ARL-TR-9129 ● Dec 2020

DEVCOM ARMY RESEARCH LABORATORY

Hands-on Cybersecurity Studies: Uncovering and Decoding Malware Communications—Malware Analysis with Ghidra

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-9129 ● DEC 2020

Hands-on Cybersecurity Studies: Uncovering and Decoding Malware Communications—Malware Analysis with Ghidra

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—Malware Analysis with Ghidra

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-9129

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 second of three hands-on exercises on basic software reverse engineering with the ultimate objective of learning the way 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. 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 second 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. The previous exercise identified and extracted malware using the open-source software tools Wireshark and Volatility. Effects and communications of RATs are demonstrated, and participants are guided through a series of steps focused on analyzing this extracted malware, an infection file, using the National Security Agency's Ghidra binary analysis software.

15. SUBJECT TERMS
Ghidra, binary 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	22	19b. TELEPHONE NUMBER (Include area code) (575) 993-2375

Contents

List of Figures

1. Introduction

This report presents the second 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 second 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 the first exercise, Wireshark, a network protocol analyzer, is used to analyze the malware traffic between the C2 server and the infected machine (Bot), then 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 this second exercise, Ghidra, the National Security Agency's (NSA) 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 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 software reverse-engineering basics, including the ability to use a disassembler to convert binary code into

human-readable instructions; specifically, in this case, the Intel x86 instruction set architecture. While there is less emphasis on the assembly in this exercise, there are many steps in the exercise that provide additional information for more advanced participants, including the reasoning behind the x86 stack and how it works.

- Participants learn about the Microsoft Portable Executable format, and how executables and dynamic-link libraries are wrapped in this file format.
- Participants learn how to use Ghidra and several of its features, including its decompiler, automated binary analysis, textual and graphical view modes, and how to find and cross-reference strings in binaries. An understanding of these capabilities and their usage provides the knowledge needed to decompose and then reconstruct simple communication schemes used by malware samples.

4. Methodology

In creating these software reverse-engineering exercises, we started with the desire to provide a hands-on learning approach to the 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 in 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 creating a VM installed with the Ubuntu 18.04 LTS Linux 64-bit operating system. Subsequently, we installed Java, specifically OpenJDK, using the standard Ubuntu apt repositories using the following command:

sudo apt-get install default-jdk

Afterward, we installed the NSA's Ghidra software reverse-engineering tool and its dependencies using the binary installation package provided on their Ghidra website.[5] Lastly, we created a shortcut script in the user's home directory called ghidraRun that would instantiate the software.

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.

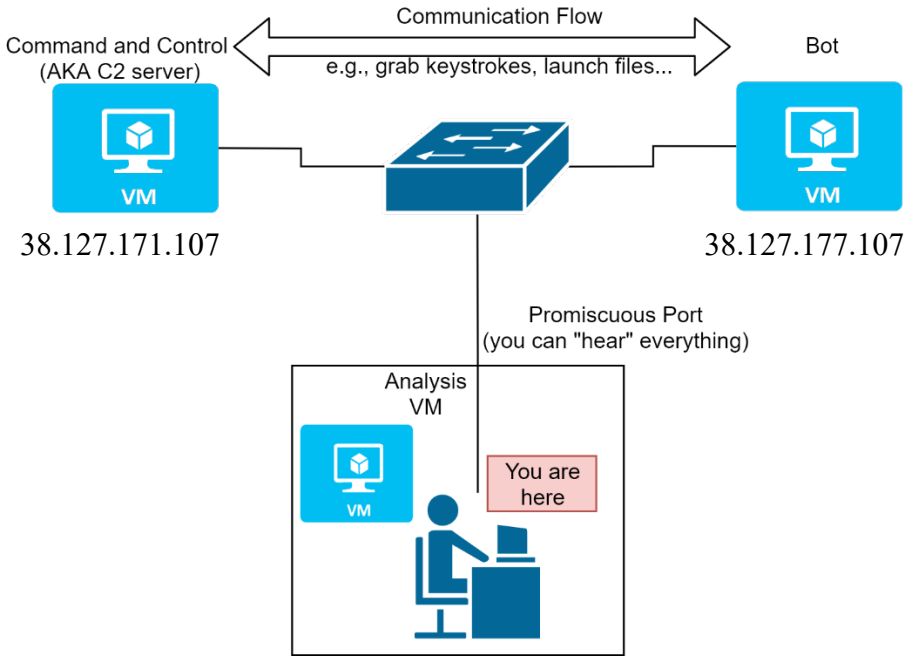

Fig. 1 Reverse-engineering exercise network scenario

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 second 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 Gear Up: Reverse Engineer Malware Communications

You must now reverse engineer the binary you found to learn the encoding used in the communication.

1) Start the Ghidra binary analysis tool by opening a terminal and executing the following command in the terminal:

   ghidraRun

   After the splash screen and tip screen, you should see the window in Fig. 2.

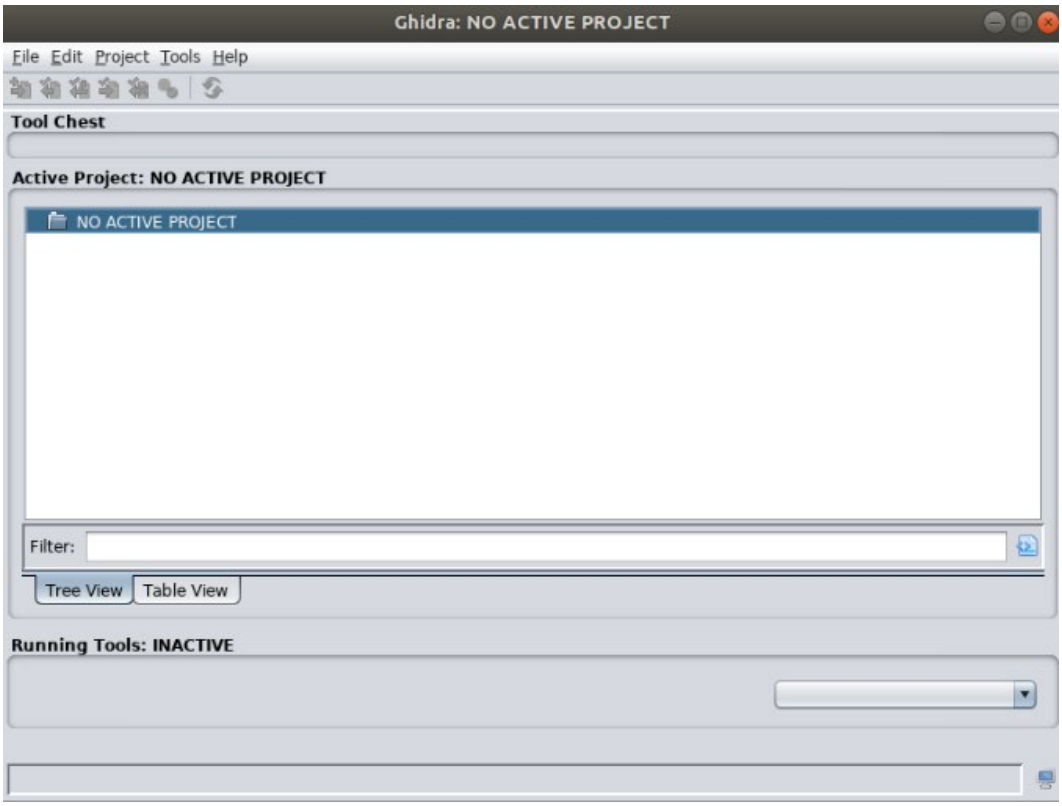

Fig. 2 Project window

2) Click on ***File***-> ***New Project***, select ***Non-Shared Project***, and press ***Next***.

   Name the project **archdeus** and press the ***Finish*** button as shown in Fig. 3.

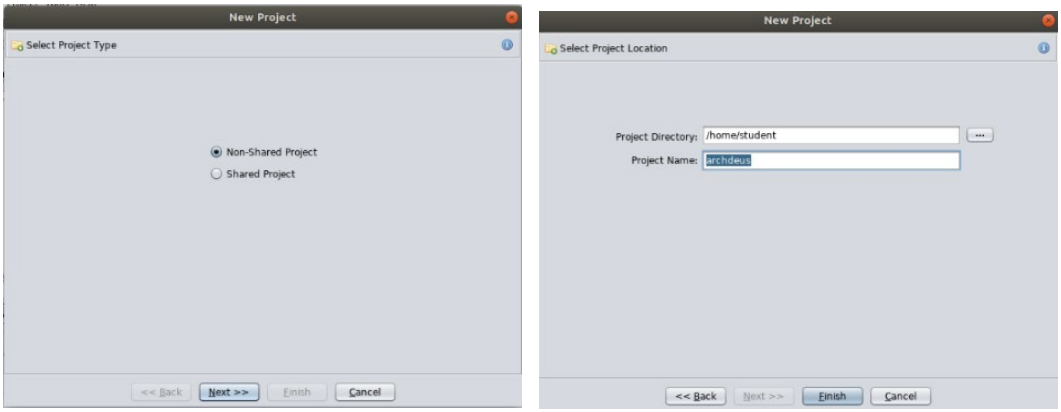

Fig. 3 Create Project and Name Project windows

3) At the main screen, select ***File***-> ***Import File…*** and then choose the file that you moved to the desktop in the previous exercise: **module.3144.1fd64520.6f4d0000.dll**. Leave all settings as defaults and click ***OK*** on the remaining windows as shown in Fig 4.

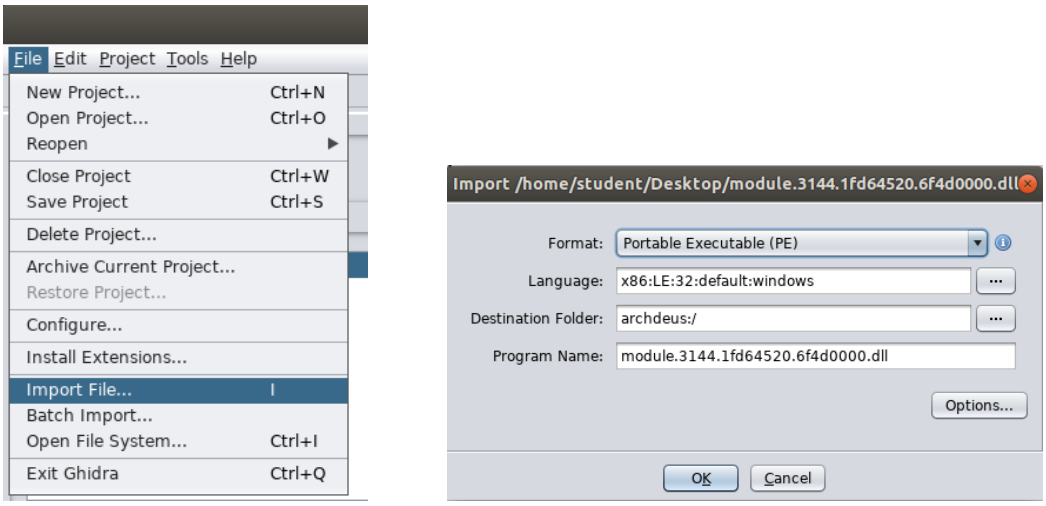

Fig. 4 Import windows

4) Double-click on the file you imported. Eventually, Ghidra will bring up the ***Analysis Options*** window (shown in Fig. 5). Do not change any settings, just click ***Analyze***.

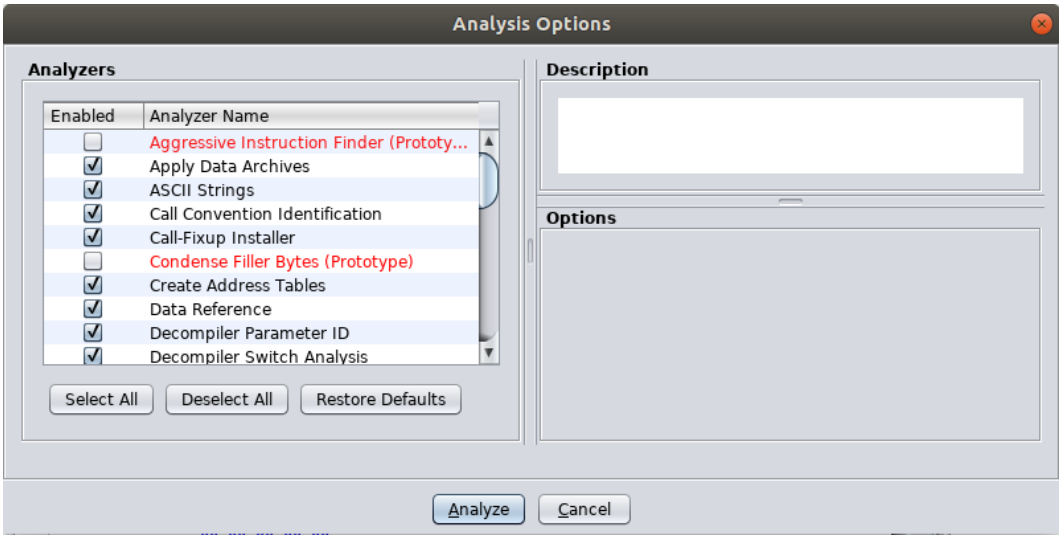

Fig. 5 Analysis Options window

5) Once the analysis is complete, you will see an ***Auto Analysis Summary*** window. Click ***OK*** until you see the ***Code Browser*** window shown in Fig. 6.

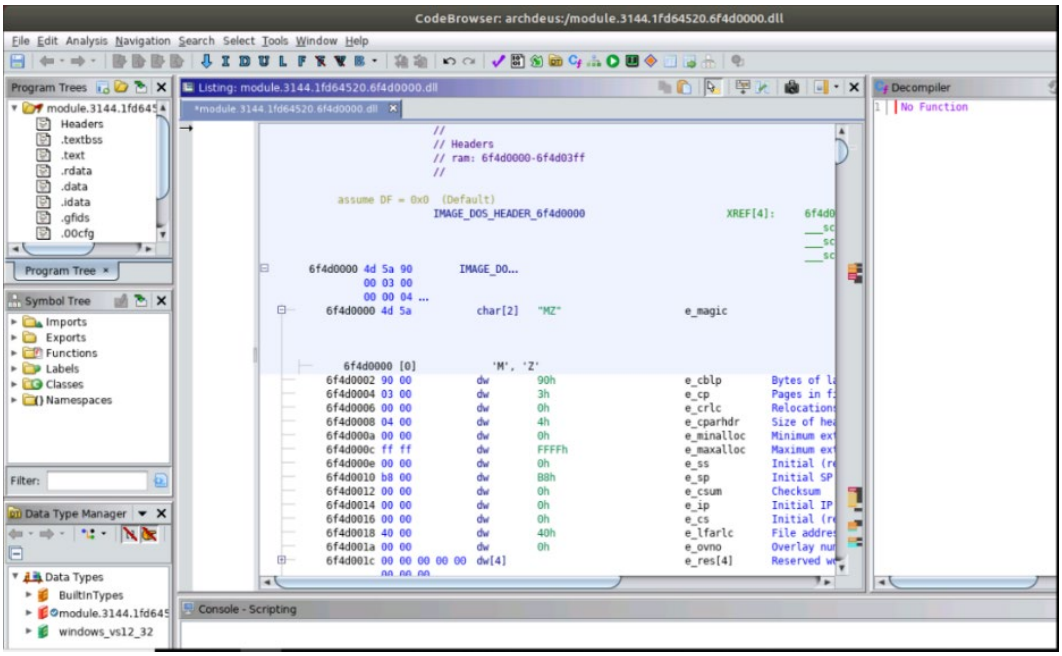

Fig. 6 Code Browser window

****

Ghidra is a very powerful tool that allows an analyst to look at the inner workings of compiled applications. It was publicly released in 2019 by the NSA and is unique in that it is open-source (free), has a nice graphical interface, has extensive support for various binary files, and comes prepackaged with a decompiler. Its features include disassembly, which converts the bytes of a file into readable assembly instructions, as well as a decompiler, which converts the bytes to high-level coding language. Other features include instruction cross-reference, as well as function, string, and class identification.[5]

In this exercise, we want to know **how** the malware is **encoding** the **communication**. We will start with what we know: the **domain name** (goog1e.com), so we will use the string search feature and then we will see **where in the code** it is being used.

****

6) On the menu bar select ***Search*** -> ***Memory***. Click on the ***String*** radio button and enter **goog1e.com** as the search value (see Fig. 7). Click ***Search All***.

this is a number "one"

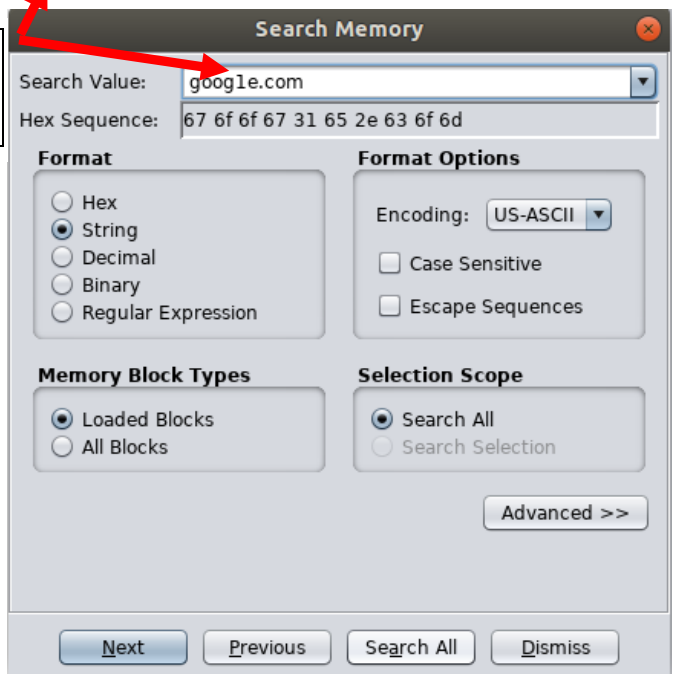

Fig. 7 Search Memory window

7) Double-click on the result (there should be only one) and then go back to the ***Code Browser*** window. You will now see the string in the code as shown in Fig. 8.

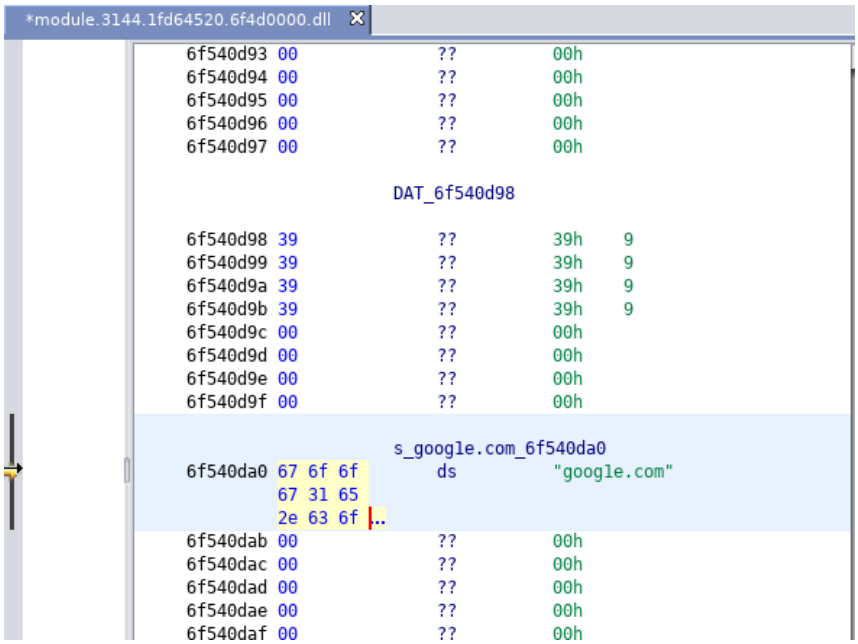

Fig. 8 Code view for goog1e.com

8) Highlight and right click on ***s_googl1e.com_6f540da0*** and then select ***References*** -> ***Show References to s_googl1e.com_6f540da0***. Double-click on the result (there should only be one, as shown in Fig. 9).

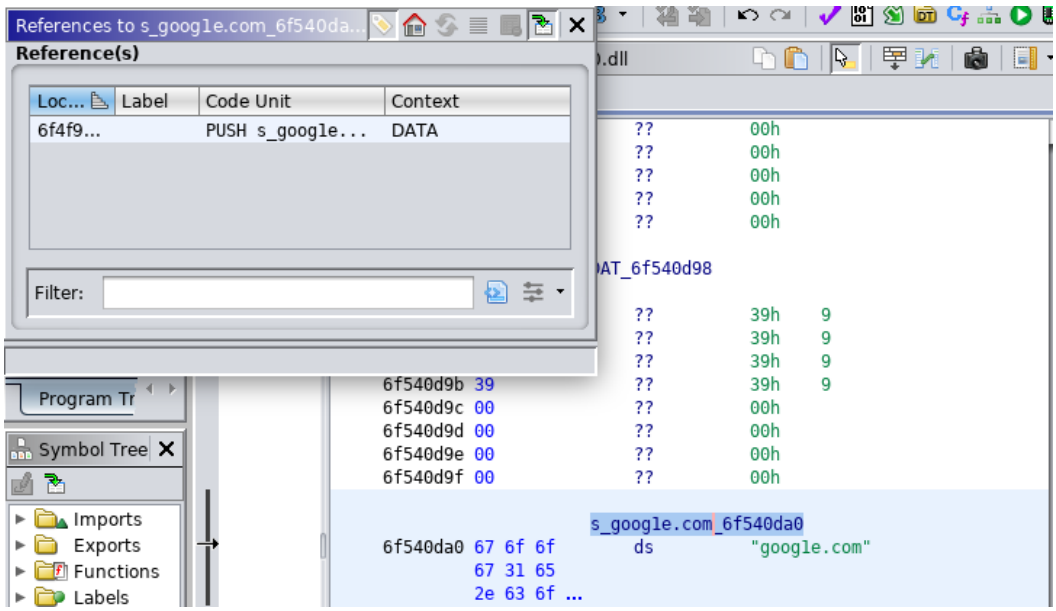

Fig. 9 Cross-references window

9) Close the ***Search*** window and expand the ***Decompiler*** subwindow (this is the right-most window). Notice the use of the string at **line 62**. Look through the code at your leisure and then scroll down to **line 117** and double-click on ***thunk_FUN_6f4faac0***.

   Now the Code Browser is showing the contents of ***FUN_6f4faac0.*** It seems the author of this malware printed some cleartext to the screen.

10) Look for the line of code where the author calls a function to **print** the results of the decoded string "decoding string:…". What is the name of the function and variable_name? (*Hint: function format resembles:* function_name(<"printable-text">, (char) <variable-name>);)

    function_name: _____ variable_name: _____

11) It turns out that the variable you found in 10) contains the **decoded** communication string. Look for the function above this, where it is likely that the string was actually decoded (it should have the same variable name).

    function_name: _____

12) Double-click on the function name you found in 11). If your decompiler screen looks like Fig. 10, you have found the encoding/decoding code! It seems that the encoding is a **simple shift cipher**. Write a short description (preferably pseudo-code) of the encoding. *(Hint 1: an ASCII chart like this may help:* https://www.petefreitag.com/cheatsheets/ascii-codes/*.) (Hint 2: the ascii character \r is the decimal number 13.*)

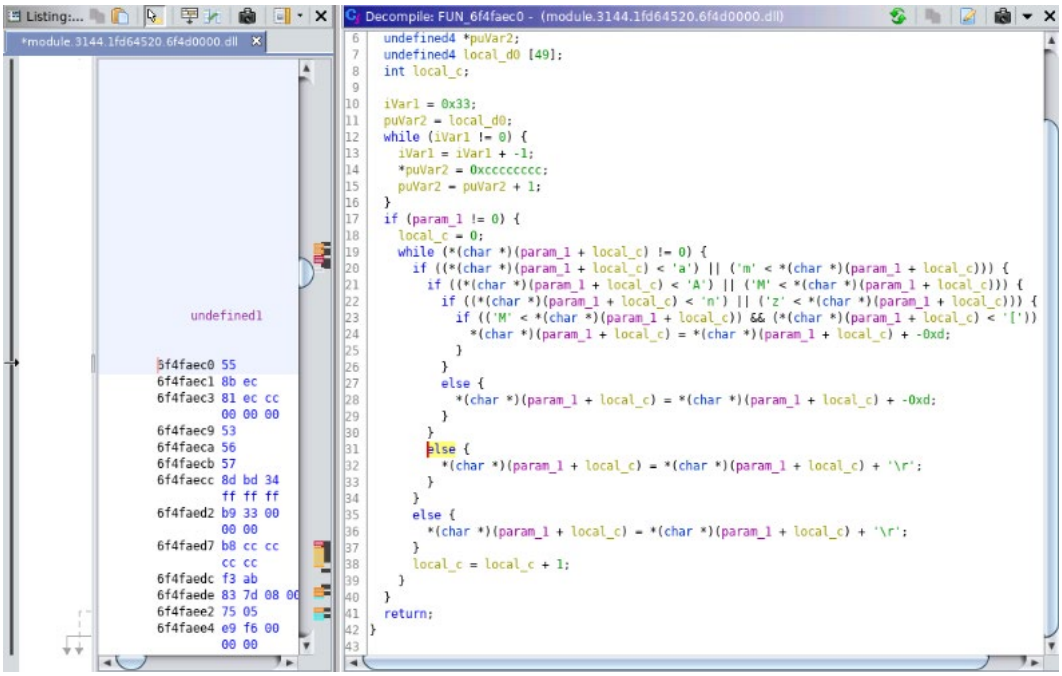

Fig. 10 Decompiler window

5.3 Speak to the Ancients

Test what you have learned: send a message to the C2 server and decode its language.

1) By hand, encode the text "**list commands**" using the cipher algorithm that you identified (spaces are not encoded, and do not include the quotation marks).

_____

2) Start a new terminal by pressing **Ctrl+Alt+t**

****

Netcat (or nc) is a powerful network testing tool that allows analysts to quickly start clients and servers, and has many more Transmission Control Protocol/User Datagram Protocol (TCP/UDP) uses.[10] You can check out a listing of some of these here: https://www.sans.org/security-resources/sec560/netcat_cheat_sheet_v1.pdf.

****

3) Use the Netcat tool to establish a connection with the C2 server by executing the following command:

   nc <ip address of C2 server> 9999

   Recall that the client is constantly connecting and disconnecting to the server (at 30-second intervals in fact). If you do not receive a message within 3 seconds, press Ctrl+c and keep trying.

Once you receive a message, type in your answer to question 1) and press Enter.

4) Write the response that you get here:

   _____

5) Decode the response and write it here:

   _____

6) Describe the response and what you think it is:

   _____

****

Uber question: Can you **list** any other **masters** (C2 servers)? What are their IP addresses?

****

Great job! You have successfully connected to the C2 server and obtained some additional live information!

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 obfuscation of the data is used in attempts to hide the encoding scheme. Participants learned one possible way to uncover, deconstruct, and then reconstruct this communication using the Ghidra reverse-engineering tool. 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 to 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—initial analysis with Wireshark and Volatility. DEVCOM Army Research Laboratory (US); 2020 Dec. Report No.: ARL-TR-9128.
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. Netcat [accessed 2020 Dec]. http://netcat.sourceforge.net/.

List of Symbols, Abbreviations, and Acronyms

ARL	Army Research Laboratory
Bot	infected machine
C2	command and control
CyberRIG	Cybersecurity Rapid Innovation Group
DEVCOM	US Army Combat Capabilities Development Command
IP	Internet Protocol
nc	Netcat
NSA	National Security Agency
RAT	remote access trojan
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
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