A proposed investigative, generic broad-scoped Windows memory analysis methodology

R. Carbone
Infosec Analyst and Researcher
EC-Council CHFI / SANS GCIH & GREM
DRDC – Valcartier Research Centre

Defence Research and Development Canada

Scientific Report
DRDC-RDDC-2017-R041
March 2017

IMPORTANT INFORMATIVE STATEMENTS

The content of this report is not advice and should not be treated as such.

Her Majesty the Queen in right of Canada, as represented by the Minister of National Defence ("Canada"), makes no representations or warranties, express or implied, of any kind whatsoever, and assumes no liability for the accuracy, reliability, completeness, currency or usefulness of any information, product, process or material included in this report. Moreover, nothing in this report should be interpreted as an endorsement of the specific use of any of the tools or techniques examined in it.

Any reliance on, or use of, any information, product, process or material included in this report is at the sole risk of the person so using it or relying on it.

Canada does not assume any liability in respect of any damages or losses arising out of or in connection with the use of, or reliance on, any information, product, process or material included in this report.

Abstract

This work describes a functional, generic, broad-scoped investigative methodology for Windows memory analysis. The methodology applies equally to functional and damaged, or corrupted memory images and relies on Volatility, although other analysis frameworks will do. It is based on the author's various memory analysis case studies. Summing it up succinctly, the methodology aids the forensic practitioner in squeezing the maximum amount of possible evidence from a memory image. The proposed methodology is suitable for analysts at all levels of investigative capability. It provides guidance in extracting maximum evidence using simple, commonplace tools and techniques familiar to digital forensic practitioners. As with all methodologies, nothing is written in stone; the forensic practitioner must be flexible and agile in responding to ever-changing investigative requirements.

Significance to defence and security

Due to an overreliance on frameworks and other easy-to-use tools, investigators often struggle to analyse damaged images. The proposed methodology addresses the issues and problems arising from analysing such images, although it equally applies to functional images. This work will assist to Canadian Forces Network Operation Centre, charged with analysing memory images from defence systems. It may also assist the broader Canadian Law Enforcement community. This report was performed in collaboration with the Royal Canadian Mounted Police, as part of the Platform-to-Assembly Secured Systems (PASS) project under Joint Force Development.

Résumé

Dans ces travaux, on présente la description d'une méthode d'enquête fonctionnelle, générique et de large portée pour analyser la mémoire de Windows. Cette méthode s'applique autant aux images mémoire fonctionnelles qu'aux images altérées ou corrompues et s'appuie sur Volatility, même si d'autres cadres d'analyse feront tout aussi bien l'affaire. Elle est fondée sur diverses études de cas d'analyse de la mémoire menées par l'auteur. En bref, pour résumer, la méthode aide le praticien légiste à relever le maximum de traces possibles d'une image mémoire. La méthode proposée convient aux analystes à tous les niveaux de capacité d'enquête. Elle fournit des indications pour extraire le maximum de traces à l'aide d'outils et de techniques simples et bien connus des experts judiciaires en informatique. Comme pour toute méthode, rien n'est figé dans le béton; le praticien légiste doit faire preuve d'adaptabilité et de flexibilité pour répondre à des besoins en matière d'enquêtes en perpétuel changement.

Importance pour la défense et la sécurité

Le recours excessif aux cadres et autres outils d'utilisation facile fait en sorte que les enquêteurs s'évertuent souvent à analyser des images altérées. La méthode proposée porte sur les questions et les difficultés que pose l'analyse de telles images, même si elle s'applique aussi bien aux images fonctionnelles. Ces travaux faciliteront la tâche du Centre d'opérations des réseaux des Forces canadiennes chargé d'analyser les images mémoire provenant des systèmes de défense. Ils pourront également aider la communauté canadienne plus vaste chargée de l'application de la loi. Le présent rapport a été produit encollaboration avec la Gendarmerie royale du Canada, dans le cadre du projet *Platform-to-Assembly Secured Systems* (PASS) pour le Développement de la Force interarmées.

Table of contents

Acknowledgements

The author would like to thank Mr. Martin Salois, Defence Scientist, for acting as associate editor for this report. Large thanks are extended to the peer reviewer as well.

Disclaimer policy

This report examines computer forensic technologies and therefore is not without risk. The reader must be familiar with both computer and memory forensic techniques and methodologies to avoid harming potential digital evidence. The reader should never work from original copies of digital evidence.

The reader must neither construe nor interpret this work as an endorsement for the various techniques, methodologies and tools examined herein as suitable for any specific purpose, construed, implied or otherwise. The use of any software discussed herein is neither an endorsement nor a recommendation for said software. Only individuals knowledgeable of the inherent risks in using said software should use it.

Furthermore, the author of this report absolves himself in all ways conceivable with respect to how the reader may use, interpret or construe this report. The author assumes absolutely no liability or responsibility, implied or explicit. The onus is entirely on the reader who must be adequately equipped and knowledgeable in the application of digital forensics.

The author and the Government of Canada are henceforth absolved from all wrongdoing, whether intentional, unintentional, construed or misunderstood on the part of the reader or any other party. If the reader does not agree to these terms, then his copy of this Scientific Report must be destroyed or returned to its source of origin. Only if the reader agrees to these terms should he continue in reading this report.

Finally, it is further assumed by all participants that if the reader has not read said Disclaimer upon reading this report 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.

Exclusions and limitations

This report examines and discusses Windows memory analysis, with emphasis placed on the detection of malware and other less commonly identified evidence. Using a qualitative approach, this report attempts to combine many of the author's public case studies into memory analysis in conjunction with various complex memory analyses that were never formalised.

This report continues with work and research pursuant to the 005A PASS RCMP "Live Forensics" Collaboration.

This report is not an introduction to digital forensics. It is expected that the reader is already familiar with the application of digital forensics.

The author uses various approaches in building a methodology. Some of it is heavily reliant on the command line, including the use and development of scripts. However, it also capitalizes on plugin-based frameworks to perform some of the heavy lifting, assuming that the memory image is in fact intact. It has been fully tested on Linux and is expected to work on Windows, assuming the necessary tools and utilities are available (e.g., Cygwin).

This report only examines post-mortem memory analysis. Direct live system memory analysis is outside the scope of this work.

The author neither suggests nor endorses any anti-virus or anti-malware scanner, framework, tool, or scripting language. It is entirely for the reader to decide which to use. However, the reader is responsible for ensuring that all software he uses respect the various licenses to which they are accorded.

The use of masculine throughout this report is for simplification.

Due to the complex and varying nature of memory analyses and investigations, case report writing is not examined in this report.

The use of multiple anti-virus scanners is essential; detecting malware without is very difficult, very time consuming and largely the realm of software reverse engineers. To bring this capability to the "masses" scanners become an important part of the proposed methodology. Some will argue against this approach—other approaches have been tried and were identified by the author as providing the most consistent results.

Finally, this report does not take into account Windows 10 or Windows Server 2012, although it is expected to apply equally to them.

1 Background

1.1 Objective

The primary objective is to provide a functionally generic, flexible and multipronged methodological approach to conducting memory analyses. It also aims to provide a state-of-the-art approach concerning maximum evidentiary extraction and analysis. Through it, forensic analytical support is provided to the Canadian Armed Forces, Canadian Forces Network Operation Centre (CFNOC). It is expected to benefit both the Royal Canadian Mounted Police (RCMP) and by extension, the general Law Enforcement community.

This report does not examine Law Enforcement-specific techniques.

1.2 Executive summary

As far as can be determined, there is no agreed-upon best technique for conducting post-mortem memory analysis. Books (e.g., The Art of Memory Forensics [1]) and information security training courses (e.g., SANS 526 [2] and 610 [3]) do not provide a sufficiently multifaceted approach for analysing memory images when an acquisition or standard analysis fail. This methodology hopes to bridge that gap.

It is based largely on the author's previously published memory analysis case studies examining Zeus [4], Prolaco and SpyEye [5], 0zapftis (R2D2) [6], Stuxnet [7], Tigger [8], IVYL [9], Jynx2 [10] and KBeast [11]. Many other successful, but non-formalised studies, contributed to this methodology (Trojans and RATs, rootkits and spyware).

This report attempts to combine these various case studies into a more formalized methodology although it remains qualitative in nature. Although it places much emphasis on identifying and extracting malware-specific evidence, it is sufficiently generic to allow for extracting much additional information and evidence. While it makes avid use of Volatility and its myriad plugins, broader analysis will typically maximize evidentiary extraction to better fill gaps in the investigation.

Various memory analysis frameworks exist, including, but not limited to, free and open source software (FOSS) solutions (e.g., Volatility, Rekall.) There also exist various commercial frameworks. The primary difference between these paradigms is that the FOSS solutions usually allow an investigator or analyst to modify plugins or write new ones using high-level languages, to modify or improve their capabilities.

As there is no one-size-fits-all approach, investigators must be versatile in memory forensics, so that, when necessary, deviation from the proposed methodology will not result in the loss of analytical focus or capability.

1.3 Audience

The direct audience are CFNOC's analysts, RCMP decision makers and the Law Enforcement community. By making much of the work conducted under the 005A PASS RCMP "Live Forensics" Collaboration public, the audience is expanded to the general computer forensic community.

As this report is highly technical, it is likely only of use to those knowledgeable in information security or computer forensics and therefore is not an introduction to digital forensics.

2 Proposed investigative methodology for Windows

2.1 Background

The author's initial Windows-specific investigative methodology was first proposed in the Zeus report [4]. It was later refined in Prolaco & SpyEye [5] and then further clarified in Stuxnet [7] and Tigger [8]. These early models were a first step towards a generic analytical approach. Since then, it has been vastly improved, tested and genericized. Additional revisions improved its focus for handling the complexities of malware memory investigations. However, it does not discuss Law Enforcement specific techniques or methodology. Instead, it provides a clear approach for conducting generic investigations for non-reverse engineers, computer forensic investigators and analysts.

Various steps are proposed in this methodology. Some are mandatory, others optional. It is broken down into ten specific steps, most of which provide an opportunity to cease the investigation and conduct a wrap-up. These steps can be readily rearranged or altered to suit the reader's needs as they meant to be fluid. Additional processing can be applied wherever necessary. While it is aptly suited to malware, it is entirely appropriate for non-malware investigations too.

The tools and techniques described in the methodology are familiar to forensic practitioners at both ends of the knowledge/capability spectrum. Often taken for granted, they can be readily used in extended memory analysis.

Memory analysis can be complex and time consuming, particularly when done manually using command line driven analysis frameworks (e.g., Volatility, Rekall). This is in contrast to automated or semi-automated frameworks that remove the investigator or analyst as much as possible (e.g., CounterTack ResponderPro, Mandiant Redline). Which to choose is a matter of needs vs. available resources. In situations where ample resources are available, manual analysis is an excellent manner for investigators and analysts to maintain and sharpen their skills.

Either way, there is an overreliance on automated tools. This leads to situations where investigators and analysts cannot explain the production of certain results. While such frameworks and tools certainly speed up triaging and analysis, they are unlikely to catch highly complex or stealthy malware; this has been corroborated using ResponderPro, although mileage may vary with other similar frameworks. In such cases, only a manual analysis conducted by a competent investigator can find evidence or indications of its presence.

While reading the proposed methodology, the reader may wonder why data carving is used, considering it is an outdated method for conducting memory analysis. The reason is simple—frameworks are only capable of processing functional memory images and images from supported systems. Frameworks are limited in their ability to find memory artifacts; some are from long since terminated programs or threads and libraries, dependencies and configuration files. Highly advanced and stealthy malware can completely unload[1] themselves from memory.

[1] In this context, unload means actually unloaded from memory, not hidden or removed from one or more of the various memory housekeeping lists used by Windows.

Identifying such evidence is especially difficult using automated analysis frameworks. However, carving and associated methods often provide useful results.

At the same time, anti-virus and other malware scanning software play a key role in identifying malware. Typically, in manually driven memory analysis, it is not possible to differentiate between the many hundreds or even thousands of files generated, some of which may contain malicious or suspicious content. For this reason, malware identification must occur using some mechanism. While various online malware scanning resources exist (e.g., VirusTotal), it is not always possible or desirable to upload and potentially share malware with publicly available resources.

The methodology explores the many facets for processing and analyzing the various data files generated, including those extracted using carving. These techniques include the use of file hashing and similarity (fuzzy) matching as well as file type identification, which when used appropriately, can further augment an investigator or analyst's ability to identify which files to study further.

The key to a successful manual investigation relies primarily on information correlation, not generating excess data and information. It is about the investigator or analyst's ability to correlate information into a "big picture." Software exists which can help perform this correlation but it is not examined here, though the reader is free to do so. In lieu of such software, various scripts can be written that readily reduce the amount of information generated.

Most of the various tools or classes of software examined in the methodology can typically be interchanged for other similar software. However, MD5 should not be used under any circumstance; it is no longer secure, and hash collisions with it are becoming more common. SHA-1 is considered superior to MD5 although specific types of collisions can be engineered for it. SHA-2, while computationally more expensive than the previous hash algorithms, is currently considered secure from hash collisions.

Some may wonder why the National Software Reference Library (NSRL) has not been used in this methodology. It was, originally. However, due to the inexact nature of carving the likelihood of obtaining matches against are low, and often not worth the effort. The reader is, of course, free to do so and integrate it into the methodology.

It could be argued that Internet web activities should be done sooner in the methodology but in the interest of objectivity, the methodology is not implemented this way. The same argument can be made for testing an image's intactness prior to commencing a manual analysis. It boils down to sidestepping shortcuts to avoid skewing potential facts or losing investigative focus. Of course, the reader is free to make changes as needed.

The concepts explored in this methodology are entirely applicable to the pagefile, with the exception of using a memory analysis framework, and was tested numerous times by the author.

Finally, the methodology does take into consideration the examination of corresponding disk images parallel to a given memory analysis. Where necessary, it should be done, but doing so is complex and outside the scope of this work.

2.2 Step (1): Pre-triage—Memory image safeguarding

Ensure that the memory image (or dump) has been set as immutable to prevent accidental changes or modifications from occurring:

1. Ensure that the memory image matches the cryptographic hash (minimum SHA-1) of the original memory image:

    A) Hash using the SHA-2 algorithm. **Do not ever use MD5**. If possible, avoid SHA-1 as collision attacks are now possible for it.

2. Ensure correct file permissions are used:

    A) Use DACs and ACLs to ensure that only those who should have access are permitted access.

    B) If the machine is networked and the image file and case directory are shared, ensure the correct NFS or SMB permissions and attributes have been set, including DACs and ACLs.

3. Set the memory image as immutable:

    A) This guarantees the file's intactness atop the current filesystem.

    B) It does not safeguard against hardware failure or bit-rot—these require additional hardware/software to protect against.

    C) The ability to set immutability varies according to the underlying operating system and supported filesystem features:

        i. Neither Windows nor NTFS provide mechanisms for setting file immutability. The best that can be done is applying appropriate ACLs.

        ii. Linux, BSD and some UNIX systems provide mechanisms for setting file immutability.

2.3 Step (2): Triaging (Part 1)—Preliminary memory image malware scanning

Analyse the memory image using multiple anti-virus (AV) or other anti-malware scanners:

1. Use as many quality scanners as reasonably possible:

    A) A minimum of six AV scanners using different types of heuristics and signature mechanisms is advised.

B) Command line scanners are the most preferable to use, which for Linux is the most commonly encountered.

C) Eight or more is preferable:

i. Various commercial solutions are available.

D) They should be command line scanners as these are easier to script.

E) Ensure that each scanner is set to generate a report or log of its scan:

i. Sometimes not all output is recorded with problematic scanners. In such cases, output recording can be forced by appending the following to the end of the scanner command line:

1. >report.txt&>report.txt

F) Save all scanner output and logs to an appropriate location.

2. Be aware that some scanners use significantly different command line parameters:

A) Some use very long and complex—mileage varies.

3. Be cautious of instances where only one scanner identifies a memory image as infected or suspicious:

A) Only a select few can do this.

B) In some instances, one or more scanners may accurately identify an underlying malware infection in the memory image.

C) Some scanners are capable of performing in-depth scans of arbitrary binary data files that can cause false results.

4. Correlate all scanner results:

A) This can usually be done manually.

B) In some instances correlation may need the use of scripts:

i. Once these scripts have been developed, they can be further refined for future investigations.

C) Save all output to an appropriate location.

5. Decide whether continued analysis is warranted:

A) In certain instances of triaging, malware scanning of a memory image may suffice as infection specifics may have been identified.

B) Scanners are not typically capable of identifying an infected memory image; therefore, it is suggested to continue triaging.

C) Ensure proper documentation and that it has been stored to an appropriate location(s).

D) If appropriate, terminate triaging, processing and close the current analysis.

E) Otherwise, continue on to the next step.

2.4 Step (3): Extract strings (optional)

Extract 7, 8, 16 and 32-bit strings from the memory image. Many tools exist do this for Linux and Windows.

String files can be enormous, sometimes as large as the image itself.

If desired, tools can be used to isolate Base64 text and convert it back to ASCII or binary. This part of the step is completely optional.

Bulk_Extractor [12] can be used to do this as it readily handles repetitive string occurrences and can identify Base64 data. The UNIX ***strings*** command can also be used.

Strings with whitespaces should be split into appropriately sized pieces. Sometimes *strings* will not always do this by itself—this is done by replacing whitespace with a newline ('\n') character.

This entire step can be done independently of Step (11).

While this step is optional some file types (encrypted or password protected) may rely on its results.

It is important if using strings to ensure that all duplicates are removed; this is done using ***sort*** and ***uniq***.

2.5 Step (4): Triaging (Part 2)—Data carving, malware scanning and hashing

Memory images do not have filesystems from which data can be readily recovered though they may contain filesystem segments, buffers and cache. Thus, data carving is typically the only mechanism for immediately recovering data from it. Using an advanced data carving utility, carve all possible data files from the suspect image:

1. Use only one highly capable data-carving tool instead of several mediocre tools and carve as much as possible:

   A) Many capable commercial and FOSS tools exist, for Windows and Linux.

B) Typically, hundreds or even thousands of data files may be carved from a given memory image.

C) What to carve:

i. Both the size of the memory image and what was loaded into will largely determine what and how much will be recovered from it.

ii. Much of what is recovered will be nonsensical; nevertheless, useful data will often be retrieved:

1. This is because often memory pages have been paged or "swapped" out to disk.

iii. It is suggested to carve files relevant to Windows malware investigation:

1. Executables—Windows runnable programs, libraries (DLLs), drivers, etc.

2. Event logs.

3. Registry files.

4. Archived (e.g., ZIP, RAR, TAR, CPIO, LHA) and compressed files (e.g., BZ2, GZ, ZIP, RAR, Z):

a. This includes files detected as PGP, GPG or other form of detectable encryption.

5. Text-like files including:

a. Text and configuration files.

b. Source code files of various languages.

c. Compiled bytecode files including Java class and JAR files, etc.

d. HTML and JavaScript like files.

6. Web caches and history files (we'll look at this in Step (11)).

7. Image and multimedia files of all sorts.

8. PDFs, Flash files, OLE and office document files.

9. In addition, any other file type thought possibly to be harboring an infection or containing pertinent information (e.g., encrypted files and executables).

D) If the tool generates a log of what has been recovered and the settings used (i.e., settings or configuration file), keep a copy and place in an appropriate location:

i. If not, note the settings used and store it in an appropriate location. Make a full listing of everything carved from the memory image, including file sizes.

2. Consider using Windows memory page sizes for carving, 4 KiB:

   A) Other appropriate sizes may include 512 bytes:

      i. This will likely result in even more data being recovered, but also increasing the likelihood that recovered data may be corrupt or invalid.

3. Hash everything extracted, carved or recovered:

   A) Use SHA-2, if available. At a minimum, use nothing less than SHA-1.

   B) Check for hash collisions:

      i. If SHA-1 or 2 is used then collisions are usually duplicates.

   C) Use fuzzy hashing to identify very similar files:

      i. Set matching threshold to >90% or higher.

   D) Store this information in an appropriate location.

   E) Duplicates and threshold matches:

      i. If duplicate files are found, delete them but make note of which ones they were and why there were deleted, and save these notes to an appropriate location.

      ii. If threshold matches are found, it can indicate similarity but not necessarily a direct relationship. Thresholds greater or equal to 98% typically indicate the files very likely have the same origins and differ only slightly.

4. Analyse the carved data files using multiple AV scanners:

   A) Use the same scanners as in Step (2).

   B) Save all scanner output to an appropriate location. If necessary, force shell output logging (see Step (2) Part 1) E) for details).

   C) Some scanners are capable of performing in-depth scans of arbitrary binary data files.

   D) Cross correlate results:

      i. This can sometimes be done manually when only small amount of data are output.

      ii. Usually, this should be done using scripts, as there may be hundreds or thousands of lines of scanner output to process:

1. Once the scripts have been developed, they can be further refined for future investigations.

iii. If multiple scanners indicate that a carved data file is suspicious or malicious, then it is of interest:

    1. The same applies to password-protected archives and executables, which some scanners will successfully identify:

        a. Password protected executables are rarely encountered; packed is far more common. Go to Step (6) for processing these types of files.

        b. This may include files flagged as containing exploit or shell code.

        c. Record this information in an appropriate location for future use.

    2. The more scanners that corroborate a carved file is infected or suspected of infection, the more it should be considered relevant to the investigation.

    3. Files of interest should be copied out to an appropriate location where they will be easy to identify:

        a. Use scripts where possible.

iv. Files picked up by only one scanner, especially scanners prone to false positives, should be considered as false positives, until shown to be otherwise, due to the inexact nature of data carving and scanning:

    1. These include overly generic scanner messages of infection or suspiciousness.

    2. False positives can sometimes be examined using a strings-based (7, 8, 16 and 32-bit strings) approach:

        a. Look at the output while seeking out common functions and APIs used by malicious software.

        b. If nothing notable is identified then the file(s) can be considered "harmless":

            i. This requires a basic knowledge of reverse engineering or malware fundamentals.

        c. Encrypted executables will show little human-readable text:

            i. Identification of too "many" nonsensical strings may be indicative of executable file encryption or packing—go to Step (6) for processing these encrypted executables.

E) Save all output and copy all suspected and extracted files to an appropriate location and generate a file listing:

   i. This listing will be used in Step (5).

5. Determine if additional analysis is necessary:

   A) Based on the results obtained from Steps (2) and (4), there may be no overt indications of infection or suspicious activity.

   B) At this point, with or without confirmed infection, there may be nothing else to do:

      i. If appropriate, terminate triaging, processing and close the current analysis:

         1. Ensure proper documentation.

      ii. Otherwise, continue on to the next step.

2.6 Step (5): Analysis (Part 1)—File analysis, extraction and scanning

Building on Step (4), we continue with the analysis, moving on to file type determination, extraction and malware scanning:

1. Run the ***file*** command or a similar command against all files extracted/carved from the memory image:

   A) Save all output to an appropriate location.

   B) This is a very useful signature detection program. It is not foolproof, however.

2. Ensure the following identified file types are also copied to an appropriate location(s):

   A) Copy all files containing embedded file types or OLE for subsequent analysis. These include:

      i. PDFs: They can contain infected Flash, JavaScript, images, multimedia or other embedded data type, including shell code.

      ii. Office (MS, OpenOffice, LibreOffice, etc.) documents: These files can contain macros, OLE and shell code.

      iii. Java class files and JARs:

         1. In general, these can be easily reverse-engineered back to a source code-like format—in some cases this may be of use.

   B) Packed executables not caught by the scanners:

i. Most scanners have difficulty with all but a few packed executable types.

C) Common archive and compression types:

i. Do not rely on file extensions; rely on signatures or visual inspection of data.

ii. Includes TAR, GZ, BZ2, RAR, ZIP, ARC, LHZ, LHA, and many others.

D) Registry files.

E) Internet history files.

F) Event logs.

G) Do not consider MSI or Innosetup-like (e.g., MSI, InstallShield) executables or files unless there is reason to; i.e., suspected infection.

H) Encrypted and password protected files and executables identified in Step (4).

I) Scripts will likely be required for correlating and copying:

i. Once these scripts have been developed, they can be further refined in future investigations.

3. Take inventory of all the files copied in this step and combine with those from Step (4) Part 4) Subsection E):

A) With this "master list," file, archive, object and OLE, packed executable and other extractions can be carried out.

4. Now, it is time to extract data from the identified files:

A) This can be very long.

B) Do this against PDFs, OLE files, Internet history files, Registry files, Event logs, and other identified files.

C) Many are in a proprietary format but many tools exist to extract textual information from them:

i. Be sure to capture all output from these tools.

ii. Some tools extract text, images, scripts or code, etc.

D) Extraction should be done in the same location as the container file (archive, compressed or packed file).

E) Many tools exist to extract archived and compressed data:

i. An overwhelming number of archives and compressed file types are supported under Linux, although some software may need to be installed or compiled to provide the necessary support.

ii. Some may require passwords. Generate a list of password-protected archives:

1. Process these in Step (6).

F) Most of these tools are not fully automated:

i. Many can be scripted.

ii. Others, based on the tool's output can be scripted to use other tools to perform the actual extraction.

G) Some packed or compressed executables can be directly unpacked or uncompressed:

i. For examples, files identified as UPX can be directly unpacked, assuming the carved file is intact from the carving process.

ii. Many packed/compressed executables are not extractable without significant effort, often requiring reverse engineering:

1. If they cannot be extracted, these files should continue be considered suspicious upon detection.

iii. There are rarely reasons to use heavily packed files, unless a context warrants it:

1. For example, professional game players and developers are known to rely on these to protect their products.

iv. Some packed file types are very difficult to extract and may require reverse engineering due to various anti-debugging and anti-extraction mechanisms in place:

1. Generate a list of such files.

2. Depending on available resources, it may be necessary to pass these files on to experienced reverse engineers.

3. If the appropriate tool or extraction technique becomes available then this sub-step can be revisited, otherwise, this avenue should be closed without help or additional resources.

H) Decisions about extracting Java code from bytecode files needs to be made:

i. Probably not worth the effort unless a bytecode file was identified as infected or suspicious.

ii. Probably not worth the effort to immediately examine extracted source code files, including HTML and JavaScript, unless previous identified as infected or suspicious.

I) This might be the right time to correlate textual information:

i. For example, information obtained from Event logs and Internet histories may coincide, as may Registry information.

ii. Otherwise, save for a later step.

5. All extracted, decompressed, unpacked, uncompressed or extracted files and executables should be scanned using the same scanners used in previous steps:

A) Based on scanner results, it might be possible to corroborate unpacked files with detected infections from data carving and scanning.

B) Scripts will likely be required for correlating infections.

C) Save all output to an appropriate location(s).

6. All files extracted from the copied archives, compressed files or executables are to be hashed:

A) If possible, use SHA-2, but remaining consistent with the hash algorithm already in use in the investigation.

B) Fuzzy hash these files.

C) Then correlate hashes (SHA and fuzzy) looking for collisions and threshold (>90%) matches between these files:

i. Log all output.

D) Duplicates and threshold matches:

i. If duplicate files are found, delete them but make note of which ones they were and why there were deleted, and save these notes to an appropriate location.

ii. If threshold matches are found, it can indicate similarity but not necessarily a relationship. Threshold greater or equal to 98% typically indicate the files very likely have the same origins and differ only slightly.

iii. Log all collisions, deletions and threshold matches.

E) Check hashes (SHA and fuzzy) against those from Step (4) Part 3). If hashes match or fuzzy hashes have >=98% similarity then:

i. Log all output but do not delete duplicates (for hash matches only).

F) For fuzzy hashes, if threshold matches are equal or greater to 98% then this typically indicate the files very likely have the same origins and differ only slightly.

7. Determine if additional analysis is necessary:

   A) At this point, with or without confirmed infection, there may be nothing else to do as everything of interest may be have already been identified:

      i. If appropriate, terminate triaging, processing and close the current analysis. Ensure proper documentation.

      ii. Otherwise, continue on to the next step.

2.7 Step (6): Password cracking (optional)

Password cracking is computationally expensive and requires dedicated systems, often equipped with many GPUs and CPUs—the more the better. There simply may be insufficient resources, time, energy or even interest to pursue password cracking, in which, skip to the next step.

The software required for password cracking is often expensive timewise and can take days, weeks, months, or years, even on extremely powerful systems (>16 GPUs and >128 CPUs). Commercial password cracking software (e.g., Passware, Elcomsoft) typically have more capability than many of the FOSS tools (e.g., John the Ripper, Rainbowcrack)—mileage will vary as there are notable FOSS password cracking tools.

If strings extraction was done in Step (3) then these files can be used as potential password lists for use with both commercial and FOSS cracking software.

Several major vendors provide quality and highly capable software for cracking a wide assortment of cryptographic algorithms and implementations.

If password cracking succeeds against a given file or executable, repass it through the various units of Step (5) to ensure it is correctly processed and assessed against all the other known files, malware signatures and hashes (SHA and fuzzy), to date.

Save all notes in an appropriate location(s). Screenshots may be appropriate if working in a graphical interface (Windows, X Windows, etc.).

If encryption keys are detected in a memory image, it may be possible to directly use it to decrypt an archive or executable. Sometimes these keys can be directly passed to the password-cracking program.

If LSA passwords are obtained, crack these as they may have been used elsewhere for encryption and password protection.

2.8 Step (7): Analysis (Part 2)—Volatility plugin memory analysis

If a given memory image continues to remain suspect (i.e., evidence or indications of infection were identified), then a full memory analysis should be conducted in order to establish how the infection occurred, if possible. Registry analysis using Volatility is done in Step (10). Non-Registry Volatility analysis, based largely on the reports mentioned in the Executive Summary (Section 1.2) consists of:

1. Identify if the memory image is intact. Here, we are using Volatility, but it could be some other framework:

    A) This is done using basic plugins such as ***imageinfo*** and ***pslist***:

        i. Running ***pslist*** helps to identify damaged or corrupt memory images.

    B) Although Volatility can often auto-detect the memory image type, it might be necessary to specify it at the command line.

    C) It may also be necessary to specify a memory profile.

    D) If after specifying the appropriate memory image type and profile, and the aforementioned plugins fail, then the memory image can be considered as damaged, incomplete or corrupt:

        i. If this is the case, skip to the next step.

        ii. If appropriate, terminate triaging, processing and close the current analysis:

            1. Ensure proper documentation.

2. There are many reasons a memory image could be corrupt:

    A) It may be the result of an incomplete or incorrect acquisition:

        i. Due to software or operator error.

    B) It may be because malware was residing in memory and redirected inconsistent data back to the acquisition tool.

    C) It could be that the version of Windows is unsupported (i.e., alpha, beta, pre-release version of Windows) or has been significantly altered.

    D) It could be from corrupt or damaged media used for acquiring the memory image.

    E) In such cases, framework-based analysis of the memory image will yield no useful results or information. Trying with a completely different framework is unlikely to result in better results.

F) Depending on policy or procedure, it may be pertinent to conduct a strings-based analysis (Step (3)) against the memory image or conduct optional evidence collection and extraction (Step (11)).

3. For a **functional memory** image, there are many avenues of framework analysis to be conducted:

   A) Using non-reverse engineering memory analysis plugins, find and extract as much information as possible concerning the underlying system, processes and process listings, Internet Explorer history, Event logs, threads, communications channels, open files, mutexes, drivers, services, etc.:

      i. Generating a timeline at the beginning using the appropriate plugin and correlating this information to the date/time information obtained from other plugins may yield additional clues:

         1. Timelining should be done early in the plugin-analysis process.

      ii. Encryption, and by extension disk encryption, have become commonplace. Because of this, Volatility offers plugins to find and identify TrueCrypt (TC) and RSA keys:

         1. BitLocker key identification is not yet available from Volatility:

            a. Some commercial password cracking software support this by directly extracting the key(s) from the memory image (see Step (6)).

         2. Sometimes, the TC password is cached in memory; if so, Volatility might be able to find it.

         3. Other non-Volatility command line tools exist to do this too.

      iii. The ability to use these and other plugins is highly dependent on the investigator or analyst's knowledge of Volatility and memory forensics in general. The better the knowledge and skills the more plugins can be used, understood, and potentially useful information or evidence obtained.

      iv. Save all plugin output in an appropriate location(s).

      v. Typically, taken only as individual plugins, information obtained is typically insufficient to put the pieces together, unless something is particularly obvious:

         1. Complex analyses may be required to identify non-obvious potential indicators of infection.

         2. The information plugins provides, taken together as a whole, can provide an investigator or analyst with a "big picture" view.

vi. The information obtained from the various plugins require correlation:

1. The more plugin output there is the longer it will take to correlate.

2. Look for links or relationships between the information obtained from the numerous plugins and link them:

   a. This can be a very time consuming process.

   b. It can also be a very difficult process, examining the data for sinuous threads of causality.

   c. Identify if correlation can be made with information extracted in Step (5) concerning Event logs and Internet web histories.

3. Automated tools may help the investigator or analyst, if available:

   a. Currently, there is little available in the marketplace with respect to memory analysis correlation tools or software.

B) Once one or more suspected files, processes, threads, DLLs, drivers, processes, process memory, etc., have been identified, it is time to dump them:

i. Various plugins exist to dump a variety of data and files from a memory image:

1. The investigator or analyst must choose the correct method of dumping the data or file.

2. Specific dump-based plugins exist to enable an investigator to access and acquire additional data types from a memory image.

ii. Process all dumped files as per Step (5) and correlated against all existing hashes.

iii. Text plugin output for various logs, histories, events, or other textual output is not dumped; the information is stored to text by redirecting output.

C) Based on all the information at hand attempt to corroborate the information and dumped data obtained from the plugins:

i. Connections will likely be difficult to establish.

D) If no suspicious or malicious content can be found in the dumped memory sample(s) then:

i. Optionally, go on to Steps (8) and (9).

ii. Otherwise, it may be appropriate to terminate the investigation at this point or continue on to Step (10).

iii. In either case, ensure that all work has been thoroughly documented, that all dumped files or newly generated (extracted or unpacked) data files or executables have been copied to an appropriate location(s).

2.9 Step (8): Volatility plugin memory dumping and carving (optional)

Although not yet discussed, it is possible to use the Volatility's *memdump* plugin to dump process and kernel memory and addressing space.

The *memdump* plugin can dump all addressable memory for a given process. It works by enumerating the process' page tables and dumping them to a destination file. This is a very useful plugin for scanning and carving. The dumped files, one per process, will contain all the memory associated with that process, if it was not paged out. However, the data is not likely to be contiguous. Portions of Steps (4) and (5) can be used to carve and analyze any resulting files.

The reader may wonder why not just use this plugin rather than go through the whole process of carving in Step (4). The reason is simple: much data and evidence still reside in a memory image in "unallocated" or unused space long after a process has terminated. Direct carving provides access to these files. However, because carvers are not aware of memory pages and their associated structures, they treat memory images as one contiguous block of data.

2.10 Step (9): Post—Volatility plugin analysis (optional)

If nothing has yet been identified as possibly suspicious or infected using Volatility or no new information has been identified beyond what was found in previous steps, then it may be necessary to perform alternative analyses:

1. Conducting string and hexadecimal analysis can be helpful:

    A) Very often, malicious software, particularly for DLLs, drivers, threads and processes additional analysis can be done:

        i. They may contain suspicious keywords, values, strings or specific Windows API functions.

        ii. Whitelists and blacklists for Windows APIs and commonly used strings can be used to identify known strings:

            1. These are to be used with caution.

            2. The context of potentially suspicious strings may be dependent on the underlying programming language used.

2. If nothing is found or identified to date, then it may be time to stop the analysis or try using other tools or frameworks:

A) Other tools or frameworks (Rekall, CounterTack DNA, etc.) are not covered herein as there are several choices to make concerning capable analysis frameworks.

B) Ensure that everything has been thoroughly documented and saved to an appropriate location.

C) At this point, with or without confirmed infection, there may be nothing else to do as everything of interest may be have already been identified:

   i. If appropriate, terminate triaging, processing and close the current analysis.

   ii. Otherwise, continue on to the next step.

2.11 Step (10): Analysis (Part 3)—Examining the Windows Registry

The Windows Registry can be long and difficult to analyse, but various frameworks including Volatility provide numerous plugins for extracting Registry information from a memory image:

1. Determine which Registry keys are being sought out:

   A) Consider the reasons for examining the Windows Registry:

      i. Identify possible mechanisms of malware persistence across reboots and shutdowns:

         1. This can be achieved using one or more Registry keys, in combination or alone.

      ii. Determine possible malware configuration options:

         1. Identify potential lists of Command & Control servers or IP addresses.

         2. Identify potential list of capabilities or “orders” that the malware may have.

         3. Identify certain previously cleaned infections.

      iii. Corroborate findings from the previously described steps of analysis:

         1. Certain scanners can identify infections or malware that use specific Registry keys. This can confirm the presence of such infections.

         2. Possibly identify the malware’s modus operandi.

         3. Corroborate intelligence sources against identified malware or infection specific Registry keys.

B) Very often, malicious software, particularly for DLLs, drivers, threads and processes additional analysis can be done:

   i. They may contain suspicious keywords, values, strings or specific Windows API functions.

   ii. Whitelists and blacklists for Windows APIs and commonly used strings can be used to identify known strings:

      1. These are to be used with caution.

      2. The context of potentially suspicious strings may be dependent on the underlying programming language used.

2. Identify which Registry hives are available in the memory image:

   A) If no pre-identification efforts are made then it may be necessary to sift through many tens of thousands of keys.

   B) In general, malware has a tendency to use a few keys quite regularly.

   C) Otherwise, lots of additional work may lie ahead in this analysis.

   D) Many sources of information exist on the web:

      i. These often include details about readily obtainable from well-established sources including scanner and anti-malware vendors.

      ii. Malware researchers often make available their own lists of Registry keys.

      iii. Some sources are more reputable than others.

      iv. Document all Registry key sources used.

      v. These lists are not always up to date.

      vi. Malware authors are often aware of such lists and therefore sometimes use different techniques to hide their use of the Registry:

         1. They may encode the values used by their malware.

         2. They may use less common Registry keys.

         3. They may identify new Registry keys to use.

3. Extract UserAssist and ShellBags keys from the memory image using corresponding plugins:

   A) This list will be used to identify which sets of Registry keys are available for extraction.

B) Some frameworks provide the ability to query a memory image for a list of available Registry hives:

   i. Those that do are typically straightforward, with respect to both querying and the listing of available Registry hives.

   ii. Those that do not should instead be directly examined using one or more generated scripts based on one or more lists of potential malware-specific Registry keys.

4. Dump LSA passwords from the memory image:

   A) In frameworks that do not support these scripts can be used to identify and extract said keys.

5. Armed with sufficient information, generate one or more scripts to query the previously identified Registry hives for specific keys:

   A) This is a straightforward plugin.

   B) Dumped passwords are hashed.

   C) These hashed passwords can be cracked using password cracking software (see Step (6)).

6. Run the scripts (or manually typed-in commands):

   A) This is relatively straightforward.

   B) This assumes the reader is sufficiently knowledgeable to write or "auto-generate" such scripts.

   C) It is advised to use one script per Registry hive to keep things simple.

   D) Broad or precise Registry key searches can be carried out.

   E) Scripts can be written using various languages, including shell scripting and PowerShell.

   F) The reasons for using scripts over manual Registry key processing are:

      i. There may be too many keys to examine or extract manually.

      ii. Speed up overall processing.

      iii. Reduce errors.

      iv. Easier to generate acquirable output.

v. Maintain processing and transaction log for documentation purposes.

7. Analyse script (or manual command) output:

   A) Capture all generated output:

      i. If necessary, force output capture.

   B) Typically takes only a few minutes to run to completion.

   C) Make note of any errors and attempt to identify their cause(s):

      i. If identified, correct for it (them) and rerun script(s).

      ii. Typically straightforward to debug.

8. Determine if additional analysis is necessary:

   A) Typically takes only a few minutes.

   B) Registry keys of importance to the investigation are often easy to identify.

   C) Some malware use non-standard key locations and encoding:

      i. Important to know alternate key locations likely to be used by malware authors.

      ii. Keys may be encrypted or encoded differently:

         1. ROT13 has been used with some success by malware writers.

         2. Sometimes certain Registry key values can be XOR'red.

   D) Correlate pertinent information against what has already been identified in previous steps.

9. If nothing is found or identified to date, then it may be time to stop the analysis or try using other tools or frameworks:

   A) Based on the results from this analysis, there may be no overt indications of infection.

   B) At this point, there may be nothing else to do.

   C) If appropriate, for various reasons, terminate and close the current analysis. Ensure proper documentation has been conducted.

   D) Otherwise, continue on to the next step.

2.12 Step (11): Analysis (Part 4)—Post-processing and analysis (optional)

Once the aforementioned analyses have completed, it may be of interest to conclude by attempting to identify additional potential information or evidence from the memory image. This step is entirely optional. Although in previous steps the analyst could have waited until now, only parts of this analysis may be possible to complete:

1. Run post-processing and analysis routines on images (pictures) extracted/carved from memory:

   A) It may be of interest to run pornography detection software on the carved or plugin-dumped images.

   B) It may be of interest to run image similarity matching looking to seek out image patterns or comparative levels of sameness.

2. If required, run encryption key detection:

   A) Numerous FOSS and COTS software exist to identify and extract different types of encryptions keys embedded within a memory image:

      i. Commonly used FOSS tools include ***Bulk_Extractor***, ***aeskeyfind***, ***rsakeyfind*** and ***interrogate***.

      ii. Commonly used and extractable keys include AES, RSA, Serpent and Twofish, as well as others.

   B) Keep in mind that much legitimate software uses encryption, sometimes for illicit activities, whether on the local system or online. Many malware utilise various means for encrypting their network communications.

   C) Sometimes, keys can be damaged—software exists to rebuild, at least in part, moderately damaged keys.

   D) COTS software typically find encryption keys in the memory image that is then passed on to the tool's decryption component (see Step (6)).

   E) Unidentified encryption-like keys identified in a given memory image may be network-specific encryption keys:

      i. Implementing these keys requires knowledge of the program that used it and reverse engineering in order to exploit it.

   F) Ensure program settings and documentation have been done.

3. If rootkits or malicious device drivers were found, attempt to establish their underlying capabilities:

A) If suspicious or malicious software (Trojan, worm, device driver or rootkit, etc.) was dumped from the memory image, using several of Volatility's very low-level plugins (***driverirp***, ***eventhooks***, ***handles***, ***impscan***, ***mutantscan***, etc.), it may be possible for to learn more about it:

   i. Some of these plugins can be used with little knowledge of reverse engineering.

B) Ensure proper documentation and that all results are stored in an appropriate location.

4. Consider using ***Bulk_Extractor***:

   A) This highly versatile, multithreaded, multiplatform tool can be used to extract many things from a memory image.

   B) A full list of what it currently supports can be found at http://www.forensicswiki.org/wiki/Bulk_extractor.

   C) Of particular interest are email address, domain names, IP addressed, URLs and URL searches.

   D) By default, all results are saved in text files. Ensure copies are stored in an appropriate location:

      i. These files can be used as password lists (see Step (6)).

   E) Ensure all documentation and recording of program settings has been done.

5. Finally, correlate generated, identified and isolated information:

   A) At this point, if it has not already been done, or done piecemeal, it is time to try to correlate as many disparate elements as possible.

   B) How to do this is extremely complex and depends on the investigative context and how much information or potential evidence was identified.

   C) This is where the analytical abilities of the investigator or analyst are key.

2.13 Wrap up and additional points of consideration

The most difficult part of any analysis is correlating the various pieces of information. Some may seem disparate from the rest, yet they may still fit in. This is all too true where advanced persistent threats are concerned. Unfortunately, there is no one-size-fits-all solution for proceeding when correlating information. Key to a successful investigation is being methodical and ruling out unrelated information.

With this general methodology, it should be possible to dissect almost any memory image. Nevertheless, making sense of the information takes understanding and experience. The best way to gain these are to work with some of the many publicly-available memory images and analyse

them, looking at first for what is obvious and comparing results with the community. Then, as expertise increases, begin by delving deeper and deeper into the memory images. There are some highly specialised memory forensics courses out there, which can really help.

At the end of the day, if malware resided in memory, it will leave some trace of itself, unless it completely wiped itself from memory and pagefile or the system was left running for too long. However, this is almost never the case; there is just about always some hint of its existence.

3 Conclusion and discussion

We have provided many case studies into memory analysis, and from them, assembled this memory analysis methodology. It is suitable for use by investigators and analysts at all ranges in the spectrum.

With it, it should be possible to handle even the most difficult of memory investigations. But, like all methodologies, it is subject to change as per requirements. It was never the author's intention to set it in stone; rather it was meant to be a "living" document in that it is flexible and readily adapted to changing requirements or circumstances.

It is difficult to say when it should be used. In cases where analyses are expected to be simple and straightforward, using it may unnecessarily lengthen an investigation. In situations where basic analysis has turned up nothing of value then perhaps it should be implemented.

Finally, it is hoped that it will be of use to the community. For the moment, this methodology concludes our work into memory analysis.

This page intentionally left blank.

References

[1] Ligh, Michael Hale; Case, Andrew; Levy, Jamie and Walters, AAron. The Art of Memory Forensics. Book. First edition. Wiley & Sons Publishing. ISBN-13: 9781118825099. 2014.

[2] Kornblum, Jesse and Torres, Alissa. Memory Forensics 526 Training Manuals. Training manuals. SANS.org. 2014.

[3] Zeltser, Lenny. Reverse Engineering Malware Forensics 610 Training Manuals. Training manuals. SANS.org. 2012.

[4] Carbone, R. Malware memory analysis for non-specialists: Investigating a publicly available memory image of the Zeus Trojan horse. Technical Memorandum. DRDC Valcartier TM 2013-018. Defence R&D Canada – Valcartier. April 2013. Last Accessed: January 2017. http://cradpdf.drdc-rddc.gc.ca/PDFS/unc166/p801024_A1b.pdf.

[5] Carbone, R. Malware memory analysis for non-specialists: Investigating publicly available memory images for Prolaco and SpyEye. Technical Memorandum. DRDC Valcartier TM 2013-155. Defence R&D Canada – Valcartier. October 2013. Last Accessed: January 2017. http://cradpdf.drdc-rddc.gc.ca/PDFS/unc166/p801025_A1b.pdf.

[6] Carbone, R. Malware memory analysis for non-specialists: Investigating publicly available memory image 0zapftis (R2D2). Technical Memorandum. DRDC Valcartier TM 2013-177. Defence R&D Canada – Valcartier. October 2013. Last Accessed: January 2017. http://cradpdf.drdc-rddc.gc.ca/PDFS/unc166/p800963_A1b.pdf.

[7] Carbone, R. Malware memory analysis for non-specialists: Investigating publicly available memory image for the Stuxnet worm. DRDC Scientific Report. DRDC-RDDC-2013-R1. Defence Research and Development Canada. November 2013. Last Accessed: January 2017. http://cradpdf.drdc-rddc.gc.ca/PDFS/unc165/p538612_A1b.pdf.

[8] Carbone, R. Malware memory analysis for non-specialists: Investigating publicly available memory image for the Tigger Trojan horse. DRDC Scientific Report. DRDC-RDDC-2013-R28. Defence Research and Development Canada. June 2014. Last Accessed: January 2017. http://cradpdf.drdc-rddc.gc.ca/PDFS/unc166/p800199_A1b.pdf.

[9] Carbone, R. Malware memory analysis of the IVYL Linux rootkit: Investigating a publicly available Linux rootkit using the Volatility memory analysis framework. DRDC Scientific Report. DRDC-RDDC-2015-R060. Defence Research and Development Canada. April 2015. Last Accessed: January 2017. http://cradpdf.drdc-rddc.gc.ca/PDFS/unc189/p801882_A1b.pdf.

[10] Carbone, R. Malware memory analysis of the Jynx2 Linux rootkit: Investigating a publicly available Linux rootkit using the Volatility memory analysis framework. DRDC Scientific Report. DRDC-RDDC-2014-R176. Defence Research and Development Canada. October 2014. Last Accessed: January 2017. http://cradpdf.drdc-rddc.gc.ca/PDFS/unc167/p800829_A1b.pdf.

[11] Carbone, R. Memory analysis of the KBeast Linux rootkit: Investigating publicly available Linux rootkit using the Volatility memory analysis framework. DRDC Scientific Report. DRDC-RDDC-2015-R064. Defence Research and Development Canada. June 2015. Last Accessed: January 2017. http://cradpdf.drdc-rddc.gc.ca/PDFS/unc189/p801869_A1b.pdf.

[12] Bulk extractor. ForensicsWiki. June 2015. Last Accessed: March 2017. http://www.forensicswiki.org/wiki/Bulk_extractor.

Bibliography

1. Almarri, Saeed and Sant, Paul. Optimised Malware Detection in Digital Forensics. Technical paper. International Journal of Network Security & Its Application, Vol. 6, No. 1. January 2014. Last Accessed: January 2017. http://airccse.org/journal/nsa/6114nsa01.pdf.

2. Bhatnagar, Richa; Ansari, Mariya Khurshid; Bhatnagar, Sakshi and Barik, Harshbardhan. An Expert Anti-Malware Detection System. Technical paper. International Journal of Soft Computing and Engineering, Volume 2, Issue 5, November 2012. Last Accessed: January 2017. https://pdfs.semanticscholar.org/e9bf/7dc2c1f6854902ae25081680af3ef1291802.pdf.

3. Bilby, Darren. Low Down and Dirty: Anti-forensic Rootkits. Presentation. Presented at BlackHat Japan 2006. 2006. Last accessed: January 2017. https://www.blackhat.com/presentations/bh-jp-06/BH-JP-06-Bilby-up.pdf.

4. Blount, Jonathan Joseph. Adaptive rule-based malware detection employing learning classifier systems. Master's thesis. Missouri University of Science and Technology. 2011. Last Accessed: January 2017. http://scholarsmine.mst.edu/cgi/viewcontent.cgi?article=6007&context=masters_theses.

5. Brezinski, D. and Killalea, T. Guidelines for Evidence Collection and Archiving. Request for Comment (RFC). The Internet Society. 2002. Last accessed: January 2017. http://www.ietf.org/rfc/rfc3227.txt.

6. Broder, Evan. Transparent Detection of Computer Malware using Virtualization. Technical paper. MIT Computer Science and Artificial Intelligence Lab. May 2010. Last Accessed: January 2017. http://web.mit.edu/broder/Public/6.uap-report.pdf.

7. Butler, Jamie. Memory acquisition and the pagefile(s). Online informational web page. FireEye. July 2010. Last accessed: January 2017. https://www.fireeye.com/blog/threat-research/2010/07/memory-acquisition-pagefiles-part-ii.html.

8. Carbone, R. and Bean, C. Application of Bloom Filters in Digital Forensics: Identifying hard-to-detect data from Partial Evidentiary Hashes. Magazine article. Issue 24, August 2015. Digital Forensics Magazine. August 2015.

9. Carbone, R. and Bean, C. Modified Bloom Filter Case Studies. Magazine article. Issue 25, October 2015. Digital Forensics Magazine. October 2015.

10. Carbone, R. Malware memory analysis for non-specialists: Investigating a publicly available memory image of the Zeus Trojan horse. Technical Memorandum. DRDC Valcartier TM 2013-018. Defence R&D Canada – Valcartier. April 2013. Last Accessed: January 2017. http://cradpdf.drdc-rddc.gc.ca/PDFS/unc166/p801024_A1b.pdf.

11. Carbone, R. Malware memory analysis for non-specialists: Investigating publicly available memory images for Prolaco and SpyEye. Technical Memorandum. DRDC Valcartier

TM 2013-155. Defence R&D Canada – Valcartier. October 2013. Last Accessed: January 2017. http://cradpdf.drdc-rddc.gc.ca/PDFS/unc166/p801025_A1b.pdf.

12. Carbone, R. Malware memory analysis for non-specialists: Investigating publicly available memory image 0zapftis (R2D2). Technical Memorandum. DRDC Valcartier TM 2013-177. Defence R&D Canada – Valcartier. October 2013. Last Accessed: January 2017. http://cradpdf.drdc-rddc.gc.ca/PDFS/unc166/p800963_A1b.pdf.

13. Carbone, R. Malware memory analysis for non-specialists: Investigating publicly available memory image for the Stuxnet worm. DRDC Scientific Report. DRDC-RDDC-2013-R1. Defence Research and Development Canada. November 2013. Last Accessed: January 2017. http://cradpdf.drdc-rddc.gc.ca/PDFS/unc165/p538612_A1b.pdf.

14. Carbone, R. Malware memory analysis for non-specialists: Investigating publicly available memory image for the Tigger Trojan horse. DRDC Scientific Report. DRDC-RDDC-2013-R28. Defence Research and Development Canada. June 2014. Last Accessed: January 2017. http://cradpdf.drdc-rddc.gc.ca/PDFS/unc166/p800199_A1b.pdf.

15. Carbone, R. Malware memory analysis of the IVYL Linux rootkit: Investigating a publicly available Linux rootkit using the Volatility memory analysis framework. DRDC Scientific Report. DRDC-RDDC-2015-R060. Defence Research and Development Canada. April 2015. Last Accessed: January 2017. http://cradpdf.drdc-rddc.gc.ca/PDFS/unc189/p801882_A1b.pdf.

16. Carbone, R. Malware memory analysis of the Jynx2 Linux rootkit: Investigating a publicly available Linux rootkit using the Volatility memory analysis framework. DRDC Scientific Report. DRDC-RDDC-2014-R176. Defence Research and Development Canada. October 2014. Last Accessed: January 2017. http://cradpdf.drdc-rddc.gc.ca/PDFS/unc167/p800829_A1b.pdf.

17. Carbone, R. Memory analysis of the KBeast Linux rootkit: Investigating publicly available Linux rootkit using the Volatility memory analysis framework. DRDC Scientific Report. DRDC-RDDC-2015-R064. Defence Research and Development Canada. June 2015. Last Accessed: January 2017. http://cradpdf.drdc-rddc.gc.ca/PDFS/unc189/p801869_A1b.pdf.

18. Carbone, R. State of the art concerning memory acquisition software: A detailed examination of Linux, BSD and Solaris live memory acquisition. Technical Memorandum. DRDC Valcartier TM 2012-008. Defence R&D Canada – Valcartier. March 2012. Last Accessed: January 2017. http://cradpdf.drdc-rddc.gc.ca/PDFS/unc150/p536645_A1b.pdf.

19. Carbone, R. The definitive guide to Linux-based live memory acquisition tools: An addendum to "State of the art concerning memory acquisition software: A detailed examination of Linux, BSD and Solaris live memory acquisition". Technical Memorandum. DRDC Valcartier TM 2012-319. Defence R&D Canada – Valcartier. September 2013. Last Accessed: January 2017. http://cradpdf.drdc-rddc.gc.ca/PDFS/unc160/p800486_A1b.pdf.

20. Carbone, R.; Salois, M. and Bean, C. An in-depth analysis of the cold boot attack: Can it be used for sound forensic memory acquisition? Technical Memorandum. DRDC Valcartier TM 2010-296. Defence R&D Canada – Valcartier. January 2011. Last accessed: January 2017. http://cradpdf.drdc-rddc.gc.ca/PDFS/unc188/p534323_A1b.pdf.

21. Carbone, Richard and Bourdon-Richard, Sébastien. The definitive guide to Linux-based live memory acquisition tools: An addendum to "State of the art concerning memory acquisition software: A detailed examination of Linux, BSD and Solaris live memory acquisition". Technical Memorandum. DRDC Valcartier TM 2012-319. Defence R&D Canada – Valcartier and Royal Canadian Mounted Police (Integrated Technological Crime Unit). September 2013. Last accessed: January 2017. http://cradpdf.drdc-rddc.gc.ca/PDFS/unc160/p800486_A1b.pdf.

22. Case, Andrew and Richard III, Golden G. Advancing Mac OS X rootkit detection. Technical paper. DFRWS 2015 US. 2015. Last Accessed: January 2017. https://www.dfrws.org/sites/default/files/session-files/paper-advancing_mac_os_x_rootkit_detection.pdf.

23. Chan, Ellick; Wan, Winston; Chaugule, Amey and Campbell, Roy. A Framework for Volatile Memory Forensics. Technical paper. University of Illinois at Urbana-Champaign. 2009. Last Accessed: January 2017. http://www.sigsac.org/ccs/CCS2009/pd/abstract_25.pdf.

24. Chau, Duen Horng; Nachenberg, Carey; Wilhelm, Jeffrey; Wright, Adam and Faloutsos, Christos. Polonium: Tera-Scale Graph Mining and Inference for Malware Detection. Technical paper. Carnegie Mellon University & Symantec. December 2012. Last Accessed: January 2017. http://www.cc.gatech.edu/~dchau/polonium/polonium_sdm2011.pdf.

25. Cohen, Michael. How to stop memory acquisition by changing one byte. Blog. Rekall Memory Forensics blog. March 2014. Last Accessed: January 2017. http://rekall-forensic.blogspot.ca/2014/03/how-to-stop-memory-acquisition-by.html.

26. Cohen, Michael. Windows Virtual Address Translation and the Pagefile. Online informational web page. Rekall. 2014. Last Accessed: January 2017. http://www.rekall-forensic.com/posts/2014-10-25-pagefile.html.

27. Damodaran, Anusha. Combining Dynamic and Static Analysis for Malware Detection. Master's thesis. San Jose State University. May 2015. Last Accessed: January 2017. http://scholarworks.sjsu.edu/cgi/viewcontent.cgi?article=1391&context=etd_projects.

28. Damshenas, Moshen; Dehghantanha, Ali and Mahmoud, Ramlan. A SURVEY ON MALWARE PROPAGATION, ANALYSIS, AND DETECTION. Technical paper. International Journal of Cyber-Security and Digital Forensics (IJCSDF) 2(4): 10–29 & The Society of Digital Information and Wireless Communications, 2013 (ISSN: 2305-0012). December 2013. Last Accessed: January 2017. http://sdiwc.us/digitlib/journal_paper.php?paper=00000875.pdf.

29. Dell SecureWorks Counter Threat Intelligence Unity Threat Intelligence. Stegoloader: A Stealth Information Stealer. Threat analysis. Dell SecureWorks Counter Threat Intelligence Unity Threat Intelligence. June 2015. Last Accessed: January 2017. https://www.secureworks.com/research/stegoloader-a-stealthy-information-stealer.

30. Demme, John; Maycock, Matthew; Schmitz, Jared; Tang, Adrian; Waksman, Adam; Sethumadhavan, Simha and Stolfo, Salvatore. On the Feasibility of Online Malware

Detection with Performance Counters. Technical paper. Department of Computer Science, Columbia University. 2013. Last Accessed: January 2017. http://www.cs.columbia.edu/~jdd/papers/isca13_malware.pdf.

31. Dias, Ricardo. Intelligence-Driven Incident Response with YARA. SANS Gold paper, GCFA. SANS.org. October 14. Last Accessed: January 2017. https://www.sans.org/reading-room/whitepapers/forensics/intelligence-driven-incident-response-yara-35542.

32. Dulaunoy, Alexandre. An Introduction to Incident Detection and Response: Memory Forensic Analysis. Presentation. Computer Incident Response Center, Luxembourg. February 2015. Last Accessed: January 2017. http://www.foo.be/cours/dess-20142015/memory-analysis.pdf.

33. Embleton, Shawn; Sparks, Sherri and Zou, Cliff. SMM Rootkits: A New Breed of OS Independent Malware. Technical paper. Presented at SecureComm 2008, September 22–25, 2008, Istanbul, Turkey. September 2008. Last accessed: January 2017. http://www.eecs.ucf.edu/~czou/research/SMM-Rootkits-Securecom08.pdf.

34. EMC Corporation. RSA ECAT: Signature-less malware Detection. Data sheet. EMC Corporation. 2013. Last Accessed: January 2017. https://www.emc.com/collateral/software/data-sheet/ds-ecat-final.pdf.

35. EMC Corporation. RSA Incident Response: Threat Detection Techniques – Backoff Point of Sale Malware. White paper. EMC Corporation. August 2014. Last Accessed: January 2017. https://blogs.rsa.com/wp-content/uploads/2014/12/threat-detection-report-backoff-pos.pdf.

36. Farmer, Dan and Venema, Wietse. Forensic Discovery. Book. First edition. Pearson Education. ISBN-10: 0-201-63497-X. 2004.

37. Fedler, Rafael; Schutte, Julian and Kulicke, Marcel. An evaluation of Android antivirus apps. Technical Report. Version 1.0. April 2013. Last Accessed: January 2017. https://www.aisec.fraunhofer.de/content/dam/aisec/Dokumente/Publikationen/Studien_TechReports/deutsch/042013-Technical-Report-Android-Virus-Test.pdf.

38. FireEye, Inc. Redline User Guide. User guide. Version 1.14. 2015. Last Accessed: January 2017. https://www.fireeye.jp/content/dam/fireeye-www/services/freeware/ug-redline.pdf.

39. Flaglien, Anders Orsten. Cross-Computer Malware Detection in Digital Forensics. Presentation for Master's thesis. Gjøvik University College. June 2010. Last Accessed: January 2017. https://brage.bibsys.no/xmlui//bitstream/handle/11250/143928/Cross-Computer_Malware_Det_in_DF_Anders_OF.pdf.

40. Gruhn, Michael. Windows NT pagefile.sys Virtual Memory Analysis. Technical paper. Department of Computer Science, IT Security Infrastructure, Friedrich-Alexander University Erlangen-Nürnberg. 2014. Last accessed: January 2017. https://www1.cs.fau.de/filepool/gruhn/pagefile.pdf.

41. Haist, Robert. Restoring Windows CMD sessions from pagefile.sys. Forensics blog. Robert Haist. December 2013. Last accessed: January 2017. http://blog.roberthaist.com/2013/12/restoring-windows-cmd-sessions-from-pagefile-sys-2/.

42. Halderman, J. Alex; Schoen, Seth D.; Heninger, Nadia; Clarkson, William; William, Paul; Calandrino, Joseph A.; Feldman, Ariel J.; Appelbaum, Jacob; and Felten, Edward W. Lest We Remember: Cold Boot Attack on Encryption Keys. Technical paper. Princeton University, Electronic Frontier Foundation and Wind River Systems. 2009. Last accessed: January 2017. https://jhalderm.com/pub/papers/coldboot-sec08.pdf.

43. Harrell, Cory. Finding Malware: Like Iron Man. Presentation. SANS DFIR Summit 2013. July 2013. Last Accessed: January 2017. https://digital-forensics.sans.org/summit-archives/DFIR_Summit/Finding-Malware-Like-Iron-Man-Corey-Harrell.pdf.

44. Haruyama, Takahiro and Suzuki, Hiroshi. One-byte Modification for Breaking Memory Forensic Analysis. Presentation. Presented at BlackHat Netherlands, 2012. 2012. Last accessed: January 2017. https://media.blackhat.com/bh-eu-12/Haruyama/bh-eu-12-Haruyama-Memory_Forensic-Slides.pdf.

45. Hayon, Adi and Teller, Tomer. Enhancing Malware Analysis with Automated Memory Analysis. Presentation. BlackHat 2014 USA. Check Point Software Technologies. 2014. Last Accessed: January 2017. https://www.blackhat.com/docs/us-14/materials/arsenal/us-14-Teller-Automated-Memory-Analysis-Slides.pdf.

46. Iqbal, Hameed. Forensic Analysis of Physical Memory and Page File. Master's thesis. Department of Computer Science and Media Technology, Gjøvik University College. 2009. Last accessed: January 2017. http://brage.bibsys.no/xmlui/bitstream/handle/11250/143807/Hameed%2bIqbal.pdf?sequence=1&isAllowed=y.

47. Javid, Salman. Analysis and Detection of Heap-based Malwares Using Introspection in a Virtualized Environment. Master's thesis. University of New Orleans. August 2014. Last Accessed: January 2017. http://scholarworks.uno.edu/cgi/viewcontent.cgi?article=2947&context=td.

48. Jiang, Xuxian; Wang, Xinyuan and Xu, Dongyan. Stealthy Malware Detection Through VMM-Based "Out-of-the-Box" Semantic View Reconstruction. Technical paper. October 2007. Last Accessed: January 2017. https://cs.gmu.edu/~xwangc/Publications/CCS07-VMwatcher.pdf.

49. Jiang, Xuxian; Wang, Xinyuan and Xu, Dongyan. Stealthy Malware Detection Through VMM-BASED "Out-of-the-Box" Semantic View Reconstruction. Technical paper. Department of Information and Software Engineering, George Mason University & Department of Computer Science, Purdue University. November 2007. Last Accessed: January 2017. https://cs.gmu.edu/~xwangc/Publications/CCS07-VMwatcher.pdf.

50. Kaspersky Lab. Equation Group: Questions And Answers. Technical Report/Malware intelligence report. Version 1.5. February 2015. Last Accessed: January 2017. https://securelist.com/files/2015/02/Equation_group_questions_and_answers.pdf.

51. Kaspersky Labs. Equation Group: Questions and Answers. Technical paper. Version 1.5. Kaspersky Labs. February 2015. Last Accessed: January 2017. http://www.mcafee.com/ca/resources/solution-briefs/sb-quarterly-threats-nov-2015-1.pdf.

52. Kirat, Dhilung and Vigna, Giovanni. BareCloud: Bare-metal Analysis-based Evasive Malware Detection. Technical paper. University of California at Santa Barbara. July 2014. Last Accessed: January 2017. https://www.cs.ucsb.edu/~vigna/publications/2014_USENIX_BareCloud.pdf.

53. Kleissner, Peter. Hibernation File Format. Technical paper. Kleissner & Associates. 2010. Last Accessed: January 2017. http://stoned-vienna.com/downloads/Hibernation%20File%20Attack/Hibernation%20File%20Format.pdf.

54. Kolbitsch, Clemens; Comparetti, Paolo Milani; Kruegel, Christopher; Kirda, Engin; Zhou, Xiaoyong and Wang, XiaoFeng. Effective and Efficient Malware Detection at the End Host. Technical paper. Secure Systems Lab (TU Vienna); UC Santa Barbara; Institute Eurocom (Sophia Antipolis) and Indiana University at Bloomington. Presented at Usenix 2009. Last Accessed: January 2017. https://www.usenix.org/legacy/event/sec09/tech/full_papers/kolbitsch.pdf.

55. Kolbitsch, Clemens; Livshits, Benjamin; Zorn, Benjamin and Seifert, Christian. Rozzle: De-Cloaking Internet Malware. Technical paper. 2012 IEEE Symposium on Security and Privacy. Lastline, Inc.; Microsoft Research and Microsoft Corporation. 2012. Last Accessed: January 2017. http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6234429

56. Kolter, J. Zico and Maloof, Marcus A. Learning to Detect and Classify Malicious Executables in the Wild. Technical paper. Journal of Machine Learning Research 7 (2006) 2721–2744. December 2006. Last Accessed: January 2017. http://www.jmlr.org/papers/volume7/kolter06a/kolter06a.pdf.

57. Korkin, Igor and Nesterov, Ivan. Applying Memory Forensics to Rootkit Detection. Technical Report. National Research Nuclear Moscow Engineering & Physics Institute and Moscow Institute of Physics and Technology. May 2014. Last Accessed: January 2017. http://arxiv.org/ftp/arxiv/papers/1506/1506.04129.pdf.

58. Kornblum, Jesse and Torres, Alissa. Memory Forensics 526 Training Manuals. Training manuals. SANS.org. 2014.

59. Kornblum, Jesse D. Using Every Part of the Buffalo in Windows Memory Analysis. Technical paper. Digital Investigation, Volume 4, Issue 1, March 2007, Pages 24–29. March 2007. Last accessed: January 2017. http://www.sciencedirect.com/science/article/pii/S1742287607000047.

60. Lee, Rob. APT Attacks Exposed, Network Host Memory and Malware Analysis. Presentation. SANS DFIR Summit 2014. May 2014. Last Accessed: January 2017. http://computerforensicsblog.champlain.edu/wp-content/uploads/2014/06/APT-Attacks-Exposed-Network-Host-Memory-and-Malware-Analysis-Lee-Tilbury-Hagen-5-21-2014.pdf.

61. Lee, Seokhee; Savoldi, Antonio; Lee, Sangjin and Lim, Jongin. Windows Pagefile Collection and Analysis for a Live Forensics Context. Technical paper. Center for Information Security Technologies, Korea University, Seoul, Korea and Department for Electronics for Automation, University of Brescia, Via Branze 38, Brescia, Italy. December 2007. Last accessed: January 2017. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4426211&tag=1.

62. Liao, Haofu. Survey: Virtual Machine Introspection Based System Monitoring and Malware Detection Techniques. Technical paper. Department of Computer Science, University of Rochester. December 2015. Last Accessed: January 2017. http://www.cs.rochester.edu/u/hliao6/projects/other/os_survey.pdf.

63. Ligh, Michael Hale; Case, Andrew; Levy, Jamie and Walters, AAron. The Art of Memory Forensics. Book. First edition. Wiley & Sons Publishing. ISBN-13: 9781118825099. 2014.

64. Lorch, Jacob R.; Parno, Bryan and Wang, Helen J. SAV-V: Securing Anti-Virus with Virtualization. Technical paper. Microsoft Research & Carnegie Mellon University. September 2011. Last Accessed: January 2017. http://research.microsoft.com/pubs/153846/sav-v.pdf.

65. Ma, Weiqin; Duan, Pa; Liu, Sanmin; Gu, Guofei and Liu, Jyh-Charn. Shadow Attacks: Automatically Evading System-Call-Behaviour Based Malware Detection. Technical paper. Department of Computer Science and Engineering, Texas A&M University. December 2011. Last Accessed: January 2017. http://faculty.cse.tamu.edu/guofei/paper/ShadowAttacks_final-onecolumn.pdf.

66. Maclean, Nicholas P. Acquisition and analysis of windows memory. Master's thesis, University of Strathclyde. 2006. Last accessed: January 2017. http://4tphi.net/fatkit/papers/NickMaclean2006.pdf.

67. Mandiant Corp. MANDIANT Redline Users Guide. User guide. Version 1.7. 2012. Last Accessed: January 2017. http://web.nmsu.edu/~agarcia9/portfolio/RedlineUserGuide.pdf.

68. Matonis, Michael. Page_brute (software tool). Readme.md. January 5, 2014. Last accessed: January 2017. https://github.com/matonis/page_brute.

69. McKnight, Jim. How To Remove Malware From Your PC. How to. Jimopi.net. December 2014. Last Accessed: January 2017. http://www.gegeek.com/tech_reference/home_page/Removing%20Malware.pdf.

70. Mell, Peter; Kent, Karen and Nusbaum, Joseph. Guide to Malware Incident Prevention and Handling: Recommendations of the National Institute of Standards and Technology. Special

Publication. Publication No. 800-83. November 2005. Last Accessed: January 2017. http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-83.pdf.

71. Milković, Luka. 29c3 defeating windows memory forensics. Online Informational web site / mailing list. Volatility Project. January 2013. Last Accessed: January 2017. http://lists.volatilesystems.com/pipermail/vol-users/2013-January/000696.html.

72. Milković, Luka. Defeating Windows memory forensics 29c3. Presentation. Presented at Chaos Communication Congress in Hamburg, 2012. December 2012. Last accessed: January 2017. https://storage.googleapis.com/mwg-internal/de5fs23hu73ds/progress?id=OgFt4FQ17lztdXzP7_93Fp3ns8i3lqdq0WvV0oXggFI.

73. Mohaisen, Aziz and Alrawi, Omar. AMAL: High-Fidelity, Behavior-based Automated Malware Analysis and Classification. Technical paper. Verisign Labs & Qatar Computing Research Institute. September 2014. Last Accessed: January 2017. http://www.cse.buffalo.edu/~mohaisen/doc/tr/amal-tr.pdf.

74. Nasi, Emeric. Bypass Antivirus Dynamic Analysis. Technical paper. Sevagas.com August 2014. Last Accessed: January 2017. http://packetstorm.foofus.com/papers/virus/BypassAVDynamics.pdf.

75. Natarajan, Ramesh. Linux 101 Hacks: Practical Examples to Build a Strong Foundation in Linux. Book. First Edition. Self-Published. 2009.

76. Park, Junewon. Acquiring Digital Evidence from Botnet Attacks: Procedures and Methods. Master's thesis. School of Computing and Mathematical Sciences, AUT University. 2011. Last Accessed: January 2017. http://www.covert.io/research-papers/security/Acquiring%20Digital%20Evidence%20from%20Botnet%20Attacks:%20Procedures%20and%20Methods%20(PhD%20Thesis).pdf.

77. Parker, Steve. Shell Scripting: Expert Recipes for Linux, Bash, and More. Book. First Edition. Wiley & Sons Publishing. 2011. ISBN-13: 978-0-470-02448-5.

78. Paul, Nathanael R. Disk-Level Behavioral Malware Detection. PhD thesis. School of Engineering and Applied Sciences, University of Virginia. May 2008. Last Accessed: January 2017. http://www.cs.virginia.edu/~evans/theses/paul.pdf.

79. Perry, Brandon. Analyzing the Windows pagefile.sys from GNU/Linux. Forensics blog. Brandon Perry. October 4, 2011. Last accessed: January 2017. http://volatile-minds.blogspot.ca/2011/10/analyzing-windows-pagefilesys-from.html.

80. Prakash, Aravind; Venkataramani, Eknath; Yin, Heng and Lin, Zhiqiang. On the Trustworthiness of memory Analysis – An Empirical Study from the Perspective of Binary Execution. Technical paper. IEEE Transactions of Dependable and Secure Computing, Volume 12, Number 5, Sept/Oct 2015. September 2015. Last Accessed: January 2017. http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6942280.

81. Prakash, Aravind; Venkataramani, Eknath; Yin, Heng and Lin, Zhiqiang. Manipulating Semantic Values in Kernel Data Structures: Attack Assessments and Implications. Technical

paper. Department of EECS, Syracuse University & Department of Computer Science, University of Texas at Dallas. 2013. Last Accessed: January 2017. https://www.utdallas.edu/~zxl111930/file/DSN13.pdf.

82. Rieck, Konrad; Trinius, Philipp; Willems, Carsten and Holz, Thorsten. Automatic Analysis of Malware Behavior using Machine Learning. Technical paper. Berlin Institute of Technology, University of Mannheim & Vienna Universtiy of Technology. 2011. Last Accessed: January 2017. http://www.mlsec.org/malheur/docs/malheur-jcs.pdf.

83. Robertson, Chad. Indicators of compromise in memory forensics. SANS Gold paper. SANS.org. February 2013. Last accessed: January 2017. http://www.sans.org/reading-room/whitepapers/forensics/indicators-compromise-memory-forensics-34162.

84. Rutkowska, Joanna. Beyond the CPU: Defeating Hardware Based RAM Acquisition (part I: AMD case). Presentation. Presented BlackHat 2007. 2007. Last accessed: January 2017. https://www.first.org/conference/2007/papers/rutkowska-joanna-slides.pdf.

85. Schuster, Andreas. YARA: An Introduction. Presentation. 26th Annual FIRST Conference, Boston. June 2014. Last Accessed: January 2017. https://www.google.ca/url?url=https://www.first.org/resources/papers/conference2014/first_2014_-_schuster-_andreas_-_yara_basic_and_advanced_20140619.pdf&rct=j&frm=1&q=&esrc=s&sa=U&ved=0ahUKEwj4-5Gv__XMAhVPEllKHaEiBY44bhAWCCgwAg&usg=AFQjCNGaB4hz_epk5DcxO9bwVSS6b4D8qw.

86. Schweiger, Michael; Chung, Sam and Endicott-Popovsky, Barbara. Malware Analysis on the Cloud: Increased Performance, Reliability, and Flexibility. Graduate capstone. Institute of Technology & Center for Information Assurance and Cybersecurity, University of Washington. January 2014. Last Accessed: January 2017. https://www.tacoma.uw.edu/sites/default/files/sections/InstituteTechnology/M_Schweiger.pdf.

87. Siddiqui, Muazzam Ahmed. Data Mining Methods for Malware Detection. PhD thesis. University of Central Florida. 2008. Last Accessed: January 2017. http://etd.fcla.edu/CF/CFE0002303/Siddiqui_Muazzam_A_200808_PhD.pdf.

88. Sikorski, Michael. Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software. First Edition. Book. No Starch Press. ISBN-13: 9781593272906.

89. Sparks, Sherri and Butler, Jamie. "Shadow Walker": Raising The Bar For Rootkit Detection. Technical paper. Phrack Inc. Volume 0x0b, Issue 0x3d, Phile #0x08 of 0x14. January 2005. Last Accessed: January 2017. http://phrack.org/issues/63/8.html.

90. Sparks, Sherri and Butler, Jamie. "Shadow Walker": Raising The Bar For Rootkit Detection. Presentation. Presented at BlackHat Japan 2005. 2005. Last Accessed: January 2017. https://www.blackhat.com/presentations/bh-jp-05/bh-jp-05-sparks-butler.pdf.

91. Spensky, Chad; Hu, Hongyi and Leach, Kevin. LO-PHI: Low-Observable Instrumentation for Malware Analysis. Technical paper. MIT Lincoln Laboratory & University of California, Santa Barbara. February 2016. Last Accessed: January 2017. https://www.internetsociety.org/sites/default/files/blogs-media/lo-phi-low-observable-physical-host-instrumentation-malware-analysis.pdf.

92. Stewin, Patrick and Bystrov, Iurii. Understanding DMA Malware. Security in Telecommunications, Technical University of Berlin. April 2012. Last Accessed: January 2017. http://www.stewin.org/papers/dimvap15-stewin.pdf.

93. Stimson, Jared M. FORENSIC ANALYSIS OF WINDOW'S VIRTUAL MEMORY INCORPORATING THE SYSTEM'S PAGEFILE. Master's thesis. Naval Postgraduate School, Monterey, California. December 2008. Last accessed: January 2017. http://calhoun.nps.edu/bitstream/handle/10945/3714/08Dec_Stimson.pdf?sequence=1&isAllowed=y.

94. Stüttgen, Johannes and Cohen, Michael. Anti-forensic resilient memory acquisition. Technical paper. DFRWS 2013. Digital Investigation 10 (2013) S105–S115. 2013. Last Accessed: January 2017. https://www.dfrws.org/sites/default/files/session-files/paper-anti-forensic_resilient_memory_acquisition.pdf.

95. Stüttgen, Johannes. On the Viability of Memory Forensics in Compromised Environments. PhD thesis. Der Technischen Fakultät der Friedrich-Alexander-Universität Erlangen-Nürnberg, zur Erlangung des Grades. May 2015. Last Accessed: January 2017. https://opus4.kobv.de/opus4-fau/files/6316/dissertation_stuettgen_published.pdf.

96. Stüttgen, Johannes; Vömel, Stefan and Denzel, Michael. Acquisition and analysis of compromised firmware using memory forensics. Technical paper. Digital investigation 12 (2015 S50-S60). 2015. Last accessed: January 2017. http://ac.els-cdn.com/S1742287615000110/1-s2.0-S1742287615000110-main.pdf?_tid=37ff6762-81aa-11e5-9cae-00000aacb35e&acdnat=1446500558_24f0f320dbd3e51e9f15289054d8186c.

97. Suiche, Matthieu. Windows Hibernation File For Fun 'N' Profit. Presentation. BlackHat 2008. 2008. Last Accessed: January 2017. https://www.blackhat.com/presentations/bh-usa-08/Suiche/BH_US_08_Suiche_Windows_hibernation.pdf.

98. Suiche, Matthieu. Windows Hibernation For Fun 'N' Profit. Presentation. Presented at BlackHat 2008. 2008. Last accessed: January 2017. http://msuiche.net/con/bhusa2008/Windows_hibernation_file_for_fun_%27n%27_profit-0.6.pdf.

99. Tang, Adrian; Sethumadhavan, Simha and Stolfo, Salvatore. Unsupervised Anomaly-based Malware Detection using Hardware Features. Technical paper. Columbia University. June 2014. Last Accessed: January 2017. https://mice.cs.columbia.edu/getTechreport.php?techreportID=1565&format=pdf&.

100. Teller, Tomer and Hayon, Adi. Enhancing Automated Malware Analysis Machines with Malware Analysis. Technical paper. Check Point software Technologies. 2014. Last

Accessed: January 2017. https://www.blackhat.com/docs/us-14/materials/arsenal/us-14-Teller-Automated-Memory-Analysis-WP.pdf.

101. Teller, Tomer. Detecting the One Percent: Advanced Targeted Malware Detection. Presentation. RSA Conference, 2013. 2013. Last Accessed: January 2017. https://www.rsaconference.com/writable/presentations/file_upload/spo2-t19_spo2-t19.pdf.

102. Thomas, Vinoo; Ramagopal, Prashanth; and Mohandas, Rahul. The Rise of AutoRun-Base. McAfee, 2009. Last Accessed: February 2017. http://download.adamas.ai/dlbase/ebooks/VX_related/The%20Rise%20of%20AutoRunBased%20Malware.pdf.

103. Tratt, Laurence. What role for static analysis in malware detection? Presentation. Tratt.net. April 2011. Last Accessed: January 2017. http://crest.cs.ucl.ac.uk/cow/12/slides/LaurieTratt_cow_apr_2011.pdf.

104. Trivedi, Nisarg. Study on Pagefile.sys in Windows System. Technical paper. IOSR Journal of Computer Engineering (IOSR-JCE), e-ISSN: 2278-0661, p- ISSN: 2278-8727, Volume 16, Issue 2, Ver. V (Mar–Apr. 2014), PP 11–16. April 2014. Last accessed: January 2017. http://www.iosrjournals.org/iosr-jce/papers/Vol16-issue2/Version-5/C016251116.pdf.

105. Tushar, Shantanu and Lakshman, Sarath. Linux Shell Scripting Cookbook. Book. Second Edition. Packt Publishing. May 2013. ISBN-13: 978-1-78216-274-2.

106. Vidas, Timothy. The Acquisition And Analysis Of Random Access Memory. Technical paper. Journal of Digital Forensic Practice, Volume 1, Number 1, 2006. 2006. Last accessed: January 2017. http://users.ece.cmu.edu/~tvidas/papers/JDFP06.pdf.

107. Vinod, P; Laxmi, V. and Gaur, M.S. Survey on Malware Detection Methods. Technical paper. Department of Computer Engineering, Malaviya National Institute of Technology, India. March 2009. Last Accessed: May 2016. http://www.security.iitk.ac.in/contents/events/workshops/iitkhack09/papers/vinod.pdf.

108. Vömel, Stefan and Freiling, Felix C. A survey of main memory acquisition and analysis techniques for the windows operating system. Technical paper. Digital Investigation, Volume 8, Issue 1, July 2011, Pages 3–22. July 2011. Last accessed: January 2017. http://ac.els-cdn.com/S1742287611000508/1-s2.0-S1742287611000508-main.pdf?_tid=6e4b831a-37c6-11e5-98fc-00000aacb35d&acdnat=1438376289_962ea4932c8bc01de07d603dff0b5bb1.

109. Vömel, Stefan. Forensic Acquisition and Analysis of Volatile Data in Memory. PhD dissertation. Der Technischen Fakultät der Friedrich-Alexander-Universität Erlangen-Nürnberg, zur Erlangung des Grades. December 2013. Last accessed: January 2017. https://opus4.kobv.de/opus4-fau/files/4111/dissertation_endfassung.pdf.

110. Williams, Jake and Torres, Alissa. ADD -- Complicating Memory Forensics Through Memory Disarray. Presentation. Schmoocon 2014. Last Accessed: January 2017. http://www.mediafire.com/view/h7bmcscbtyaeb6r/ADD_Shmoocon.pdf.

111. Yee, Chan Lee; Chuan, Lee Ling; Ismail, Mahamod and Jumari, Kasmiran. Metaware – An Extensible Malware Detection and Removal Toolkit. Technical paper. Department of Electrical, Electronic Ana System Engineering, Universiti Kebangsaan Malaysia. January 2011. Last Accessed: January 2017. http://icact.org/upload/2011/0700/20110700_finalpaper.pdf.

112. Yin, Heng and Song, Dawn. Whole-system Fine-grained Taint Analysis for Automatic Malware Detection and Analysis. Technical paper. College of William and Mary & Carnegie Mellon University. July 2006. Last Accessed: January 2017. http://bitblaze.cs.berkeley.edu/papers/malware-detect.pdf.

113. YobiWiki. RAM analysis. Online informational web page. YobiWiki. October 2013. Last Accessed: January 2017. http://wiki.yobi.be/wiki/RAM_analysis.

114. Zeltser, Lenny. Reverse Engineering Malware Forensics 610 Training Manuals. Training manuals. SANS.org. 2012.

115. Zhang, R.; Wang, L. and Zhang, S. Windows Memory Analysis Based on KPCR. Technical paper. Proceedings of the Fifth International Conference on Information Assurance and Security. 2009. Last Accessed: January 2017. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5284273.

116. Zhang, Ning; Sun, Kun; Lou, Wenjing; Hou, Thomas Y. and Jajodia, Sushil. Now You See Me: Hide and Seek in Physical Address Space. Technical paper. Virginia Polytechnic Institute, College of William and Mary, and George Mason University. Published in ASIA CCS'15, April 14–17, 2015, Singapore. 2015. Last Accessed: January 2017. http://www.cnsr.ictas.vt.edu/publication/NowYouSeeMe.pdf.

117. Zhang, Shuhui; Wang, Lianhai; Zhang, Ruichao and Guo, Qiuxiang. Exploratory Study on Memory Analysis of Windows 7 Operating System. Technical paper. Proceedings of the Third International Conference on Advanced Computer Theory and Engineering (ICACTE). 2010. Last Accessed: January 2017. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5579832.

List of symbols/abbreviations/acronyms/initialisms

ACL	Access Control List
AES	Advanced Encryption Standard
API	Application Programming Interface
APT	Advanced Persistent Threat
AV	Anti-virus or Antivirus
BSD	Berkeley Software Distribution
CFNOC	Canadian Forces Network Operation Centre
CLE	Canadian Law Enforcement
COTS	Commercial of the Shelf
DLL	Dynamic Linked Library or Dynamic Link Library
DRDC	Defence Research and Development Canada
FOSS	Free and Open Source Software
GiB	Gibibyte (2^{30})
HTML	HyperText Markup Language
JAR	Java Archive
KiB	Kibibyte (2^{10})
MOU	Memorandum of Understanding
MSI	Microsoft Installer
NSRL	National Software Reference Library
NTFS	NT (New Technology) File System
OLE	Object Linking and Embedding
PDF	Portable Document Format
R&D	Research & Development
RAT	Remote Access Tool
RCMP	Royal Canadian Mounted Police
ROT13	“Rotate 13 places” or “Rotate by 13 places”
RSA	Rivest-Shamir-Adleman
SHA-1/SHA-2/SHA-3	Secure Hash Algorithm-1/2/3
TM	Technical Memorandum

UPX	Ultimate Packer for Executables
XOR	Exclusive OR

DOCUMENT CONTROL DATA

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

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

DRDC – Valcartier Research Centre
Defence Research and Development Canada
2459 route de la Bravoure
Quebec (Quebec) G3J 1X5
Canada

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

UNCLASSIFIED

2b. CONTROLLED GOODS

(NON-CONTROLLED GOODS)
DMC A
REVIEW: GCEC DECEMBER 2013

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.)

A proposed investigative, generic broad-scoped Windows memory analysis methodology

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.)
March 2017	54	12

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.)

Scientific Report

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

DRDC – Valcartier Research Centre
Defence Research and Development Canada
2459 route de la Bravoure
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.)

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-RDDC-2017-R041	

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 work describes a functional, generic, broad-scoped investigative methodology for Windows memory analysis. The methodology applies equally to functional and damaged, or corrupted memory images and relies on Volatility, although other analysis frameworks will do. It is based on the author's various memory analysis case studies. Summing it up succinctly, the methodology aids the forensic practitioner in squeezing the maximum amount of possible evidence from a memory image. The proposed methodology is suitable for analysts at all levels of investigative capability. It provides guidance in extracting maximum evidence using simple, commonplace tools and techniques familiar to digital forensic practitioners. As with all methodologies, nothing is written in stone; the forensic practitioner must be flexible and agile in responding to ever-changing investigative requirements.

Dans ces travaux, on présente la description d'une méthode d'enquête fonctionnelle, générique et de large portée pour analyser la mémoire de Windows. Cette méthode s'applique autant aux images mémoire fonctionnelles qu'aux images altérées ou corrompues et s'appuie sur Volatility, même si d'autres cadres d'analyse feront tout aussi bien l'affaire. Elle est fondée sur diverses études de cas d'analyse de la mémoire menées par l'auteur. En bref, pour résumer, la méthode aide le praticien légiste à relever le maximum de traces possibles d'une image mémoire. La méthode proposée convient aux analystes à tous les niveaux de capacité d'enquête. Elle fournit des indications pour extraire le maximum de traces à l'aide d'outils et de techniques simples et bien connus des experts judiciaires en informatique. Comme pour toute méthode, rien n'est figé dans le béton; le praticien légiste doit faire preuve d'adaptabilité et de flexibilité pour répondre à des besoins en matière d'enquêtes en perpétuel changement.

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.)

Analysis methodology; Anti-malware; Anti-virus; Computer forensics; Data carving; Forensics; Forensic investigation; Infection; Investigation; Malware; Memory analysis; Memory forensics; Plugins; Rekall; Volatility