ARL-TR-9128 ● Dec 2020

DEVCOM ARMY RESEARCH LABORATORY

Hands-on Cybersecurity Studies: Uncovering and Decoding Malware Communications—Initial Analysis with Wireshark and Volatility

by Jaime C Acosta and Daniel E Krych

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-TR-9128 ● DEC 2020

Hands-on Cybersecurity Studies: Uncovering and Decoding Malware Communications—Initial Analysis with Wireshark and Volatility

Jaime C Acosta and Daniel E Krych
Computational and Information Sciences Directorate,
DEVCOM Army Research Laboratory

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
December 2020	Technical Report	May 2019–March 2020

4. TITLE AND SUBTITLE
Hands-on Cybersecurity Studies: Uncovering and Decoding Malware Communications—Initial Analysis with Wireshark and Volatility

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Jaime C Acosta and Daniel E Krych

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
DEVCOM Army Research Laboratory
ATTN: FCDD-RLC-ND
Adelphi, MD 20783-1138

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-9128

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
ORCID ID: Jaime C Acosta, 0000-0003-2555-9989

14. ABSTRACT

This report presents the first of three hands-on software reverse-engineering exercises with the ultimate objective of learning how a particular malware (malicious software) is communicating across a network, and developing software to detect and reveal these communications in plaintext, in vivo. Remote access trojans (RATs) are a type of malware that persist on the infected machine after compromise and provide the malicious actor in control of the malware with remote access to the infected machine via established command-and-control channels. RATs are typically spread through phishing emails or websites where the software is downloaded without the user knowing; it can also spread by taking advantage of vulnerabilities in software running on the victim's devices. This report details the first of three software reverse-engineering exercises, which can be completed cumulatively or individually as each accomplishes a specific task and builds off the previous exercise. This exercise entails identifying and extracting malware that will be used in subsequent exercises. The effects and communications of RATs are demonstrated, and participants are guided through a series of steps leveraging the open-source Wireshark tool to analyze suspicious network traffic and the Volatility tool to pinpoint and extract malicious files within a previously captured memory image.

15. SUBJECT TERMS
Wireshark, Volatility, traffic analysis, memory analysis, decode, software reverse engineering, trojan, remote access trojan, RAT, malware, command-and-control, C2, hands-on cybersecurity, Cybersecurity Rapid Innovation Group, CyberRIG

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Jaime C Acosta
Unclassified	Unclassified	Unclassified	UU	23	19b. TELEPHONE NUMBER (Include area code) (575) 993-2375

Contents

List of Figures

1. Introduction

This report presents the first of three hands-on exercises on basic software reverse engineering with the ultimate objective of learning the way that a particular malware (malicious software) is communicating across a network, and developing software to detect and reveal these communications in plaintext, in vivo.

Remote access trojans (RATs) are a type of malware that persist on the infected machine ("Bot") after compromise and provide the malicious actor in control of the malware with remote access to the infected machine via established command-and-control (C2) channels. As with all malware, RATs are typically spread through phishing emails or websites where the software is downloaded without the user knowing; it can also spread by taking advantage of vulnerabilities in software running on the victim's devices. This report details the first of three software reverse-engineering exercises, which can be completed cumulatively or individually as each accomplishes a specific task and builds off the previous exercise. These exercises and their reports demonstrate the effects and communications of RATs and guide participants through a series of steps to uncover, analyze, and develop software to detect the malware.[1,2]

In this first exercise, Wireshark, a network protocol analyzer, is used to analyze the malware traffic between the C2 server and the Bot, and Volatility, a memory forensics tool, is used to analyze an image of the infected hard drive (memory dump) and find and extract the malicious program (binary).[3,4] In the second exercise, Ghidra, the National Security Agency's open-source software reverse-engineering framework, is used to reverse engineer the communications between the C2 server and the Bot by analyzing the malicious binary.[5] In the third exercise, the US Army Combat Capabilities Development Command (DEVCOM) Army Research Laboratory's (ARL's) open-source network forensic analysis framework, Dshell, is used to develop a "Dshell plugin" or "Dshell decoder", to decode the RAT malware communications, which will enable detection and support mitigation.[6]

2. Setup and Configuration

The reverse-engineering hands-on exercises consist of three virtual machines (VMs): one is used as the C2 server, one is used as the infected machine (Bot), and one is used as the Analysis VM, which is placed in between the C2 and Bot machines with a promiscuous port, allowing it to see all traffic between the C2 and Bot machines. This setup is seen in Section 5. Participants use the Analysis VM throughout these exercises to analyze malware traffic between the machines, extract

the malware from the hard disk and analyze the memory dump, reverse engineer the communications by analyzing the malware binary and, finally, develop software to detect and reveal these communications in plaintext.

The setup configuration consists of the following software elements:

- VirtualBox[7] (Version 6.0)
- Two Windows 7 Home Basic 32-bit VMs[8]
- One Ubuntu 18.04 LTS Linux 64-bit VM[9]
- Wireshark[3] (Version 3.0)
- Volatility[4] (Version 2.1)
- Ghidra[5] (Version 9.1.2)
- Dshell[6] (Python 2)
- ArchDeus (a set of scripts that mimic communications notional to RATs)

The Analysis VM was set up as an Ubuntu Linux machine with Wireshark, Volatility, Ghidra, and Dshell installed. The C2 machine was set up with scripts to communicate commands to the Bot machine. A promiscuous port was set up to allow the Analysis VM to view all of the traffic between the C2 and Bot machines.

The entire exercise runs on the DEVCOM Army Research Laboratory South Cybersecurity Rapid Innovation Group (CyberRIG) Collaborative Innovation Testbed, which provides an isolated environment, ensuring that all the environmental artifacts are segregated from any real systems. Participants access the Analysis VM via a web-based interface to allow any system with a web browser to be used, and to isolate the exercise and its contents from the participants' machines.

3. Learning Objectives

This exercise teaches participants the following:

- Participants gain a better understating of how RATs work, which entails malicious actors using C2 channels to remotely control infected machines. The effects of the malware on the sandbox environment should emphasize the importance of securing computer systems and detecting and preventing malware.

- Participants gain experience in collecting and analyzing network traffic with Wireshark, along with many of its features including filtering, sorting, and session isolation.
- Participants learn about malware communication and C2, especially the client/server common infrastructure used by RAT malware. This understanding is critical in the next stages of analysis that involve interpreting the communication protocols, payload obfuscation, and real-world mechanisms that are commonly used to defend against such malware.
- Participants learn about memory images, the reasoning behind their widespread use, and how they can be analyzed to gain an upper hand against both simple and advanced malware. The Volatility tool is used to demonstrate the processes involved in identifying and extracting malware from an infected system memory dump.

4. Methodology

In creating these software reverse-engineering exercises, we started with the desire to provide a hands-on learning approach to development of a Dshell malware decoder. By leveraging multiple analysis tools and techniques to analyze the malware and its communications, we provide a way for participants to learn and practice forensic techniques and gain enough knowledge about the malicious binary and its actions to understand its inner workings and develop a uniquely crafted Dshell decoder, which enables detection and supports mitigation.

While developing these exercises, we intended for them to be educational for participants with varying levels of experience in network security, forensics, and programming. We chose to create scripts that mimic C2 communications notional to RATs and leverage a simple rotational cipher to encrypt the data, which provide an approachable, but still educational, malware binary to uncover, analyze, and decode. Novices will learn a breadth of knowledge and be able to step through the exercise instructions, understand the overall scenario, and develop the basic Dshell decoder. Experts will move more quickly through the exercises, but still learn new tools and analysis techniques, and can aim for going above and beyond in analysis, forensics, practicing using these tools, and developing an efficient and effective Dshell decoder. We purposely used simple encryption and code so all participants can follow along, but more versed participants will notice the applicability of these same techniques for analyzing and decoding highly complex, encrypted, and obfuscated malware. This exercise could be modified to use a more complex encryption method or malware to increase its difficulty and provide more advanced educational material.

In creating this exercise, we began by prepackaging a VM with Ubuntu 18.04 LTS Linux 64-bit along with the analysis tools required for the exercises. Wireshark is a well-known open-source tool for network analysis. It can read live and stored network traffic data and it comes with over 900 dissectors, or parsers, for network protocols.[3] Currently, Wireshark is available through Ubuntu's default apt sources repository; it is installed using the following command:

sudo apt-get install wireshark

After installation, a user may invoke Wireshark through the terminal or through the shortcut icon on the Ubuntu graphical interface. It is worth noting that, by default, Wireshark will not be allowed to capture data on network interface cards in live mode. A user may either provide suitable permissions to the user invoking the tool, or run Wireshark using **sudo**.

Volatility is a memory analysis software tool that has several features for understanding the state of a system at a given point in time; that is, when a memory dump was captured. This open-source software was first released at Black Hat DC in 2007 and is widely used across the globe for fault analysis, malware analysis, and general system forensics.[4] Installing Volatility on the Ubuntu VM was also completed using the following command:

sudo apt-get install volatility

Volatility is invoked through a terminal window using the command **vol.py** followed by several arguments, depending on the intentions of the user. It is worth noting that Volatility does not store any information about the memory image in memory or otherwise. That is, every time it is instantiated, it will analyze the memory image independently of any other invocations. It is up to the user to store this information; this is one of the learning points of the exercise.

In the exercise described in this document, Wireshark provides the capabilities required to identify that there is malicious activity—asking users to identify anomalous network traffic and then extrapolate information that can be used to understand a similar malicious binary that has been frozen in the form of a memory image. Volatility is then used, in combination with the information gained during the Wireshark analysis, to further understand the malware's infection mechanism, which is known as dynamic-link library (DLL) injection. Finally, participants will extract the malicious logic in preparation for the subsequent exercise using Ghidra.

5. Exercise

5.1 Mission Briefing

The briefing for the software reverse-engineering exercise is as follows:

You are a member of the special task force Weltall-42. You have been charged with uncovering a new and deadly RAT known as ArchDeus. Your team was able to 1) recover an infected hard drive with the infection and 2) place yourselves between a malware "command and control" (C2) and a "Bot" (or victim machine) as seen in Fig. 1.

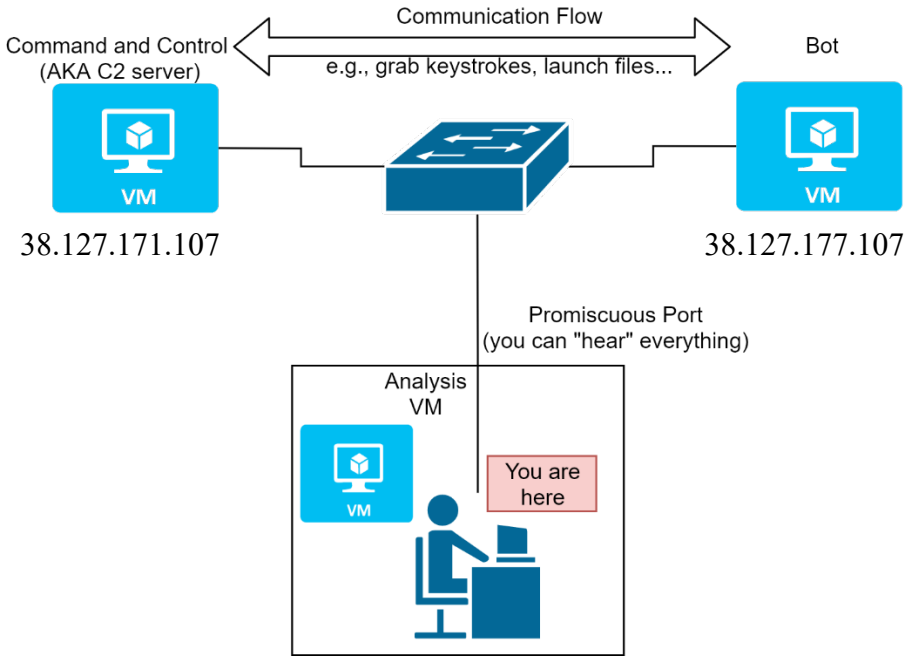


Fig. 1 Reverse-engineering exercise network scenario

The overall software reverse-engineering exercise series is separated into three main exercises, which build off each other to determine key information but can also be stand-alone exercises. This report covers the first exercise.

1) a. Analyze malware traffic between the C2 server and the Bot using a tool called Wireshark.

   b. Extract malware from a hard disk by analyzing the hard drive and pulling out the infected process using a tool called Volatility.

2) Reverse engineer the communications between the C2 server and Bot by analyzing the malware with a tool called Ghidra.

3) Develop a decoder for the malware traffic (Detection and Decoding Mechanism) using a tool called Dshell.

This exercise requires about 1.5–2 h to complete.

5.2 The Awakening: Analyze Malware Traffic

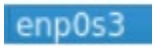
Observe malicious communication between a C2 server and a Bot.

1) Start by finding the malware communication between the control server and the Bot. Start a terminal window by pressing the following keys on the keyboard:
   Ctrl+Alt+t

2) Start Wireshark by running the following command:

   sudo wireshark

   (Enter the password **toor** when prompted.)

3) When Wireshark starts, double-click on the following label:
   

You will notice there is too much traffic and it is difficult to know which traffic is malicious. Let us isolate the traffic so that we only see packets going from the **Bot** to the **C2 server** address.

4) Look at Fig. 1 and you will notice that your team gave you some IP addresses. Write down the address of the **C2 server** and the **Bot** here:

   a. Bot IPv4 address: ___.___.___.___

   b. C2 server IPv4 address: ___.___.___.___

5) Now return to your Wireshark window and hide all packets that have these addresses by entering the following into the filter entry bar:
   

   (don't include the < or >)
   ip.src==**<your answer to question 4)a>** and ip.dst==**<your answer to question 4)b>**

****

This malware uses a stateful connection, meaning that it creates a connection using an initial "handshake" and controls the flow through acknowledgment packets (kind of like certified mail; with proof of receipt).

Every packet has **flags** or indicators to signify their purpose. Example flags include SYN (synchronize), ACK (acknowledgment), and PSH (push). Data is usually sent in packets with the **[PSH, ACK]** flags. An active connection will have packets with **[PSH, ACK]** flags set flowing across a network.

****

6) Look at the traffic in Wireshark and look for packets that have the [PSH, ACK] flags to fill in the blanks: (*Hint: look for the flags in the **Info Column***)
    a. Transport ***Protocol*** used for the communication (*Hint: three letters, **Protocol Column***)
    _____ (write these in lowercase)
    b. ***Destination Port*** used by the **Bot**
    (*Hint: Dst Port*)
    _____

7) Now let us isolate the malicious communication channel by entering a filter in Wireshark. Add a filter to hide all traffic except the packets that have our addresses **and destination port** by entering the following into the filter entry bar:

(do not include the < or >)
ip.src==**<your answer to 4)a>** and ip.dst==**<your answer to 4)b>** and **<your answer to 6)a>**.port==**<your answer to 6)b>**

8) Before we move on, let us sum up our findings so far. Fill in the blanks to describe the **malicious communication** (some of these are repeats):
    a. Transport ***Protocol*** used by the malware
    _____ (write these in lowercase)
    b. ***Destination IP Address*** and ***Destination Port*** used by the **Bot**
    i. Address: ___.___.___.___ *ii. Port:* _____
    c. **Number** of packets sent before longer pause (*Hint: look through several sequences; only look at packets with [PSH, ACK]*)
    _____
    d. **Length** of the longer pause (*Hint: timestamps*)
    _____

9) Select any packet that contains a ***[PSH, ACK]***. Now select the ***Data*** label in the middle panel as shown in Fig. 2.

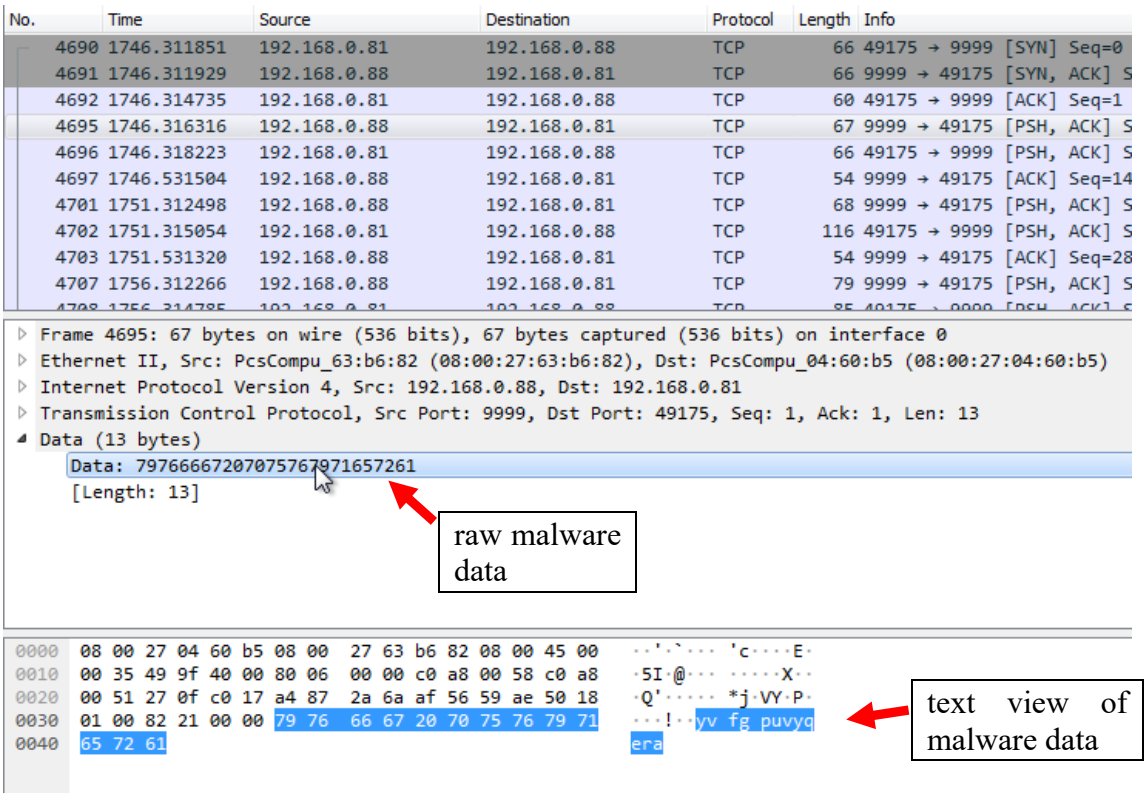


Fig. 2 Traffic view of Wireshark capture

The other parts of the packet (Frame, Ethernet II, Internet Protocol Version 4, etc.) contain information that the operating system uses to understand the packet. The data section is where the program (in this case the malware RAT) is packaging its encoded data.

10) Copy down the set of characters that appear in the ***text view*** of the data packet (as shown in Fig. 2) that you have selected. You will use this in the later part of the exercise:

_____

Great job! You have confirmed that you can view the malicious traffic. Now you will look at the hard disk image that your team collected to obtain a copy of the malware that will help you to learn how to decode the communication.

Clear the desktop and close all your windows.

5.3 The One Who Bares Fangs: Extract Malware from a Hard Disk

Your group was able to find a machine infected with the same malware that you observed in Section 5.2. You have been provided with a frozen image of the memory (a memory dump). Your job is to analyze the memory dump and find the infected program.

1) Start a new terminal by pressing **Ctrl+Alt+t**
2) Now you will setup your environment by creating a series of folders to hold information that you will extract from the infected image. Run the following commands:

   cd Desktop/

   mkdir vol_analysis

3) Next, you will obtain information from the infected image so that Volatility can be used to further dissect the data. Type the following command to obtain the image's operating system information.

   vol.py -f victim.img imageinfo

4) What is the likely profile name of this image? (*Hint: it is the first one.)*

5) Execute the following command to set the profile that Volatility will be using.

   export VOLATILITY_PROFILE=<answer to 4)>

****

Volatility has several functions that are useful for extracting information about the memory dump. This includes processes, threads, loaded libraries, network connections and several others.[4] Many of these functions are listed in the Volatility Cheat Sheet on the Desktop. Make sure to use it!

****

6) Collect data by writing Volatility output to files in your "vol_analysis" folder. For example, run the following command to extract the system registry and store it into a file called registry_hives.txt

   vol.py -f victim.img hivelist > vol_analysis/registry_hives.txt

   This redirects output to a file

7) View the content by running the **geany** text editor as follows (**vim** is also installed).

   geany vol_analysis/registry_hives.txt
   You can also navigate to the folder using a graphical file navigator (by clicking on ) and then opening the file with the default text editor by double-clicking.

8) Using what you have learned and the **Volatility Cheat Sheet** on the Desktop, extract this information to files (choose any names and write them here). Also, look through the files and give a short description of what the information provided.
   a. **Processes listing** *(Hint: use the pslist command)*
      Your Filename: _____
      Description: _____
   b. **DLL listing** *(Hint: use the dlllist command)*
      Your Filename: _____
      Description: _____
   c. **Network information** *(Hint: use the netscan command)*
      Your Filename: _____
      Description: _____
   d. **Used files listing** *(Hint: use the filescan command)*
      Your Filename: _____
      Description: _____

****

We know that the malware has a communication component—it communicated with the server. Our first step in our investigation is to figure out which process was communicating through the network.

****

9) Open your network_info file (your answer to 8)c above) and observe the contents. List the **Foreign Address** and process ID (**PID)** for the **ESTABLISHED** connection that is **NOT** 0.0.0.0 or 127.0.0.1 (these would be connections from the machine to itself). Also, do not include the "port number" (the colon or the number after it).

   Foreign Address and associated PID:

   _____

****

As you noticed, the PID for the associated connection is –1, which means it was not successfully retrieved. Keep in mind that Volatility relies on "profiles" to define the structure of the memory image, and these are not always complete or accurate.

****

10) Identify the five processes (two active, the others not active) that were run through the ***command prompt*** before the memory image was generated. Write the process names here: (*Hint: use the* ***consoles*** *command*)
   Active Process: _____ PID: _____
   Console Process: _____
   Other Process: _____
   Other Process: _____
   Other Process: _____

****

YARA is a widely used tool for pattern matching in files (it is called "the pattern-matching Swiss knife for malware researchers").[10] Volatility has built-in support for this tool. We will use it to look for simple strings, but it has many more capabilities.

****

11) Let us figure out what process had an Established Connection with the Foreign Address that you found in 9); (your answer to 9) should be one address, if you have more, check your answer again). You will do this by using the YARA pattern-matching tool. You should get two results. List the process name resulting from your search. (*Hint: The following is a sample search command that looks for the string: feifong*
   vol.py -f victim.img yarascan -Y "feifong"
   [*Note: this will not return any results in your image*])

   Process name associated with string: (*Hint: both results should have the same exe file*)
   _____

****

Release your ID!

This is somewhat unexpected. You may have thought that the Active Processes you found would have the IP addresses embedded in their code, but this is not the case. It turns out that the processes are using a Domain Name Server to resolve names to IP Addresses (e.g., the name google.com resolves to the IP address 172.217.5.206).

So, now we need to find what name was used to contact the Foreign Address.

****

12) Go back to the output of the strings search from your answer to 11), which should resemble Fig. 3, and write down the **names** that appear after the Foreign Address. (*Hint 1: look closely at "1" vs "L"*) (*Hint 2: the names both end with .com*)

_____

_____

```
Owner: Process svchost.exe Pid 1124
0x00d23f83  37 37 2e 36 36 2e 35 35 2e 34 34 09 67 6f 6f 67   77.66.55.44.goog
0x00d23f93  31 65 2e 63 6f 6d 0a 35 34 2e 33 32 2e 31 32 2e   1e.com.54.32.12.
0x00d23fa3  33 34 09 61 72 6c 2d 73 6f 75 74 68 2e 63 6f 6d   34.arl-south.com
0x00d23fb3  0a 0a 2e 31 32 2e 33 34 09 61 72 6c 2d 73 6f 75   ...12.34.arl-sou
0x00d23fc3  74 68 2e 63 6f 6d 0d 0a 0d 0a 00 00 00 00 00 00   th.com..........
0x00d23fd3  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x00d23fe3  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x00d23ff3  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x00d24003  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x00d24013  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x00d24023  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x00d24033  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x00d24043  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x00d24053  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x00d24063  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x00d24073  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
Rule: r1
Owner: Process svchost.exe Pid 3388
0x1053ff83  37 37 2e 36 36 2e 35 35 2e 34 34 09 67 6f 6f 67   77.66.55.44.goog
0x1053ff93  31 65 2e 63 6f 6d 0a 35 34 2e 33 32 2e 31 32 2e   1e.com.54.32.12.
0x1053ffa3  33 34 09 61 72 6c 2d 73 6f 75 74 68 2e 63 6f 6d   34.arl-south.com
0x1053ffb3  0a 0a 2e 31 32 2e 33 34 09 61 72 6c 2d 73 6f 75   ...12.34.arl-sou
0x1053ffc3  74 68 2e 63 6f 6d 0d 0a 0d 0a 00 00 00 00 00 00   th.com..........
```

Fig. 3 Volatility YARA scan results

13) Now using what you have learned, conduct a **string search** (add the "**-p** option" to specify a particular process) to verify that your Active Process from question 10) is the culprit (had the established connection), as shown in Fig. 4. Also note down the **Memory Address** on the leftmost column (it should start with 0x6f54…).
    a. Infected Process name: _____
    b. PID: _____
    c. Memory Address: _____

```
student@student-VirtualBox:~/Desktop/volatility$ vol.py -f victim.img yarascan -Y "google.com" -p
 3144
Volatility Foundation Volatility Framework 2.6
Rule: r1
Owner: Process rockbot.exe Pid 3144
0x6f540da0  67 6f 6f 67 31 65 2e 63 6f 6d 00 00 00 00 00 00   google.com......
0x6f540db0  67 65 74 61 64 64 72 69 6e 66 6f 20 66 61 69 6c   getaddrinfo.fail
0x6f540dc0  65 64 20 77 69 74 68 20 65 72 72 6f 72 3a 20 25   ed.with.error:.%
0x6f540dd0  64 0a 00 00 00 00 00 00 00 00 00 00 73 6f 63 6b   d...........sock
0x6f540de0  65 74 20 66 61 69 6c 65 64 20 77 69 74 68 20 65   et.failed.with.e
0x6f540df0  72 72 6f 72 3a 20 25 6c 64 0a 00 00 00 00 00 00   rror:.%ld.......
0x6f540e00  00 00 00 00 55 6e 61 62 6c 65 20 74 6f 20 63 6f   ....Unable.to.co
0x6f540e10  6e 6e 65 63 74 20 74 6f 20 73 65 72 76 65 72 21   nnect.to.server!
0x6f540e20  0a 00 00 00 00 00 00 00 42 79 74 65 73 20 72 65   ........Bytes.re
0x6f540e30  63 65 69 76 65 64 3a 20 25 64 0a 00 00 00 00 00   ceived:.%d......
0x6f540e40  43 6f 6e 6e 65 63 74 69 6f 6e 20 63 6c 6f 73 65   Connection.close
0x6f540e50  64 0a 00 00 00 00 00 00 72 65 63 76 20 66 61 69   d.......recv.fai
0x6f540e60  6c 65 64 20 77 69 74 68 20 65 72 72 6f 72 3a 20   led.with.error:.
0x6f540e70  25 64 0a 00 00 00 00 00 00 00 00 00 73 65 6e 64   %d..........send
0x6f540e80  20 66 61 69 6c 65 64 20 77 69 74 68 20 65 72 72   .failed.with.err
0x6f540e90  6f 72 3a 20 25 64 0a 00 00 00 00 00 00 00 00 00   or:.%d..........
```

Fig. 4 Volatility YARA scan process search results

****

A DLL is a library with collections of functions. Programs import these libraries at runtime. In this case, a malicious DLL was loaded into a process. Now you will extract this DLL and analyze it.

****

14) Open the dll_listing file that you created in question 8)b. Look at the ***base*** and the ***size*** columns of the infected file to figure out which DLL is using the memory **range** that includes the Memory Address from your answer to question 13)c.

    DLL Name: _____
    Base: 0x_____

15) Make a new directory and dump the DLL so that we can analyze it later using the following commands: (*Note: do not include the < and >)*

    mkdir vol_analysis/dlls
    vol.py -f victim.img dlldump -p <answer to question 13.b>
    --base=<base from question 14> --dump-dir vol_analysis/dlls

16) Move the DLL of interest to the Desktop.

    mv vol_analysis/dlls/* ~/Desktop

Excellent! You have found and extracted the malicious binary that infected the hard disk!

6. Conclusion

After completing this exercise, participants should have a better understanding of how RATs work, specifically related to the way they communicate and how they infect victim computers. This exercise and the related reverse-engineering exercises have been, and will be, shared with collaborators and partners (including professionals, faculty, and students) to help establish a common ground for studying, researching, and learning about malware, analysis tools, and analysis techniques to harden systems and develop ways to recover after compromise and detect and protect systems moving forward.

We received positive feedback on these software reverse-engineering exercises from both cybersecurity novices and experts.

Future exercises could entail a broader scope of the many tools used throughout these exercises including Wireshark, Volatility, Ghidra, and Dshell, and how to fully leverage the Dshell framework, such as by conducting advanced network forensics and analysis on a network for threat hunting, learning how to modify and customize existing decoders to fit an end-user's analysis needs, learning how an existing decoder was developed to decode a specific network protocol, or developing a decoder to detect and decode real, complex network protocols found in the wild today. Additionally, ARL developed and open-sourced a new and improved version of Dshell written in Python 3, which was publicly released on GitHub[6] in September 2020, and could be used in future exercises.

We hope the information found herein will enlighten students, researchers, and practitioners in the cybersecurity field with new analysis tools and techniques, and help spawn ideas to better their security posture and develop new and unique Dshell plugins/decoders.

7. References

1. Acosta JC, Krych DE. Hands-on cybersecurity studies: uncovering and decoding malware communications—malware analysis with Ghidra. DEVCOM Army Research Laboratory (US); 2020 Dec. Report No.: ARL-TR-9129.
2. Krych DE, Acosta JC. Hands-on cybersecurity studies: uncovering and decoding malware communications with Dshell. DEVCOM Army Research Laboratory (US); 2020 June. Report No.: ARL-TR-8986.
3. Wireshark [accessed 2020 Nov]. https://www.wireshark.org.
4. Volatility [accessed 2020 Nov]. https://www.volatilityfoundation.org.
5. Ghidra [accessed 2020 Nov]. https://ghidra-sre.org.
6. Dshell [accessed 2020 Nov]. https://github.com/USArmyResearchLab/Dshell.
7. VirtualBox [accessed 2020 Nov]. https://www.virtualbox.org/.
8. Microsoft Windows 7 [accessed 2020 Mar]. https://www.microsoft.com/en-us/software-download/windows7.
9. Ubuntu 18.04 [accessed 2020 Nov]. https://releases.ubuntu.com/18.04.4/.
10. YARA [accessed 2020 Dec]. https://virustotal.github.io/yara/.

List of Symbols, Abbreviations, and Acronyms

ACK	acknowledgment
ARL	Army Research Laboratory
Bot	infected machine
C2	command and control
CyberRIG	Cybersecurity Rapid Innovation Group
DEVCOM	US Army Combat Capabilities Development Command
DLL	dynamic-link library
ID	identification
IP	Internet Protocol
PID	process ID
PSH	push
RAT	remote access trojan
SYN	synchronize
VM	virtual machine

1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
1 (PDF)	DEVCOM ARL FCDD RLD CL TECH LIB
2 (PDF)	DEVCOM ARL FCDD RLC ND J ACOSTA D KRYCH