Malware memory analysis for non-specialists

Investigating a publicly available memory image of the Zeus Trojan horse

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

Defence R&D Canada – Valcartier

Technical Memorandum
DRDC Valcartier TM 2013-018
April 2013

Principal Author

Richard Carbone
Forensic Investigator

Approved by

Guy Turcotte
Head/Mission Critical Cyber Security Section

Approved for release by

Christian Carrier
Chief Scientist

Abstract

This technical memorandum examines how an investigator can analyse a Windows-based computer memory dump infected with malware. 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. The author has proposed a memory-specific methodology based on a simple investigative process to help fellow novice memory analysts. Once evidence or indicators of malware have been found, the author examines how Volatility can be used to undertake a given memory investigation. This technical memorandum is the first of a series of reports that will be written concerning Windows malware-based memory analysis using Volatility and various malware scanners. This specific work examines a memory image infected with the Zeus Trojan horse.

Résumé

Le présent mémorandum technique examine comment un enquêteur peut analyser une image mémoire Windows infectée par des logiciels malveillants. L'auteur étudie la façon d'effectuer une telle analyse en utilisant Volatility ainsi que d'autres outils, y compris des utilitaires pour la récupération de données et des scanneurs anti-virus. Volatility est un cadriciel à code source ouvert populaire et en constante évolution pour l'analyse de mémoire. L'auteur propose une méthodologie spécifique à l'analyse de mémoire basée sur un processus d'enquête simple afin d'aider des collègues débutants. Une fois que des preuves ou des indicateurs de la présence de logiciels malveillants ont été trouvés, l'auteur examine comment Volatility peut être utilisé pour analyser la mémoire. Ce mémorandum technique est le premier d'une série de rapports qui seront écrits au sujet de l'analyse de mémoire pour Windows en utilisant Volatility et d'autres scanneurs de logiciels malveillants. Le présent ouvrage examine une image mémoire infectée par le cheval de Troie Zeus.

This page intentionally left blank.

Executive summary

Malware memory analysis for non-specialists: Investigating a publicly available memory image of the Zeus Trojan horse

R. Carbone; DRDC Valcartier TM 2013-018; Defence R&D Canada – Valcartier; April 2013.

The author has decided to share his own investigative analysis concerning a publicly available Windows-based infected memory image with the forensic community. While memory analysis has largely been carried out by software reverse engineers and malware analysts, the advent of memory analysis 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 concrete examples how investigators can conduct meaningful memory-based investigations on their own. Moreover, the author has provided a straightforward memory-specific investigative methodology to help novice memory analysts with their own memory investigations.

This technical memorandum examines the Zeus Trojan horse and is the first of a series that will examine various Windows-based malware infected memory images, in order to build a compendium of examples that can be used by the Canadian Armed Forces as a basis for conducting their own investigations. Thus, this document serves as a learning guide, for both the author and the community. Using publicly available computer memory images infected with well-known malware, investigators will be able to apply the concepts and memory-specific methodology examined herein.

Although others have engaged in the analysis of these very same memory images, the author is of the opinion that these analyses are insufficient as learning guides. Specifically, these analyses are either too limited in their investigative scope or report too little information to be of much use. Moreover, many of these analyses leave the reader asking more questions than when he began, due to the overall lack of their having a comprehensive investigative context. Thus, the author has strived to ensure that his investigative actions and lines of inquiry were documented herein, even if some of them 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 a publicly available memory image of the Zeus Trojan horse

R. Carbone ; DRDC Valcartier TM 2013-018 ; R & D pour la défense Canada – Valcartier; avril 2013.

L'auteur a décidé de partager avec la communauté légale sa propre analyse d'enquête d'une image mémoire Windows infectée et accessible au public. Bien que jusqu'à maintenant l'analyse de mémoire ait été en grande partie réalisée par des rétro-ingénieurs et des analystes de logiciels malveillants, l'avènement de cadriciels légaux pour l'analyse de mémoire tels que Volatility, ont rendu possible pour des non-spécialistes l'analyse d'images infectées par des logiciels malveillants. En combinant Volatility avec des utilitaires de récupération de données et des scanneurs anti-virus, les analystes débutants ont tous les outils nécessaires pour mener des enquêtes basées sur l'analyse de mémoire.

L'objectif principal de l'auteur est de démontrer par des exemples concrets comment des enquêteurs peuvent mener des enquêtes significatives basées sur l'analyse de mémoire de leur propre chef. Par ailleurs, l'auteur fourni une méthodologie simple et spécifique à l'analyse de mémoire afin d'aider les analystes novices dans leur propres enquêtes.

Ce mémorandum technique analyse le cheval de Troie Zeus et est le premier d'une série qui examinera plusieurs images mémoires Windows infectées par des logiciels malveillants. Le but est de constituer un recueil d'exemples qui pourra être utilisé par les Forces armées canadiennes comme base pour la conduite de leurs propres enquêtes. Ce document sert donc de guide d'apprentissage à la fois à l'auteur et à la communauté. En utilisant des images mémoires accessibles au public et infectées par des logiciels malveillants notoires, les enquêteurs seront en mesure d'appliquer les concepts et la méthodologie spécifique à l'image mémoire examinée dans le présent document.

Bien que d'autres se soient livrés à l'analyse de ces mêmes images mémoires, l'auteur est d'avis que ces analyses sont insuffisantes pour être utilisée comme guides d'apprentissage. Plus précisément, ces analyses sont soient trop limitées dans leur portée d'enquête ou fournissent trop peu d'informations pour être d'une quelconque utilité. De plus, plusieurs de ces analyses laissent le lecteur avec plus de questions que de réponses, en raison de l'absence générale d'un contexte d'enquête détaillé. Par conséquent, l'auteur s'est assuré que ses actions et pistes de réflexion soient documentées, même celles qui se sont avérées infructueuses, afin que le contexte d'enquête utilisé soit cohérent.

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

Abstract ……. i
Résumé ……..... i
Executive summary iii
Sommaire iv
Table of contents v
List of tables vii
Acknowledgements viii
Disclaimer policy..... ix
Requirements, assumptions and exclusions..... x
Target audience xi
1 Background..... 1
1.1 Objective..... 1
1.2 Why write new tutorials?..... 1
1.3 Infected memory image information 1
1.4 Data carving..... 2
1.5 Malware and anti-virus scanners 3
1.5.1 Specifics..... 3
1.5.2 Caveat 3
1.6 Detailed list of software tools used..... 4
1.6.1 Anti-virus scanners 4
1.6.2 Data carving..... 5
1.6.3 Volatility..... 6
1.7 Investigative methodology 6
2 Zeus memory investigation..... 10
2.1 Background..... 10
2.2 Preliminary investigative steps..... 10
2.2.1 Protect the memory image 10
2.2.2 Preliminary anti-virus scanning results..... 10
2.2.3 Data carving and file hashing 11
2.2.4 Anti-virus scanning and file hashing results for data carved files 11
2.3 Volatility memory analysis..... 13
2.3.1 Background..... 13
2.3.2 First analysis endeavour: wrong turn..... 13
2.3.2.1 Imageinfo plugin..... 13
2.3.2.2 Pslist plugin..... 14
2.3.2.3 Cmdscan and consoles plugins 16
2.3.2.4 Psscan plugin 17

2.3.2.5 Differentiating the output between the pslist and psscan plugins 18
2.3.2.6 Connecting the dots with respect to VMip.exe 18
2.3.2.7 Psxview plugin 19
2.3.2.8 Threads plugin 20
2.3.2.9 Thrdscan plugin 24
2.3.2.10 Memdump, procexedump and procmemdump plugins 25
2.3.2.11 First analysis endeavour summary 26
2.3.3 Second analysis endeavour: hunting and finding the evidence 27
2.3.3.1 Connscan plugin 27
2.3.3.2 Connections plugin 28
2.3.3.3 Sockets and sockscan plugins 28
2.3.3.4 Whois suspicious IP address 30
2.3.3.5 Malfind plugin 32
2.3.3.6 Dumping the suspicious process using Volatility and malfind 34
2.3.3.7 Virus scanning and hash verification of *malfind*-dumped PID 856 34
2.3.3.8 Filescan plugin 35
2.3.3.9 Summary 36
2.3.4 Pruning the registry for more information 37
2.3.4.1 Hivelist plugin 37
2.3.4.2 Printkey plugin 39
2.3.4.3 Output from the various printkey commands 40
2.3.4.4 Userassist plugin 42
3 Memory analysis issues 43
3.1 Memory analysis problems 43
3.2 The uses of memory analysis 43
4 Conclusion 44
References 45
Annex A Anti-virus scanner logs for data carved files 47
A.1 Avast 47
A.2 AVG 47
A.3 BitDefender 48
A.4 ClamAV 48
A.5 F-Prot 51
A.6 McAfee 51
Annex B Volatility Windows-based plugins 53
Annex C NSRL file hash matches for data carved files 57
Annex D Commonly used registry keys in a typical malware infection 61
Bibliography 65
List of symbols/abbreviations/acronyms/initialisms 66

List of tables

Table 1: Infected memory image metadata. 2
Table 2: List of anti-virus scanners and their command line parameters. 4
Table 3: Photorec data carving settings. 5
Table 4: Matching of carved executable files and their detection by anti-virus scanners. 11
Table 5: Volatility output for the pslist plugin. 14
Table 6: Volatility output for the psscan plugin. 17
Table 7: Volatility output for the psxview plugin. 19
Table 8: Volatility output for the thrdscan plugin. 24
Table 9: Volatility output for the connscan plugin. 27
Table 10: Volatility output for the sockets plugin. 28
Table 11: Volatility output for the sockscan plugin. 29
Table 12: Volatility output for the hivelist plugin. 37
Table 13: Association between registry hives and their corresponding registry keys commonly used for registry-based infections as per the hivelist output. 38
Table B.1: List of Volatility 2.2 plugins. 53
Table C.1: Data carved file SHA1-filename matches as per the NSRL 57

Acknowledgements

The authors would like to thank Mr. Philippe Charland for peer reviewing this text and providing helpful comments to improve it. Thanks are also extended to Mr. Sébastien Bourdon-Richard of the RCMP's Integrated Technological Crime Units for conducting a technical review of this document.

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 at work on a corporate network or on an 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 one 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, free or 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 using any malware, whether examined herein or otherwise, in any offensive or defensive nature against any other entity, or 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 the 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, this technical memorandum will endeavour to ensure that the reader can carry out his own forensic analysis of computer memory images suspected of malware infection.

The experimentation conducted throughout this technical memorandum has been carried out atop a Fedora Core 17 64-bit Linux operating system. Six different anti-virus scanners were used throughout this investigation. They include, in alphabetical order, the AVG, Avast, BitDefender, ClamAV, FRISK F-Prot and McAfee command line scanners. As for data carving tools and utilities, the author used Photorec, a part of the Testdisk suite of data recovery tools.

It is assumed that the reader successfully obtained the publicly available infected memory image examined in Section 1.3. Moreover, it is assumed that the reader has permission to use these tools on his computer system and network. Use of these tools and the analysis of virulent software always carry some inherent risk that must be adequately managed and minimized.

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.

Target audience

The targeted audience for this technical memorandum are computer forensic investigators who must assess suspect computer memory dumps for malware infection. Although computer memory analysis is a rather new topic within the field of digital forensics, there are those who have been conducting malware analysis and software reverse engineering for years, long before this topic became popular. Those seasoned veterans are aptly skilled. A framework such as Volatility, while capable of providing great insight even by novices, is all the more capable in the hands of experts.

The author of this technical memorandum has written it for others who, like himself, are required from time to time to conduct memory malware assessments and investigations. However, the author, like many others, are 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 plugins, in order to develop an investigative how-to for non-memory analysis experts.

This page intentionally left blank.

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 suspected infected memory image. More specifically, this document examines a methodological approach a novice memory analyst could use to investigate suspected memory images.

The work carried out herein is based on a publicly available memory image containing a well-known malware, the Zeus Trojan horse. This document, the first in a series of many, examines the investigative techniques necessary for a novice to conduct such memory analyses on his own. Ultimately, these reports will provide a methodological and foundational framework that novice memory analysts and experienced investigators alike can rely as a memory analysis guide or tutorial.

1.2 Why write new tutorials?

Although various online tutorials exist in various locations concerning these and other infected memory images, the tutorials are generally written for a highly technical audience already familiar with software reverse engineering and memory forensics.

It could be argued that these currently available tutorials are altogether insufficient in aiding budding memory investigators in learning the necessary techniques required to apply the techniques of digital forensics to memory analysis. The author instead asserts that by re-examining and thoroughly documenting the steps and procedures necessary to unravel these publicly available malware infected memory images, one at a time that a compendium of information would become available to the novices within the forensic community to serve as learning guides and tutorials.

The author has made all efforts to ensure that this document and the investigation of the Zeus Trojan horse is comprehensible to general computer forensic practitioner, in the hopes of reaching as wide an audience as possible, in order to have a more significant impact.

1.3 Infected memory image information

The infected Zeus memory dump file examined herein has been procured from the following location: http://code.google.com/p/volatility/wiki/PublicMemoryImages. Its SHA1 hash in its uncompressed form is as follows:

Table 1: Infected memory image metadata.

Memory image name	Size (MiB)	SHA1 hash value
zeus.vmem	128 (exactly)	e67f018663089c05a2ad8dd8d5a2d7c53c35c4ca

1.4 Data carving

Data carving software are specific tools and utilities whose primary objective is to recover data from damaged or corrupted filesystems, partitions or from unallocated disk and filesystem space. However, this software can also be used to coax the data recovery and extraction of deleted or damaged data from raw filesystems and damaged disks or devices.

Although a variety of data carving software exists in the commercial and open source marketplace, the capabilities of several open source tools rival the best commercial tools. Specifically, Photorec, part of Testdisk suite of data recovery tools is, in the opinion of the author, superior to all other open source data carving tools, as it uses advanced pattern detection techniques and supports many hundreds of commonly used file formats. Moreover, since the work carried out herein must be reproducible, the use of open source software makes sense, as the software's configuration and functionalities can be fine-tuned and understood through code analysis, respectively. [1, 16]

Data carving should not generally be carried out against intact filesystems, as undamaged and accessible data therein will be needlessly recovered, thereby complicating the recovery and extraction of deleted or lost data and files. Instead, data recovery through data carving should only be conducted against a filesystem's unallocated space. Data carving-based recovery can also be conducted against raw or damaged filesystems where no discernible filesystem or logical access mechanism can be readily determined.

Although computer memory images may have a discernible structure, they are not easy to work with and manipulate without an appropriate memory analysis framework. These frameworks include but are not limited to Volatility. Even then, these frameworks are only of use against processes and threads which were in the midst of running or had been paused (or that turned into a zombie process). Other processes that had died or terminated may nevertheless remain relatively intact within a given memory image and can sometimes provide additional information to a memory analysis framework. However, many of the data files in use at the time of acquisition are not generally available through such a framework and using a data recovery and carving tools may help to retrieve these data from a given memory image. Moreover, data recovery and carving tools used against memory images can often recover individual processes and threads, which can then be analysed using malware scanning technology for indicators of infection.

Thus, in the author's opinion, data carvers can be used to help perform "quick and dirty" triage-based forensics in order to determine whether a given memory image should be analysed using a memory analysis framework.

Of course, data carving is not foolproof and can be highly error prone, as the success of data carving is highly dependent on the contents of the memory and the extent to which its various contents may have been paged out to the underlying suspect system's pagefile. Even if a given

process or its data has not been paged out, it may not have been allocated in a contiguous block of memory and may therefore be fragmented. Thus, data carving a given memory image will often result in partially recovered executable and data files. Nevertheless, this is often a good starting point for commencing an investigation and with multiple anti-virus scanners, the odds increase of correlating aggregated anti-virus scanner detections.

1.5 Malware and anti-virus scanners

1.5.1 Specifics

Prior to conducting any memory analysis, it is prudent to use at least several scanners to determine, from a preliminary standpoint, whether a given memory image may in fact be infected. Some scanners are highly sensitive, while others are not. As such, some scanners can be directly passed suspected memory images for immediate analysis, while others are altogether incapable of processing such images. Of course, even after the initial analysis of a given raw memory image, multiple scanners should be used against the data recovered against said memory image using a data recovery and carving tools. The use of multiple scanners is highly beneficial as each scanner uses different detection techniques that are either signature or heuristic-based, and sometimes both.

When performing the initial scan against a raw memory image, it is unlikely that most scanners will pick up anything. Instead, it is likely that the scanners will turn up evidence of infection only after the dismemberment of a memory image using a data carving or recovery tool. However, tearing memory dumps apart using a data carver and scanning the resultant data files with anti-virus scanners will often product many false positives. This is why multiple scanners must be used and their results aggregated. Thus, if a file is detected by multiple scanners, the likelihood of infection increases. Thus, this increase in detection and identification will help the investigator decide whether additional analysis against a given memory image is warranted.

Memory fragmentation, which is largely responsible for incomplete processes and threads residing within a given memory at the time of its acquisition, is due to the manner in which memory is allocated to programs and swapped out to disk by the operating system's virtual memory manager. The swapping out of data, processes and threads only complicates malware detection of recovered data using data recovery tools. Of course, a variety of factors including the amount of physical memory (RAM), pagefile size, the number of programs and applications concurrently running and the amount of memory occupied by various data files will certainly have an effect on memory fragmentation. Moreover, since memory pages on x86-based systems are typically allocated in 4 KiB blocks, rarely do entirely programs fit therein. Instead, they are generally allocated multiple blocks of contiguous memory.

1.5.2 Caveat

The author uses and suggests that instead of relying on online malware scanning resources including VirusTotal and others, investigators should use locally installed anti-virus and other malware scanning technology. The primary reason for this recommendation is that while investigating a malware-related incident within a government or corporate network, infiltration or

partial control of a network may already have occurred. Moreover, an investigator using one of these networks to submit malware samples to an online resource may inadvertently tip off the attacker, who may already be monitoring the network, that signs or suspicion of an attack may have been uncovered.

Today's government, corporate and industrial networks are commonly infected with malware, some of which is virtually undetectable and may have been sponsored by state-based actors. This is no longer the realm of fiction. All too commonly, advanced malware with command and control behaviour are able to not only permit a remote attacker to carry out various actions against a target network, but can even monitor for signs of it having been discovered.

The level of trust in an underlying network is a contentious issue and readers may be of the opinion that their network is not under any foreign influence. However, how sure can the reader be about this? The reader must consider and question who potentially stands to benefit from control of the network or some of its computing resources if they are compromised.

It is for these reasons that the author suggests that the investigator uses a standalone computer system. Moreover, it is recommended that if the analysis system must remain on the network, it should implement a form of Mandatory Access Control (MAC) in order to reduce the likelihood of an attacker compromising it. The use of a MAC-based Linux or UNIX system will only help to further reduce the possibility that a given malware sample may infect the system and since most malware the investigator will encounter will be Windows-based, using non-Windows systems for analysis will only further reduce the likelihood of infecting the analysis system. Thus, such a system, used in conjunction with locally based anti-virus and malware detection software, will further reduce the prospect of tipping off the attacker, as that system is likely to be more trustworthy and free on infection itself. [3]

Finally, major vendors of malware detection software provide Linux and UNIX versions of their software, sometimes free, although usually they are available for a nominal charge.

1.6 Detailed list of software tools used

1.6.1 Anti-virus scanners

This memorandum makes use of six anti-virus scanners, listed in the following table.

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

Anti-virus scanner	Command line parameters
AVG 2012 command line scanner	-H -P -p
Avast v.1.3.0 command line scanner	-c
BitDefender for Unices v7.90123 Linux-amd64 scanner command line	No parameters used
ClamAV 0.97.6/15618/Thus Nov 22 19:07:00 2012 command line	--detect-pua=yes --detect-broken=yes -r

Anti-virus scanner	Command line parameters
FRISK F-Prot version 6.3.3.5015 command line scanner	-u 4 -s 4 -z 10 --adware --applications
McAfee VirusScan for Linux64 Version 6.0.3.356 command line scanner	--RECURSIVE --ANALYZE --MANALYZE --MIME --PANALYZE --UNZIP --VERBOSE

The assortment of scanners used herein is sufficiently diverse to represent an adequate cross-section of various detection mechanisms necessary to detect varying malware. Each scanner was last updated on January 22, 2012, the date upon which the analysis was carried out herein.

1.6.2 Data carving

For data carving, Photorec was used. It is part of the Testdisk suite of data recovery tools, developed by Christophe Grenier. Written entirely in C, the current stable version, released November 2011, is 6.13. Originally written to recover deleted photos from disk-based media, it has since been expanded to support several hundred file formats. Moreover, it is filesystem agnostic and can be run against disk images without any discernible filesystem such as unallocated disk clusters and memory dump files.

Photorec's data carving options were set to the following:

Table 3: Photorec data carving settings.

Options	Value
Paranoid	Yes (Brute for enabled)
Allow partial last cylinder	Yes
Keep corrupted files	No
Expert Mode	Yes
Options	**Value**
Low Memory	No
File Options	**Value**
Type	Default
Search (settings)	**Value**
Filesystem type	Other
Block size	512 bytes

Support for the Photorec's file formats were left to the program's default settings.

1.6.3 Volatility

Volatility 2.2 is used throughout this work for the analysis of the memory image suspected of infection by the Zeus Trojan horse. The version of this framework, at the time of this writing, is considered the stable public release and is suitable for general use by both the public and investigators alike, although it may not necessarily have the most recent or cutting-edge plugins. It was released for public use October 2012.

Originally written by Aaron Walters of Volatile Systems, Volatility has become a full-fledged memory analysis framework. It is written entirely in Python and can therefore be run atop Windows, Linux and other operating systems supporting Python. Moreover, it has begun to support Linux-based memory analysis, although its Windows-based support should be considered more reliable. Currently, it is developed by a variety of contributors, although the most well known of these are Michael Ligh, Jamie Levy, Brendan Dolan-Gavitt, Michael Cohen, Andrew Case and Mike Auty. Furthermore, each of these individuals has made significant contributions to the digital forensic community over the last few years.

The Windows plugins currently supported by this version Volatility are described in Annex B.

1.7 Investigative methodology

The overall investigative methodology used throughout this report is simple. It can be summarised as follows:

Part 1:

- Ensure that the memory image has been set as read-only using the underlying filesystem's immutable flag to prevent accidental changes or modifications to the image.

Part 2:

- Analyse the raw suspect memory image with multiple anti-virus scanners:

    1. Some scanners[1] can perform in-depth analysis of seemingly raw data files and in many instances, determine the nature of the underlying infection. Avast is one such scanner.

    2. Save the output from the various scanners.

Part 3:

- Using at least one advanced data carving utility, carve out all potential data and files from the suspect image:

    1. It is best to use one highly capable data-carving tool rather than several mediocre tools[2].

[1] The number of scanners capable of this is rare.

2. Perform a SHA1 hash against all carved files and ensure that they are not a match against known good files hash-sets (e.g. known good hashes from the NSRL[3]). Those that match known good hashes are to be deleted in order to remove them from further analysis. Save the hashes in a data file for possible future use.

3. Consider using a CTPH[4] hash as well against the extracted data files. This information will be used in the next step.

Part 4:

– Run the anti-virus scanners against all carved data and files, with attention focused on cross-AV scanner correlation:

    1. When multiple scanners indicate that the same data files carved from a given suspect memory image contains the same or similar malware, it is likely that these files do in fact contain a significantly detectable amount of the infection.

    2. Only files picked up by more than one scanner are to be considered as possibly infected, as those detected only once are likely false positives.

    3. Save the output from the various scanners and correlate the results. Save this second analysis and associate fuzzy hashes to correlated scanner results (CTPH hashes are done using *ssdeep*).

Part 5:

– If a given memory image continues to remain suspect, e.g., evidence or indications of infection have been found, then use the Volatility memory analysis framework to better determine its state and if possible, how the system has come to be infected:

    1. Various investigative endeavours using Volatility may not yield tangible results (e.g., the memory image is corrupt, current plugins are not able to detect anything abnormal, etc.). Nevertheless, document these as they may serve as a lessons learned.

    2. This requires extracting as much information as possible about the underlying system, processes and threads that were running, communications, registry settings (if applicable), open files, etc.

    3. There are many plugins to choose from and it is unlikely they will all be used to determine more information about a given infection. Start by using plugins that are likely to be of immediate use (e.g., *imageinfo*, *pslist*, *psxview*, etc.) before using more esoteric plugins.

---

[2] The author is of the opinion that Photorec is one of the best available data carvers currently in use, even if it is free.

[3] The National Software Reference List is a list of software hashes maintained by NIST. The NSRL is the premier source of known hashes and represents many hundreds of popular software packages, tools and operating systems.

[4] CTPH is better known as fuzzy hashing. This is carried out using the *ssdeep* tool.

4. Once a suspected malware thread, process, DLL or data file has been found, hash and verify it using locally installed anti-virus scanners.

    a) If no other malware can be found, cease further analysis and ensure that all work, analyses and results are documented so that results can be reproduced by others. Save all work.

    b) If the malware is disk-resident, having since removed itself from memory and cannot be found therein, cease further analysis and recover the data file from a forensically acquired disk image corresponding to the given memory image. Ensure that all work, analyses and results are documented so that results can be reproduced by others. Save all work.

    c) Correlate the extracted malware with that discovered in 4-3. Determine if the SHA1 or CTPH hashes are the same or similar, respectively.

5. The malware may not even be in the memory image anymore, as it may have been removed from memory.

    a) This can occur if the malware is swapped out to pagefile.

    b) This can also occur if the malware detects an anomalous situation or environment for itself. For example, some malware continuously scan for a network connection and it the connection goes down, the malware unloads itself from memory.

    c) However, even if the malware is not in memory anymore, sometimes the cross-correlation of information from the various Volatility plugins may lead the investigator to suspect or determine that one or more disk-based files or network connections may have been responsible for the infection (or at least involved to some varying extent).

Part 6:

- In a worst case scenario, where little to no useful information can be determined about a given infection using Volatility:

    1. Begin by dumping processes and DLLs using the appropriate Volatility plugins (*procexedump*, *procmemdump* and *dlldump*).

    2. Use the various scanners to determine if any of the dumped executables were infected. Note any executable that has received more than one positive confirmation from more than one scanner.

3. Determine if any of the executables scanned above that received more than one positive confirmation for infection match against those data files carved out in Part 4-3, which also received more than one positive confirmation for infection against their respective SHA1 or CTPH hashes.

2 Zeus memory investigation

2.1 Background

The investigation of the memory image suspected of harbouring the Zeus Trojan horse is examined in this section, as based on the methodology as put forward in Section 1.7. Additional information concerning the malware can be found in [7, 8, 9, 10, 11, 12, 13, 14, 15, 17 and 18].

2.2 Preliminary investigative steps

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

2.2.1 Protect the memory image

The memory image *zeus.vmem* was set to immutable (attribute + i) atop an Ext4-based filesystem. The command used to perform this, carried out as the root user, was:

```
$ sudo chattr +i zeus.vmem
```

This results in the fact that the memory image can no longer be modified without resetting the file's immutable attribute, not even by the root user. This is to prevent accidental modifications from occurring to this file.

2.2.2 Preliminary anti-virus scanning results

The results of the preliminary anti-virus scanning using the six scanners outlined in Section 1.6.1 are examined herein.

The only scanner that identified the memory image *zeus.vmem* as infected was Avast. Its output is as follows:

```
zeus.vmem       [infected by: Win32:Zbot-BCW [Trj]]
#
# Statistics:
#
# scanned files: 1
# scanned directories:   0
# infected files: 1
# total file size:  128.0 MB
# virus database:         121122-1 22.11.2012
# test elapsed:   0s 346ms
#
```

Preliminary anti-virus scanner examination indicates that this memory image is in fact infected with the Zeus Trojan horse. Avast was the only scanner capable of accurately examining the image's internal structures. All anti-virus results were recorded and stored in appropriate text-based files.

2.2.3 Data carving and file hashing

Photorec succeeded in recovering 509 files carved from memory as per the author's recommended Photorec settings put forward in Section 1.6.2. Of those files recovered, 360 were PE-based files. Of those, 184 were identified as Windows 32-bit DLLs, while 176 were identified as standard Windows 32-bit PEs and device drivers. Other file types were also detected but they had no immediate use. However, their types were recorded and saved for possible future use within this analysis.

The recovered files were hashed and validated against the latest NSRL hash-set (September 2012). SHA1 hashes were obtained for all the data carved files and stored for future use. Eleven unique SHA1 hashes were confirmed as matches for the NSRL hash-set. A full listing of the NSRL filename matches as per the SHA1 hashes can be found in Annex C.

CTPH-based hashing was conducted using the *ssdeep* (fuzzy hashing) tool and stored for future use.

2.2.4 Anti-virus scanning and file hashing results for data carved 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.), the following matches were made. In order to reduce the incidence of false positives, only those detected using two or more scanners have been included. Specific logs for each scanner can be found in Annex A.1. Those filenames with three or more anti-virus matches have been bolded in the table below.

Table 4: Matching of carved executable files and their detection by anti-virus scanners.

Carved data filename	Matches found	Anti-virus scanner matches	Detected as Zbot
f0026720.dll	2	Avast, ClamAV	Yes
f0031840.dll	2	Avast, ClamAV	Yes
f0068952.dll	2	Avast, ClamAV	Yes
f0069472.dll	2	Avast, ClamAV	Yes
f0078696.exe	**5**	**Avast, AVG, BitDefender, ClamAV, FRISK**	Yes
f0083472.exe	**3**	**Avast, AVG, BitDefender**	No
f0096936.exe	2	AVG, ClamAV	No
f0104608.exe	2	AVG, ClamAV	No

Carved data filename	Matches found	Anti-virus scanner matches	Detected as Zbot
f0108688.dll	2	Avast, ClamAV	Yes
f0122376.dll	2	Avast, ClamAV	Yes
f0123288.exe	2	Avast, AVG	Yes
f0126048.exe	**3**	**AVG, BitDefender, FRISK**	Yes
f0126936.exe	2	Avast, AVG	Yes
f0135928.dll	2	AVG, ClamAV	No
f0136384.dll	2	Avast, ClamAV	Yes
f0144008.exe	2	AVG, ClamAV	No
f0152824.exe	2	AVG, BitDefender	No
f0169216.dll	2	AVG, ClamAV	No
f0169264.dll	2	Avast, ClamAV	Yes
f0176824.exe	**3**	**AVG, BitDefender, ClamAV**	No
f0179776.dll	2	Avast, ClamAV	Yes
f0186296.dll	2	Avast, ClamAV	Yes
f0189184.exe	2	AVG, ClamAV	No
f0198048.dll	**3**	**Avast, AVG, ClamAV**	Yes
f0198744.exe	**3**	**Avast, AVG, BitDefender**	No
f0202808.exe	2	AVG, BitDefender	No
f0206136.exe	2	AVG, ClamAV	No
f0215816.exe	**3**	**AVG, BitDefender, ClamAV**	No

Only those carved executables that were detected by three or more scanners are of particular interest, as they are statistically less likely to be false positives. These were found to be, as shown in the above table, *f0078696.exe*, *f0083472.exe*, *f0126048.exe*, *f0176824.exe*, *f0198048.dll*, *f0198744.exe* and *f0215816.exe*.

Of the aforementioned detected files, only two scanners picked up that some of the suspected files were infected with Zeus, detected as *Zbot*, specifically FRISK and Avast.

Using the previously generated fuzzy hashes (see Section 2.2.3), matches between scanner-correlated malware and the fuzzy hashes were established. Specifically, scanner identified malware *f0198744.exe* was correlated as having a 21% similarity with scanner identified malware *f0135928.exe*. Due to the limited similarity between these potential malware, no foregone conclusions should be drawn at this time.

Although similarities had been established between other executables, none of them had been established as warranting further consideration, since they had not been detected by two or more scanners. Thus, they can be safely ignored.

2.3 Volatility memory analysis

This subsection carries out the actual Volatility memory image analysis.

2.3.1 Background

In order to investigate this specific memory image suspected of infection by the Zeus Trojan horse, the author examines the use and output of various Volatility plugins that are likely to be of assistance in this particular case.

The Volatility plugins used throughout this section must support Windows XP. However, not all of the plugins support this specific operating system, although Windows XP remains the most supported operating system by Volatility [2]. The first plugin used in this investigation, found in the next subsection, determines some of the underlying information about the memory image (see Section 2.3.2.1).

The analysis carried out in the subsection is broken up by endeavours, where each endeavour is a distinct investigative path analysed using Volatility. If a given endeavour proves ineffective, then another investigative path is taken, whereby a productive end may be obtained.

While using certain plugins, it was possible to confirm their results by using additional plugins. For example, consider that when the *pslist* plugin was used, its results could not only be corroborated but also expanded upon by using additional process listing and process analysis plugins including *psscan* and *psxview*. Similarly, the *thrdscan* plugin was used to validate the results of the *threads* plugin.

2.3.2 First analysis endeavour: wrong turn

The investigator should begin the Volatility-based analysis using basic plugins including those that provide background information about the memory image and process listings. Using them will allow the investigator to move towards a more precise line of inquiry. These plugins can include, but are not limited to, image process listing, thread listing, background-based memory image information and process dumping.

2.3.2.1 Imageinfo plugin

This Volatility plugin is used to provide basic contextual information about a suspect memory image.

Output from the plugin, using command "*volatility imageinfo -f zeus.vmem*," is as follows:

Determining profile based on KDBG search...

Suggested Profile(s) : WinXPSP2x86, WinXPSP3x86 (Instantiated with WinXPSP2x86)
AS Layer1 : JKIA32PagedMemoryPae (Kernel AS)
AS Layer2 : FileAddressSpace (/volatility/memimgs/zeus.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 : 2010-08-15 19:17:56 UTC+0000
Image local date and time : 2010-08-15 15:17:56 -0400

This memory image appears to be running atop a Windows XP computer system with Service Pack 2. It is running with one processor and the memory image is 128 MiB in size (based on the memory image's size determined using *ls -l*). It was captured atop a system supporting a 32-bit PAE x86-based processor, on August 15, 2010 at 15:17:56 EDT.

2.3.2.2 Pslist plugin

The next step is to determine which processes were running within the memory image in order to determine if anything was out of the ordinary. The *pslist* plugin does precisely as its name implies. It provides a detailed process listing of the detected processes. It makes use of virtual memory addresses and offsets, whereas the *psscan* plugin (see Section 2.3.2.4) makes use of physical addresses and offsets.

Output from the *pslist* plugin, using command "*volatility pslist -f zeus.vmem*," is as follows:

Table 5: Volatility output for the pslist plugin.

Offset(V)	Name	PID	PPID	Thds	Hnds	Sess	Wow64	Start	Exit
0x810b1660	System	4	0	58	379	------	0		
0xff2ab020	smss.exe	544	4	3	21	------	0	8/11/2010 6:06:21	
0xff1ecda0	csrss.exe	608	544	10	410	0	0	8/11/2010 6:06:23	
0xff1ec978	winlogon.exe	632	544	24	536	0	0	8/11/2010 6:06:23	
0xff247020	services.exe	676	632	16	288	0	0	8/11/2010 6:06:24	
0xff255020	lsass.exe	688	632	21	405	0	0	8/11/2010 6:06:24	
0xff218230	vmacthlp.exe	844	676	1	37	0	0	8/11/2010 6:06:24	
0x80ff88d8	svchost.exe	856	676	29	336	0	0	8/11/2010 6:06:24	
0xff217560	svchost.exe	936	676	11	288	0	0	8/11/2010 6:06:24	
0x80fbf910	svchost.exe	1028	676	88	1424	0	0	8/11/2010 6:06:24	
0xff22d558	svchost.exe	1088	676	7	93	0	0	8/11/2010 6:06:25	
0xff203b80	svchost.exe	1148	676	15	217	0	0	8/11/2010 6:06:26	

Offset(V)	Name	PID	PPID	Thds	Hnds	Sess	Wow64	Start	Exit
0xff1d7da0	spoolsv.exe	1432	676	14	145	0	0	8/11/2010 6:06:26	
0xff1b8b28	vmtoolsd.exe	1668	676	5	225	0	0	8/11/2010 6:06:35	
0xff1fdc88	VMUpgradeHelper	1788	676	5	112	0	0	8/11/2010 6:06:38	
0xff143b28	TPAutoConnSvc.e	1968	676	5	106	0	0	8/11/2010 6:06:39	
0xff25a7e0	alg.exe	216	676	8	120	0	0	8/11/2010 6:06:39	
0xff364310	wscntfy.exe	888	1028	1	40	0	0	8/11/2010 6:06:49	
0xff38b5f8	TPAutoConnect.e	1084	1968	1	68	0	0	8/11/2010 6:06:52	
0x80f60da0	wuauclt.exe	1732	1028	7	189	0	0	8/11/2010 6:07:44	
0xff3865d0	explorer.exe	1724	1708	13	326	0	0	8/11/2010 6:09:29	
0xff3667e8	VMwareTray.exe	432	1724	1	60	0	0	8/11/2010 6:09:31	
0xff374980	VMwareUser.exe	452	1724	8	207	0	0	8/11/2010 6:09:32	
0x80f94588	wuauclt.exe	468	1028	4	142	0	0	8/11/2010 6:09:37	
0xff224020	cmd.exe	124	1668	0	-------	0	0	8/15/2010 19:17:55	2010-08-15 19:17:56

Looking at the above process listing, it can be readily determined that memory image *zeus.vmem* was running within a VMware virtual machine. Moreover, a study of the process listing yields no readily recognizable suspicious processes. Thus, additional plugins will be required to further analyse this memory image.

Based on the time indicated for the memory image's acquisition, listed as August 15, 2010 at 15:17:56 EDT by plugin *imageinfo* (see Section 2.3.2.1), this date and time coincides with the data and time that *cmd.exe* (PID 124) terminated. It is interesting to note that in the above process listing, PID 124 is a child process of PID 1668 (*vmtoolsd.exe*). This information has been highlighted in the table. With this information, it is possible to assume that the individual who acquired this memory image attempted to query the name of the host system from within the virtual machine. To do this, the individual may have used the following command:

> C:\Program Files\VMware\VMware Tools\> vmtoolsd.exe --cmd "info-get guestinfo.hypervisor.hostname"

If no command shell history can be found relating to this process (*vmtoolsd.exe* or *cmd.exe* from where it may be have launched), then it is likely that the process was launched from the *Start -> Run*. This will be confirmed in the upcoming steps.

Additional information that can be found in the above process listing is that the virtual machine appears to have been running since August 11, 2010 at 6:06:00 EDT.

The next step an investigator could undertake might be to determine if the aforementioned command shell, or any other command shells previously run but that have gone undetected by the *pslist* plugin, have left behind a command shell history. Moreover, the connection between *cmd.exe* and *vmtoolsd.exe* should be further investigated.

2.3.2.3 Cmdscan and consoles plugins

The plugins *cmdscan* and *consoles* may reveal more information about commands typed into a command shell. Querying a memory image using the *cmdscan* plugin is carried out by executing the command "*volatility cmdscan -f zeus.vmem*." This yields the following output:

```
**************************************************
CommandProcess: csrss.exe Pid: 608
CommandHistory: 0xf786f8 Application: TPAutoConnect.exe Flags: Allocated
CommandCount: 0 LastAdded: -1 LastDisplayed: -1
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x448
```

This output indicates that process *csrss.exe* (PID 608) spawned process *TPAutoConnect.exe* from a shell, but that no command shell history is available.

Querying the memory image using the *consoles* plugin is carried out by executing the command "*volatility consoles -f zeus.vmem*." This yields the following output:

```
**************************************************
ConsoleProcess: csrss.exe Pid: 608
Console: 0x4e23b0 CommandHistorySize: 50
HistoryBufferCount: 1 HistoryBufferMax: 4
OriginalTitle: C:\Program Files\VMware\VMware Tools\TPAutoConnSvc.exe
Title: C:\Program Files\VMware\VMware Tools\TPAutoConnSvc.exe
AttachedProcess: TPAutoConnect.e Pid: 1084 Handle: 0x448
----
CommandHistory: 0xf786f8 Application: TPAutoConnect.exe Flags: Allocated
CommandCount: 0 LastAdded: -1 LastDisplayed: -1
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x448
----
Screen 0x4e2ab0 X:80 Y:25
Dump:
TPAutoConnect User Agent, Copyright (c) 1999-2009 ThinPrint AG, 7.17.512.1
**************************************************
ConsoleProcess: csrss.exe Pid: 608
Console: 0xf78958 CommandHistorySize: 50
HistoryBufferCount: 2 HistoryBufferMax: 4
OriginalTitle: ??ystemRoot%\system32\cmd.exe
Title:
```

This output appears to support the assertion made concerning the *cmdscan* plugin. However, no command shell history was found. These plugins provide no additional clues regarding processes *cmd.exe* and *vmtoolsd.exe*.

Since these plugins were of little help, the next step is to determine if other memory-based process listing plugins can provide additional information.

2.3.2.4 Psscan plugin

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

Consider the following output from the *psscan* plugin, using command "*volatility psscan -f zeus.vmem*."

Table 6: Volatility output for the psscan plugin.

Offset(P)	Name	PID	PPID	PDB	Time created	Time exited
0x01214660	System	4	0	0x00319000		
0x06238020	cmd.exe	124	1668	0x06cc02a0	8/15/2010 19:17:55	8/15/2010 19:17:56
0x05f027e0	alg.exe	216	676	0x06cc0240	8/11/2010 6:06:39	
0x04be97e8	VMwareTray.exe	432	1724	0x06cc02e0	8/11/2010 6:09:31	
0x04b5a980	VMwareUser.exe	452	1724	0x06cc0300	8/11/2010 6:09:32	
0x010f7588	wuauclt.exe	468	1028	0x06cc0180	8/11/2010 6:09:37	
0x05471020	smss.exe	544	4	0x06cc0020	8/11/2010 6:06:21	
0x066f0da0	csrss.exe	608	544	0x06cc0040	8/11/2010 6:06:23	
0x066f0978	winlogon.exe	632	544	0x06cc0060	8/11/2010 6:06:23	
0x06015020	services.exe	676	632	0x06cc0080	8/11/2010 6:06:24	
0x05f47020	lsass.exe	688	632	0x06cc00a0	8/11/2010 6:06:24	
0x06384230	vmacthlp.exe	844	676	0x06cc00c0	8/11/2010 6:06:24	
0x0115b8d8	svchost.exe	856	676	0x06cc00e0	8/11/2010 6:06:24	
0x04c2b310	wscntfy.exe	888	1028	0x06cc0200	8/11/2010 6:06:49	
0x063c5560	svchost.exe	936	676	0x06cc0100	8/11/2010 6:06:24	
0x01122910	svchost.exe	1028	676	0x06cc0120	8/11/2010 6:06:24	
0x049c15f8	TPAutoConnect.e	1084	1968	0x06cc0220	8/11/2010 6:06:52	
0x061ef558	svchost.exe	1088	676	0x06cc0140	8/11/2010 6:06:25	
0x06499b80	svchost.exe	1148	676	0x06cc0160	8/11/2010 6:06:26	
0x06945da0	spoolsv.exe	1432	676	0x06cc01a0	8/11/2010 6:06:26	
0x069d5b28	vmtoolsd.exe	1668	676	0x06cc01c0	8/11/2010 6:06:35	
0x04a065d0	explorer.exe	1724	1708	0x06cc0280	8/11/2010 6:09:29	
0x010c3da0	wuauclt.exe	1732	1028	0x06cc02c0	8/11/2010 6:07:44	
0x0655fc88	VMUpgradeHelper	1788	676	0x06cc01e0	8/11/2010 6:06:38	
0x069a7328	VMip.exe	1944	124	0x06cc0320	8/15/2010 19:17:55	8/15/2010 19:17:56
0x0211ab28	TPAutoConnSvc.e	1968	676	0x06cc0260	8/11/2010 6:06:39	

The listing from the *psscan* appears approximately the same as the output from the *pslist* plugin (see Section 2.3.2.2 for details). Moreover, it confirms that PID 124 (*cmd.exe*) is a child of PID 1668 (*vmtoolsd.exe*), as also seen by plugin *pslist* (see Section 2.3.2.2 for details). Process PID 1944 (*VMip.exe*) is a child process of PID 124 (*cmd.exe*). However, *VMip.exe* is a process that was found exclusively by the *psscan* plugin. These three processes have been highlighted in the table above to emphasize the relationship between them.

2.3.2.5 Differentiating the output between the pslist and psscan plugins

Highlighting the differences between the output from the *pslist* and *psscan* plugins, as seen in sections 2.3.2.2 and 2.3.2.4, may not always be obvious. For this task, shell-based text processing is of significant use. By using the following commands, it will be possible to determine which differences were found:

```
$ cat psscan.txt | awk '{print $2"\t"$3"\t"$4"\t"$6"\t"$7}' | grep -v "\-\-\-\-\-\-\-\-\-\-" |
grep -v PPID | sort > psscan.txt

$ cat pslist.txt | awk '{print $2"\t"$3"\t"$4"\t"$9"\t"$10}' | grep -v "\-\-\-\-\-\-\-\-\-\-" |
grep -v PPID | sort > pslist_sorted.txt

$ diff psscan_sorted.txt pslist_sorted.txt > pslist_psscan_diff.txt
```

The output from file *pslist_psscan_diff.txt* yields the following output:

```
18d17

< VMip.exe      1944    124     2010-08-15      19:17:55
```

Thus, by using these commands, it was possible to determine that the difference between these two plugins (*pslist* and *psscan*) is the *VMip.exe* process.

2.3.2.6 Connecting the dots with respect to VMip.exe

Based on the information obtained in sections 2.3.2.2, 2.3.2.4 and 2.3.2.5, it can be determined that the chain of processes for the instantiation of *VMip.exe* (PID 1944) was:

```
PID 632 -> PID 676 -> PID 1668 -> PID 124 -> PID 1944
```

This translates to the following actual process names:

```
winlogon.exe -> services.exe -> vmtoolsd.exe -> cmd.exe -> VMip.exe
```

Although this may have seemed obvious to an investigator commonly working with virtual machines and VMware technology, this serves as a useful example of tracing back process instantiation. Thus, looking back on the command proposed in Section 2.3.2.2:

```
C:\Program Files\VMware\VMware Tools\> vmtoolsd.exe --cmd "info-get
guestinfo.hypervisor.hostname"
```

It is far more likely, based on the above chain of execution, that *VMip.exe* was actually instantiated by *vmtoolsd.exe* directly and not by an intervening user or investigator. Consider that in order for *VMip.exe* to run, *vmtoolsd.exe* called *cmd.exe* to execute *VMip.exe*, an action that occurs regularly where one process runs a shell command to instantiate another process. However, this becomes more obvious upon examining sections 2.3.2.7 and 2.3.2.8, where additional plugins lend credence to this claim.

2.3.2.7 Psxview plugin

Volatility provides the ability to detect hidden running processes, such as *VMip.exe*, through the *psxview* plugin.

The *psxview* plugin provides a detailed listing of which processes were running in the memory image and by which method they were found. Currently, the Volatility 2.2 *psxview* plugin supports five methods: *pslist*, *psscan*, *thrdproc*, *pspcdid* and *csrss*.

A hidden process, for example, would be a process that was invisible to not only the *pslist* plugin, but to most of the other aforementioned plugins, but which would have to be visible to at least one, in order to be detected.

Consider the following output from the *psxview* plugin, using the command "*volatility psxview -f zeus.vmem*."

Table 7: Volatility output for the psxview plugin.

Offset(P)	Name	PID	pslist	psscan	thrdproc	pspcdid	csrss
0x06499b80	svchost.exe	1148	TRUE	TRUE	TRUE	TRUE	TRUE
0x04b5a980	VMwareUser.exe	452	TRUE	TRUE	TRUE	TRUE	TRUE
0x05f027e0	alg.exe	216	TRUE	TRUE	TRUE	TRUE	TRUE
0x0655fc88	VMUpgradeHelper	1788	TRUE	TRUE	TRUE	TRUE	TRUE
0x0211ab28	TPAutoConnSvc.e	1968	TRUE	TRUE	TRUE	TRUE	TRUE
0x04c2b310	wscntfy.exe	888	TRUE	TRUE	TRUE	TRUE	TRUE
0x061ef558	svchost.exe	1088	TRUE	TRUE	TRUE	TRUE	TRUE
0x06945da0	spoolsv.exe	1432	TRUE	TRUE	TRUE	TRUE	TRUE
0x05471020	smss.exe	544	TRUE	TRUE	TRUE	TRUE	FALSE
0x069d5b28	vmtoolsd.exe	1668	TRUE	TRUE	TRUE	TRUE	TRUE
0x06384230	vmacthlp.exe	844	TRUE	TRUE	TRUE	TRUE	TRUE
0x010f7588	wuauclt.exe	468	TRUE	TRUE	TRUE	TRUE	TRUE
0x066f0da0	csrss.exe	608	TRUE	TRUE	TRUE	TRUE	FALSE
0x010c3da0	wuauclt.exe	1732	TRUE	TRUE	TRUE	TRUE	TRUE
0x06238020	cmd.exe	124	TRUE	TRUE	FALSE	TRUE	FALSE
0x06015020	services.exe	676	TRUE	TRUE	TRUE	TRUE	TRUE
0x04a065d0	explorer.exe	1724	TRUE	TRUE	TRUE	TRUE	TRUE
0x049c15f8	TPAutoConnect.e	1084	TRUE	TRUE	TRUE	TRUE	TRUE
0x0115b8d8	svchost.exe	856	TRUE	TRUE	TRUE	TRUE	TRUE

Offset(P)	Name	PID	pslist	psscan	thrdproc	pspcdid	csrss
0x01214660	System	4	TRUE	TRUE	TRUE	TRUE	FALSE
0x01122910	svchost.exe	1028	TRUE	TRUE	TRUE	TRUE	TRUE
0x04be97e8	VMwareTray.exe	432	TRUE	TRUE	TRUE	TRUE	TRUE
0x05f47020	lsass.exe	688	TRUE	TRUE	TRUE	TRUE	TRUE
0x063c5560	svchost.exe	936	TRUE	TRUE	TRUE	TRUE	TRUE
0x066f0978	winlogon.exe	632	TRUE	TRUE	TRUE	TRUE	TRUE
0x069a7328	VMip.exe	1944	FALSE	TRUE	FALSE	FALSE	FALSE

Based on the output from this plugin, it can be confirmed that process *VMip.exe* (PID 1944), highlighted in red, cannot be seen by any of the other process listing plugins, except when using *psscan*.

A quick web search reveals that process 1944 (*VMip.exe*) is likely part of the VMware tools, fitting well with the information already established in Section 2.3.2.6 concerning the chain of its instantiation. However, prior to jumping to conclusions concerning *VMip.exe*, the *threads* plugin may reveal additional information about some of the processes examined in Section 2.3.2.6. Other process listing plugins such as *pstree* are not likely to be of use at this time. The *pstree* plugin is useful to view the chain of execution as it creates a process tree.

2.3.2.8 Threads plugin

The *threads* plugin is useful as it has the ability to provide detailed information about processes and threads that have since terminated or that may be hidden. Specifically, additional information should be queried for processes PIDs 124, 1668 and 1944 (*cmd.exe*, *vmtoolsd.exe* and *VMip.exe*, respectively).

The *threads* plugin can be used by the investigator to ensure that nothing out of the ordinary is going with one more processes and its threads.

Output from the *threads* plugin for PID 124, using command "*volatility threads -f zeus.vmem -p 124,*" is as follows:

```
[x86] Gathering all referenced SSDTs from KTHREADs...
Finding appropriate address space for tables...
------
ETHREAD: 0xff3b1d80 Pid: 124 Tid: 972
Tags:
Created: 2010-08-15 19:17:55
Exited: 2010-08-15 19:17:56
Owning Process: cmd.exe
Attached Process: cmd.exe
State: Terminated
BasePriority: 0x8
Priority: 0x10
TEB: 0x00000000
```

StartAddress: 0x7c810867 UNKNOWN
ServiceTable: 0x80552140
  [0] 0x80501030
  [1] 0xbf997600
  [2] 0x00000000
  [3] 0x00000000
Win32Thread: 0x00000000
CrossThreadFlags: PS_CROSS_THREAD_FLAGS_TERMINATED

Output from the *threads* plugin for PID 1668, using command "*volatility threads -f zeus.vmem -p 1668,*" is as follows:

[x86] Gathering all referenced SSDTs from KTHREADs...
Finding appropriate address space for tables...

ETHREAD: 0xff1b88b0 Pid: 1668 Tid: 1672
Tags:
Created: 2010-08-11 06:06:35
Exited: 1970-01-01 00:00:00
Owning Process: vmtoolsd.exe
Attached Process: vmtoolsd.exe
State: Waiting:Executive
BasePriority: 0x8
Priority: 0x8
TEB: 0x7ffdd000
StartAddress: 0x7c810867 UNKNOWN
ServiceTable: 0x80552140
  [0] 0x80501030
  [1] 0xbf997600
  [2] 0x00000000
  [3] 0x00000000
Win32Thread: 0xe1d73690
CrossThreadFlags:

ETHREAD: 0xff14bbf8 Pid: 1668 Tid: 1844
Tags:
Created: 2010-08-11 06:06:38
Exited: 1970-01-01 00:00:00
Owning Process: vmtoolsd.exe
Attached Process: vmtoolsd.exe
State: Waiting:WrLpcReceive
BasePriority: 0x8
Priority: 0x9
TEB: 0x7ffdb000

StartAddress: 0x7c810856 UNKNOWN
ServiceTable: 0x80552140
  [0] 0x80501030
  [1] 0xbf997600
  [2] 0x00000000
  [3] 0x00000000
Win32Thread: 0xe127ab18
CrossThreadFlags:
Eip: 0x7c90eb94
  eax=0x00167a88 ebx=0x00000000 ecx=0x00167ae8 edx=0xfe7b5598 esi=0x0015e298 edi=0x00000100
  eip=0x7c90eb94 esp=0x015dfe1c ebp=0x015dff80 err=0x00000000
  cs=0x1b ss=0x23 ds=0x23 es=0x23 gs=0x00 efl=0x00000246
  dr0=0x00000000 dr1=0x00000000 dr2=0x00000000 dr3=0x00000000 dr6=0x00000000 dr7=0x00000000

ETHREAD: 0xff379bc0 Pid: 1668 Tid: 1380
Tags:
Created: 2010-08-11 06:07:14
Exited: 1970-01-01 00:00:00
Owning Process: vmtoolsd.exe
Attached Process: vmtoolsd.exe
State: Waiting:UserRequest
BasePriority: 0x8
Priority: 0x9
TEB: 0x7ffd8000
StartAddress: 0x7c810856 UNKNOWN
ServiceTable: 0x80552180
  [0] 0x80501030
  [1] 0x00000000
  [2] 0x00000000
  [3] 0x00000000
Win32Thread: 0x00000000
CrossThreadFlags:
Eip: 0x7c90eb94
  eax=0x77e3e70d ebx=0x00000000 ecx=0x00000000 edx=0x00000000 esi=0x00000000 edi=0x00000102
  eip=0x7c90eb94 esp=0x01c9ff78 ebp=0x01c9ffb4 err=0x00000000
  cs=0x1b ss=0x23 ds=0x23 es=0x23 gs=0x00 efl=0x00000286
  dr0=0x00000000 dr1=0x00000000 dr2=0x00000000 dr3=0x00000000 dr6=0x00000000 dr7=0x00000000

ETHREAD: 0xff1fc230 Pid: 1668 Tid: 1760

Tags:
Created: 2010-08-11 06:06:38
Exited: 1970-01-01 00:00:00
Owning Process: vmtoolsd.exe
Attached Process: vmtoolsd.exe
State: Running
BasePriority: 0x8
Priority: 0x8
TEB: 0x7ffdc000
StartAddress: 0x7c810856 UNKNOWN
ServiceTable: 0x80552140
  [0] 0x80501030
  [1] 0xbf997600
  [2] 0x00000000
  [3] 0x00000000
Win32Thread: 0xe174c4a0
CrossThreadFlags:

ETHREAD: 0xff14ada8 Pid: 1668 Tid: 1872
Tags:
Created: 2010-08-11 06:06:38
Exited: 1970-01-01 00:00:00
Owning Process: vmtoolsd.exe
Attached Process: vmtoolsd.exe
State: Waiting:WrLpcReceive
BasePriority: 0x8
Priority: 0x8
TEB: 0x7ffd9000
StartAddress: 0x7c810856 UNKNOWN
Win32StartAddress: 0x00009505
ServiceTable: 0x80552180
  [0] 0x80501030
  [1] 0x00000000
  [2] 0x00000000
  [3] 0x00000000
Win32Thread: 0x00000000
CrossThreadFlags:
Eip: 0x7c90eb94
  eax=0x00000000 ebx=0x00000000 ecx=0x0015e298 edx=0x000003c0 esi=0x0015e298
edi=0x00000100
  eip=0x7c90eb94 esp=0x017dfe1c ebp=0x017dff80 err=0x00000000
  cs=0x1b ss=0x23 ds=0x23 es=0x23 gs=0x00 efl=0x00000246
  dr0=0x00000000 dr1=0x00000000 dr2=0x00000000 dr3=0x00000000 dr6=0x00000000
dr7=0x00000000

Output from the *threads* plugin for PID 1994, using command "*volatility threads -f zeus.vmem -p 1944,*" is as follows:

```
[x86] Gathering all referenced SSDTs from KTHREADs...
Finding appropriate address space for tables...
------
ETHREAD: 0x010fcda8 Pid: 1944 Tid: 1208
Tags: ScannerOnly
Created: 2010-08-15 19:17:55
Exited: 2010-08-15 19:17:56
Owning Process: VMip.exe
Attached Process: VMip.exe
State: Terminated
BasePriority: 0x8
Priority: 0x10
TEB: 0x00000000
StartAddress: 0x7c810867 UNKNOWN
ServiceTable: 0x80552140
  [0] 0x80501030
  [1] 0xbf997600
  [2] 0x00000000
  [3] 0x00000000
Win32Thread: 0x00000000
CrossThreadFlags: PS_CROSS_THREAD_FLAGS_TERMINATED
```

Examining this output, it is apparent that both processes 124 and 1944 (*cmd.exe* and *VMip.exe*, respectively) have already terminated. However, process 1668 (*vmtoolsd.exe*) has not yet terminated, as its threads continue to execute.

2.3.2.9 Thrdscan plugin

The purpose of this section is to corroborate the results obtained from Section 2.3.2.8, where it was determined based on *threads* plugin that processes 124 and 1944 have since terminated but that process 1668 is still active. To validate these results Volatility command "*volatility thrdscan -f zeus.vmem | grep -P '(\ 124\ |\ 1668\ |1944\ )' | sort -k 2 -n*" yields the following output:

Table 8: Volatility output for the thrdscan plugin.

Offset(P)	PID	TID	Start Address	Create Time	Exit Time
0x04419d80	124	972	0x7c810867	2010-08-15 19:17:55	2010-08-15 19:17:56
0x003f3bf8	1668	1844	0x7c810856	2010-08-11 06:06:38	
0x004a0da8	1668	1872	0x7c810856	2010-08-11 06:06:38	
0x04a55bc0	1668	1380	0x7c810856	2010-08-11 06:07:14	
0x06560230	1668	1760	0x7c810856	2010-08-11 06:06:38	

Offset(P)	PID	TID	Start Address	Create Time	Exit Time
0x069d58b0	1668	1672	0x7c810867	2010-08-11 06:06:35	
0x010fcda8	1944	1208	0x7c810867	2010-08-15 19:17:55	2010-08-15 19:17:56
0x010fcda8	1944	1208	0x7c810867	2010-08-15 19:17:55	2010-08-15 19:17:56

Thus, it can be confirmed that processes 124 and 1944 (*cmd.exe* and *VMip.exe*) have indeed terminated but that process 1668 and its threads remain active. The next step will be to attempt to dump their process space.

Although nothing appears out of the ordinary about these processes, it is important that the investigator follow through on leads. This is especially true since PID 1944 (*VMip.exe*) appears as a hidden process, and even though it has since terminated, PID 124 (*cmd.exe*) was not found to be hidden. Therefore, in order to be thorough, the investigator should finish following these leads prior to moving on to other avenues of the investigation.

2.3.2.10 Memdump, procexedump and procmemdump plugins

Based on the analyses conducted thus far, it may be that processes 124, 1668 and 1944 are potentially malicious. In order to validate this assumption, it is necessary to dump their process space and memory using the *memdump*, *procexedump* and *procmemdump* plugins.

Dumping the process space for PIDs 124, 1668 and 1944 may reveal additional information about them. If successfully dumped, anti-virus scanners can then be used to determine if they contain malicious code. Even though two of the three processes have since terminated, they may continue to occupy space in memory and may therefore be dumped.

In order to dump these potentially malicious processes and memory space, the following four commands are required, three of which are Volatility-specific.

```
$ mkdir memdump; mkdir procexedump; mkdir procmemdump

$ volatility memdump -f zeus.vmem -p 124,1668,1944 --dump-dir=memdump

$ volatility procexedump -f zeus.vmem -p 124,1668,1944 --dump-dir=procexedump

$ volatility procmemdump -f zeus.vmem -p 124,1668,1944 --dump-dir=procmemdump
```

These commands attempt to dump the processes' memory space to the various working directories (created as per the *mkdir* commands). It was possible to dump the memory space occupied by *cmd.exe* (PID 124) and *vmtoolsd.exe* (PID 1668) via the *memdump* plugin. However, for the *procexedump* and *procmemdump* plugins, it was only possible to dump the memory for *vmtoolsd.exe*. In all, four files were generated. It was not possible, however, to dump the memory for *VMip.exe* (PID 1944) using any of the aforementioned plugins. These dumps were then hashed for their SHA1 and fuzzy hash values.

Using all six scanners (see Section 1.6.1 for details), only the Avast scanner detected that one of the dumped processes was potentially infected, specifically file *memdump/1668.dmp*. Avast's output was seen as:

```
../memdump/1668.dmp      [infected by: Win32:Zbot-BCW [Trj]]
```

Since only one scanner picked up this infection, it is unlikely that process PID 1668 is itself infected. Instead, the author posits that older memory space from the active malware, which has since moved on elsewhere in memory, was taken up by PID 1668.

Thus, if one or more memory pages of PID 1668 contained heuristically detectable code or signatures, then it is plausible that a scanner would pick these up, as is the case here. Moreover, this can be readily confirmed by the fact that the process space for PID 1688 dumped by plugins *procexedump* and *procmemdump* were not found to contain any malware, further confirming the author's supposition. Specifically, if the executable code contained therein dumped by the two aforementioned plugins had malicious code, it would be likely that at least one of the scanners would have detected this. Thus, since they were not detected as malicious, it is logical to conclude that for this specific process (PID 1668) the detected Zeus code was a remnant of another possibly infected process.

Comparing the fuzzy hashes of the data carved from the memory image in Section 2.2.3 yielded no similarities with the fuzzy hashes of the data dumped using the *memdump*, *procexedump* and *procmemdump* plugins.

However, comparing the fuzzy hashes of the dumped memory samples in the previous subsection (using plugins *memdump*, *procmemdump* and *procexedump*) revealed that there was an 86% match between the *memdump* of process 1668 and 124. This should not be surprising considering that PID 124 was spawned from PID 1668 and likely inherited memory and DLLs from it. Recall that when using the *memdump* plugin, all of a process' addressable memory is dumped. Thus, it is expected that such similarities will be a common occurrence.

Moreover, a 99% match was found between the fuzzy hashes of the *procexedump* and *procmemdump* dumps for process 1668. This finding is altogether normal. Recall that the *procmemdump* plugin dumps a process' executable code, stack, memory and slack space while the *procexedump* plugin dumps only the process' executable code. Thus, there is a likelihood of similarity between them.

The four data files generated from the *memdump*, *procexedump* and *procmemdump* plugins were compared against the current NSRL hash-set as per their SHA1 hashes. No matches were found. Moreover, the SHA1 hash values for these memory dumps were also compared against those obtained against the data carving of the memory image (see Section 2.2.3 for details). No matches were found in this comparison either.

2.3.2.11 First analysis endeavour summary

The approach undertaken by the author in this section was obviously a wrong turn. However, it provided a useful lesson concerning the thorough investigation of potential leads and demonstrated how to examine in detail processes and threads.

However, another approach must be undertaken to find direct evidence of the malware in memory. Direct process examination turned up little. Prior to examining other potential avenues with Volatility, the use of process and thread listings should be exhausted first. The author considers process memory dumping plugins to be akin to process listings.

The examination of state-based information concerning the memory image may prove to be of use now that it has been determined that the current line of inquiry has turned up no tangible leads.

2.3.3 Second analysis endeavour: hunting and finding the evidence

Upon having completed a primary survey of the infected memory image using process and thread listing plugins, the investigator should consider examining the memory image using state-specific plugins. These plugins are sometimes able to detect anomalies with respect to the memory image's last running known state.

Plugins of use in this section can include command histories[5], open files, devices in use, list of DLLs, drivers and services, event logs, network communications, etc. The first plugin that would have ordinarily been suggested by the author would have been the command history-based plugins, but these were used in Section 2.3.2.

The choice of plugins to use all depends on the evidence trail and logical flow of the investigation under Volatility, some of which have already been established thus far in the investigation.

2.3.3.1 Connscan plugin

The first Volatility plugin that should be used is the *connscan* plugin. It is used to verify for the existence of ongoing network connections. It scans a memory image for current or recently terminated connections. Using command "*volatility connscan -f zeus.vmem*" yields the following output:

Table 9: Volatility output for the connscan plugin.

Offset(P)	Local Address	Remote Address	PID
0x02214988	172.16.176.143:1054	193.104.41.75:80	856
0x06015ab0	0.0.0.0:1056	193.104.41.75:80	856

This plugin reveals several important pieces of new information. The first is that the suspect computer system from whence the memory image originated from has been established by what appears to be an HTTP connection (using port 80) with a remote system with IP address 193.104.41.75. The system from whence the memory image came from has IP address 172.16.176.143.

[5] Command histories generally provide state-based information for command shells, but since the process and thread listing plugins used in Section 2.3.2 found that *cmd.exe* had been used, it made sense to take advantage of Volatility's command history plugins immediately to determine if additional information could be readily obtained.

Upon closer examination, process PID 856 is in fact *svchost.exe*, as based on the information obtained in Section 2.3.2.2. It appears that PID 856 is being used as a cover process for some hidden process carrying out what appears to be a subversive communication. This process, however, should never communicate on this port, thus marking this activity as particularly suspicious. Using other network state-based plugins such as *connections*, *sockets* and *sockscan*, additional information can be determined about this communication channel.

2.3.3.2 Connections plugin

The *connections* plugin can be used to determine information concerning not only ongoing communications, but also for recently terminated network communications and sessions. It therefore makes sense to use this plugin in order to query the memory image for additional network-based information. Using command "*volatility connections -f zeus.vmem*" yielded no output whatsoever.

Although output from the *connscan* plugin indicated that that the HTTP communication appeared to have been instantiated by *svchost.exe* (see Section 2.3.3.1), because it was not seen by the *connections* plugin indicates that it is very likely a covert communication channel.

The use of other network-related plugins may help to reveal or isolate the process conducting this potentially convert communication.

2.3.3.3 Sockets and sockscan plugins

Volatility offers two other network-based plugins, *sockets* and *sockscan*. The *sockets* plugin prints open sockets that may provide additional information about the convert network channel, while the *sockscan* plugin scans a suspect memory image for all TCP sockets.

Consider the following output from the *sockets* plugin, using command "*volatility sockets -f zeus.vmem*":

Table 10: Volatility output for the sockets plugin.

Offset(V)	PID	Port	Proto	Protocol	Address	Create Time
0x80fd1008	4	0	47	GRE	0.0.0.0	8/11/2010 6:08
0xff258008	688	500	17	UDP	0.0.0.0	8/11/2010 6:06
0xff367008	4	445	6	TCP	0.0.0.0	8/11/2010 6:06
0x80ffc128	936	135	6	TCP	0.0.0.0	8/11/2010 6:06
0xff37cd28	1028	1058	6	TCP	0.0.0.0	8/15/2010
0xff20c478	856	29220	6	TCP	0.0.0.0	8/15/2010
0xff225b70	688	0	255	Reserved	0.0.0.0	8/11/2010 6:06
0xff254008	1028	123	17	UDP	127.0.0.1	8/15/2010
0x80fce930	1088	1025	17	UDP	0.0.0.0	8/11/2010 6:06
0xff127d28	216	1026	6	TCP	127.0.0.1	8/11/2010 6:06

Offset(V)	PID	Port	Proto	Protocol	Address	Create Time
0xff206a20	1148	1900	17	UDP	127.0.0.1	8/15/2010
0xff1b8250	688	4500	17	UDP	0.0.0.0	8/11/2010 6:06
0xff382e98	4	1033	6	TCP	0.0.0.0	8/11/2010 6:08
0x80fbdc40	4	445	17	UDP	0.0.0.0	8/11/2010 6:06

Consider the following output from the *sockscan* plugin, using command "*volatility sockscan -f zeus.vmem*":

Table 11: Volatility output for the sockscan plugin.

Offset(P)	PID	Port	Proto	Protocol	Address	Create Time
0x007c0a20	1148	1900	17	UDP	172.16.176.143	8/15/2010 19:15
0x01120c40	4	445	17	UDP	0.0.0.0	8/11/2010 6:06
0x01131930	1088	1025	17	UDP	0.0.0.0	8/11/2010 6:06
0x01134008	4	0	47	GRE	0.0.0.0	8/11/2010 6:08
0x011568a8	4	138	17	UDP	172.16.176.143	8/15/2010 19:15
0x0115f128	936	135	6	TCP	0.0.0.0	8/11/2010 6:06
0x02daad28	216	1026	6	TCP	127.0.0.1	8/11/2010 6:06
0x04863458	4	139	6	TCP	172.16.176.143	8/15/2010 19:15
0x04864578	1028	68	17	UDP	172.16.176.143	8/15/2010 19:17
0x04864a08	4	137	17	UDP	172.16.176.143	8/15/2010 19:15
0x04a4be98	4	1033	6	TCP	0.0.0.0	8/11/2010 6:08
0x04a51d28	1028	1058	6	TCP	0.0.0.0	8/15/2010 19:17
0x04be7008	4	445	6	TCP	0.0.0.0	8/11/2010 6:06
0x05dee200	1028	123	17	UDP	127.0.0.1	8/15/2010 19:15
0x05e33d68	1148	1900	17	UDP	127.0.0.1	8/15/2010 19:15
0x05f44008	688	500	17	UDP	0.0.0.0	8/11/2010 6:06
0x05f48008	1028	123	17	UDP	127.0.0.1	8/15/2010 19:17
0x06236e98	1028	68	17	UDP	172.16.176.143	8/15/2010 19:17
0x06237b70	688	0	255	Reserved	0.0.0.0	8/11/2010 6:06
0x06450478	856	29220	6	TCP	0.0.0.0	8/15/2010 19:17
0x06496a20	1148	1900	17	UDP	127.0.0.1	8/15/2010 19:17
0x069d5250	688	4500	17	UDP	0.0.0.0	8/11/2010 6:06

Information for PID 856 has been highlighted in tables 10 and 11 above.

Based on the output from the *sockets* and *sockscan* plugins, the information obtained using the *connscan* plugin (Section 2.3.3.1) could not be confirmed. Thus, it can be concluded that the process conducting the suspicious network communication to remote system 193.104.41.75 is covert, as plugins that should have been able to detect it could not. Specifically, using the *connections*, *sockets* and *sockscan* plugins, none of them could detect this suspicious network communication.

Moreover, as based on the information examined thus far in this section, process *svchost.exe* presents no useful information about the covert communication channel discovered in Section 2.3.3.1. This should lead the investigator to conclude that process *svchost.exe* is not actually initiating the concealed communication, but that some hidden process that seized control of *svchost.exe* very likely did. It is highly probable that process *svchost.exe* was hijacked (possibly with injected code) in order to instantiate a hidden process that appears as *svchost.exe*.

The next step an investigator should undertake is to determine more information about this remote IP address. A web *whois* search of this IP address may reveal much.

2.3.3.4 Whois suspicious IP address

A web search at http://Whois.net quickly reveals that this IP address currently resides in Transnistria, a territory bordering Ukraine and Moldavia. Moreover, this country has little political autonomy from Russia and geopolitically is an ideal location for a command and control botnet server. Specifically, the output from http://whois.net/ip-address-lookup/193.104.41.75 is as follows:

```
[Querying whois.ripe.net]
[whois.ripe.net]
% This is the RIPE Database query service.
% The objects are in RPSL format.
%
% The RIPE Database is subject to Terms and Conditions.
% See http://www.ripe.net/db/support/db-terms-conditions.pdf

% Note: this output has been filtered.
%       To receive output for a database update, use the "-B" flag.

% Information related to '193.104.41.0 - 193.104.41.255'

inetnum:        193.104.41.0 - 193.104.41.255
netname:        VVPN-NET
descr:          PE Voronov Evgen Sergiyovich
country:        MD
org:            ORG-PESV2-RIPE
admin-c:        ESV1-RIPE
```

```
tech-c:         ESV1-RIPE
status:         ASSIGNED PI
mnt-by:         VVPN-MNT
mnt-by:         RIPE-NCC-END-MNT
mnt-lower:      RIPE-NCC-END-MNT
mnt-routes:     VVPN-MNT
mnt-domains:    VVPN-MNT
source:         RIPE # Filtered

organisation:   ORG-PESV2-RIPE
org-name:       PE Voronov Evgen Sergiyovich
org-type:       OTHER
descr:          PE Evgen Sergeevich Voronov
address:        25 October street, 118-15
address:        Tiraspol, Transdnistria
phone:          +373 533 50404
admin-c:        ESV1-RIPE
tech-c:         ESV1-RIPE
mnt-ref:        VVPN-MNT
mnt-by:         VVPN-MNT
source:         RIPE # Filtered

person:         Evgen Sergeevich Voronov
address:        25 October street, 118-15
address:        Tiraspol, Transdnistria
phone:          +373 533 50404
nic-hdl:        ESV1-RIPE
mnt-by:         VVPN-MNT
source:         RIPE # Filtered

% Information related to '193.104.41.0/24AS49934'

route:          193.104.41.0/24
descr:          PE Voronov Evgen Sergiyovich
origin:         AS49934
mnt-by:         VVPN-MNT
source:         RIPE # Filtered

% This query was served by the RIPE Database Query Service version 1.51.1 (WHOIS1)
```

The next step the investigator should examine is the process space of *svchost.exe* (PID 856). However, since this process has likely been injected with malicious code, it is unlikely that any thread-based plugins will be of much help. Instead, the investigator should consider using a Volatility plugin designed to seek out malware, including injected malware. The easiest of these to use, as it requires no specific software or malware reverse engineering knowledge, is the *malfind* plugin.

2.3.3.5 Malfind plugin

Volatility's *malfind* plugin was specifically designed to search for potentially hidden malware residing within a memory image. Moreover, it may help the investigator extract the actual process (es) associated with PID 856, possibly including hidden process (es). Recall that PID 856 is process *svchost.exe* and that thus far, it appears that it has been hijacked or injected with malicious code. This likely explains the mysterious network communication with remote system 193.104.41.75 (see Section 2.3.3.1).

Using command "*volatility malfind -f zeus.vmem -p 856*" results in the following output, where the memory address for each instance of potential malware detected by the *malfind* plugin for this process has been highlighted.

Process: svchost.exe Pid: 856 Address: 0xb70000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 38, MemCommit: 1, PrivateMemory: 1, Protection: 6

```
0x00b70000  4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00   MZ..............
0x00b70010  b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00   ........@.......
0x00b70020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x00b70030  00 00 00 00 00 00 00 00 00 00 00 00 d0 00 00 00   ................
```

```
0xb70000 4d          DEC EBP
0xb70001 5a          POP EDX
0xb70002 90          NOP
0xb70003 0003          ADD [EBX], AL
0xb70005 0000          ADD [EAX], AL
0xb70007 000400          ADD [EAX+EAX], AL
0xb7000a 0000          ADD [EAX], AL
0xb7000c ff          DB 0xff
0xb7000d ff00          INC DWORD [EAX]
0xb7000f 00b800000000    ADD [EAX+0x0], BH
0xb70015 0000          ADD [EAX], AL
0xb70017 004000          ADD [EAX+0x0], AL
0xb7001a 0000          ADD [EAX], AL
0xb7001c 0000          ADD [EAX], AL
0xb7001e 0000          ADD [EAX], AL
0xb70020 0000          ADD [EAX], AL
0xb70022 0000          ADD [EAX], AL
0xb70024 0000          ADD [EAX], AL
0xb70026 0000          ADD [EAX], AL
0xb70028 0000          ADD [EAX], AL
0xb7002a 0000          ADD [EAX], AL
0xb7002c 0000          ADD [EAX], AL
0xb7002e 0000          ADD [EAX], AL
0xb70030 0000          ADD [EAX], AL
0xb70032 0000          ADD [EAX], AL
0xb70034 0000          ADD [EAX], AL
```

```
0xb70036 0000          ADD [EAX], AL
0xb70038 0000          ADD [EAX], AL
0xb7003a 0000          ADD [EAX], AL
0xb7003c d000          ROL BYTE [EAX], 0x1
0xb7003e 0000          ADD [EAX], AL
```

Process: svchost.exe Pid: 856 Address: 0xcb0000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 1, MemCommit: 1, PrivateMemory: 1, Protection: 6

```
0x00cb0000  b8 35 00 00 00 e9 cd d7 c5 7b 00 00 00 00 00 00   .5.......{......
0x00cb0010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x00cb0020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x00cb0030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
```

```
0xcb0000 b835000000      MOV EAX, 0x35
0xcb0005 e9cdd7c57b      JMP 0x7c90d7d7
0xcb000a 0000          ADD [EAX], AL
0xcb000c 0000          ADD [EAX], AL
0xcb000e 0000          ADD [EAX], AL
0xcb0010 0000          ADD [EAX], AL
0xcb0012 0000          ADD [EAX], AL
0xcb0014 0000          ADD [EAX], AL
0xcb0016 0000          ADD [EAX], AL
0xcb0018 0000          ADD [EAX], AL
0xcb001a 0000          ADD [EAX], AL
0xcb001c 0000          ADD [EAX], AL
0xcb001e 0000          ADD [EAX], AL
0xcb0020 0000          ADD [EAX], AL
0xcb0022 0000          ADD [EAX], AL
0xcb0024 0000          ADD [EAX], AL
0xcb0026 0000          ADD [EAX], AL
0xcb0028 0000          ADD [EAX], AL
0xcb002a 0000          ADD [EAX], AL
0xcb002c 0000          ADD [EAX], AL
0xcb002e 0000          ADD [EAX], AL
0xcb0030 0000          ADD [EAX], AL
0xcb0032 0000          ADD [EAX], AL
0xcb0034 0000          ADD [EAX], AL
0xcb0036 0000          ADD [EAX], AL
0xcb0038 0000          ADD [EAX], AL
0xcb003a 0000          ADD [EAX], AL
0xcb003c 0000          ADD [EAX], AL
0xcb003e 0000          ADD [EAX], AL
```

This output is not particularly revealing to non-software reverse engineering specialists. However, what is important to know is that an MZ PE header was detected at memory address *0xb70000*.

This type of file signature is used to identify Windows-based executables. This is highly indicative of an injected process within the process space of PID 856. Thus, within the memory space of *svchost.exe*, at linear decimal byte offset 11,993,088, one suspicious PE header was found. The other possible malware detected by this plugin is very likely a false positive, as no PE header was found in the above output associated with memory address offset *0xcb0000*. Moreover, based on the above *malfind* output, it is likely that these two processes are hidden.

2.3.3.6 Dumping the suspicious process using Volatility and malfind

Upon finding two suspicious processes within the memory space of *svchost.exe* using the *malfind* plugin, rather than have to carve them out manually using esoteric sub-process byte address conversion to physical memory addresses, the *malfind* plugin can readily perform this on behalf of the investigator. This is done by rerunning the plugin and appending the "*--dump-dir*" parameter to the command. This parameter specifies a directory location to dump the suspicious process. In so doing, it is hoped that the *malfind* plugin will succeed in dumping these two hidden processes associated with PID 856. Running the command "*volatility malfind -f zeus.vmem -p 856 --dump-dir=*." generates the following two files:

process.0x80ff88d8.0xb70000.dmp

process.0x80ff88d8.0xcb0000.dmp

The two filenames listed above correspond to the order in which they were displayed by Volatility's *malfind* plugin and the memory address offsets where they were located within PID 856's memory space.

Performing a SHA1 hash of the two dump files above and comparing them against the current NSRL hash-set resulted in no matches being established. SHA1 comparisons between them and the data carved from the memory image (see Section 2.2.3 for details) also resulted in no established matches. Moreover, conducting a fuzzy hash comparison of the two dumped processes above against those obtained from the data carving of the memory image (see Section 2.2.3 for details) also resulted in no partial matches being established.

Note that the files for both process dumps consist of *0x80ff88d8*. This virtual memory address was found to be in use for *svchost.exe* (PID 876) as found in Section 2.3.2.2 (Table 5). This indicates that these dumped processes can be directly attributed to PID 856.

The next step is to determine if the various anti-virus scanners can reveal about these two dumped processes.

2.3.3.7 Virus scanning and hash verification of *malfind*-dumped PID 856

Using the six anti-virus scanners (see Section 1.6.1 for details) against the two *malfind*-extracted files produced in Section 2.3.3.5, the following scanner messages and alerts were produced:

Avast:

process.0x80ff88d8.0xb70000.dmp [infected by: Win32:Zbot-BCW [Trj]]

process.0x80ff88d8.0xcb0000.dmp [OK]

AVG:

process.0x80ff88d8.0xb70000.dmp Virus found Win32/Heri

BitDefender:

process.0x80ff88d8.0xb70000.dmp infected: Gen:Variant.Graftor.22830

process.0x80ff88d8.0xcb0000.dmp ok

ClamAV:

process.0x80ff88d8.0xb70000.dmp: OK

process.0x80ff88d8.0xcb0000.dmp: OK

FRISK:

[Found security risk] <W32/Zbot.AG.gen!Eldorado (generic, not disinfectable)> process.0x80ff88d8.0xb70000.dmp

McAfee:

process.0x80ff88d8.0xb70000.dmp ... Found the PWS-Zbot.gen.ub trojan !!!

It appears that five of the six scanners have found infections within the *malfind*-based dump process file *process.0x80ff88d8.0xb70000.dmp* for PID 856. Three of the six scanners identified it as the Zeus Trojan horse. It is interesting to note that while five of the six scanners identified the dump file *process.0x80ff88d8.0xb70000.dmp* as infected, its SHA1 and fuzzy hash signatures do not match (see Section 2.3.3.6 for more information), even partially, any of those files carved from the memory image that were identified as Zeus-based in Section 2.2.3.

Not surprisingly, the second process dumped by the *malfind* plugin, found within PID 856's memory space at address offset *0xcb0000*, was found to be uninfected.

2.3.3.8 Filescan plugin

Now that the Zeus infection has been found in memory, including both in the actual process where it was running and in the process where it injected code to hide itself (see sections 2.3.3.5, 2.3.3.6 and 2.3.3.7 for details), it is time to attempt to determine which files on disk were responsible for this infection.

To determine this, the *filescan* plugin can be used. This plugin searches memory for open file handles. Unfortunately, it is not able to directly link files to processes. The best manner for finding indications is twofold. First, using keywords (e.g. Zeus, infection, rootkit, etc.) it may be possible to find the infection, as malware programmers do not always remember to use innocuous looking filenames. Of course, this is at best a hit and miss approach. Secondly, attempt to detect suspect files based on their names and locations. This process requires that the investigator have a

good working knowledge of the operating system from whence the memory image originated, as just looking blindly at filenames[6] is not likely to produce meaningful results.

Nevertheless, with the necessary knowledge of file listing and hash-sets, this plugin may help investigators pinpoint the actual file(s) causing the infection. Based on the information found in [7, 9, 17 and 18] concerning the Zeus botnet, it is known that "*variant 4*" instances of Zeus-based infections rely on filenames of *sdra*, *lowsec\user.ds* and *lowsec\local.ds*. With this knowledge, the investigator is equipped with the necessary knowledge to find additional evidence of this specific malware infection using the *grep* command.

Running the command "*volatility filescan -f zeus.vmem | grep -i '(zeus|sdra|lowsec)'* " in order to look for "*variant 4*" based evidence of Zeus generates the following output:

```
0x061abef8      1      0 R--r-d \Device\HarddiskVolume1\Documents and Settings\Administrator\Desktop\ZeuS_binary_5767b2c6d84d87a47d12da03f4f376ad.exe

0x029d9b40    1    1 R----- \Device\HarddiskVolume1\WINDOWS\system32\sdra64.exe

0x029d9cf0                    1                    0    -WD--- \Device\HarddiskVolume1\WINDOWS\system32\sdra64.exe

0x01061028                    1                    0    RW-r-- \Device\HarddiskVolume1\WINDOWS\system32\lowsec\user.ds

0x0115ab90                    1                    1    R----- \Device\HarddiskVolume1\WINDOWS\system32\lowsec\user.ds

0x02bbe470                    1                    1    R----- \Device\HarddiskVolume1\WINDOWS\system32\lowsec\local.ds
```

Of course, had the infection not been "*variant 4*" based, the other sources of Zeus information cited herein may have been of use. If not, the investigator must exercise painstaking attention to the details found by the *filescan* plugin and attempt to spot filenames that simply do belong in the *\WINDOWS\system32* directory.

2.3.3.9 Summary

This subsection succeeded in locating and extracting the malware suspected of having infected the memory image. The malware seems to be based on a known version of the Zeus Trojan horse, although many versions of this malware are known to exist with differing detection signatures. As no disk image was available for examination in this analysis, it is not possible to attempt to match disk-based malware signatures with the known malware signatures for Zeus. Relying on memory-

[6] Recall that a good source of filenames is the NSRL hash-set. It is broken by product and operating system.

extracted malware signatures and comparing them against known disk-based signatures would be ineffective, as memory-resident executables are modified while running in memory.

Specifically, Zeus is a botnet-based Trojan horse primarily intended for stealing victims' banking information. It is stealthy and difficult to detect while running on a given Windows-based system even with up to date virus scanners, as seen in the analysis carried out herein. [4]

Even though the infection has been found and extracted, the final part of the investigation is to determine, if possible, how it was loaded by the system. It is likely that it was loaded by the registry. As with many other malware, once a system becomes infected, the malware makes changes to the victim Windows system's registry so that it remains persistent. These changes are generally made to the registry settings affecting system boot-up or user logons.

2.3.4 Pruning the registry for more information

The Windows registry can serve to both complicate and facilitate the investigator's work. It is commonly used by malware to store its settings and configure the victim's system to load it at boot up or user login. However, the difficulty in working with the registry lies in knowing where to look. The registry is spread out across many data files (also commonly known as registry hives) in various locations and each serves a specific purpose with respect to system, application and user configurations. Reference [6] provides additional background information.

2.3.4.1 Hivelist plugin

The purpose in using the *hivelist* plugin is to determine which registry hives[7] are available in the memory image. Running the command "*volatility hivelist -f zeus.vmem*" generates the following output:

Table 12: Volatility output for the hivelist plugin.

Virtual	Physical	Name
0x8066e904	0x0066e904	[no name]
0xe1008978	0x01824978	[no name]
0xe101b008	0x01867008	\Device\HarddiskVolume1\WINDOWS\system32\config\system
0xe13ae580	0x01bbd580	[no name]
0xe1537b60	0x06ae4b60	\SystemRoot\System32\Config\SECURITY
0xe153ab60	0x06b7db60	\Device\HarddiskVolume1\WINDOWS\system32\config\software
0xe1542008	0x06c48008	\Device\HarddiskVolume1\WINDOWS\system32\config\default
0xe1544008	0x06c4b008	\Device\HarddiskVolume1\WINDOWS\system32\config\SAM
0xe1a33008	0x01f98008	\Device\HarddiskVolume1\Documents and Settings\NetworkService\NTUSER.DAT

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

Virtual	Physical	Name
0xe1a39638	0x021eb638	\Device\HarddiskVolume1\Documents and Settings\NetworkService\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
0xe1c41b60	0x04010b60	\Device\HarddiskVolume1\Documents and Settings\LocalService\NTUSER.DAT
0xe1c49008	0x036dc008	\Device\HarddiskVolume1\Documents and Settings\LocalService\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
0xe1da4008	0x00f6e008	\Device\HarddiskVolume1\Documents and Settings\Administrator\NTUSER.DAT
0xe1e158c0	0x009728c0	\Device\HarddiskVolume1\Documents and Settings\Administrator\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat

In the above table, the *Virtual* and *Physical* columns refer to virtual and physical memory address offsets, respectively. The column *Name* refers to data file locations (registry hives) of the underlying registry hives.

It is important to note that not all the registry hives generated from the *hivelist* plugin are of immediate use. This is based in part on the experience of the investigator and on what he may hope to find as evidence. The following table provides an overview of the various registry hives contained within this memory image that are, in the opinion of the author, the most likely to contain registry-based evidence of malware infection. This is based on the associated root registry keys and virtual memory address offsets, as seen in Annex D and Table 12 (above), respectively.

Table 13: Association between registry hives and their corresponding registry keys commonly used for registry-based infections as per the hivelist output.

Registry data file (hive)	Associated root registry key	Virtual Offset
\Device\HarddiskVolume1\Documents and Settings\LocalService\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat	HKEY_CURRENT_USER\SOFTWARE	0xe1a39638
\Device\HarddiskVolume1\Documents and Settings\LocalService\NTUSER.DAT	HKEY_CURRENT_USER\SOFTWARE	0xe1a33008
\Device\HarddiskVolume1\Documents and Settings\NetworkService\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat	HKEY_CURRENT_USER\SOFTWARE	0xe1a39638
\Device\HarddiskVolume1\Documents and Settings\NetworkService\NTUSER.DAT	HKEY_CURRENT_USER\SOFTWARE	0xe1c41b60
\Device\HarddiskVolume1\Documents and Settings\Administrator\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat	HKEY_CURRENT_USER\SOFTWARE	0xe1e158c0
\Device\HarddiskVolume1\Documents and Settings\Administrator\NTUSER.DAT	HKEY_CURRENT_USER\SOFTWARE	0xe1da4008

Registry data file (hive)	Associated root registry key	Virtual Offset
\Device\HarddiskVolume1\WINDOWS\system32\config\software	HKEY_LOCAL_MACHINE\SOFTWARE	0xe153ab60
\Device\HarddiskVolume1\WINDOWS\system32\config\system	HKEY_LOCAL_MACHINE\SYSTEM	0xe101b008

2.3.4.2 Printkey plugin

The purpose of the *printkey* plugin is to extract registry key information from specific registry hives, as found in the memory image, using the *hivelist* plugin results of the preceding section.

Based on the information provided by Annex D and Section 2.3.4.1, the following *HKLM\SOFTWARE*-specific registry keys are examined using the *printkey* plugin as based on the following Volatility commands:

Command Set (1):

```
volatility -printkey -o 0xe153ab60 -K 'Microsoft\Windows NT\CurrentVersion\Winlogon' -f zeus.vmem

volatility -printkey -o 0xe153ab60 -K 'Microsoft\Windows NT\CurrentVersion\Winlogon\Notify' -f zeus.vmem

volatility -printkey -o 0xe153ab60 -K 'Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects' -f zeus.vmem

volatility -printkey -o 0xe153ab60 -K 'Microsoft\Windows\CurrentVersion\Explorer\SharedTaskScheduler' -f zeus.vmem

volatility -printkey -o 0xe153ab60 -K 'Microsoft\Windows\CurrentVersion\Policies\Explorer\Run' -f zeus.vmem

volatility -printkey -o 0xe153ab60 -K 'Microsoft\Windows\CurrentVersion\Run' -f zeus.vmem

volatility -printkey -o 0xe153ab60 -K 'Microsoft\Windows\CurrentVersion\ShellServiceObjectDelayLoad' -f zeus.vmem
```

Based on the information provided in Annex D and Section 2.3.4.1, the following *HKLM\SYSTEM*-specific registry keys are examined using the *printkey* plugin as based on the following Volatility commands:

Command Set (2):

```
volatility -printkey -o 0xe101b008 -K 'ControlSet001\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile\AuthorizedApplications\List' -f zeus.vmem

volatility -printkey -o 0xe101b008 -K 'CurrentControlSet\Services' -f zeus.vmem
```

Based on the information provided in Annex D and Section 2.3.4.1, the following *HKCU\SOFTWARE*-specific registry keys are examined using the *printkey* plugin as based on the following Volatility commands:

Command Set (3):

```
volatility -printkey -o 0xe1c41b60 -K
'Microsoft\Windows\CurrentVersion\Explorer\RunMRU' -f zeus.vmem

volatility -printkey -o 0xe1c41b60 -K
'Microsoft\Windows\CurrentVersion\Explorer\UserAssist' -f zeus.vmem

volatility -printkey -o 0xe1c41b60 -K 'Microsoft\Windows\CurrentVersion\Run' -f
zeus.vmem

volatility -printkey -o 0xe1e158c0 -K
'Microsoft\Windows\CurrentVersion\Explorer\RunMRU' -f zeus.vmem

volatility -printkey -o 0xe1e158c0 -K
'Microsoft\Windows\CurrentVersion\Explorer\UserAssist' -f zeus.vmem

volatility -printkey -o 0xe1e158c0 -K 'Microsoft\Windows\CurrentVersion\Run' -f
zeus.vmem
```

The various memory addresses, as specified in the abovementioned commands, have been highlighted to aid in the differentiation between the virtual address offsets used in the previously mentioned commands.

2.3.4.3 Output from the various printkey commands

The output, based on the *HKLM\SOFTWARE* hives registry keys and the *Command Set (1)*, after pruning duplicate and non-useful output, has resulted in the following evidence:

```
REG_DWORD    AutoRestartShell : (S) 1
REG_SZ       DefaultDomainName : (S) BILLY-DB5B96DD3
REG_SZ       DefaultUserName : (S) Administrator
REG_SZ       LegalNoticeCaption : (S)
REG_SZ       LegalNoticeText : (S)
REG_SZ       PowerdownAfterShutdown : (S) 0
REG_SZ       ReportBootOk : (S) 1
REG_SZ       Shell : (S) Explorer.exe
REG_SZ       ShutdownWithoutLogon : (S) 0
REG_SZ       System : (S)
```

REG_SZ Userinit : (S) C:\WINDOWS\system32\userinit.exe,C:\WINDOWS\system32\sdra64.exe,

REG_SZ VmApplet : (S) rundll32 shell32,Control_RunDLL "sysdm.cpl"

REG_DWORD SfcQuota : (S) 4294967295

REG_SZ allocatecdroms : (S) 0

REG_SZ allocatedasd : (S) 0

REG_SZ allocatefloppies : (S) 0

REG_SZ cachedlogonscount : (S) 10

REG_DWORD forceunlocklogon : (S) 0

REG_DWORD passwordexpirywarning : (S) 14

REG_SZ scremoveoption : (S) 0

REG_DWORD AllowMultipleTSSessions : (S) 1

REG_EXPAND_SZ UIHost : (S) logonui.exe

REG_DWORD LogonType : (S) 1

REG_SZ Background : (S) 0 0 0

REG_SZ AutoAdminLogon : (S) 0

REG_SZ DebugServerCommand : (S) no

REG_DWORD SFCDisable : (S) 0

REG_SZ WinStationsDisabled : (S) 0

REG_DWORD HibernationPreviouslyEnabled : (S) 1

REG_DWORD ShowLogonOptions : (S) 0

REG_SZ AltDefaultUserName : (S) Administrator

REG_SZ AltDefaultDomainName : (S) BILLY-DB5B96DD3

Information highlighted in red above pinpoints the registry key responsible for re-loading the Zeus malware into memory. Specifically, the system becomes re-infected every time the Administrator user logs in to the system.

The output, based on the *HKLM\SYSTEM* hives and registry keys from *Command Set (2)*, after pruning duplicate and non-useful output, has resulted in the following evidence:

REG_SZ %windir%\system32\sessmgr.exe : (S) %windir%\system32\sessmgr.exe:*:enabled:@xpsp2res.dll,-22019

REG_DWORD EnableFirewall : (S) 0

From this output, it is not possible to determine if the system's firewall was already disabled or if it was disabled by the Zeus Trojan horse, as it is known to have this ability [8].

Finally, the output based on the *HKCU\SOFTWARE* hives and registry keys and the *Command Set (3)* resulted in no useful information.

2.3.4.4 Userassist plugin

The final Volatility plugin that will be run against the memory image is *userassist*. This plugin has the potential to provide, among other things, additional registry-based information pertaining to programs run and files opened by the user. Its output, after pruning it of non-useful information, has resulted in the following evidence:

```
REG_BINARY    UEME_RUNPATH:C:\Documents and
Settings\Administrator\Desktop\ZeuS_binary_5767b2c6d84d87a47d12da03f4f376ad.exe
```

This output indicates that a *UserAssist* registry artifact was found and its name is indicative of the Zeus infection. Moreover, it coincides directly with the evidence found using the *filescan* plugin (see Section 2.3.3.8). Furthermore, since this evidence was found in the *UEME_RUNPATH*, it was executed. Whether this executable contains an actual instance of the Zeus Trojan horse is not known at this time, as the file is not available for analysis. However, based on the name of the file, it is not a part of any Windows or known application installation. Thus, it can be said that this executed file is in some way related to the infection.

3 Memory analysis issues

3.1 Memory analysis problems

Although memory forensics has begun to change how computer forensic investigations are conducted, there is much work yet to be done. It is still largely a field predominantly comprised of software reverse engineers. Moreover, even once an infection has been isolated, an in-depth understanding of the infection is predominantly obtained through the reverse engineering of the malware.

Unfortunately, each memory analysis framework is quite different. Further complicating the matter is the fact that the memory analysis capabilities of the frameworks provided by the main vendors (e.g. FTK, EnCase, Paraben) have not yet caught up with the capabilities of their competitors (e.g. Volatility, HBGary Responder and DNA, Mandiant Redline, etc.). This has the added effect of creating a fragmented marketplace. In addition, each framework is distinct with its own learning curve and nuances. Moreover, some have been designed for non-memory specialists, while others are difficult to use and comprehend by anyone other than software reverse engineers. The documentation of these various frameworks, whether commercial, free or open source, is largely lacking and of poor quality. The provided literature with these products is always obvious even to non-experienced memory specialists.

Finally, further complicating the matter is that these frameworks primarily support Windows-based systems, although Volatility does provide some non-Windows support. It is worth mentioning that Volatility's Linux-based support is continuing to improve.

3.2 The uses of memory analysis

Memory analysis, when working against a given memory image, can readily enable the investigator to determine a variety of facts about a suspect system's state at the time of the memory's acquisition. It can be used to determine what documents a suspect was working on, what network activities he was currently or recently involved in. Pictures can be readily detected and extracted using data carving techniques. Evidence of malicious activity such as computer malware infections can be found and used to corroborate evidence found on disk or to detect newer strains of malware that never write to disk. Memory analysis can even reveal encryption keys and passwords that can be used to decrypt locked volumes and files, including accessing user files and shared network drives.

Finally, computer memory forensics is the latest chapter in computer-based forensics and there is still a great deal of work, research and innovation to be coaxed from memory acquisition and analysis.

4 Conclusion

What can be concluded from this work is that using solid investigative footwork, combined with the capabilities of the Volatility memory forensics framework, investigators can readily analyse and investigate memory-based infections. The Zeus Trojan horse was not particularly obvious to find with respect to process-based listings, but it left actual traces of its activity through its extensive use of the registry and its cover communications channel. Of course, armed with various virus reports, it was possible to determine other potential sources of evidence.

Moreover, using Volatility's malware-finding and dumping plugin *malfind*, it was possible to not only find the process that had been hijacked but even to dump the actual Zeus process. Throughout this document, the author has demonstrated the manner in which a forensic memory analysis could be conducted by non-memory specialists using a comprehensive, yet easy to follow, memory analysis methodology. Thus, even novice memory investigators can successfully examine relatively difficult memory analyses, when armed with a usable technique and methodology, as well as the necessary background information concerning the infection.

However, not all analyses to be conducted will be able to rely on many well-prepared virus reports. Furthermore, not all investigations will be carried out against known malware, as malware is constantly evolving. Nevertheless, the techniques and methodology presented herein will be of use against even these newer malware.

This document, the first in a series of many, walks the reader through various malware memory infections in the hope of building a sufficient compendium of knowledge for memory analysis. While the degree of difficulty will vary substantially throughout these analyses, they will provide a means for investigators to rely on in learning how to carry out their own memory investigations.

References

[1] Carbone, Richard. File recovery and data extraction using automated data recovery tools: A balanced approahc using Windows and Linux when working with an unknown disk image and filesystem. Technical Memorandum. DRDC Valcartier. TM 2009-161. August 2009.

[2] Volatility. Command Reference 2.2. Online article. Volatility 2012. http://code.google.com/p/volatility/wiki/CommandReference22.

[3] Carbone, Richard and Vincent, Simon. Installing and configuring a declassification system: A solution combining Trusted Solaris and Free and Open Source Software. Technical Memorandum. DRDC Valcartier. TM 2009-086. January 2013.

[4] Wikipedia. Zeus (Trojan horse). Online encyclopaedic article. Wikimedia Foundation. February 2013. http://en.wikipedia.org/wiki/Zeus_(Trojan_horse).

[5] F-Secure. Top10 malware registry launchpoints. Online article. F-Secure.com. June 2007. http://www.f-secure.com/weblog/archives/00001207.html.

[6] Wikipedia. Windows Registry. Online encyclopaedic article. Wikimedia Foundation. February 2013. http://en.wikipedia.org/wiki/Windows_Registry.

[7] IOActive Inc. Reversal and Analysis of Zeus and SpyEye Banking Trojans. Technical wgitepaper. IOActive Inc. 2012. http://www.ioactive.com/pdfs/ZeusSpyEyeBankingTrojanAnalysis.pdf.

[8] PC Tools by Symantec. Zeus Almighty's Handcrafter PDF Files. Online article. PC Tools by Symantec. 2012. http://www.pctools.com/security-news/zeus-almightys-handcrafted-pdf-files/.

[9] Binsalleeh, H., Ormerod, T., et al. On the analysis of the Zeus Botnet Crimeware Toolkit. Technical paper. Computer Security Laboratory, Concordia University. http://www.ncfta.ca/papers/On_the_Analysis_of_the_Zeus_Botnet_Crimeware.pdf.

[10] BitDefender. Trojan.Spy.Zeus.W. Online article. BitDefender. http://www.bitdefender.com/VIRUS-1000496-en--Trojan-Spy-Zeus-W.html.

[11] Zeustracker.abuse.ch. ZeuS Tracker :: FAQ. Online article / FAQ. June 2009. Zeustracker.abuse.ch. https://zeustracker.abuse.ch/faq.php.

[12] Waheed, Shahzad. Implementation and evaluation of a botnet analysis and detection methods in a virtual environment. Technical paper. Document No.: 01007306. August 2012. Edinburgh Hapier University. http://researchrepository.napier.ac.uk/5667/1/Waheed.pdf.

[13] Mcafee. Summary: PWS-Zbot. Online article. June 2012. https://kc.mcafee.com/resources/sites/MCAFEE/content/live/PRODUCT_DOCUMENTATION/23000/PD23030/en_US/McAfee_Labs_Threat_Advisory_PWS_ZBot.pdf.

[14] Wyke, James. What is Zeus? Technical paper. SophosLabs UK. May 2011. http://www.sophos.com/medialibrary/PDFs/technical%20papers/Sophos%20what%20is%20zeus%20tp.pdf.

[15] Falliere, Nicolas and Chien, Eric. Zeus: King of the Bots. Technical paper. Symantec. http://courses.isi.jhu.edu/malware/papers/ZEUS.pdf.

[16] Michaud, F. and Carbone, R. Practical verification & safeguard tools for C/C++. Technical Report. TR 2006-735. DRDC Valcartier. http://cradpdf.drdc-rddc.gc.ca/PDFS/unc69/p528977.pdf.

[17] Wallisch, Phil. Physical Memory Standard Operating Procedures: HBGary Memory Forensic Tools. Technical paper (draft). Morgan Stanley. May 2010. http://info.publicintelligence.net/HBGary-MorganStanley.pdf.

[18] McMahon, David. PSTP08-0107eSec Combating Robot Networks and Their Controllers: A Study for the Public Security and Technical Program (PSTP). Technical paper. PSTP 08-0107ESEC Version 3.0. Bell Canada for Defence R&D Canada Public Security S&T Summer Symposium 2009. December 2010.

Annex A Anti-virus scanner logs for data carved files

A.1 Avast

carving/recup_dir.1/f0198744.exe [infected by: Win32:Malware-gen]
carving/recup_dir.1/f0198048.dll [infected by: Win32:Zbot-BCW [Trj]]
carving/recup_dir.1/f0126936.exe [infected by: Win32:Zbot-BCW [Trj]]
carving/recup_dir.1/f0179568.dll [infected by: Win32:Zbot-BCW [Trj]]
carving/recup_dir.1/f0186296.dll [infected by: Win32:Zbot-BCW [Trj]]
carving/recup_dir.1/f0122376.dll [infected by: Win32:Zbot-BCW [Trj]]
carving/recup_dir.1/f0083472.exe [infected by: Win32:Malware-gen]
carving/recup_dir.1/f0169264.dll [infected by: Win32:Zbot-BCW [Trj]]
carving/recup_dir.1/f0179776.dll [infected by: Win32:Zbot-BCW [Trj]]
carving/recup_dir.1/f0069472.dll [infected by: Win32:Zbot-BCW [Trj]]
carving/recup_dir.1/f0026720.dll [infected by: Win32:Zbot-BCW [Trj]]
carving/recup_dir.1/f0102992.exe [infected by: Win32:Zbot-BCW [Trj]]
carving/recup_dir.1/f0123288.exe [infected by: Win32:Zbot-BCW [Trj]]
carving/recup_dir.1/f0078696.exe [infected by: Win32:SwPatch [Wrm]]
carving/recup_dir.1/f0108688.dll [infected by: Win32:Zbot-BCW [Trj]]
carving/recup_dir.1/f0016384.exe [infected by: Win32:Zbot-BCW [Trj]]
carving/recup_dir.1/f0136384.dll [infected by: Win32:Zbot-BCW [Trj]]
carving/recup_dir.1/f0068952.dll [infected by: Win32:Zbot-BCW [Trj]]
carving/recup_dir.1/f0031840.dll [infected by: Win32:Zbot-BCW [Trj]]
carving/recup_dir.1/f0009000.exe [infected by: Win32:SwPatch [Wrm]]
carving/recup_dir.2/f0127576.dll [infected by: Win32:Zbot-BCW [Trj]]

A.2 AVG

carving/recup_dir.1/f0198744.exe Virus found Win32/Heri
carving/recup_dir.1/f0189184.exe Trojan horse Pakes.AW
carving/recup_dir.1/f0198048.dll Virus identified Win32/Cryptor
carving/recup_dir.1/f0197560.dll Virus found Win32/Heur
carving/recup_dir.1/f0126936.exe Virus identified Win32/Cryptor
carving/recup_dir.1/f0169216.dll Virus found Win32/Heur
carving/recup_dir.1/f0106544.exe Virus found Win32/Heur
carving/recup_dir.1/f0126048.exe Virus found Win32/Heri
carving/recup_dir.1/f0083472.exe Virus found Win32/Heri
carving/recup_dir.1/f0176824.exe Virus found Win32/Heri
carving/recup_dir.1/f0076592.exe Virus found Win32/Heri
carving/recup_dir.1/f0080496.exe Virus found Win32/Heri
carving/recup_dir.1/f0096936.exe Virus found Win32/Heri
carving/recup_dir.1/f0197216.exe Virus found Win32/Heri
carving/recup_dir.1/f0062328.exe Virus found Win32/Heri

carving/recup_dir.1/f0123288.exe Virus identified Win32/Cryptor
carving/recup_dir.1/f0105344.exe Virus found Win32/Heri
carving/recup_dir.1/f0152824.exe Virus found Win32/Heri
carving/recup_dir.1/f0104608.exe Virus found Win32/Heur
carving/recup_dir.1/f0070384.exe Virus found Win32/Heri
carving/recup_dir.1/f0078696.exe Virus found Win32/Heri
carving/recup_dir.1/f0169152.dll Virus found Win32/Heur
carving/recup_dir.1/f0135928.dll Virus found Win32/Heur
carving/recup_dir.1/f0144008.exe Virus found Win32/Heri
carving/recup_dir.1/f0095216.exe Virus found Win32/Heri
carving/recup_dir.2/f0215816.exe Trojan horse Pakes.AW
carving/recup_dir.2/f0202808.exe Virus found Win32/Heri
carving/recup_dir.2/f0202184.exe Virus found Win32/Heri
carving/recup_dir.2/f0206136.exe Virus found Win32/Heri
carving/recup_dir.2/f0217400.exe Virus found Win32/Heur
carving/recup_dir.2/f0209592.exe Virus found Win32/Heri
carving/recup_dir.2/f0214736.dll Virus found Win32/Heur

A.3 BitDefender

carving/recup_dir.1/f0215816.exe infected:Gen:Trojan.Heur.FU.hqW@aahezco
carving/recup_dir.1/f0202808.exe infected: Gen:Trojan.Heur.JP.hqW@aqTeVHc
carving/recup_dir.1/f0198744.exe infected: Trojan.Generic.8251755
carving/recup_dir.1/f0053360.exe infected: Backdoor.Bot.156746
carving/recup_dir.1/f0126048.exe infected: Gen:Trojan.Heur.FU.hqW@aqTeVHc
carving/recup_dir.1/f0083472.exe infected: Trojan.Generic.7400965
carving/recup_dir.1/f0176824.exe infected: Gen:Trojan.Heur.JP.hqW@aqTeVHc
carving/recup_dir.1/f0152824.exe infected: Gen:Trojan.Heur.FU.hqW@aqTeVHc
carving/recup_dir.1/f0078696.exe infected: Gen:Trojan.Heur.FU.hqW@aqTeVHc

A.4 ClamAV

carving/recup_dir.1/f0152992.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0144496.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0144144.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0189184.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0026008.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0198048.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0005728.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0146736.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0137232.dll: PUA.Win32.Packer.Msvcpp FOUND
carving/recup_dir.1/f0102264.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0169616.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0132888.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND

carving/recup_dir.1/f0169216.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0013544.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0007368.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0185552.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0145024.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0091520.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0186296.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0122376.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0169264.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0179776.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0176824.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0063288.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0146176.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0144536.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0118024.dll: PUA.Win32.Packer.BorlandDelphiKo FOUND
carving/recup_dir.1/f0096936.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0069472.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0118440.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0008976.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0189832.dll: PUA.Win32.Packer.Msvcpp FOUND
carving/recup_dir.1/f0159152.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0026848.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0112520.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0026720.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0024864.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0018848.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0048840.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0094808.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0184400.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0174120.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0050760.dll: PUA.Win32.Packer.Msvcpp FOUND
carving/recup_dir.1/f0056096.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0179536.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0169136.dll: PUA.Win32.Packer.Msvcpp FOUND
carving/recup_dir.1/f0142112.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0088992.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0186488.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0059664.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0165528.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0104608.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0150376.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0144488.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0134080.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0078696.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0137240.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND

carving/recup_dir.1/f0118992.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0108688.dll: PUA.Win32.Packer.Msvcpp FOUND
carving/recup_dir.1/f0110056.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0102544.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0159528.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0018152.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0130352.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0135928.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0136384.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0153976.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0068952.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0031840.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0075624.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0191400.exe: PUA.Win32.Packer.Msvcpp FOUND
carving/recup_dir.1/f0144008.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0192048.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0081112.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0138992.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0144928.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0024528.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0063824.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0030696.exe: PUA.Win32.Packer.Msvcpp FOUND
carving/recup_dir.1/f0042984.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0058272.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0141512.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0008576.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0089600.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0102320.exe: PUA.Win32.Packer.Msvcpp FOUND
carving/recup_dir.1/f0083224.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.1/f0157160.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.2/f0215816.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.2/f0203632.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.2/f0260944.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.2/f0252472.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.2/f0219592.exe: PUA.Win32.Packer.NspackDotnetNor-1 FOUND
carving/recup_dir.2/f0210064.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.2/f0204448.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.2/f0216392.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.2/f0206136.exe: PUA.Win32.Packer.Msvcpp FOUND
carving/recup_dir.2/f0213520.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.2/f0250272.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.2/f0203192.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.2/f0203408.dll: PUA.Win32.Packer.Msvcpp FOUND
carving/recup_dir.2/f0243048.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.2/f0209976.dll: PUA.Win32.Packer.Msvcpp FOUND

carving/recup_dir.2/f0221920.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.2/f0203288.dll: PUA.Win32.Packer.Msvcpp FOUND
carving/recup_dir.2/f0202984.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.2/f0254128.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.2/f0218960.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.2/f0219024.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.2/f0209968.dll: PUA.Win32.Packer.Msvcpp FOUND
carving/recup_dir.2/f0209608.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.2/f0218064.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.2/f0204816.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.2/f0250240.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND
carving/recup_dir.2/f0241064.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND

A.5 F-Prot

[Found security risk] <W32/Zbot.AG.gen!Eldorado (generic, not disinfectable)>
carving/recup_dir.1/f0126048.exe
[Found security risk] <W32/Zbot.AG.gen!Eldorado (generic, not disinfectable)>
carving/recup_dir.1/f0078696.exe

A.6 McAfee

McAfee was the only anti-virus scanner unable to detect any malware whatsoever for the data carved files recovered.

This page intentionally left blank.

Annex B Volatility Windows-based plugins

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

Table B.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
gditimers	Print installed GDI timers and callbacks

Plugin	**Capability (as per *Volatility --help* output)**
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
printkey	Print a registry key, and its subkeys and values
procexedump	Dump a process to an executable file sample

Plugin	**Capability (as per *Volatility --help* output)**
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
windows	Print Desktop Windows (verbose details)
wintree	Print Z-Order Desktop Windows Tree
wndscan	Pool scanner for tagWINDOWSTATION (window stations)

Plugin	**Capability (as per *Volatility --help* output)**
yarascan	Scan process or kernel memory with Yara signatures

Annex C NSRL file hash matches for data carved files

This annex provides a listing of those data carved files obtained in Section 2.2.3 that matched the SHA1 hashes of the NSRL hash-set (September 2012). In all, eleven NSRL SHA1 matches were obtained. In turn, these eleven matches resulted in 793 SHA1-filename matches. However, after taking into account repeating SHA1-filename matches, 65 unique matches were found. These unique SHA1-filename matches, based on the eleven NSRL SHA1 hash matches obtained in Section 2.2.3 are as follows:

Table C.1: Data carved file SHA1-filename matches as per the NSRL

SHA1 hash	File name
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
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
9537335B7EDA9AE3D1C125BE7BAC3161D5B853B8	comctl.man
9537335B7EDA9AE3D1C125BE7BAC3161D5B853B8	COMCTL.MAN
9537335B7EDA9AE3D1C125BE7BAC3161D5B853B8	X86_POLICY.6.0.MICROSOFT.WINDOWS.COMMON-CONTROLS_6595B64144CCF1DF_6.0.2600.2180_X-
A8139A5A5BCC413090176ECAF41510AA0FFBB987	Windows Catalog.lnk
B97B75F861EE499D00CBD547AEDE672B8F8BD08D	__0X0056
C5B52B71F4C5F933815D7D606175EA0BB37DC548	controls.man
C5B52B71F4C5F933815D7D606175EA0BB37DC548	CONTROLS.MAN
C5B52B71F4C5F933815D7D606175EA0BB37DC548	X86_MICROSOFT.WINDOWS.COMMON-CONTROLS_6595B64144CCF1DF_6.0.2600.2180_X-
D10440930CC994409E920D94C7C45F0405D60422	8.0.50727.762.policy

SHA1 hash	File name
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
DFC37F6C15612F7AB155E53A028A69FB5987199A	Program Compatibility Wizard.lnk
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
FA52F823B821155CF0EC527D52CE9B1390EC615E	__0X001A
FA52F823B821155CF0EC527D52CE9B1390EC615E	__0X001B
FA52F823B821155CF0EC527D52CE9B1390EC615E	__0X001C
FA52F823B821155CF0EC527D52CE9B1390EC615E	__0X001D
FA52F823B821155CF0EC527D52CE9B1390EC615E	__0X001E
FA52F823B821155CF0EC527D52CE9B1390EC615E	__0X001F
FA52F823B821155CF0EC527D52CE9B1390EC615E	__0X0020
FA52F823B821155CF0EC527D52CE9B1390EC615E	__0X0021
FA52F823B821155CF0EC527D52CE9B1390EC615E	__0X0022
FA52F823B821155CF0EC527D52CE9B1390EC615E	__0X0023
FA52F823B821155CF0EC527D52CE9B1390EC615E	__0X0024
FA52F823B821155CF0EC527D52CE9B1390EC615E	__0X0025
FA52F823B821155CF0EC527D52CE9B1390EC615E	__0X0085
FA52F823B821155CF0EC527D52CE9B1390EC615E	__0X0087
FA52F823B821155CF0EC527D52CE9B1390EC615E	__0X0089
FA52F823B821155CF0EC527D52CE9B1390EC615E	__0X008B
FA52F823B821155CF0EC527D52CE9B1390EC615E	__0X008D
FA52F823B821155CF0EC527D52CE9B1390EC615E	__0X008F
FA52F823B821155CF0EC527D52CE9B1390EC615E	__0X00BB

SHA1 hash	File name
FA52F823B821155CF0EC527D52CE9B1390EC615E	__0X00BD
FA52F823B821155CF0EC527D52CE9B1390EC615E	__0X00BF
FA52F823B821155CF0EC527D52CE9B1390EC615E	__0X00C1
FA52F823B821155CF0EC527D52CE9B1390EC615E	__0X00C3
FA52F823B821155CF0EC527D52CE9B1390EC615E	__0X00C5
FA52F823B821155CF0EC527D52CE9B1390EC615E	__0X00DF
FA52F823B821155CF0EC527D52CE9B1390EC615E	__0X00E1
FA52F823B821155CF0EC527D52CE9B1390EC615E	__0X00E3
FA52F823B821155CF0EC527D52CE9B1390EC615E	__0X00E5
FA52F823B821155CF0EC527D52CE9B1390EC615E	__0X00E7
FA52F823B821155CF0EC527D52CE9B1390EC615E	__0X00E9
FA52F823B821155CF0EC527D52CE9B1390EC615E	__0X0408
FA52F823B821155CF0EC527D52CE9B1390EC615E	__0X040A
FA52F823B821155CF0EC527D52CE9B1390EC615E	__0X040C
FA52F823B821155CF0EC527D52CE9B1390EC615E	__0X040E
FA52F823B821155CF0EC527D52CE9B1390EC615E	__0X0410
FA52F823B821155CF0EC527D52CE9B1390EC615E	__0X0412

This page intentionally left blank.

Annex D Commonly used registry keys in a typical malware infection

Due to the complexity inherent in working with the Windows registry, this document should not be construed as a registry tutorial. The registry is simply too complex to be fully explained in a few pages. However, certain registry locations are frequently used by malware. Common locations, as based on reference [5] include:

Registry Set (1):

A) HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run

B) HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon

   HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Notify

   HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\SharedTaskScheduler

   HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects

   HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\Explorer\Run

   HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run

   HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\ShellServiceObjectDelayLoad

C) HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services

Other common registry locations of general forensic interest, although not necessarily of use by malware, as based on those the author regularly uses in his own investigations include:

Registry Set (2):

A) HKEY_CURRENT_USER\SOFTWARE\Microsoft\Internet Explorer\Download Directory

   HKEY_CURRENT_USER\SOFTWARE\Microsoft\Internet Explorer\Main

   HKEY_CURRENT_USER\SOFTWARE\Microsoft\Internet Explorer\TypedURLs

   HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\RunMRU

   HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\UserAssist

HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\ComputerDescriptions

B) HKEY_LOCAL_MACHINE\SOFTWARE\ Microsoft\WZCSVC\Parameters\Interfaces

C) HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile\AuthorizedApplications\List

HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\Tcpip\Parameters\Interfaces\

HKEY_LOCAL_MACHINE\SYSTEM\ControlSet00x\Enum\USBSTOR

HKEY_LOCAL_MACHINE\SYSTEM\MountedDevices

However, upon combining the various registry keys from Registry Set (1) and (2), based on the author's interpretation of registry-based malware forensics, the following registry keys should be regularly examined for evidence of malware infection:

Aggregated Registry Keys:

A) HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\RunMRU

HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\UserAssist

HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run

B) HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Notify

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\SharedTaskScheduler

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\Explorer\Run

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\ShellServiceObjectDelayLoad

C) HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile\AuthorizedApplications\List

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services

Moreover, consider that the registry keys presented in Registry Set (1) and (2) and Aggregated Registry Keys could be readily regrouped into the following root registry keys:

Root Registry Keys:

HKEY_CURRENT_USER\SOFTWARE

HKEY_LOCAL_MACHINE\SOFTWARE

HKEY_LOCAL_MACHINE\SYSTEM

The use of these root registry keys is of immediate use to Section 2.3.4.

This page intentionally left blank.

Bibliography

AccessData. AccessData Supplemental Appendix: Understanding the UserAssist Registry Key. Technical documentation. 2008. http://ad-pdf.s3.amazonaws.com/UserAssist%20Registry%20Key%209-8-08.pdf.

Delorie, DJ. EXE format. Online informational article. Delorie.com. September 2010. http://www.delorie.com/djgpp/doc/exe/.

Insecure.com. Nmap socket services. Socket service listing. Insecure.com. https://svn.nmap.org/nmap/nmap-services.

Wikipedia. Flat memory model. Online encyclopaedic article. Wikimedia Foundation. July 2012. http://en.wikipedia.org/wiki/Flat_memory_model.

Wikipedia. Memory management unit. Online encyclopaedic article. Wikimedia Foundation. November 2012. http://en.wikipedia.org/wiki/Memory_management_unit.

Wikipedia. Network socket. Online encyclopaedic article. Wikimedia Foundation. January 2013. http://en.wikipedia.org/wiki/Network_socket.

Wikipedia. Page (compute memory). Online encyclopaedic article. Wikimedia Foundation. November 2012. http://en.wikipedia.org/wiki/Page_(computing).

Wikipedia. Page cache. Online encyclopaedic article. Wikimedia Foundation. October 2012. http://en.wikipedia.org/wiki/Page_cache.

Wikipedia. Page table. Online encyclopaedic article. Wikimedia Foundation. September 2012. http://en.wikipedia.org/wiki/Page_table.

Wikipedia. Paging. Online encyclopaedic article. Wikimedia Foundation. November 2012. http://en.wikipedia.org/wiki/Page_file.

Wikipedia. Portable Executable. Online encyclopaedic article. Wikimedia Foundation. November 2012. http://en.wikipedia.org/wiki/Portable_Executable.

Wikipedia. Shared memory. Online encyclopaedic article. Wikimedia Foundation. November 2012. http://en.wikipedia.org/wiki/Shared_memory.

Wikipedia. Virtual address space. Online encyclopaedic article. Wikimedia Foundation. October 2012. http://en.wikipedia.org/wiki/Virtual_address.

Wikipedia. Virtual memory. Online encyclopaedic article. Wikimedia Foundation. November 2012. http://en.wikipedia.org/wiki/Virtual_memory.

Wikipedia. Zeus (Trojan horse). Online encyclopaedic article. Wikimedia Foundation. October 2012. http://en.wikipedia.org/wiki/Zeus_(trojan_horse).

List of symbols/abbreviations/acronyms/initialisms

AV	Anti-Virus or Antivirus
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
EDT	Eastern Daylight Time
EXT4	Fourth Extended Filesystem
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
NIST	National Institute of Standards and Technology
NSRL	National Software Reference Library
PE	Portable Executable
PID	Process ID
R&D	Research & Development
RAM	Random Access Memory
RCMP	Royal Canadian Mounted Police
RDDC	Recherche et Développement pour la Défense Canada
SHA1	Secure Hash Algorithm 1
TID	Thread ID

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

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 R&D Canada – Valcartier
2459 Pie-XI Blvd North
Quebec (Quebec)
G3J 1X5 Canada

2. SECURITY CLASSIFICATION (Overall security classification of the document including special warning terms if applicable.)

UNCLASSIFIED
(NON-CONTROLLED GOODS)
DMC A
REVIEW: GCEC JUNE 2010

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 a publicly available memory image of the Zeus Trojan horse

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.)
April 2013	82	18

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 R&D 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 2013-018	

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 a Windows-based computer memory dump infected with malware. 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. The author has proposed a memory-specific methodology based on a simple investigative process to help fellow novice memory analysts. Once evidence or indicators of malware have been found, the author examines how Volatility can be used to undertake a given memory investigation. This technical memorandum is the first of a series of reports that will be written concerning Windows malware-based memory analysis using Volatility and various malware scanners. This specific work examines a memory image infected with the Zeus Trojan horse.

Le présent mémorandum technique examine comment un enquêteur peut analyser une image mémoire Windows infectée par des logiciels malveillants. L'auteur étudie la façon d'effectuer une telle analyse en utilisant Volatility ainsi que d'autres outils, y compris des utilitaires pour la récupération de données et des scanneurs anti-virus. Volatility est un cadriciel à code source ouvert populaire et en constante évolution pour l'analyse de mémoire. L'auteur propose une méthodologie spécifique à l'analyse de mémoire basée sur un processus d'enquête simple afin d'aider des collègues débutants. Une fois que des preuves ou des indicateurs de la présence de logiciels malveillants ont été trouvés, l'auteur examine comment Volatility peut être utilisé pour analyser la mémoire. Ce mémorandum technique est le premier d'une série de rapports qui seront écrits au sujet de l'analyse de mémoire pour Windows en utilisant Volatility et d'autres scanneurs de logiciels malveillants. Le présent ouvrage examine une image mémoire infectée par le cheval de Troie Zeus.

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.)

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