Using Memory Forensics to detect Malware processes.

Stephen Nwagwughiagwu, Rhoda Ajayi, Tarun Chaitanya Talluri

University of New Haven, 300 Boston Post Rd., West Haven, CT, 06516

Abstract

Digital investigators could easily discover and analyze malicious code on computer systems with relative ease in past times. Trojan horse programs, and UNIX rootkits did little to counteract forensic analysis of compromised systems. Since majority of malware functionality was easily observable at this time, there was little or no need for a digital investigator to perform an in-depth analysis or investigation. Now, intruders have become more cognizant of digital forensic techniques. Hence, malicious programs are increasingly designed to obstruct meaningful analysis by employing advanced techniques that thwart reverse engineering, encode and conceal network traffic, as well as minimizing the traces left on file systems during a malware attack. These malicious code developers are making both discovery and forensic analysis more difficult. These forms of malware are proliferating in nature, automatically spreading worms and providing remote control access such as the trojan horse and backdoor, and sometimes concealing their activities on the compromised host (rootkit behavior). Furthermore, malware has evolved to undermine security measures, by disabling antivirus tools and bypassing firewalls by connecting from within the network to external command and control servers.

In this project, we present an efficient and effective method to carry out the study of the memory of a computer system in order to identify malicious processes. This will be very useful for analyzing memory resident malware which never writes any information to disk and thus can go unnoticed. Furthermore, the main objective is to demonstrate and explain the importance of memory forensics of live machines and artifacts which can be found as well as methods and an efficient tool which can be used for extracting and analyzing data from Random Access Memory (RAM). In addition, it will be shown that in most forensic investigations, data contained in RAM can contain adequate evidence to solve a whole security incident and actually be everything a digital forensics investigator really needs.

Keywords: Malware, Disk, Memory Forensics, Artifacts

1. Introduction

Malware attacks are a nightmare for every modern organization and our current digital infrastructure. Malware poses to be a serious concern to the security posture of any organization. Adversaries are carrying out more advanced malware attacks on critical business infrastructures and data centers in such a way that it is difficult to trace. Malware analysis and memory forensics are powerful analytic concepts that can be combined together during an investigation. These have become a must-have skillset for combating advanced malware and overall security breaches. Memory forensics as applicable to malware analysis is the art of analyzing a memory image acquired from a targeted machine where the malware has been executed to obtain relevant number of artifacts that provide visibility into how, when and where malicious activities were performed on the concerned host machine. During an incident response, taking an image of a device can be time consuming coupled with the hurdle of transferring same image which could be over a 100GB in size to a forensic lab where it will be analyzed. This will take a large amount of time to get this done by the Incident Response team having in mind that time is of the essence in a forensic investigation.

During an incident response triggered by a malware attack, it is imperative that volatile data must be collected first as stated in (Malin et al., 2012). If the Random Access Memory (RAM) dump is not acquired first, there is every maximum likelihood that the system state might change because of the volatile nature of memory. It is advisable that incident responders should have a toolkit and a standard, repeatable, and reliable process [1]that can be used to collect system information and subsequently, further forensic analysis should be completed. Most importantly, it is very crucial to collect data in a way that others can review and analyze. The toolkit used should also create a hash of the file for integrity purposes.

Considering that part of computer crime evidence cannot be completely obtained from the system's physical

*Corresponding author.

Email addresses: snwag1@unh.newhaven.edu (Stephen Nwagwughiagwu), rajay1@unh.newhaven.edu (Rhoda Ajayi), ttall4@unh.newhaven.edu (Tarun Chaitanya Talluri)

URL: www.newhaven.edu (Tarun Chaitanya Talluri)

[1]Belkasoft Ram Capturer, http://forensic.belkasoft.com/en/ram-capturer

hard disk. It is recommended to access the physical memory to find important and critical information such as network connections which include IP Addresses, network protocols, time stamps as well as information about running malicious programs, worms, trojans and so on. Furthermore, we can extract sensitive information from email and messaging tools from the dumped physical memory. However, if the computer system is turned off, all these important information will be lost and cannot be retrieved as memory is highly volatile. Therefore, the research of forensics and analysis of physical memory has a high importance and relevance on the development of computer forensics technology and in the long run promoting the frontiers of digital forensics.

In this project we have used the following:

- Memory image of a Windows 7 machine gotten from Tek Defense
- Volatility framework 2.6.1 installed and configured on an Ubuntu VM (Virtual Machine).

2. Related Work

2.1. Memory Acquisition & Malware Analysis

In (Carvey, 2014), the concept of live forensics, live response, file analysis, malware detection, timeline was introduced and how to investigate a malware attack. Similarly, (Huseinović and Ribić, 2013) illustrates the process of obtaining the virtual machine memory dump for analysis. This work also proves that physical memory acquired can contain various data such as passwords, encryption keys, browser activity and relevant artifcats that might be of great interest to a forensic investigation. Also, it demonstrates how to preserve volatile data from VMware Workstation and Oracle Virtual box.

In the last couple of years, research has also been geared towards detecting malware written in Objective-C on Mac OS X as well as other programming languages. In the research, (Case and Richard III, 2016) a novel forensic technique was developed to properly examine the state of the Objective-C runtime inside of associated targeted processes. This work also identified a large number of suspicious activities ranging from keystroke logging to pointer swizzling. Furthermore, the model examined and compared the developed technique against other memory samples in the targeted Mac OS X attacks that are infected with the associated malware.

2.2. Memory Forensics: Volatility Framework

The Volatility Framework [2] is a completely open collection of tools, it is an open source project written in Python under the GNU General Public License, for the extraction of digital artifacts from volatile memory (RAM) samples.The Volatility Framework supports Windows, Linux, and MacOS. It is can be downloaded on Github and has version running on Python 2.6 or later and recently not 3.0 called Volatility 3[3].

Several research papers have shown how to analyze a memory image using the volatility framework and this has been of great impact to reserach in memory forensics. For instance, (Cai et al., 2013) depicts the importance of forensics analysis on physical memory and some related software and hardware tools that are used. This work shows in detail, the usage of the *"Dumpit"*[4] software to get the complete memory dump of a computer also the process of analysing the memory dump using the open source forensic analysis tool Volatility. Some research have also used Volatility to perform live memory forensics and analyze the memory image of mobile devices such as Android platform and iOS. In (Macht, 2013),(Chang et al., 2013) the critical artifacts were extracted from Dalvik Virtual Machine Instances that made it possible for writing a universal Volatility plugin which was used to analyze every single process or system class within the android application. The process ID was gotten using a linux plugin named *plugin linux_pslist*. Similarly, with the help of the plugin, the desired memory artifacts where extracted and printed. This whole process entails traversal within the graphical layout of the android image to detect any malware processes.

There have also been recent developments to the the Volatility Framework through building of plugins and proposing a more efficient and user-friendly way to conduct memory forensics. In (Logen et al., 2012), a Graphical User Interface (GUI) was presented which addressed the concern of some forensic investigators not been able to use the command line interface utility. This offered additional utilities such as storage of the forensic results or artifcats gathered in a database as well as shortcuts for long Volatility command sequences and provided new commands which were based on data stored on the database. In (Magnet Forensics, 2020), Subsequently, FTK Imager was used at the end of each major round of testing in the Windows VM to capture its physical disk image. FTK and Comae DumpIt tools were also used to acquire the VM's memory when the application was active and terminated. Finally, Magnet Acquire was used to collect a physical image of the Android and a logical image of the iOS device. It is important to note that even though the iPhone was jailbroken, Magnet Acquire only offered support to acquire a logical image of the device.

3. Apparatus

The hardware and software used to conduct this research are presented on Table A.1 in Appendix A.

[2] https://www.volatilityfoundation.org/

[3] https://github.com/volatilityfoundation/volatility3

[4] https://github.com/thimbleweed/All-In-USB/tree/master/utilities/DumpIt

4. Methodology

Malware analysis and memory forensics for this project was conducted in three phases: scenario creation and setup, memory acquisition, memory and malware analysis. The details of these four phases and results are found in the next two subsections and Sections 5 and 6.

4.1. Setup & Scenario Creation

The memory image of the Remote Access Trojan (RAT) was gotten from a company caled Tek Defense [5]. While trying to download a sample, we were redirected to request access to the Tek Defense company. This malware associated with this Remote Access Trojan is known as *Darkcomet*. The Remote Access Trojan malware sample was then installed on the Windows 10 machine. On the same virtualization environment, we installed an Ubuntu operating system and installed volatility 2.6.1 which runs on Python 2.7.Before installing volatility, the command *sudo python2 get-pip.py* was used to install python 2.7. Next, the command *sudo python2 -m pip install -U setuptools wheel* was used to install the setup tools needed to run the volatility tool. Also, *python2 -m pip install -U distorm3 yara pycrypto pillow openpyxl ujson pytz ipython capstone* command was used to install the dependencies such as *yara*, *distorm3*, *pycrypto*, *openpyxl*, *et-xmlfile*, *simplegeneric* and *scandir*. Furthermore, all the modules needed for volatility was cloned from the Github repository using the command *python2 -m pip install -U git+https://github.com/volatilityfoundation/volatility.git*. The memory image is associated with a Windows 7 operating system. The malware sample was also installed and executed on a sandbox environment (a Windows operating system). The memory image was analyzed using Volatility. The following steps were taken in order to investigate the running processes in the image file. Use of the imageinfo plugin to validate and confirm the operating system of the memory dump under investigation. This was accomplished by using the command: *vol.py -f DarkcometRAT.raw imageinfo* (see Figure 2).

4.2. Identifying Processes in Memory

The list of processes present in the infected host machine as at when the live acquisition was viewed using the *pstree* plugin (see Figure 4). The *pstree* plugin prints all running processes by following the EPROCESS lists. Figure 3 gives a high level overview of the basic process resources and the underlying principles with respect to which components of the operating system are involved during the phase of identifying processes present in memory. Upon listing the running processes, we did a search on the internet regarding all the processes, such as Winlogon.exe, LogonUI.exe, etc., and found these to be legitimate processes. However, there were no records of any legitimate processes with the name "runddl32.exe". This piqued our curiosity and led us to investigate further this suspicious-looking process. Next, the *pstree* plugin was used to validate processes that are running and display their corresponding parent process using the command: *python2 vol.py –profile=Win7SP1x86_23418 -f DarkcometRAT.raw pstree*. (see Appendix B.5) Based on the *pstree* results, it was observed that "runddl32.exe" has a parent process ID "3220" and this appears to be no longer present as we tried searching for the process using the processID. There was no trace of what process initiated the initial lunch of the "runddl32.exe" process. Upon further investigation on the pstree results, we see a process *notepad.exe* and a couple of command-line shells launching under the "runddl32.exe" process. We performed a memory dump of the notepad process using the *memdump* plugin. We also performed a string search for the "runddl32.exe" process. We found the content of the .dmp file having some information about the location where the process was hidden in the hard disk (see Appendix B.6).

4.3. Malware Extraction

After noticing several traces of the "runddl32.exe" process in the memory dump of the notepad process, we seek to create an executable sample of this process in order to confirm the suspicion that we have about the process. We have used the *procdump* plugin to dump the process to an executable file sample. The command used to achieve this is: *python2 vol.py -f DarkcometRAT.raw –profile=Win7SP1x86_23418 procdump -p 1524 -D /Downloads/volatility-master/* The –D flag was used to save the executable file to a local directory on the Ubuntu virtual machine.(see Appendix B.7).

4.4. Malware Research

The executable extracted was uploaded on Virus Total to analyze the file and verify our suspicion and search for malicious content using its antivirus engines. The .exe file was transferred to a Windows 11 workstation. Real-time protection for Microsoft Defender flagged the file as a Backdoor-Remote Access Trojan (RAT) (see Appendix B.9) . This executable program sends login details to an attacker to enable them to take full control of a computer. Virus total gave a comprehensive detail of the executable file with a list of antivirus engines that flagged it as malicious (see Appendix B.10). From our research, we found out that this is a Remote Access Trojan called **DARKCOMET**.

5. Experimental Results and Analysis

After the extracted executable have been fed into Virus Total [6], we conducted more experimental results to identify configuration files and other programs that allows the

[5] http://www.tekdefense.com/downloads/malware-samples

[6] www.virustotal.com

```
stephenforensics@ubuntu:~/Downloads/volatility-master$ sudo python2 -m pip install yara
DEPRECATION: Python 2.7 reached the end of its life on January 1st, 2020. Please upgrade your Python as Python 2.7 is no longer maintained. pip
21.0 will drop support for Python 2.7 in January 2021. More details about Python 2 support in pip can be found at https://pip.pypa.io/en/latest/
development/release-process/#python-2-support pip 21.0 will remove support for this functionality.
Collecting yara
  Downloading yara-1.7.7.tar.gz (387 kB)
     |████████████████████████████████| 387 kB 2.5 MB/s
Building wheels for collected packages: yara
  Building wheel for yara (setup.py) ... done
  Created wheel for yara: filename=yara-1.7.7-py2-none-any.whl size=124252 sha256=321b6585db153746fa84a5dc3f376e145bea3223d45141ef95360699ef4afc
de
  Stored in directory: /root/.cache/pip/wheels/fa/aa/9d/2de77fd090c26066b9457b5b8413e3df3f60ded7d7ed3d184f
Successfully built yara
Installing collected packages: yara
Successfully installed yara-1.7.7
stephenforensics@ubuntu:~/Downloads/volatility-master$ sudo ln -s /usr/local/lib/python2.7/dist-packages/usr/lib/libyara.so /usr/lib/libyara.so
stephenforensics@ubuntu:~/Downloads/volatility-master$ sudo ln -s /usr/local/lib/python2.7/dist-packages/usr/lib/libyara.so /usr/lib/libyara.so
ln: failed to create symbolic link '/usr/lib/libyara.so': File exists
stephenforensics@ubuntu:~/Downloads/volatility-master$ python2 -m pip install -U git+https://github.com/volatilityfoundation/volatility.git
DEPRECATION: Python 2.7 reached the end of its life on January 1st, 2020. Please upgrade your Python as Python 2.7 is no longer maintained. pip
21.0 will drop support for Python 2.7 in January 2021. More details about Python 2 support in pip can be found at https://pip.pypa.io/en/latest/
development/release-process/#python-2-support pip 21.0 will remove support for this functionality.
Defaulting to user installation because normal site-packages is not writeable
Collecting git+https://github.com/volatilityfoundation/volatility.git
  Cloning https://github.com/volatilityfoundation/volatility.git to /tmp/pip-req-build-Q4PogR
  Running command git clone -q https://github.com/volatilityfoundation/volatility.git /tmp/pip-req-build-Q4PogR
Building wheels for collected packages: volatility
  Building wheel for volatility (setup.py) ... done
  Created wheel for volatility: filename=volatility-2.6.1-py2-none-any.whl size=6563372 sha256=d16d9906024385492abc90c8ed6f759667ecf7e5276bcc316
152557777cefafe
  Stored in directory: /tmp/pip-ephem-wheel-cache-bijzvO/wheels/7a/c4/2a/0a32e376b4c5a05335e0659f1633938e1f7ec4b2cd8708b7bc
Successfully built volatility
Installing collected packages: volatility
Successfully installed volatility-2.6.1
```

Figure 1: Successful installation of Volatility 2.6.1 with Python 2.7

```
stephenforensics@ubuntu:~/Downloads/volatility-master$ vol.py -f DarkcometRAT.raw imageinfo
Volatility Foundation Volatility Framework 2.6.1
INFO    : volatility.debug    : Determining profile based on KDBG search...
          Suggested Profile(s) : Win7SP1x86_23418, Win7SP0x86, Win7SP1x86_24000, Win7SP1x86
                     AS Layer1 : IA32PagedMemoryPae (Kernel AS)
                     AS Layer2 : FileAddressSpace (/home/stephenforensics/Downloads/volatility-master/DarkcometRAT.raw)
                      PAE type : PAE
                           DTB : 0x185000L
                          KDBG : 0x82968c28L
          Number of Processors : 1
     Image Type (Service Pack) : 1
                KPCR for CPU 0 : 0x82969c00L
             KUSER_SHARED_DATA : 0xffdf0000L
           Image date and time : 2014-02-03 12:31:36 UTC+0000
     Image local date and time : 2014-02-03 07:31:36 -0500
stephenforensics@ubuntu:~/Downloads/volatility-master$
```

Figure 2: Memory image information displayed using the *imageinfo* plugin.

Darkomet work the way it does. With this in mind, we performed a string search for the keyword "Darkcomet" on the dumped *runndl32.exe* process given as *1524.dmp* using the command: *strings -a 1524.dmp—grep -A 22 "DARKCOMET"*. The results of the string search displayed some configurations pertaining to persistence, command and control, alteration of date metadata. Furthermore, we noticed Network data settings: *NETDATA=test213.no-ip.info:1604* in the configuration file (see Figure Appendix B.11). With this information, we investigated the network connections (TCP UDP) that has been established by the Remote Access Trojan. This was investigated using the command, python2 vol.py -f DarkcometRAT.raw –profile=Win7SP1x86_23418 netscan. From Appendix B.8, it shows the C2 (command and control) functionality of the malware. The victim machine with IP address, 192.168.26.136 connected with a C2 server, 176.106.48.182 on port 1604. This is the same port that was found on NETDATA as well when we performed a string search on the dumped "runddl32" process. Based on this analysis, we have looked up the IP address and the port of the C2 server through which the victim's computer has established a connection. From our findings using the website https://www.speedguide.net/port.php?port=1604[7], it was reported that the Darkcomet Remote Access Trojan uses the port 1604 for its remote access administration (see Appendix B.12. We also looked up the IP address of the C2 server on Virus Total and noticed that it is associated with the *"runddl32.exe"* process that we identified earlier. Virus Total also identified the domain "test213.no-ip.info" associated with the malicious IP address, 176.106.48.182 (see Figure Appendix B.13)

6. Conclusion/Discussion

From our results, we identified the Darkcomet Remote Access Trojan as a malware program that includes a backdoor for remote administrative control over a target computer. From the memory analysis, the compromised host

[7]https://www.speedguide.net/port.php?port=1604

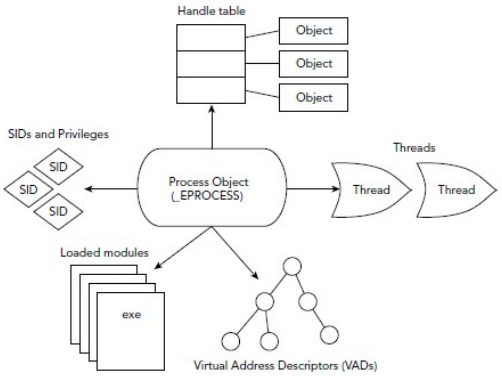


Figure 3: A high level diagram showing basic process resources

```
0x85039810 dllhost.exe            2012    484     13      191      0      0 2014-02-03 09:04:29 UTC+0000
0x858ef030 msdtc.exe               840    484     12      145      0      0 2014-02-03 09:04:31 UTC+0000
0x8594ab18 SearchIndexer.         1712    484     14      680      0      0 2014-02-03 09:04:34 UTC+0000
0x85a00d40 LogonUI.exe            2516    388      5      156      0      0 2014-02-03 09:05:25 UTC+0000
0x850e9870 svchost.exe            2644    484     14      356      0      0 2014-02-03 09:06:23 UTC+0000
0x859636f8 svchost.exe            1248    484      7      109      0      0 2014-02-03 09:08:27 UTC+0000
0x85a68030 taskhost.exe            140    484      9      253      1      0 2014-02-03 12:13:31 UTC+0000
0x84ab64a0 TPAutoConnect.         4044   1688      5      121      1      0 2014-02-03 12:13:32 UTC+0000
0x858e2540 conhost.exe            3916    380      1       33      1      0 2014-02-03 12:13:32 UTC+0000
0x849f29c0 dwm.exe                 340    792      5      129      1      0 2014-02-03 12:13:32 UTC+0000
0x850d56f8 explorer.exe           2052   1808     31      974      1      0 2014-02-03 12:13:32 UTC+0000
0x84ab3428 VMwareTray.exe         4092   2052      5       75      1      0 2014-02-03 12:13:55 UTC+0000
0x84ae5b28 vmtoolsd.exe           2116   2052      5      170      1      0 2014-02-03 12:13:55 UTC+0000
0x859b6630 wuauclt.exe            2280    844      3       88      1      0 2014-02-03 12:14:02 UTC+0000
0x84524030 audiodg.exe            3936    752      4      127      0      0 2014-02-03 12:20:49 UTC+0000
0x85659af0 cmd.exe                3656   3220      1       19      1      0 2014-02-03 12:27:17 UTC+0000
0x84535d40 cmd.exe                1128   3220      1       19      1      0 2014-02-03 12:27:17 UTC+0000
0x85a72188 conhost.exe            2752    380      2       47      1      0 2014-02-03 12:27:17 UTC+0000
0x8459d9a0 conhost.exe            2088    380      2       46      1      0 2014-02-03 12:27:17 UTC+0000
0x84536030 runddl32.exe           1524   3220     10      161      1      0 2014-02-03 12:27:18 UTC+0000
0x84506480 notepad.exe            1896   1524      2       57      1      0 2014-02-03 12:27:18 UTC+0000
0x84a70440 DumpIt.exe             3060   2052      5       38      1      0 2014-02-03 12:31:34 UTC+0000
0x85a04570 conhost.exe            2272    380      2       49      1      0 2014-02-03 12:31:34 UTC+0000
stephenforensics@ubuntu:~/Downloads/volatility-master$
```

Figure 4: List of processes in the memory image with "runddl32.exe" process identified.

system had established a remote connection to a command and control server built by the attacker to distribute RATs to other vulnerable systems within. The behavior of this malware is similar to that of a Botnet. Furthermore, RATs can be difficult to detect because the actions that they perform are similar to those of legitimate programs. Memory forensics plays a significant role in detecting malware having such kind of behavior. Memory forensics made the investigation of this seemingly legitimate malware "DARKCOMET". much more easier and optimal. Acquiring memory is far quicker than imaging an entire file system. Additionally, more cyber attacks are taking place in RAM only and without memory forensics, an investigator would not normally find what they are looking for. Preventive measures against RATs are not limited to keeping antivirus software up to date and refraining from downloading programs or opening attachments that are not from trusted sources. At the network level, it is always good practice to block unused network ports and turn off services not in use, and efficient monitoring of outbound traffic.

7. Acknowledgements

We would like to express our gratitude to Dr. Modhuparna Manna who has been ever supportive and impactful throughout the Introduction to Computer Security class and has also done a great job by introducing us to the several concepts of security. We are also grateful to have gotten the go ahead to work on a memory forensics project. This has really made us to do extensive research and indeed expanded our horizon.

References

Cai, L., Sha, J. and Qian, W. (2013), Study on forensic analysis of physical memory, *in* 'Proc. 2nd International Symposium on Com-

puter, Communication, Control and Automation (3CA 2013)', Citeseer.

Carvey, H. (2014), *Windows forensic analysis toolkit: advanced analysis techniques for Windows 8*, Elsevier.

Case, A. and Richard III, G. G. (2016), 'Detecting objective-c malware through memory forensics', *Digital Investigation* **18**, S3–S10.

Chang, Y.-T., Chung, M.-J., Lee, C.-F., Huang, C.-T. and Wang, S.-J. (2013), Memory forensics for key evidence investigations in case illustrations, *in* '2013 Eighth Asia Joint Conference on Information Security', IEEE, pp. 96–101.

Huseinović, A. and Ribić, S. (2013), Virtual machine memory forensics, *in* '2013 21st Telecommunications Forum Telfor (TELFOR)', IEEE, pp. 940–942.

Logen, S., Höfken, H. and Schuba, M. (2012), Simplifying ram forensics: A gui and extensions for the volatility framework, *in* '2012 Seventh International Conference on Availability, Reliability and Security', IEEE, pp. 620–624.

Macht, H. (2013), 'Live memory forensics on android with volatility', *Friedrich-Alexander University Erlangen-Nuremberg* .

Magnet Forensics (2020), 'Magnet acquire'.
URL: *https://www.magnetforensics.com/resources/magnet-acquire/*

Malin, C. H., Casey, E. and Aquilina, J. M. (2012), *Malware forensics field guide for Windows Systems: Digital forensics field guides*, Elsevier.

Appendix A. Apparatus

Table A.1: Apparatus

Hardware/Software	**Use**	**Company**	**Software Version**
VirtualBox	Hosted Windows Virtual Machine	Oracle VM VirtualBox	6.1.4
Volatility	Desktop Memory Analysis	Volatility Foundation	Volatility 2.6.1 & Volatility 3 1.0.0-beta.1

Appendix B.

Associated Figures

```
stephenforensics@ubuntu:~/Downloads/volatility-master$ python2 vol.py --profile=Win7SP1x86_23418 -f DarkcometRAT.raw pstree
Volatility Foundation Volatility Framework 2.6.1
Name                                                 Pid   PPid   Thds   Hnds Time
-------------------------------------------------- ------ ------ ------ ------ ----
 0x855a7bc0:wininit.exe                              388    328      3     81 2014-02-03 09:03:54 UTC+0000
. 0x85a00d40:LogonUI.exe                            2516    388      5    156 2014-02-03 09:05:25 UTC+0000
. 0x855dc030:services.exe                            484    388      7    199 2014-02-03 09:03:56 UTC+0000
.. 0x84a29d40:spoolsv.exe                           1288    484     13    347 2014-02-03 09:04:14 UTC+0000
.. 0x85047290:vmtoolsd.exe                          1516    484      8    280 2014-02-03 09:04:18 UTC+0000
.. 0x85a68030:taskhost.exe                           140    484      9    253 2014-02-03 12:13:31 UTC+0000
.. 0x85661848:svchost.exe                            792    484     16    367 2014-02-03 09:04:03 UTC+0000
... 0x849f29c0:dwm.exe                               340    792      5    129 2014-02-03 12:13:32 UTC+0000
.. 0x850acb90:TPAutoConnSvc.                        1688    484     10    139 2014-02-03 09:04:23 UTC+0000
... 0x84ab64a0:TPAutoConnect.                       4044   1688      5    121 2014-02-03 12:13:32 UTC+0000
.. 0x850416c8:svchost.exe                           1796    484      6     92 2014-02-03 09:04:27 UTC+0000
.. 0x852c3910:svchost.exe                           1316    484     20    301 2014-02-03 09:04:14 UTC+0000
.. 0x8594ab18:SearchIndexer.                        1712    484     14    680 2014-02-03 09:04:34 UTC+0000
.. 0x85667030:svchost.exe                            820    484     12    543 2014-02-03 09:04:03 UTC+0000
.. 0x858ef030:msdtc.exe                              840    484     12    145 2014-02-03 09:04:31 UTC+0000
.. 0x856214f8:svchost.exe                            588    484     10    354 2014-02-03 09:04:01 UTC+0000
.. 0x856695a0:svchost.exe                            844    484     30   1084 2014-02-03 09:04:03 UTC+0000
... 0x859b6630:wuauclt.exe                          2280    844      3     88 2014-02-03 12:14:02 UTC+0000
.. 0x85039810:dllhost.exe                           2012    484     13    191 2014-02-03 09:04:29 UTC+0000
.. 0x859636f8:svchost.exe                           1248    484      7    109 2014-02-03 09:08:27 UTC+0000
.. 0x85634030:svchost.exe                            664    484      7    270 2014-02-03 09:04:02 UTC+0000
.. 0x85652030:svchost.exe                            752    484     19    476 2014-02-03 09:04:03 UTC+0000
... 0x84524030:audiodg.exe                          3936    752      4    127 2014-02-03 12:20:49 UTC+0000
.. 0x850e9870:svchost.exe                           2644    484     14    356 2014-02-03 09:06:23 UTC+0000
.. 0x84a29030:svchost.exe                           1176    484     15    489 2014-02-03 09:04:11 UTC+0000
. 0x855e0030:lsass.exe                               492    388      6    539 2014-02-03 09:03:57 UTC+0000
. 0x855e2860:lsm.exe                                 500    388     10    147 2014-02-03 09:03:57 UTC+0000
 0x852e74c8:csrss.exe                                336    328      9    394 2014-02-03 09:03:53 UTC+0000
 0x84a21530:winlogon.exe                             424    372      3    118 2014-02-03 09:03:54 UTC+0000
 0x855a4388:csrss.exe                                380    372     10    294 2014-02-03 09:03:54 UTC+0000
. 0x85a04570:conhost.exe                            2272    380      2     49 2014-02-03 12:31:34 UTC+0000
. 0x85a72188:conhost.exe                            2752    380      2     47 2014-02-03 12:27:17 UTC+0000
. 0x858e2540:conhost.exe                            3916    380      1     33 2014-02-03 12:13:32 UTC+0000
. 0x8459d9a0:conhost.exe                            2088    380      2     46 2014-02-03 12:27:17 UTC+0000
 0x85659af0:cmd.exe                                 3656   3220      1     19 2014-02-03 12:27:17 UTC+0000
 0x84536030:runddl32.exe                            1524   3220     10    161 2014-02-03 12:27:18 UTC+0000
. 0x84506480:notepad.exe                            1896   1524      2     57 2014-02-03 12:27:18 UTC+0000
 0x84535d40:cmd.exe                                 1128   3220      1     19 2014-02-03 12:27:17 UTC+0000
 0x850d56f8:explorer.exe                            2052   1808     31    974 2014-02-03 12:13:32 UTC+0000
. 0x84ae5b28:vmtoolsd.exe                           2116   2052      5    170 2014-02-03 12:13:55 UTC+0000
```

Figure B.5: List of processes that are running with corresponding parent processes.

```
stephenforensics@ubuntu:~/Downloads/volatility-master$ python2 vol.py -f DarkcometRAT.raw --profile=Win7SP1x86_23418 memdump --dump-dir=./ -p 1896
Volatility Foundation Volatility Framework 2.6.1
************************************************************************
Writing notepad.exe [  1896] to 1896.dmp
stephenforensics@ubuntu:~/Downloads/volatility-master$ strings -e l ./1896.dmp | grep "runddl32"
runddl32
\device\harddiskvolume1\users\tek defense\appdata\local\temp\msdcsc\runddl32.exe
\Device\HarddiskVolume1\Users\TEKDEF~1\AppData\Local\Temp\MSDCSC\runddl32.exe
\Device\HarddiskVolume1\Users\TEKDEF~1\AppData\Local\Temp\MSDCSC\runddl32.exe
\Device\HarddiskVolume1\Users\TEKDEF~1\AppData\Local\Temp\MSDCSC\runddl32.exe
\device\harddiskvolume1\users\tek defense\appdata\local\temp\msdcsc\runddl32.exe
runddl32.exe
runddl32.exe
C:\Users\TEKDEF~1\AppData\Local\Temp\MSDCSC\runddl32.exe
runddl32.exe
\Device\HarddiskVolume1\Users\Tek Defense\AppData\Local\Temp\MSDCSC\runddl32.exe
runddl32.exe
runddl32.exe
runddl32.exe
runddl32.exe
runddl32.exe
runddl32.exe
runddl32.exe
runddl32.exe
!ek Defense\AppData\Local\Temp\MSDCSC\runddl32.exe
\Users\TEKDEF~1\AppData\Local\Temp\MSDCSC\runddl32.exe
\Device\HarddiskVolume1\Users\Tek Defense\AppData\Local\Temp\MSDCSC\runddl32.exe
C:\Users\TEKDEF~1\AppData\Local\Temp\MSDCSC\runddl32.exe
runddl32.exe
\Users\TEKDEF~1\AppData\Local\Temp\MSDCSC\runddl32.exe
C:\Users\TEKDEF~1\AppData\Local\Temp\MSDCSC\runddl32.exe
\??\C:\Users\TEKDEF~1\AppData\Local\Temp\MSDCSC\runddl32.exe
\Device\HarddiskVolume1\Users\TEKDEF~1\AppData\Local\Temp\MSDCSC\runddl32.exe
runddl32
```

Figure B.6: String search on the memory dump of the notepad process.

```
stephenforensics@ubuntu:~/Downloads/volatility-master$ python2 vol.py -f DarkcometRAT.raw --profile=Win7SP1x86_23418 -p 1524 procdump -D ~/Downloads/volatility-master/
Volatility Foundation Volatility Framework 2.6.1
Process(V) ImageBase  Name                 Result
---------- ---------- -------------------- ------
0x84536030 0x00400000 runddl32.exe         OK: executable.1524.exe
stephenforensics@ubuntu:~/Downloads/volatility-master$
```

Figure B.7: Extraction of the runddl32.exe process from memory as an executable .

```
0x3ed25098      TCPv6    :::49155                :::0                  LISTENING       844     svchost.exe
0x3ee9c588      TCPv4    0.0.0.0:445             0.0.0.0:0             LISTENING       4       System
0x3ee9c588      TCPv6    :::445                  :::0                  LISTENING       4       System
0x3eea50b0      TCPv4    0.0.0.0:49156           0.0.0.0:0             LISTENING       484     services.exe
0x3eea50b0      TCPv6    :::49156                :::0                  LISTENING       484     services.exe
0x3eea7880      TCPv4    0.0.0.0:49156           0.0.0.0:0             LISTENING       484     services.exe
0x3eef1628      TCPv4    192.168.26.136:49744    176.106.48.182:1604   SYN_SENT        -1
0x3f598d50      UDPv4    127.0.0.1:55753         *:*                                   1248    svchost.exe    2014-02-03 09:08:28 UTC+0000
0x3f599488      UDPv4    127.0.0.1:1900          *:*                                   1248    svchost.exe    2014-02-03 09:08:28 UTC+0000
0x3faf01f8      UDPv4    0.0.0.0:59764           *:*                                   1176    svchost.exe    2014-02-03 12:32:17 UTC+0000
0x3fb13868      UDPv4    127.0.0.1:61756         *:*                                   0                      2014-02-03 12:20:48 UTC+0000
0x3fb4ba70      TCPv4    192.168.26.136:49735    176.106.48.182:1604   CLOSED          -1
stephenforensics@ubuntu:~/Downloads/volatility-master$
```

Figure B.8: Netscan results with the infected host machine sending a SYN packet to a public IP address.

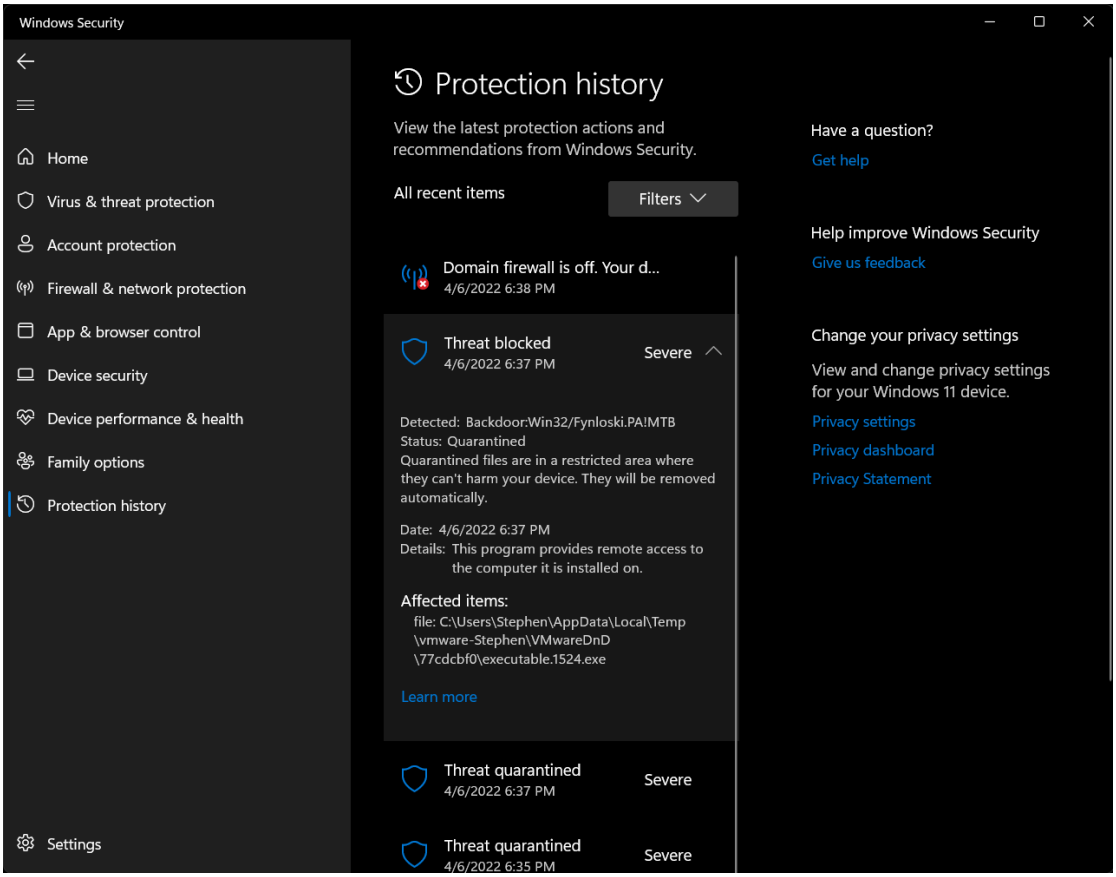


Figure B.9: Malware quarantined by Microsoft Defender when moved to a Windows 11 workstation.

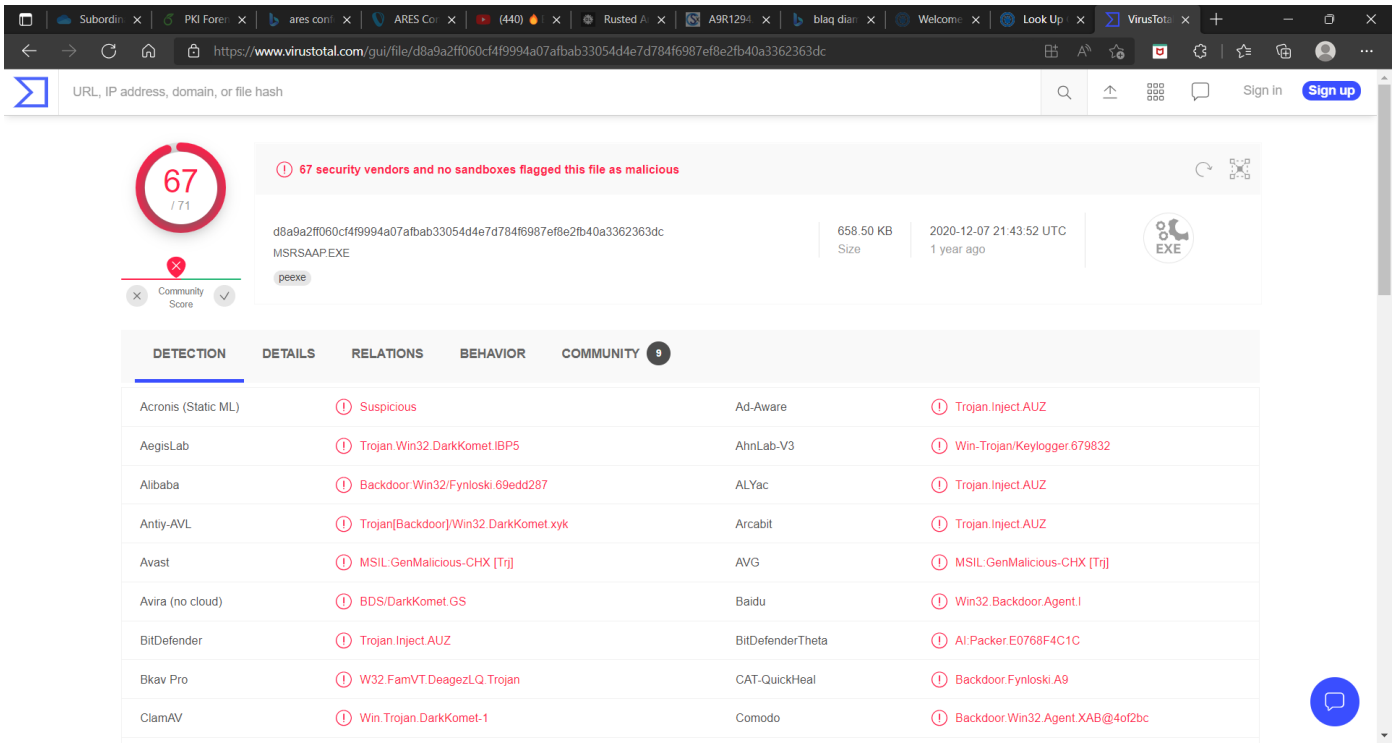
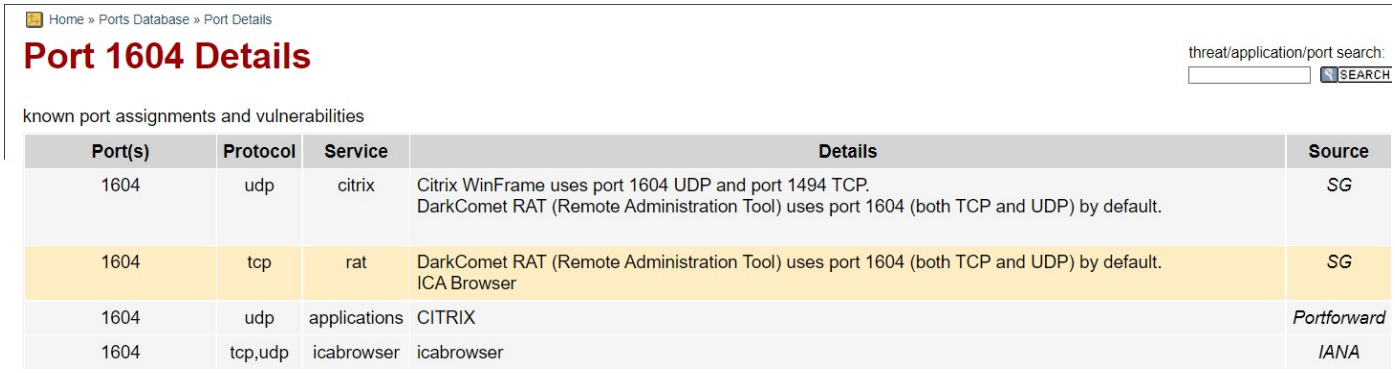


Figure B.10: Virus Total results of the extracted malware executable.

```
stephenforensics@ubuntu:~/Downloads/volatility-master$ strings -a 1524.dmp | grep -A 22 "DARKCOMET"
#BEGIN DARKCOMET DATA --
MUTEX={DC_MUTEX-KHNEW06}
SID={Guest16}
FWB={0}
NETDATA={test213.no-ip.info:1604}
GENCODE={F6FE8i2BxCpu}
INSTALL={1}
COMBOPATH={10}
EDTPATH={MSDCSC\runddl32.exe}
KEYNAME={MicroUpdate}
EDTDATE={16/04/2007}
PERSINST={1}
MELT={0}
CHANGEDATE={1}
DIRATTRIB={6}
FILEATTRIB={6}
SH1={1}
CHIDEF={1}
CHIDED={1}
PERS={1}
OFFLINEK={1}
#EOF DARKCOMET DATA --
```

Figure B.11: Darkcomet configuration file found using string search.



Home » Ports Database » Port Details

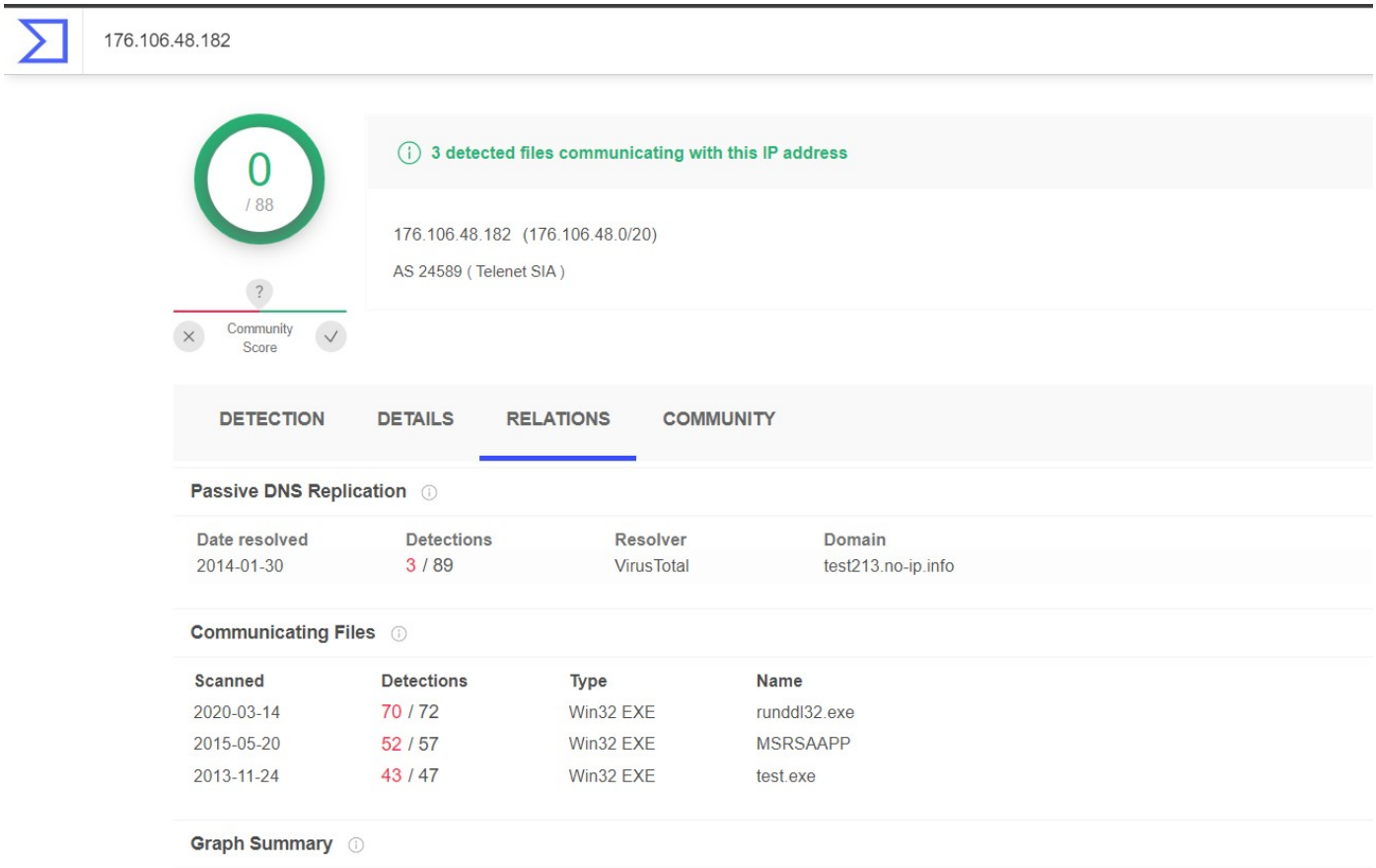
Port 1604 Details

threat/application/port search: SEARCH

known port assignments and vulnerabilities

Port(s)	Protocol	Service	Details	Source
1604	udp	citrix	Citrix WinFrame uses port 1604 UDP and port 1494 TCP. DarkComet RAT (Remote Administration Tool) uses port 1604 (both TCP and UDP) by default.	*SG*
1604	tcp	rat	DarkComet RAT (Remote Administration Tool) uses port 1604 (both TCP and UDP) by default. ICA Browser	*SG*
1604	udp	applications	CITRIX	*Portforward*
1604	tcp,udp	icabrowser	icabrowser	*IANA*

Figure B.12: Lookup of the public IP address on *www.speedguide.net*



176.106.48.182

0 / 88

Community Score

3 detected files communicating with this IP address

176.106.48.182 (176.106.48.0/20)

AS 24589 (Telenet SIA)

DETECTION DETAILS RELATIONS COMMUNITY

Passive DNS Replication

Date resolved	Detections	Resolver	Domain
2014-01-30	3 / 89	VirusTotal	test213.no-ip.info

Communicating Files

Scanned	Detections	Type	Name
2020-03-14	70 / 72	Win32 EXE	runddl32.exe
2015-05-20	52 / 57	Win32 EXE	MSRSAAPP
2013-11-24	43 / 47	Win32 EXE	test.exe

Graph Summary

Figure B.13: Investigation of the IP Address:176.106.48.182 on Virus Total