http://www.symantec.com/connect/blogs/backdoorr2d2-long-arm-law.

[8] Chaos Computer Club. Analyse Einer Regierungs – Malware. Technical report. Chaos Computer Club. October 2011. http://www.ccc.de/system/uploads/76/original/staatstrojaner-report23.pdf.

[9] Network Security Investigations. R2D2 Dropper. Sandbox incidence report. Evild3ad.com. Unknow date. http://www.evild3ad.com/Downloads/R2D2-Dropper/NSI-Sandbox.pdf.

[10] Farivar, Cyrus. German company behind govermment spyware admits sale to Bavaria. Online news article. DW. October 2011. http://www.dw.de/german-company-behind-government-spyware-admits-sale-to-bavaria/a-15453150-1.

[11] Rieger, Frank. Anatomy of a digital pest. Technical paper. Feuilleton. October 2011. http://www.edge.org/3rd_culture/FAZ2011/Trojaner_englisch.pdf.

[12] Network World. German Federal Trojan (0zapftis/Bundestrojaner) Eavesdrops On Skype, IE, Firefox, MSN Messenger & More. Online article. October 2011. http://www.darknet.org.uk/2011/10/german-federal-trojan-0zapftisbundestrojaner-eavesdrops-on-skype-ie-firefox-msn-messenger-more/.

[13] Meyer, Julien. R2D2, analyse d'un cheval de Troie gouvernemental. Journal article. Actu Sécu/XMCO. April 2012. http://www.xmco.fr/actu-secu/XMCO-ActuSecu-31-R2D2-Pharmacies_fictives.pdf.

[14] Dewald, Andreas; Freiling, Felix C. et al. Analyse und Vergleich von BckR2D2-I und II. Journal article. Universität Mannheim, Friedrich-Alexander Universität Erlangen-Nürnberg, Ruhr-Universität Bochum. Journal: Lecture Notes in Informatics. March 2012. ISBN: 978-88579-289-5. http://subs.emis.de/LNI/Proceedings/Proceedings195/P-195.pdf.

[15] Volatility. CommandReference: Example usage cases and output for Volatility 2.0 commands. Online command reference. Volatility. February 2012. http://code.google.com/p/volatility/wiki/CommandReference.

[16] Volatility. CommandReference23: Example usage cases and output for Volatility 2.3 commands. Online command reference. Volatility. Unknown date. http://code.google.com/p/volatility/wiki/CommandReference23.

[17] SpeedGuide Inc. Port 6666 Details. Informational article. SpeedGuide Inc. 2013. http://www.speedguide.net/port.php?port=6666.

[18] Wikipedia. Chaos Computer Club. Online encyclopaedic entry. Wikimedia Foundation Inc. September 2013. http://en.wikipedia.org/wiki/Chaos_Computer_Club.

[19] Chaos Computer Club. Addendum Staatstrojaner. Informational article. October 2011. Chaos Computer Club. http://www.ccc.de/de/updates/2011/addendum-staatstrojaner.

[20] DW. Several German states admit to use of controversial spy software. Online news article. DW. October 2011. http://www.dw.de/several-german-states-admit-to-use-of-controversial-spy-software/a-15449054-1.

[21] DW. German company behind government spyware admits sale to Bavaria. Online news article. DW. October 2011. http://www.dw.de/german-company-behind-government-spyware-admits-sale-to-bavaria/a-15453150.

[22] Privacy International. DigiTask. Online article. 2012. Privacy International. http://bigbrotherinc.org/v1/Germany/DigiTask/.

[23] Thomas, Michael. Remote Forensic Software. Presentation. Digitask GmbH. September 2008. http://cryptome.org/0005/michaelthomas.pdf.

[24] Cisco. Backdoor:W32/R2D2.A. Informational article. Cisco. October 2011. http://tools.cisco.com/security/center/viewAlert.x?alertId=24352.

[25] Laskov, Pavel. Intrusion Detection and Malware Analysis: Course Introduction/Overview of Security Threats. Course presentation. Wilhelm Schickard Institute for Computer Science. October 2011. http://www.cogsys.cs.uni-tuebingen.de/lehre/ws11/ids-malware/01-intro.pdf.

Annex A Volatility Windows-based plugins

The following is a complete list of the default Windows-based plugins provided with Volatility version 2.2:

Table A.1: List of Volatility 2.2 plugins.

Plugin	**Capability (as per *Volatility --help* output)**
apihooks	Detect API hooks in process and kernel memory
atoms	Print session and window station atom tables
atomscan	Pool scanner for _RTL_ATOM_TABLE
bioskbd	Reads the keyboard buffer from Real Mode memory
callbacks	Print system-wide notification routines
clipboard	Extract the contents of the windows clipboard
cmdscan	Extract command history by scanning for _COMMAND_HISTORY
connections	Print list of open connections [Windows XP and 2003 Only]
connscan	Scan Physical memory for _TCPT_OBJECT objects (tcp connections)
consoles	Extract command history by scanning for _CONSOLE_INFORMATION
crashinfo	Dump crash-dump information
deskscan	Poolscaner for tagDESKTOP (desktops)
devicetree	Show device tree
dlldump	Dump DLLs from a process address space
dlllist	Print list of loaded dlls for each process
driverirp	Driver IRP hook detection
driverscan	Scan for driver objects _DRIVER_OBJECT
envars	Display process environment variables
eventhooks	Print details on windows event hooks
evtlogs	Extract Windows Event Logs (XP/2003 only)
filescan	Scan Physical memory for _FILE_OBJECT pool allocations
gahti	Dump the USER handle type information

Plugin	**Capability (as per *Volatility --help* output)**
gditimers	Print installed GDI timers and callbacks
gdt	Display Global Descriptor Table
getservicesids	Get the names of services in the Registry and return Calculated SID
getsids	Print the SIDs owning each process
handles	Print list of open handles for each process
hashdump	Dumps passwords hashes (LM/NTLM) from memory
hibinfo	Dump hibernation file information
hivedump	Prints out a hive
hivelist	Print list of registry hives.
hivescan	Scan Physical memory for _CMHIVE objects (registry hives)
idt	Display Interrupt Descriptor Table
imagecopy	Copies a physical address space out as a raw DD image
imageinfo	Identify information for the image
impscan	Scan for calls to imported functions
kdbgscan	Search for and dump potential KDBG values
kpcrscan	Search for and dump potential KPCR values
ldrmodules	Detect unlinked DLLs
lsadump	Dump (decrypted) LSA secrets from the registry
malfind	Find hidden and injected code
memdump	Dump the addressable memory for a process
memmap	Print the memory map
messagehooks	List desktop and thread window message hooks
moddump	Dump a kernel driver to an executable file sample
modscan	Scan Physical memory for _LDR_DATA_TABLE_ENTRY objects
modules	Print list of loaded modules
mutantscan	Scan for mutant objects _KMUTANT
patcher	Patches memory based on page scans

Plugin	**Capability (as per *Volatility --help* output)**
printkey	Print a registry key, and its subkeys and values
procexedump	Dump a process to an executable file sample
procmemdump	Dump a process to an executable memory sample
pslist	Print all running processes by following the EPROCESS lists
psscan	Scan Physical memory for _EPROCESS pool allocations
pstree	Print process list as a tree
psxview	Find hidden processes with various process listings
raw2dmp	Converts a physical memory sample to a windbg crash dump
screenshot	Save a pseudo-screenshot based on GDI windows
sessions	List details on _MM_SESSION_SPACE (user logon sessions)
shimcache	Parses the Application Compatibility Shim Cache registry key
sockets	Print list of open sockets
sockscan	Scan Physical memory for _ADDRESS_OBJECT objects (tcp sockets)
ssdt	Display SSDT entries
strings	Match physical offsets to virtual addresses (may take a while, VERY verbose)
svcscan	Scan for Windows services
symlinkscan	Scan for symbolic link objects
thrdscan	Scan physical memory for _ETHREAD objects
threads	Investigate _ETHREAD and _KTHREADs
timers	Print kernel timers and associated module DPCs
userassist	Print userassist registry keys and information
userhandles	Dump the USER handle tables
vaddump	Dumps out the vad sections to a file
vadinfo	Dump the VAD info
vadtree	Walk the VAD tree and display in tree format
vadwalk	Walk the VAD tree
volshell	Shell in the memory image

Plugin	**Capability (as per *Volatility --help* output)**
windows	Print Desktop Windows (verbose details)
wintree	Print Z-Order Desktop Windows Tree
wndscan	Pool scanner for tagWINDOWSTATION (window stations)
yarascan	Scan process or kernel memory with Yara signatures

Annex B Anti-virus scanner logs for carved data files

This following are the anti-virus scanner logs for the carved data files, carried out in Section 2.2.4.

In all, two virus matches were identified between the various scanners. These matches are indicated below.

B.1 Avast

./recup_dir.1/f0181456.exe [infected by: Win32:R2D2-F [Trj]] **<- Match 2**
./recup_dir.1/f0215376.dll [infected by: Win32:R2D2-E [Trj]]
./recup_dir.1/f0148080.dll [infected by: Win32:R2D2-L [Trj]]

B.2 AVG

./recup_dir.2/f0443800.dll Virus found Win32/Heur
./recup_dir.1/f0141320.dll Virus found Win32/Heur
./recup_dir.1/f0149536.dll Virus found Win32/Heur
./recup_dir.1/f0140512.dll Virus found Win32/Heur
./recup_dir.1/f0140472.exe Virus found Win32/Heur **<- Match 1**
./recup_dir.1/f0148824.dll Virus found Win32/Heur
./recup_dir.1/f0148904.dll Virus found Win32/Heur

B.3 BitDefender

./recup_dir.1/f0140472.exe infected: Gen:Variant.FakeAlert.47 **<- Match 1**
./recup_dir.1/f0150032.exe infected: Gen:Variant.FakeAlert.47

B.4 Comodo

./recup_dir.1/f0181456.exe ---> Found Virus, Malware Name is Backdoor.Win32.R2D2.~C **<- Match 2**
./recup_dir.1/f0414032.exe ---> Found Virus, Malware Name is Packed.Win32.MUPX.Gen
./recup_dir.1/f0148072.dll ---> Found Virus, Malware Name is TrojWare.Win32.FraudPack.P

B.5 F-Prot

./recup_dir.2/f0501736.dll <W32/Heuristic-COC!Eldorado (not disinfectable)>
./recup_dir.1/f0009912_ntkrnlpa.exe <W32/Heuristic-CO3!Eldorado (not disinfectable)>
./recup_dir.1/f0215376.dll <W32/Heuristic-COC!Eldorado (not disinfectable)>
./recup_dir.1/f0408232.dll <W32/MalwareHiderPatched-based!M>

B.6 McAfee

./recup_dir.1/f0181456.exe ... Found the BackDoor-FCA!sys trojan !!! **<- Match 2**

Annex C NSRL file hash matches for carved data files

This annex provides a listing of those carved data files obtained in Section 2.2.3 that matched the SHA1 hashes of the NSRL hash-set 2.41 (June 2013). In all, 28 unique NSRL-based SHA1-filename matches were obtained. The NSRL hash-based filename matches are as follows:

Table C.1: SHA1 hash vs. NSRL filename for carved data files.

SHA1 hash	Filename
048ABF0A35FFFEB7A43696EFB78290C2923F6069	ICMP.DLL
09105C886A83677E49CE6EF47F8CF1A047214AED	8.0.50727.762.POLICY
09105C886A83677E49CE6EF47F8CF1A047214AED	MANIFEST.8.0.50727.762.68B7C6D9_1DF2_54C1_FF1F_C8B3B9A1E18E
09105C886A83677E49CE6EF47F8CF1A047214AED	UL_MANIFEST.68B7C6D9_1DF2_54C1_FF1F_C8B3B9A1E18E
09105C886A83677E49CE6EF47F8CF1A047214AED	X1SW1O0K.9HI
09105C886A83677E49CE6EF47F8CF1A047214AED	Z1SW1O0K.9HI
0C52F6D1FB3F253821DFB6BF4CDF7830F429F273	DXMRTP.MAN
0C52F6D1FB3F253821DFB6BF4CDF7830F429F273	X86_POLICY.5.2.MICROSOFT.WINDOWS.NETWORKING.DXMRTP_6595B64144CCF1DF_5.2.2.3_X-WW_CF59288D.MANIFEST
15740B197555BA8E162C37A60BA655151E3BEBAE	INDEX.DAT
172E07C564B2BF0DB2333A8CEBE7EC4D8F82180C	VMWAREFILTERS.TXT
2439C395AEDC84E421049775A8D2743BF6CA7AD6	GDIPLUS.MAN
2439C395AEDC84E421049775A8D2743BF6CA7AD6	X86_MICROSOFT.WINDOWS.GDIPLUS_6595B64144CCF1DF_1.0.2600.2180_X-WW_522F9F82.MANIFEST
2E058F605FF909BFACA2676D4F5A5B59D6704E59	RTCDLL.MAN
2E058F605FF909BFACA2676D4F5A5B59D6704E59	X86_POLICY.5.2.MICROSOFT.WINDOWS.NETWORKING.RTCDLL_6595B64144CCF1DF_5.2.2.3_X-WW_5F924D7B.MANIFEST
305330837DE6C91E5DBA87168653C9EFD30C8385	RTCDLL.MAN
305330837DE6C91E5DBA87168653C9EFD30C8385	X86_MICROSOFT.WINDOWS.NETWORKING.RTCDLL_6595B64144CCF1DF_5.2.2.3_X-WW_D6BD8B95.MANIFEST
3F85EC97F05C84781219F548D253BED5464FE8FF	DXMRTP.MAN
3F85EC97F05C84781219F548D253BED5464FE8FF	X86_MICROSOFT.WINDOWS.NETWORKING.DXMRTP_6595B64144CCF1DF_5.2.2.3_X-WW_468466A7.MANIFEST
40F6B4D98F237F0A1B53656659F93ECF8A249622	DEFAULT.MAN

SHA1 hash	Filename
40F6B4D98F237F0A1B53656659F93ECF8A249622	X86_POLICY.5.1.MICROSOFT.WINDOWS.SYSTEMCOMPATIBLE_6595B64144CCF1DF_5.1.2600.2000_X-WW_E037A8A.MANIFEST
50A9E047E7BFDA5A71608EDAE9086FC7C7930609	LZ32.DLL
6D239DC8A3C78670A544BEA19BB9E6B5BBE844BC	GOOGLEDESKTOP.TXT
830D6459350DD1AB3B1F070135425A93395782B1	MANIFEST.8.0.50727.762.74FD3CE6_2A8D_0E9C_FF1F_C8B3B9A1E18E
830D6459350DD1AB3B1F070135425A93395782B1	MFC80LOC_MAN.7643D2EA_8E33_4EBC_B95C_9E5DF999A535
830D6459350DD1AB3B1F070135425A93395782B1	UL_MANIFEST.74FD3CE6_2A8D_0E9C_FF1F_C8B3B9A1E18E
830D6459350DD1AB3B1F070135425A93395782B1	X86_MICROSOFT.VC80.MFCLOC_1FC8B3B9A1E18E3B_8.0.50727.762_X-WW_91481303.MANIFEST
83C15BD58DFF36C08DB093F81ECFD431C404A933	SYSTEM.INI
86B3D8696B5DA354EF42C8AB4A9D21CDAAF0DDA1	0526
86B3D8696B5DA354EF42C8AB4A9D21CDAAF0DDA1	AMD64_MICROSOFT-WINDOWS-FONT-VECTOR_31BF3856AD364E35_6.0.6000.16386_NONE_8F7C8248A32BA279_SCRIPT.FON_0C43F9EC
86B3D8696B5DA354EF42C8AB4A9D21CDAAF0DDA1	AMD64_MICROSOFT-WINDOWS-FONT-VECTOR_31BF3856AD364E35_6.0.6001.18000_NONE_91B34444A016B34D_SCRIPT.FON_0C43F9EC
86B3D8696B5DA354EF42C8AB4A9D21CDAAF0DDA1	AMD64_MICROSOFT-WINDOWS-FONT-VECTOR_31BF3856AD364E35_6.1.7600.16385_NONE_91899A68016A48BE_SCRIPT.FON_0C43F9EC
86B3D8696B5DA354EF42C8AB4A9D21CDAAF0DDA1	AMD64_MICROSOFT-WINDOWS-FONT-VECTOR_31BF3856AD364E35_6.2.8250.0_NONE_19CC16AF45E27012_SCRIPT.FON_0C43F9EC
86B3D8696B5DA354EF42C8AB4A9D21CDAAF0DDA1	AMD64_MICROSOFT-WINDOWS-FONT-VECTOR_31BF3856AD364E35_6.2.9200.16384_NONE_8E5E5025717D780E_SCRIPT.FON_0C43F9EC
86B3D8696B5DA354EF42C8AB4A9D21CDAAF0DDA1	SCRIPT.FO!
86B3D8696B5DA354EF42C8AB4A9D21CDAAF0DDA1	SCRIPT.FON
86B3D8696B5DA354EF42C8AB4A9D21CDAAF0DDA1	X86_MICROSOFT-WINDOWS-FONT-VECTOR_31BF3856AD364E35_6.0.5270.9_NEUTRAL_4E708706B2C34447_SCRIPT.FON_0C43F9EC
86B3D8696B5DA354EF42C8AB4A9D21CDAAF0DDA1	X86_MICROSOFT-WINDOWS-FONT-VECTOR_31BF3856AD364E35_6.0.5384.4_NONE_6395095C1981A3F5_SCRIPT.FON_0C43F9EC
86B3D8696B5DA354EF42C8AB4A9D21CDAAF0DDA1	X86_MICROSOFT-WINDOWS-FONT-VECTOR_31BF3856AD364E35_6.0.6000.16386_NONE_335DE6C4EACE3143_SCRIPT.FON_0C43F9EC

SHA1 hash	Filename
86B3D8696B5DA354EF42C8AB4A9D21CDAAF0DDA1	X86_MICROSOFT-WINDOWS-FONT-VECTOR_31BF3856AD364E35_6.1.7600.16385_NONE_356AFEE4490CD788_SCRIPT.FON_0C43F9EC
86B3D8696B5DA354EF42C8AB4A9D21CDAAF0DDA1	X86_MICROSOFT-WINDOWS-FONT-VECTOR_31BF3856AD364E35_6.2.8250.0_NONE_BDAD7B2B8D84FEDC_SCRIPT.FON_0C43F9EC
9537335B7EDA9AE3D1C125BE7BAC3161D5B853B8	COMCTL.MAN
9537335B7EDA9AE3D1C125BE7BAC3161D5B853B8	X86_POLICY.6.0.MICROSOFT.WINDOWS.COMMON-CONTROLS_6595B64144CCF1DF_6.0.2600.2180_X-WW_EB84B25E.MANIFEST
AD2347170F3D50C2FB6306E61972039497AE861B	RTCRES.MAN
AD2347170F3D50C2FB6306E61972039497AE861B	X86_MICROSOFT.WINDOWS.NETWORKING.RTCRES_6595B64144CCF1DF_5.2.2.3_EN_16A24BC0.MANIFEST
B4BC7F8E9991FC3C22206DD8938289979184D97E	__0X0069
C1A0C4D1030190A2D141B4E2D28D1329A8368A0A	GDIPLUS.MAN
C1A0C4D1030190A2D141B4E2D28D1329A8368A0A	X86_POLICY.1.0.MICROSOFT.WINDOWS.GDIPLUS_6595B64144CCF1DF_1.0.2600.2180_X-WW_5FF735E2.MANIFEST
C5B52B71F4C5F933815D7D606175EA0BB37DC548	CONTROLS.MAN
C5B52B71F4C5F933815D7D606175EA0BB37DC548	X86_MICROSOFT.WINDOWS.COMMON-CONTROLS_6595B64144CCF1DF_6.0.2600.2180_X-WW_A84F1FF9.MANIFEST
CFCBB8E4E0C32500FA5BF9382B103C4199DF0276	__0X0034
CFCBB8E4E0C32500FA5BF9382B103C4199DF0276	__0X0098
CFCBB8E4E0C32500FA5BF9382B103C4199DF0276	__0X041B
D10440930CC994409E920D94C7C45F0405D60422	8.0.50727.762.POLICY
D10440930CC994409E920D94C7C45F0405D60422	MANIFEST.8.0.50727.762.63E949F6_03BC_5C40_FF1F_C8B3B9A1E18E
D10440930CC994409E920D94C7C45F0405D60422	UL_MANIFEST.63E949F6_03BC_5C40_FF1F_C8B3B9A1E18E
D10440930CC994409E920D94C7C45F0405D60422	XXGS54WE.KJ4
D10440930CC994409E920D94C7C45F0405D60422	ZXGS54WE.KJ4
D88E50A3D3F232FB591010AB83C6D0DC3F820ECE	__0X005B
D9AA29288951E94773CAA1054237D29734E79F34	CSRSS.EXE
E6499F9C89A77383456BD1F54D75753350C6F91F	DEFAULT.MAN
E6499F9C89A77383456BD1F54D75753350C6F91F	X86_MICROSOFT.WINDOWS.SYSTEMCOMPATIBLE_6595B64144CCF1DF_5.1.2600.2000_X-WW_BCC9A281.MANIFEST
E9B4BDF28634345FD172530EA072CB0BDE0BAEFE	__0X0461

SHA1 hash	Filename
E9B4BDF28634345FD172530EA072CB0BDE0BAEFE	__0X0467
F081561658705610ADAD4C30E757312491EDF9E0	8.0.50727.762.POLICY
F081561658705610ADAD4C30E757312491EDF9E0	MANIFEST.8.0.50727.762.D2730D3F_3C41_5884_FF1F_C8B3B9A1E18E
F081561658705610ADAD4C30E757312491EDF9E0	UL_MANIFEST.D2730D3F_3C41_5884_FF1F_C8B3B9A1E18E
FA8B09CDEDED6B393B160F04816354863CAABFAD	VGA.DLL

Annex D Anti-virus scanner logs for dumped instances of mfc42ul.dll

The following are the scanner logs for the dumped *dlldump*-based instances of *mfc42ul.dll*.

D.1 Avast

module.184.18d63d0.390000.dll [infected by: Win32:R2D2-E [Trj]]
module.192.1a0b478.390000.dll [infected by: Win32:R2D2-E [Trj]]
module.228.1a233c8.10000000.dll [infected by: Win32:R2D2-E [Trj]]
module.400.17e7be0.10000000.dll [infected by: Win32:R2D2-E [Trj]]
module.544.19a34b0.10000000.dll [infected by: Win32:R2D2-E [Trj]]
module.676.18da020.10000000.dll [infected by: Win32:R2D2-E [Trj]]
module.688.15c4020.10000000.dll [infected by: Win32:R2D2-E [Trj]]
module.832.1972ca8.10000000.dll [infected by: Win32:R2D2-E [Trj]]
module.848.187e9d0.10000000.dll [infected by: Win32:R2D2-E [Trj]]
module.964.18c6da0.10000000.dll [infected by: Win32:R2D2-E [Trj]]
module.1260.19937e0.10000000.dll [infected by: Win32:R2D2-E [Trj]]
module.1444.1954990.10000000.dll [infected by: Win32:R2D2-E [Trj]]
module.1920.17c4da0.10000000.dll [infected by: Win32:R2D2-E [Trj]]
module.1956.15bcda0.10000000.dll [infected by: Win32:R2D2-L [Trj]]

D.2 AVG

module.184.18d63d0.390000.dll Trojan horse BackDoor.Generic14.BTVX
module.192.1a0b478.390000.dll Trojan horse BackDoor.Generic14.BTVX
module.228.1a233c8.10000000.dll Trojan horse BackDoor.Generic14.BBFR
module.400.17e7be0.10000000.dll Trojan horse BackDoor.Generic14.BBFR
module.544.19a34b0.10000000.dll Trojan horse BackDoor.Generic14.BBFR
module.676.18da020.10000000.dll Trojan horse BackDoor.Generic14.BBFR
module.688.15c4020.10000000.dll Trojan horse BackDoor.Generic14.BBFR
module.832.1972ca8.10000000.dll Trojan horse BackDoor.Generic14.BBFR
module.848.187e9d0.10000000.dll Trojan horse BackDoor.Generic14.BBFR
module.964.18c6da0.10000000.dll Trojan horse BackDoor.Generic14.BBFR
module.1260.19937e0.10000000.dll Trojan horse BackDoor.Generic14.BBFR
module.1444.1954990.10000000.dll Trojan horse BackDoor.Generic14.BBFR
module.1920.17c4da0.10000000.dll Trojan horse BackDoor.Generic14.BBFR
module.1956.15bcda0.10000000.dll Trojan horse BackDoor.Generic14.BBFR

D.3 BitDefender

module.184.18d63d0.390000.dll infected: Gen:Variant.Barys.1660

module.192.1a0b478.390000.dll infected: Gen:Variant.Barys.1660
module.228.1a233c8.10000000.dll infected: Trojan.Generic.KDV.395230
module.400.17e7be0.10000000.dll infected: Trojan.Generic.KDV.395230
module.544.19a34b0.10000000.dll infected: Trojan.Generic.KDV.395230
module.676.18da020.10000000.dll infected: Trojan.Generic.KDV.395230
module.688.15c4020.10000000.dll infected: Trojan.Generic.KDV.395230
module.832.1972ca8.10000000.dll infected: Trojan.Generic.KDV.395230
module.848.187e9d0.10000000.dll infected: Trojan.Generic.KDV.395230
module.964.18c6da0.10000000.dll infected: Trojan.Generic.KDV.395230
module.1260.19937e0.10000000.dll infected: Trojan.Generic.KDV.395230
module.1444.1954990.10000000.dll infected: Trojan.Generic.KDV.395230
module.1920.17c4da0.10000000.dll infected: Trojan.Generic.KDV.395230
module.1956.15bcda0.10000000.dll infected: Trojan.Generic.KDV.378678

D.4 Comodo

module.184.18d63d0.390000.dll ---> Found Virus, Malware Name is Backdoor.Win32.R2D2.~B1
module.192.1a0b478.390000.dll ---> Found Virus, Malware Name is Backdoor.Win32.R2D2.~B1
module.228.1a233c8.10000000.dll ---> Found Virus, Malware Name is Backdoor.Win32.R2D2.~B1
module.400.17e7be0.10000000.dll ---> Found Virus, Malware Name is Backdoor.Win32.R2D2.~B1
module.544.19a34b0.10000000.dll ---> Found Virus, Malware Name is Backdoor.Win32.R2D2.~B1
module.676.18da020.10000000.dll ---> Found Virus, Malware Name is Backdoor.Win32.R2D2.~B1
module.688.15c4020.10000000.dll ---> Found Virus, Malware Name is Backdoor.Win32.R2D2.~B1
module.832.1972ca8.10000000.dll ---> Found Virus, Malware Name is Backdoor.Win32.R2D2.~B1
module.848.187e9d0.10000000.dll ---> Found Virus, Malware Name is Backdoor.Win32.R2D2.~B1
module.964.18c6da0.10000000.dll ---> Found Virus, Malware Name is Backdoor.Win32.R2D2.~B1
module.1260.19937e0.10000000.dll ---> Found Virus, Malware Name is Backdoor.Win32.R2D2.~B1
module.1444.1954990.10000000.dll ---> Found Virus, Malware Name is Backdoor.Win32.R2D2.~B1
module.1920.17c4da0.10000000.dll ---> Found Virus, Malware Name is Backdoor.Win32.R2D2.~B1
module.1956.15bcda0.10000000.dll ---> Found Virus, Malware Name is Backdoor.Win32.R2D2.~B1

D.5 F-Prot

F-Prot did not succeed in finding any infected files.

D.6 McAfee

module.184.18d63d0.390000.dll ... Found the BackDoor-FCA trojan !!!
module.192.1a0b478.390000.dll ... Found the BackDoor-FCA trojan !!!
module.228.1a233c8.10000000.dll ... Found the BackDoor-FCA trojan !!!
module.400.17e7be0.10000000.dll ... Found the BackDoor-FCA trojan !!!
module.544.19a34b0.10000000.dll ... Found the BackDoor-FCA trojan !!!
module.676.18da020.10000000.dll ... Found the BackDoor-FCA trojan !!!
module.688.15c4020.10000000.dll ... Found the BackDoor-FCA trojan !!!

module.832.1972ca8.10000000.dll ... Found the BackDoor-FCA trojan !!!
module.848.187e9d0.10000000.dll ... Found the BackDoor-FCA trojan !!!
module.964.18c6da0.10000000.dll ... Found the BackDoor-FCA trojan !!!
module.1260.19937e0.10000000.dll ... Found the BackDoor-FCA trojan !!!
module.1444.1954990.10000000.dll ... Found the BackDoor-FCA trojan !!!
module.1920.17c4da0.10000000.dll ... Found the BackDoor-FCA trojan !!!
module.1956.15bcda0.10000000.dll ... Found the BackDoor-FCA trojan !!!

This page intentionally left blank.

Annex E Commonly used registry keys in a typical malware infection

E.1 Recommended registry keys for use with Volatility

Based on the author's own use and research of various Windows registry keys commonly used by malware, the following keys are recommended for evaluation. These keys are readily integrated into scripts using appropriate Volatility *printkey* plugin-based commands.

The reader's success in using these keys will undoubtedly vary according to the underlying Windows platform to be analysed and the malware's propensity for using the registry.

The proposed keys have been aggregated and their preceding *HKLM\Software*, *HKLM\System*, *HKCU\Software* and *HKCU* based information were stripped so that they can be readily scripted.

The following keys have been evaluated against R2D2. Two registry keys in the list below are highlighted in red because they refer to possible locations for malicious driver *winsys32.sys*.

- `Classes\Local Settings\Software\Microsoft\Windows\Shell\MuiCache`
- `Control Panel\Desktop`
- `Control Panel\Desktop\ScreenSaveActive`
- `ControlSet001\Enum\Root\LEGACY_malware\0000`
- `ControlSet001\services\malware`
- `ControlSet001\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile\AuthorizedApplications\List`
- `ControlSet002\services\malware`
- `CurrentControlSet\Control\Session Manager\AppCertDlls`
- `CurrentControlSet\Control\Session Manager\AppCompatCache\AppCompatCache`
- `CurrentControlSet\Control\Session Manager\AppCompatibility\AppCompatCache`
- `CurrentControlSet\Control\SessionManager\Memory Management`
- `CurrentControlSet\Services`
- `Microsoft\Active Setup\Installed Components`
- `Microsoft\DirectPlugin`
- `Microsoft\Internet Explorer\CustomizeSearch`
- `Microsoft\Internet Explorer\Main`
- `Microsoft\Internet Explorer\Main\Default_Page_URL`
- `Microsoft\Internet Explorer\Main\Default_Search_URL`
- `Microsoft\Internet Explorer\Main\HomeOldSP`
- `Microsoft\Internet Explorer\Main\Local Page`
- `Microsoft\Internet Explorer\Main\Search Bar`
- `Microsoft\Internet Explorer\Main\Search Page`
- `Microsoft\Internet Explorer\Main\SearchAssistant`
- `Microsoft\Internet Explorer\Main\SearchURL`
- `Microsoft\Internet Explorer\Main\Start Page`
- `Microsoft\Internet Explorer\Main\Use Search Asst`
- `Microsoft\Internet Explorer\PhishingFilter`

- Microsoft\Internet Explorer\Recovery
- Microsoft\Internet Explorer\Search
- Microsoft\Internet Explorer\Search Bar
- Microsoft\Internet Explorer\Search\CustomizeSearch
- Microsoft\Internet Explorer\Search\SearchAssistant
- Microsoft\Internet Explorer\SearchURL
- Microsoft\Internet Explorer\Toolbar
- Microsoft\Internet Explorer\TypedURLs
- Microsoft\Windows NT\CurrentVersion\Terminal Server\Install\Software\Microsoft\Windows\CurrentVersion\Run
- Microsoft\Windows NT\CurrentVersion\Terminal Server\Install\Software\Microsoft\Windows\CurrentVersion\Runonce
- Microsoft\Windows NT\CurrentVersion\Terminal Server\Install\Software\Microsoft\Windows\CurrentVersion\RunonceEx
- Microsoft\Windows NT\CurrentVersion\Windows
- Microsoft\Windows NT\CurrentVersion\Windows\AppInit_DLLs
- Microsoft\Windows NT\CurrentVersion\Windows\Load
- Microsoft\Windows NT\CurrentVersion\Winlogon
- Microsoft\Windows NT\CurrentVersion\Winlogon\Notify
- Microsoft\Windows NT\winlogon\userinit
- Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects
- Microsoft\Windows\CurrentVersion\Explorer\ComDlg32\LastVisitedMRU
- Microsoft\Windows\CurrentVersion\Explorer\ComDlg32\OpenSaveMRU
- Microsoft\Windows\CurrentVersion\Explorer\RecentDocs
- Microsoft\Windows\CurrentVersion\Explorer\RunMRU
- Microsoft\Windows\CurrentVersion\Explorer\SharedTaskScheduler
- Microsoft\Windows\CurrentVersion\Explorer\ShellExecuteHooks
- Microsoft\Windows\CurrentVersion\Explorer\UserAssist
- Microsoft\Windows\CurrentVersion\Internet Settings
- Microsoft\Windows\CurrentVersion\Internet Settings\EnableAutodial
- Microsoft\Windows\CurrentVersion\Internet Settings\EnableHttp1_1
- Microsoft\Windows\CurrentVersion\Internet Settings\MaxConnectionsPer1_0Server
- Microsoft\Windows\CurrentVersion\Internet Settings\MaxConnectionsPerServer
- Microsoft\Windows\CurrentVersion\Internet Settings\ProxyEnable
- Microsoft\Windows\CurrentVersion\Internet Settings\ProxyHttp1.1
- Microsoft\Windows\CurrentVersion\Internet Settings\ProxyOverride
- Microsoft\Windows\CurrentVersion\Internet Settings\ProxyServer
- Microsoft\Windows\CurrentVersion\Internet Settings\Zones\0
- Microsoft\Windows\CurrentVersion\Internet Settings\Zones\1
- Microsoft\Windows\CurrentVersion\Internet Settings\Zones\2
- Microsoft\Windows\CurrentVersion\Policies\Explorer\Run
- Microsoft\Windows\CurrentVersion\Run
- Microsoft\Windows\CurrentVersion\RunOnce
- Microsoft\Windows\CurrentVersion\RunOnce\Setup
- Microsoft\Windows\CurrentVersion\RunOnceEx
- Microsoft\Windows\CurrentVersion\RunServices
- Microsoft\Windows\CurrentVersion\RunServicesOnce
- Microsoft\Windows\CurrentVersion\SharedDLLs

- Microsoft\Windows\CurrentVersion\ShellServiceObjectDelayLoad
- Microsoft\Windows\CurrentVersion\URL
- Microsoft\Windows\CurrentVersion\URL\DefaultPrefix
- Microsoft\Windows\CurrentVersion\URL\Prefixes
- Microsoft\Windows\ShellNoRoam\MUICache

E.2 Scripting

These keys can be readily integrated into scripts. For example, consider the following Volatility *printkey* command:

```
$ volatility -f 0zapftis.vmem printkey -o 0xe1991b60 -K
“Microsoft\Windows\CurrentVersion\RunServices”
```

A script built using such commands requires only a few minutes to construct, based on the physical memory addresses listed in the Table 23, used in conjunction with various command line tools including *cat*, *awk* and *sed*.

E.3 Root Registry Keys

The author proposed registry keys are based on the following root registry keys:

```
HKEY_CURRENT_USER
HKEY_CURRENT_USER\Software
HKEY_LOCAL_MACHINE\Software
HKEY_LOCAL_MACHINE\System
```

This page intentionally left blank.

Bibliography

Carbone, Richard. Malware memory analysis for non-specialists: Investigating a publicly available memory image of the Zeus Trojan horse. Technical Memorandum. Defence R&D Canada – Valcartier. TM 2013-018. April 2013.

Carbone, Richard. Malware memory analysis for non-specialists: Investigating publicly available memory images for Prolaco and SpyEye. Technical Memorandum. Defence R&D Canada – Valcartier. TM 2013-155. October 2013.

Volatility. CommandReference: Example usage cases and output for Volatility 2.0 commands. Online command reference. Volatility. February 2012. http://code.google.com/p/volatility/wiki/CommandReference.

List of symbols/abbreviations/acronyms/initialisms

AES	Advanced Encryption Standard
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
AV	Anti-Virus or Antivirus
C&C	Command & Control
CCC	Chaos Computer Club
CFNOC	Canadian Forces Network Operations Centre
CORFC	Centre d'opérations des réseaux des Forces canadiennes
CTPH	Context Triggered Piecewise Hash Sometimes known as fuzzy hash or *ssdeep* hash
DLL	Dynamically Loaded Library
DND	Department of National Defence
DRDC	Defence Research & Development Canada
DRDKIM	Director Research and Development Knowledge and Information Management
DW	Deutsche Welle
ECB	Electronic Codebook
EDT	Eastern Daylight Time
EXT4	Fourth Extended Filesystem
FOSS	Free and Open Source Software
FTP	File Transfer Protocol
GICT	Groupe intégré de la criminalité technologique
GRC	Gendarmerie Royale du Canada
HKCU	HKEY_LOCAL_USER
HKLM	HKEY_LOCAL_MACHINE
ID	Identification
IP	Internet Protocol
ITCU	Integrated Technological Crime Unit
MAC	Mandatory Access Control
MiB	Mebibyte

N/A	Not Available
NIST	National Institute of Standards and Technology
NSRL	National Software Reference Library
NTP	Network Time Protocol
PAE	Physical Address Extension
PE	Portable Executable
PGP	Pretty Good Privacy
PID	Process ID
PPID	Parent Process ID
R&D	Research & Development
RAM	Random Access Memory
RCMP	Royal Canadian Mounted Police
RDDC	Recherche et Développement pour la Défense Canada
RDP	Remote Desktop Protocol
SHA1	Secure Hash Algorithm-1
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
TID	Thread ID
UDP	User Datagram Protocol
UPX	Ultimate Packer for eXecutables
URL	Uniform Resource Locator
UTC	Coordinated Universal Time
VAD	Virtual Address Descriptor
VMEM	Virtual Memory

Glossary

_Eprocess

See Eprocess.

_Ethread

See Ethread.

_Kthread

See Kthread.

Anti-Virus

An Anti-virus, AV, or AV scanner is a software system or framework which is used to, at a minimum, scan a given system for signs of malware infection. This software may be more than just a scanner; it may also include system-protection and anti-malware detection and prevention capability.

AV Scanner

See Anti-Virus.

Computer Memory Image

See Memory Image.

Context Triggered Piecewise Hash

See Fuzzy Hash.

Data Carving

Commonly known as file carving, data carving is the process or act of recovering known data structures, generally based on recognized file patterns. Data carving only works on contiguous data structures as the recovery of fragmented data is not supported by most of today's data recovery software and those that do support a very limited number of file formats.

DLL Injection

DLL injection is a type of process injection. It is a method which allows a DLL to inject its code into the virtual address space of another process. In so doing, the DLL hijacks the process forcing the program to run in a manner inconsistent with its design. Under Windows, various methods exist for implementing this, some through the registry while others are carried out using APIs.

Eprocess

The Eprocess is a kernel-based process-specific data structure that encompasses a process' state-based information. This structure has a forward and backward pointer to active processes.

Ethread

An Ethread is a thread/process-based management kernel-specific data structure used to identify threads to be worked on. Its structure describes the various aspects of the process or thread and it is a semi-opaque data structure. Unlike a Kthread structure, it is processor agnostic.

Ext4

Ext4 is the latest Ext-based filesystem of the Linux operating system and supersedes Ext2/3. It provides filesystem journaling and greater performance, reliability and allows for much larger file and filesystem sizes. This filesystem is natively supported by Linux.

Fuzzy Hash

This is a specific type of file hashing which has the ability to identify file similarities, usually represented as a percentage. It is based on Context Triggered Piecewise Hashing, first proposed by Dr. Andrew Tridgell.

Handle

A handle is a pointer-like resource-based reference used to a specific system resource. Handles are abstract references to resources available within a given computer operating system. Under Windows, many types of handles exist but common examples pertain to files, directories, registry and system based devices. It should not be confused with file handles.

Hash

A hash, commonly referred to as a file hash, is a reduced representation of some arbitrary data produced by passing it through some cryptographic hashing algorithm. In so doing, a unique hash value is generated by the hashing program and it can be used to identify and authenticate a given file's integrity and uniqueness against a set of hashes, commonly known as a hash-set. SHA1 and CTPH hashes are examples of hashing algorithms.

IRP Hook

An IRP Hook is a kernel-based interception technique some rootkits, viruses and Trojan horses use in order to hide themselves from detection.

Kthread

A Kthread is a thread/process-based management kernel-specific data structure used to identify threads to be worked on. It is similar to an Ethread but contains processor-specific data structures. Its structure describes the various aspects of the process or thread to be worked on, including underlying processor specific features and is more opaque than an Ethread data structure.

Memory Image

A memory image or computer memory image is a bit-copy of a computer system's RAM and is acquired using a memory-imaging program. In virtualized environments, memory can be acquired by an imaging program or by saving or dumping the virtual machine's memory state.

Mutex

A mutex is a Windows-based object used to provide exclusive access to a shared system resource. These resources can only be accessed one at a time, thus by issuing a mutex or mutual exclusion, a process or thread can be allocated said resource when it becomes available for use.

SHA1 (Secure Hash Algorithm-1)

The SHA1 hash is a 160-bit cryptographic hash commonly used for forensic file identification and authentication.

SSL (Secure Sockets Layer)

SSL is a client-server TCP/IP Application Layer protocol. It is commonly used for the exchange of cryptographic keys that will be used to establish a "secure" communications channel between two systems.

Strings (the command)

The *strings* command is a UNIX-based command used to extract 7, 8, 16 and 32-bit text patterns from arbitrary data files that are text or binary based. 7-bit extraction represents the first 128 ASCII characters while 8-bit extraction represents the extended ASCII character set. 16 and 32-bit strings are typically reserved for Unicode-based text. Thus, the command line parameters required to instruct the *strings* command to perform 7, 8, 16 or 32-bit text extraction are *-s*, *-S*, *-l* and *-L*, respectively.

Thread

A thread is typically a subset process. A thread contains only the code necessary to perform a set of instructions. In single-threaded programs, a thread represents the program's executable code and stack while in multi-threaded applications a thread performs just one piece of the work that is distributed across multiple threads. These threads then typically communicate with each other through various inter-process mechanisms.

Trojan horse

A Trojan horse is a malicious non-replicating infectious computer program. It infects a computer when the delivery software is run at which time a payload is instantiated that does the actual infecting. However, Trojan's do not typically infect computers the way viruses do. As such, they do not generally infect computer files. The program delivering the payload is known as a dropper. The payload achieves its objective by gaining some form of administrative level privileges in the target's operating system, typically through subversion. A Trojan's typical objective is to provide backdoor access but it can also be used for other capabilities including data and information theft, arbitrary or specific data file encryption and

it can inflict damage to the operating system or its data files. In rare cases, it can even attempt to damage a system's hardware components.

Unlinked DLL (or file)

Unlinking a DLL or other file such as an executable or library is a common method malware and other malicious processes use to hide the fact that they may be using one of these resources covertly. Volatility's *ldrmodules* plugin supports several unlinked validation tests. It should be used to test for the existence of unlinked files associated to a process.

UPX

UPX is an open source data compression algorithm used to compress executable files. UPX executable file packers exist for Windows, Linux, Mac OS X and other platforms.

Vmem

A Vmem file is a VMware virtual machine-based paged memory file. It is generated when a virtual machine's state is saved and contains the entire RAM allocated to that virtual machine.

Worm

Sometimes known as a computer or network worm, a worm is a malicious program designed to spread to as many computer systems as possible, usually by means of a network. Worms do not typically cause much, if any, damage to the underlying computer system. Instead, due to their need to replicate they often consume not only a network's available bandwidth but crash underlying computer systems as they sometimes overwhelm a system's resources as it attempts to propagate. Worms typically spread only to systems susceptible to the vulnerabilities necessary for their infection to take hold. Thus, unaffected systems do not become infected.

This page intentionally left blank.

DOCUMENT CONTROL DATA

(Security markings for the title, abstract and indexing annotation must be entered when the document is Classified or Designated)

1. ORIGINATOR (The name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.)

Defence Research and Development Canada – Valcartier
2459 Pie-XI Blvd North
Quebec (Quebec)
G3J 1X5 Canada

2a. SECURITY MARKING (Overall security marking of the document including special supplemental markings if applicable.)

UNCLASSIFIED

2b. CONTROLLED GOODS

(NON-CONTROLLED GOODS)
DMC A
REVIEW: GCEC APRIL 2011

3. TITLE (The complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C or U) in parentheses after the title.)

Malware memory analysis for non-specialists : Investigating publicly available memory image 0zapftis (R2D2)

4. AUTHORS (last name, followed by initials – ranks, titles, etc. not to be used)

Carbone, R.

5. DATE OF PUBLICATION (Month and year of publication of document.)	6a. NO. OF PAGES (Total containing information, including Annexes, Appendices, etc.)	6b. NO. OF REFS (Total cited in document.)
October 2013	84	25

7. DESCRIPTIVE NOTES (The category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Technical Memorandum

8. SPONSORING ACTIVITY (The name of the department project office or laboratory sponsoring the research and development – include address.)

Defence Research and Development Canada – Valcartier
2459 Pie-XI Blvd North
Quebec (Quebec)
G3J 1X5 Canada

9a. PROJECT OR GRANT NO. (If appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.)	9b. CONTRACT NO. (If appropriate, the applicable number under which the document was written.)
31XF20 « MOU RCMP "Live Forensics" »	

10a. ORIGINATOR'S DOCUMENT NUMBER (The official document number by which the document is identified by the originating activity. This number must be unique to this document.)	10b. OTHER DOCUMENT NO(s). (Any other numbers which may be assigned this document either by the originator or by the sponsor.)
DRDC Valcartier TM 2013-177	

11. DOCUMENT AVAILABILITY (Any limitations on further dissemination of the document, other than those imposed by security classification.)

Unlimited

12. DOCUMENT ANNOUNCEMENT (Any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in (11) is possible, a wider announcement audience may be selected.))

Unlimited

13. ABSTRACT (A brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual.)

This technical memorandum examines how an investigator can analyse an infected Windows memory dump. The author investigates how to carry out such an analysis using Volatility and other investigative tools, including data carving utilities and anti-virus scanners. Volatility is a popular and evolving open source-based memory analysis framework upon which the author has proposed a memory-specific methodology for aiding fellow novice memory analysts. The author examines how Volatility can be used to find evidence and indicators of infection. This technical memorandum is the third in a series concerning Windows malware-based memory analysis. This current work examines the 0zapftis (R2D2) infected memory image.

Ce mémorandum technique examine comment un investigateur peut analyser une image mémoire d'une machine Windows infectée. L'auteur investigue les techniques d'analyse utilisant Volatility et d'autres outils tels que les utilitaires de récupération de données et les scanneurs anti-virus. Volatility est un cadre populaire d'analyse de mémoire en source libre sur lequel l'auteur s'appuie pour proposer une méthodologie spécifique à la mémoire pour aider ses collègues analystes novices. L'auteur examine comment Volatility peut être utilisé pour trouver des preuves ou des indicateurs d'infection. Ce mémorandum technique est le troisième d'une série visant la découverte de maliciel par le biais d'une analyse de la mémoire. Le présent travail examine l'image de mémoire infectée par 0zapftis (R2D2).

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (Technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus, e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

0zapftis; Antivirus; Anti-virus; Computer forensics; Digital forensics; Digital forensic investigations; Forensics; Infection; Malware; Memory analysis; Memory image; R2D2; Rootkit; Scanners; Trojan horse; Virus scanner; Volatility; Windows