# MALWARE ANALYSIS USING GHIDRA

**A PROJECT REPORT**

***Submitted by***

**ANAND PANDEY (22BCY10122)**
**AYUSH MONGA (22BCY10267)**
**VIKASH KUMAR (22BCY10030)**
**ROSHNI SHARMA (22BCY10203)**
**AHMED AZAAN KHAN (22BCY10245)**

*in partial fulfillment for the award of the degree of*

**BACHELOR OF TECHNOLOGY**

**in**

**COMPUTER SCIENCE AND ENGINEERING**
**(Cyber Security and Digital Forensics)**

**SCHOOL OF COMPUTING SCIENCE AND ENGINEERING**

**VIT BHOPAL UNIVERSITY**

**KOTHRIKALAN, SEHORE**
**MADHYA PRADESH - 466114**

MAY 24

**VIT BHOPAL UNIVERSITY, KOTHRI KALAN,**
**SEHORE MADHYA PRADESH – 466114**

## BONAFIDE CERTIFICATE

Certified that this project report titled **“MALWARE ANALYSIS USING GHIDRA”** is the bonafide work of **“ROSHNI SHARMA (22BCY10203), VIKASH KUMAR (22BCY10030), AHMED AZAAN KHAN (22BCY10245), ANAND PANDEY (22BCY10122), and AYUSH MONGA (22BCY10267)”** who carried out the project work under my supervision. Certified further that to the best of my knowledge the work reported at this time does not form part of any other project/research work based on which a degree or award was conferred on an earlier occasion to this or any other candidate.

**PROGRAM CHAIR**
**Dr. D. Saravanan**
Program Chair,
Division of Cyber Security and
Digital Forensics
School of Computer Science and
Engineering
VIT BHOPAL UNIVERSITY

**PROJECT SUPERVISOR**
**Dr. Hariharasitaraman S**
Supervisor,
Division of Cyber Security and
Digital Forensics
School of Computer Science and
Engineering
VIT BHOPAL UNIVERSITY

The Project Exhibition 1 Examination is held on___________________.

# ACKNOWLEDGEMENT

First and foremost, I would like to thank the Lord Almighty for his presence and immense blessings throughout the project work.

I wish to express my heartfelt gratitude to **Dr D. Saravanan**, Program Chair, Cyber Security and Digital Forensics for much of his valuable support encouragement in carrying out this work

I would like to thank my internal guide Dr. Hariharasitaraman.S, for continually guiding and actively participating in my project, giving valuable suggestions to complete the project work.

I would also like to thank our two reviewers Dr. Ganeshan.R and Dr. Rizwan Ur Rahaman.
Last but not the least, I would like to thank all the technical and teaching staff of the School of Computing Science, who extended directly or indirectly all their support.

# LIST OF ABBREVIATIONS

| | |
|---|---|
| **URL-** | Uniform Resource Locator |
| **DLL-** | Dynamic Link Library |
| **DNS-** | Domain Name System |
| **PE-** | Portable Executor |
| **ASCII-** | American Standard Code for Information Interchange |
| **SHA-** | Secure Hash Algorithm |
| **IOC-** | Indicators of Compromise |
| **APT-** | Advance Persistent Threat |
| **Exe-** | Executable |

# LIST OF FIGURES AND GRAPHS

# TABLE OF CONTENTS

# ABSTRACT

Malware analysis plays a crucial role in cybersecurity by providing insights into malicious software, enabling threat detection, and enhancing cybersecurity defenses. This abstract explores the utilization of Ghidra, an open-source software reverse engineering framework, in the context of malware analysis. Ghidra's features, including disassembly, decompilation, and scripting capabilities, empower analysts to dissect malware samples, understand their functionalities, and identify potential security vulnerabilities.

By leveraging Ghidra for malware analysis, analysts can extract actionable intelligence from malicious code, map out attack techniques employed by threat actors, and develop effective mitigation strategies. This abstract delves into the benefits of using Ghidra in malware analysis, highlights its role in enhancing cybersecurity readiness, and underscores the importance of leveraging powerful tools like Ghidra in combating evolving cyber threats. Through a comprehensive exploration of Ghidra's capabilities in malware analysis, this abstract aims to showcase the significance of incorporating advanced tools and techniques in cybersecurity practices for proactive threat mitigation and incident response..

Comments:

- The proposed name of the topic is clear.
- The objective is relevant.
- Includes research findings on the topic.

# CHAPTER 1

# PROJECT DESCRIPTION AND OUTLINE

## 1.1 Introduction

**Malware** is an abbreviated form of “malicious software.” It is developed as harmful software that invades or corrupts one’s computer network. The goal of malware is to cause havoc and steal information or resources for monetary gain or sheer sabotage intent. It is specifically designed to gain access to or damage a computer, usually without the knowledge of the owner. The term "malware" was reportedly first coined by Yisrael Radai in 1990. Yisrael Radai is an Israeli security expert who used the term in a Usenet newsgroup discussion. Since then, "malware" has become a widely accepted and commonly used term in the field of cybersecurity to refer to all types of malicious software. Now, over 25 million new types of malwares registered since the beginning of 2022 alone. Classifying malware into distinct types or categories based on its characteristics, behavior, and intended purpose helps cybersecurity professionals, researchers, and antivirus software developers better understand, analyse, and respond to different types of malwares. Ghidra is a free and open-source **software reverse engineering (SRE)** tool developed by the National Security Agency (NSA) of the United States. It essentially allows you to crack open a program and understand how it works, line by line.

- **Disassembles code:** Imagine a program as a complex recipe written in a secret code. Ghidra translates this code (machine code) into a more readable format called assembly language, like a recipe with basic instructions.
- **Decompiles:** While not perfect, Ghidra can try to convert the assembly language back into the original programming language the software was written in. This gives you a closer look at the programmer's intent.
- **Analyzes the code:** Ghidra offers various tools to dissect the program's functionality. You can identify data structures, follow how the program flows, and even search for specific patterns within the code.

- **Extends with scripting:** Ghidra allows you to write scripts to automate repetitive tasks or create custom functionalities, making the process more efficient.

Ghidra workbench with various tools for examining the inner workings of a program. It's particularly useful for security researchers analyzing malware, engineers debugging complex software, or anyone interested in understanding how programs function at their core.

## 1.2 Motivation for the work

The motivation behind this project of ours is that there has always been a problem related to malicious software known as malware. So, it is important to note that while studying malware is essential for cybersecurity, it should always be done responsibly and within legal boundaries. Malware analysis is required to gain insights into how specific malware strains work, theirmethods and behaviours. The need to safeguard critical infrastructure has led to creating this project. So, let us quickly know more about the project in detail. Reverse engineering is a complex and intellectually stimulating field. Ghidra provides a platform to tackle challenging problems and delve into the intricate workings of software. Disassembling code and piecing together its functionality can be like solving a puzzle, offering a sense of accomplishment when you crack the code. Ghidra is an open-source tool, meaning anyone can contribute to its development. This fosters a collaborative and supportive community of developers and users. You can learn from others, contribute your expertise, and be part of a movement shaping the future of reverse engineering tools.

## 1.3 Problem Statement

In Ghidra we have to tackle various problem and we are analyzing 5 different types of crackme files and analyze them and we have to found password with code provided in C language Crackme files are often heavily obfuscated to make analysis difficult. This can involve techniques like packing, encryption, and control flow obfuscation. Disassembling and understanding the logic behind such code can be a significant challenge. Ghidra is primarily a disassembler and not a full-fledged decompiler. While it can provide pseudocode for some instructions, complex or custom logic might not be easily decompiled. This can make it harder to follow the flow of the program and identify key functionalities. Crackmes often rely on flags or triggers that determine if the program is running in a debugger or cracked state. Ghidra might not always detect these checks effectively, making it difficult to bypass them during analysis. We will also see the use Ghidra scripting and plugins for string search and vulnerable function in which we will create a custom code for string searching and vulnerable function searching and show how to run it in Ghidra.

## 1.4 Objective

The objectives of the project are to:

- To perform Ghidra scripting, for string searching and vulnerable function search.
- To use Ghidra, a software reverse engineering tool, to analyze a crackme file.
- To obtain a dataset of malicious and non-malicious samples.
- To extract the features from the respective samples.
- To create a catalogue of known malicious behaviours.

- To analyse the extracted features and compare them with the catalogue of known malicious behaviours.

## 1.5 Scope

The scopes involved in the project are threat classification, malware behavioural analysis, vulnerability exploitation.

Basically, the overall process of this project can be viewed as below:

- Collect the data
- Extract features from the samples
- Build a Malware Behavior Catalogue
- Analyse the behavior of the extracted features
- Test and evaluate
- Alert and reporting

## 1.6 Report Organization

This thesis consists of five (5) chapters. Chapter 1 will discuss the introduction to the system which will explain the introduction, problem statement, objective, and scope. For Chapter 2, it will discuss the literature review, definition, and types of malwares. For Chapter 3, it will discuss the methodology and requirements of the project. For Chapter 4, it will discuss the design and implementation of the project. Chapter 5, it will show the results and discussion.

**SUMMARY**

Our project focuses on malware analysis using Ghidra, a software reverse engineering tool in which we will use its different feature such as C decompiler and will tell how to perform reverse engineering using that by solving various crackme files. We will also create a Ghidra script for string search and vulnerable function analysis for a malware.

# CHAPTER 2:
# RELATED WORK INVESTIGATION

## 2.1 Introduction

In this chapter, we are going to focus on discussing the results or findings based on the article, journals, or any other related reference material. Some original words from the reference material may be cited to enhance the review. The purpose of this chapter is to explain about the selected project. We focus on malware reverse engineering using a technique called as scripting. Ghidra supports a powerful scripting API that helps users programmatically dissect programs using Ghidra scripts written in Java or Python 2.7 (via Jython). Through our project, we aim to reveal the inner workings of malware functions using scripting. By using existing knowledge as a guide, we seek to contribute to ongoing efforts to improve cyber security, share our knowledge and protect our digital environment from malware threats. Basically, it is divided into a few sub-sections as well. Those sub- section include some little explanation of basic concepts of the selected project, research of some already existing similar problem or solution done by others and the hardware, technique or method which will be applied or used in the selected project. This chapter explains in detail the techniques or technologies which are suitable to be adapted into the project. This chapter contains information about the study of the project in general.

LINKS:

- https://apps.dtic.mil/sti/trecms/pdf/AD1119396.pdf
- https://www.techtarget.com/searchsecurity/feature/How-to-use-Ghidra-for-malware-analysis-reverse-engineering

## 2.2 What is Malware Classification?

Malware classification is the process of categorizing malicious software (malware) into distinct types or categories based on its characteristics, behavior, and intended purpose. This classification helps cybersecurity professionals, researchers, and antivirus software developers better understand, analyze, and respond to different types of malware.

Here are some common categories of malware:

- Worms: Worms are self-replicating malware that can spread across networks or the internet without the need for user interaction. They often exploit vulnerabilities to propagate.
- Viruses: Viruses are malicious programs that attach themselves to legitimate files or software. They can replicate and spread to other files or systems when the infected file is executed.
- Trojans: Trojans disguise themselves as legitimate software but have malicious functionality hidden within. They can perform a variety of harmful actions, such as data theft, remote control, or system damage.

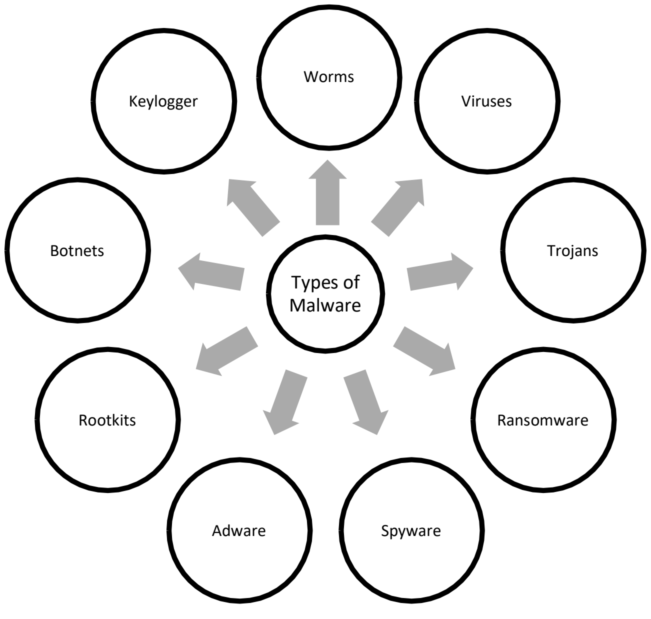


**Fig 1: Malware Classification**

- Ransomware: Ransomware encrypts a victim's files or locks them out of their system, demanding a ransom payment in exchange for the decryption key. It's known for its extortion tactics.
- Spyware: Spyware is designed to secretly monitor and gather information about a user's activities, often without their consent. It can record keystrokes, capture screenshots, and more.
- Other: Adware, Rootkits, Botnets, Keyloggers, etc.

# CHAPTER: 3
# REQUIREMENT ARTIFACTS

## 3.1 Introduction

A system architecture is a conceptual model that defines the structure, behavior, and interactions of a system. It tells us about various components of a system and how they interact with each other. It is a blueprint for how the system will be built and deployed. The system architecture should be designed to meet the specific needs of the system and its users.

A system architecture is a conceptual model that defines the structure, behavior, and interactions of a system. It tells us about various components of a system and how they interact with each other. It is a blueprint for how the system will be built and deployed. The system architecture should be designed to meet the specific needs of the system and its users. The system architecture is typically defined using a variety of diagrams and models. These diagrams and models show the different components of the system, how they are connected, and how they interact with each other. The system architecture may also include descriptions of the system's data flows, control flows, and performance characteristics. The system architecture is important because it helps to ensure that the system is well-designed and that it will meet the needs of its users. It also helps to identify any potential problems with the system early on, before they become too difficult to fix.

## 3.2 Tools and Environments used

For malware analysis various tools, environments have been used as specified below:

1. **ORACLE Virtual Box**: VM Virtual Box is cross-platform virtualization software. It allows users to extend their existing computer to run multiple operating systems including Microsoft Windows, Mac OS X, Linux, and Oracle Solaris, at the same time. Designed for IT professionals and developers, Oracle VM Virtual Box is ideal for testing, developing, demonstrating, And deploying solutions across multiple platforms from one machine.

2. **Operating System**: The Linux Operating System is a type of operating system that is similar to Unix, and it is built upon the Linux Kernel. The Linux Kernel is like the brain of the operating system because it manages how the computer interacts with its hardware and resources. It makes sure everything works smoothly and efficiently. But the Linux Kernel alone is not enough to make a complete operating system. To create a full and functional system, the Linux Kernel is combined with a collection of software packages and utilities, which are together called Linux distributions. These distributions make the Linux Operating System ready for users to run their applications and perform tasks on their computers securely and effectively. Linux distributions come in different flavors, each tailored to suit the specific needs and preferences of users.

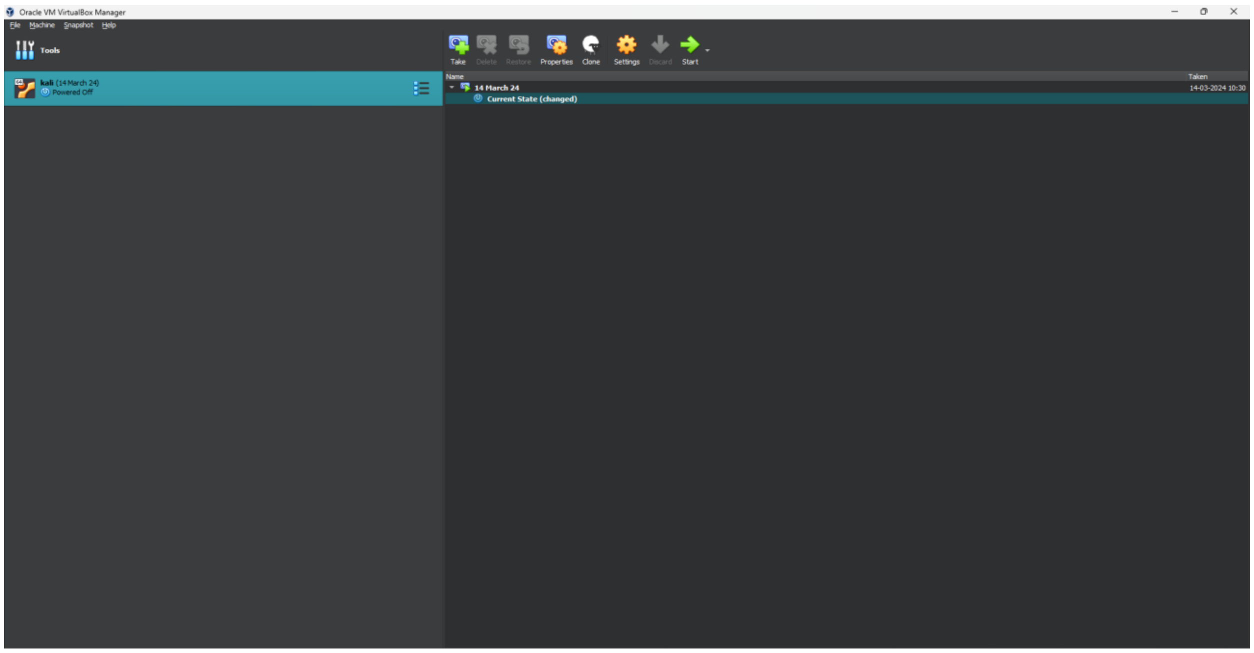
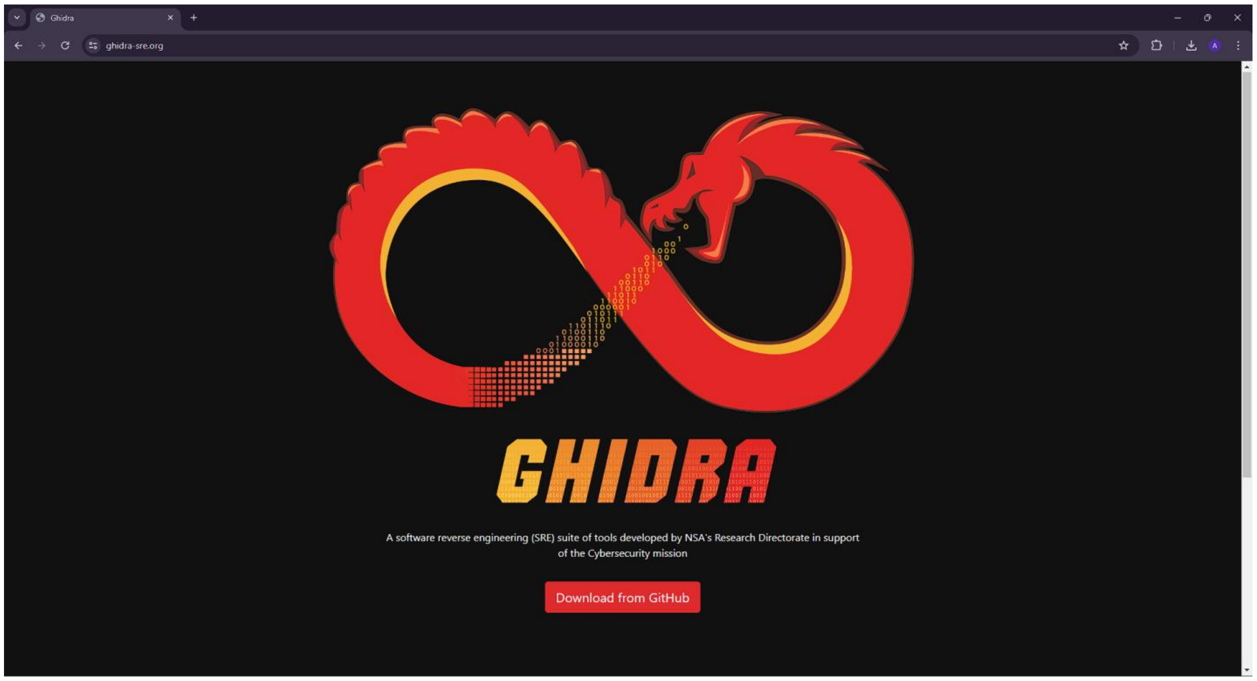
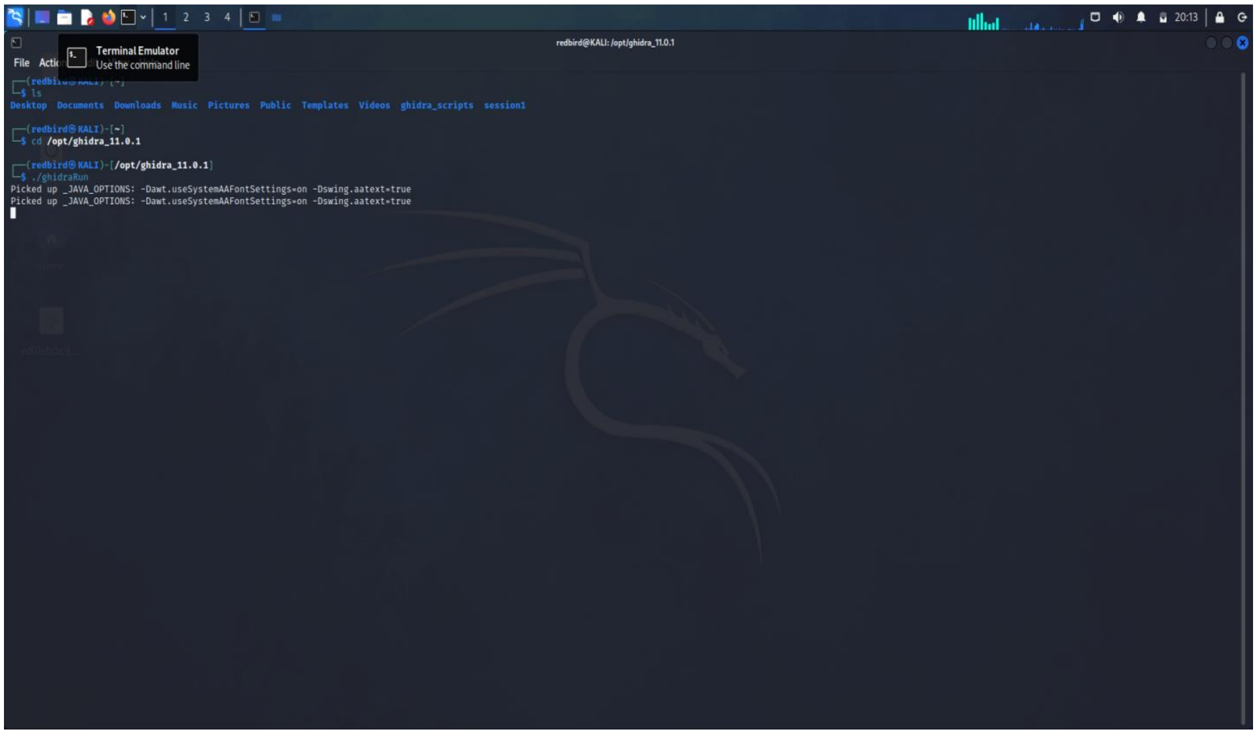

Oracle VM VirtualBox Manager
File Machine Snapshot Help
Tools
kali (14 March 24)
Powered Off
Take Delete Restore Properties Clone Settings Discard Start
Name
Taken
14 March 24
14-03-2024 10:30
Current State (changed)

Fig 2: Installing Oracle VM and setting up Linux through ISO file



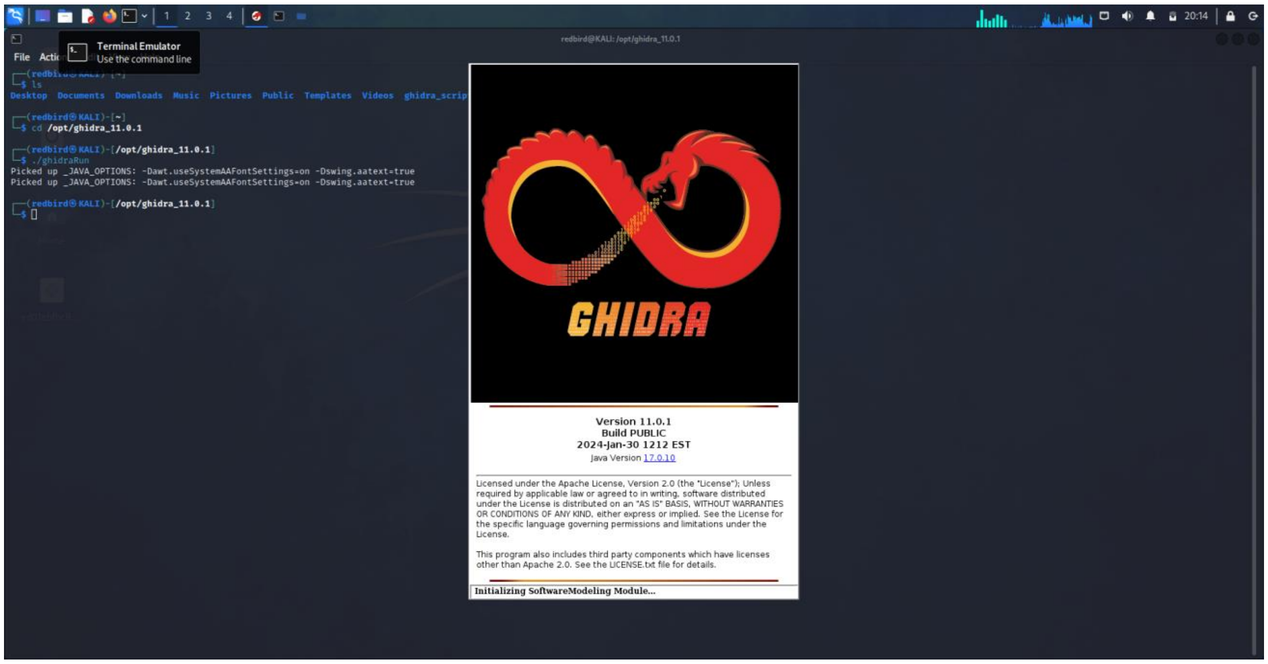
A software reverse engineering (SRE) suite of tools developed by NSA's Research Directorate in support of the Cybersecurity mission

Fig 3: Downloading Ghidra



```
┌──(redbird㉿KALI)-[~]
└─$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos  ghidra_scripts  session1

┌──(redbird㉿KALI)-[~]
└─$ cd /opt/ghidra_11.0.1

┌──(redbird㉿KALI)-[/opt/ghidra_11.0.1]
└─$ ./ghidraRun
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
```

Fig 4: Installing Ghidra



```
┌──(redbird㉿KALI)-[~]
└─$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos  ghidra_scrip

┌──(redbird㉿KALI)-[~]
└─$ cd /opt/ghidra_11.0.1

┌──(redbird㉿KALI)-[/opt/ghidra_11.0.1]
└─$ ./ghidraRun
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true

┌──(redbird㉿KALI)-[/opt/ghidra_11.0.1]
└─$
```

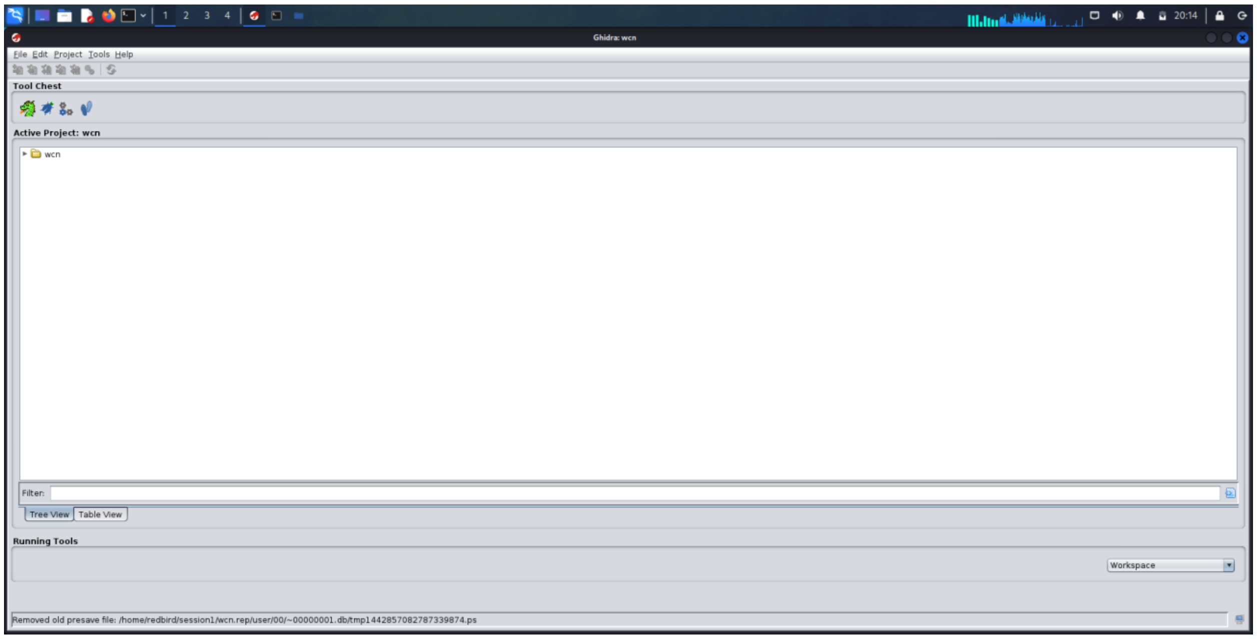
Version 11.0.1
Build PUBLIC
2024-Jan-30 1212 EST
Java Version 17.0.10

Licensed under the Apache License, Version 2.0 (the "License"); Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

This program also includes third party components which have licenses other than Apache 2.0. See the LICENSE.txt file for details.

Initializing SoftwareModeling Module...

Fig 5: Opening Ghidra



```
Removed old presave file: /home/redbird/session1/wcn.rep/user/00/~00000001.db/tmp1442857082787339874.ps
```

Fig 6: Running Ghidra

# CHAPTER 4 :
# DESIGN METHODOLOGY AND ITS NOVELTY

## 4.1 Introduction

This chapter contains section 3.2 depicting the overall flowchart of the program which is the step-by-step analysis of the malware. Section 3.2 contains hardware and software requirements for "Malware Analysis".

**Methodology:**

A systematic approach to analyzing a file with Ghidra, incorporating some novel techniques:

**Preparation:**

1. **Installation:** Ensure you have the latest Ghidra and Java 11 installed.

**Project Setup:**

1. **Launch Ghidra:** Open "runGhidra.bat" (Windows) or follow the instructions for your OS.
2. **Project Creation:** Go to "File" -> "New Project" and choose "Non-Shared" or "Shared" based on your needs. Give it a relevant name.

**Import and Initial Analysis:**

1. **File Import:** Drag and drop your file (executable, library, etc.) into the project window or use "File" -> "Import File". Accept default import options if prompted.
2. **Automated Analysis:** Ghidra will automatically analyze the file. This stage involves:
   - **Disassembly:** Converting machine code instructions into assembly language for human understanding.
   - **Data Identification:** Recognizing data structures like variables, strings, and constants within the file.

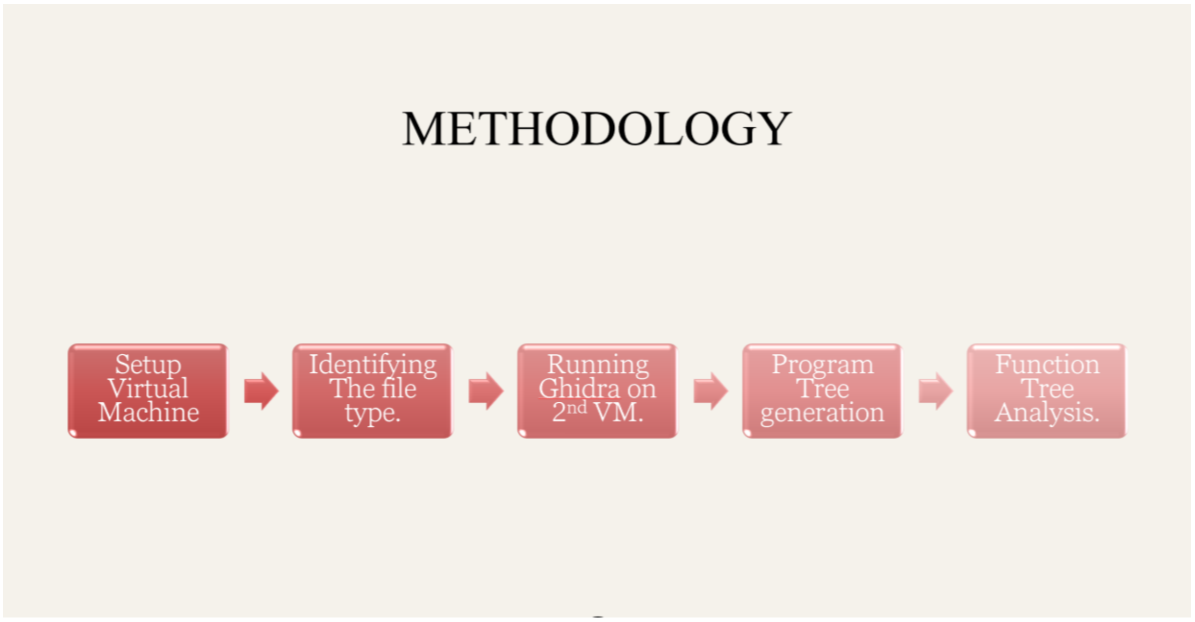
    - **Control Flow Analysis:** Understanding how the program flow works, including branches and loops.

**Novelty in Initial Analysis:**

- **Machine Learning-Assisted Disassembly:** Explore Ghidra's experimental features that utilize machine learning to improve disassembly accuracy, especially for complex or obfuscated code. This can provide a more reliable foundation for further analysis.

**Novelty in Deep Dive Analysis:**

- **Scripting for Automation:** Ghidra's scripting capabilities can be leveraged to automate repetitive tasks like data pattern identification or function analysis. This can significantly improve efficiency, especially when dealing with large files.
- **Custom Ghidra Plugins:** For unique analysis needs, consider developing custom Ghidra plugins. These can automate complex workflows, identify specific malware characteristics, or visualize data in novel ways.



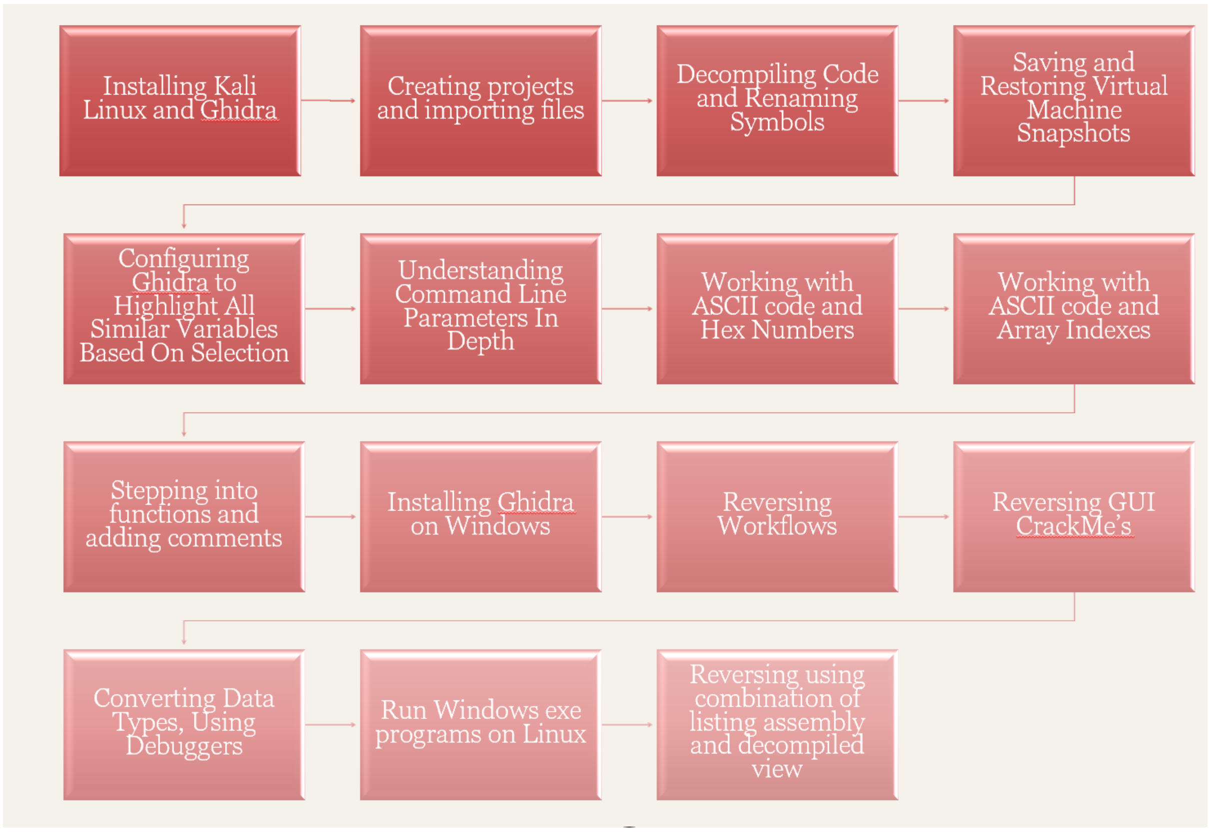

Installing Kali Linux and Ghidra
Creating projects and importing files
Decompiling Code and Renaming Symbols
Saving and Restoring Virtual Machine Snapshots
Configuring Ghidra to Highlight All Similar Variables Based On Selection
Understanding Command Line Parameters In Depth
Working with ASCII code and Hex Numbers
Working with ASCII code and Array Indexes
Stepping into functions and adding comments
Installing Ghidra on Windows
Reversing Workflows
Reversing GUI CrackMe's
Converting Data Types, Using Debuggers
Run Windows exe programs on Linux
Reversing using combination of listing assembly and decompiled view

Fig. 7: FLOWCHART FOR MALWARE ANALYSIS USING GHIDRA

## 4.2 Requirements

- . Install Virtual Box and Kali Linux
- .Download Java 11 and above version
- .Set up Ghidra in Kali Linux
- Code to install Ghidra in Linux (sudo apt install ghidra)
- . Analyzing Crakme file
- .Analyzing Ghidra Scripting

# CHAPTER-5
# TECHNICAL IMPLEMENTATION AND ANALYSIS

## 5.1 Introduction

Ghidra is a powerful open-source software reverse engineering framework developed by the National Security Agency (NSA). It aids in analyzing malicious code and software vulnerabilities. With features like disassembly, decompilation, and scripting, Ghidra is widely used by cybersecurity professionals and researchers to understand how software works, identify security vulnerabilities, and enhance cybersecurity practices. Its flexibility and extensibility make it a valuable tool in the field of cybersecurity and reverse engineering.

## 5.2 Advantages and Disadvantages of Ghidra

Advantages of Ghidra:

1. Open-Source: Being open-source, Ghidra is freely available to the cybersecurity community, fostering collaboration, transparency, and continuous improvement.
2. Feature-Rich: Ghidra offers a range of powerful features like disassembly, decompilation, debugging, and scripting, providing comprehensive support for software reverse engineering tasks.
3. Cross-Platform: It is available on multiple operating systems, allowing users to work with Ghidra on different platforms.
4. Community Support: There is an active user community around Ghidra, providing resources, tutorials, and sharing knowledge that helps users maximize their utility of the tool.

Disadvantages of Ghidra:

1. Learning Curve: Ghidra's feature-rich environment can be overwhelming for beginners, requiring time and effort to master effectively.

2. Performance: For extremely large binaries, Ghidra's analysis process may be resource-intensive and time-consuming.

3. User Interface: Some users find Ghidra's user interface to be less intuitive compared to other reverse engineering tools, which may impact user experience and productivity for those not familiar with it.

4. Limited Plugins: While Ghidra supports scripting and plugins, the availability of third-party plugins and extensions is not as extensive as some other reverse engineering tools.

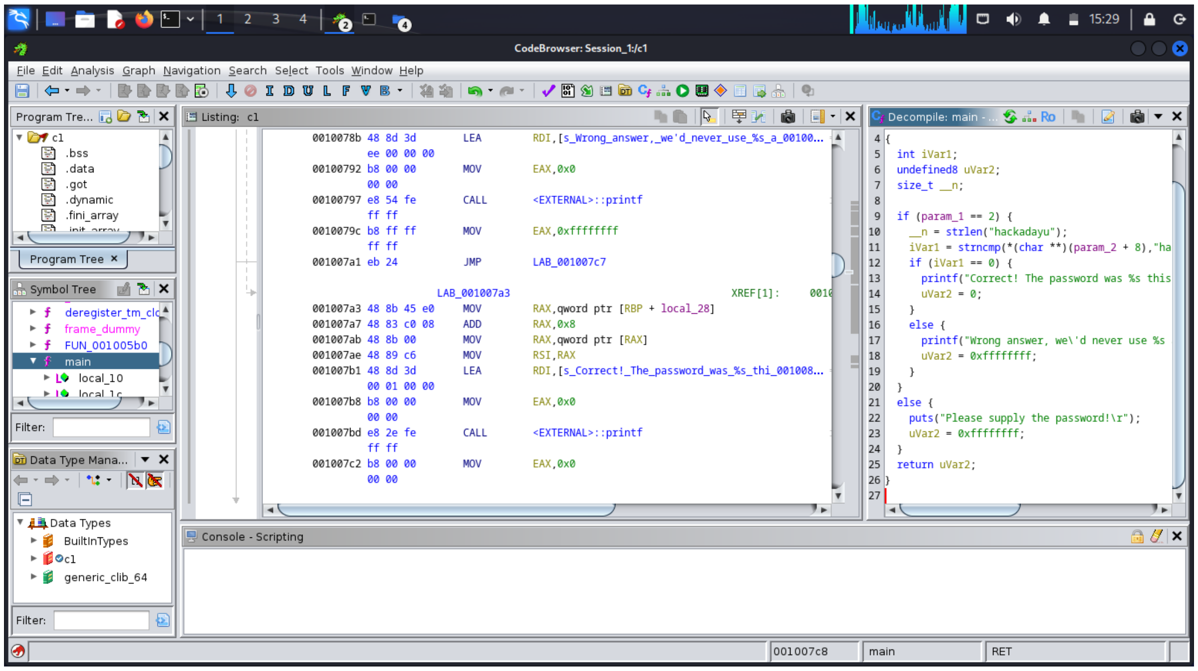
Overall, Ghidra's strengths lie in its accessibility, robust features, and community support, with considerations for its learning curve and performance for more extensive project

## 5.3 Crackme Files Analysis:

CrackMe files are challenges designed to test one's ability to reverse engineer software and bypass certain security mechanisms. Typically created by software developers and cybersecurity enthusiasts, CrackMe files often require participants to uncover a specific password, serial key, or other hidden information by analyzing the program's code and behavior. These challenges provide valuable hands-on experience for individuals looking to sharpen their skills in software reverse engineering and cybersecurity by encouraging them to think creatively and develop solutions to bypass various protection mechanisms.

Basic steps to start:

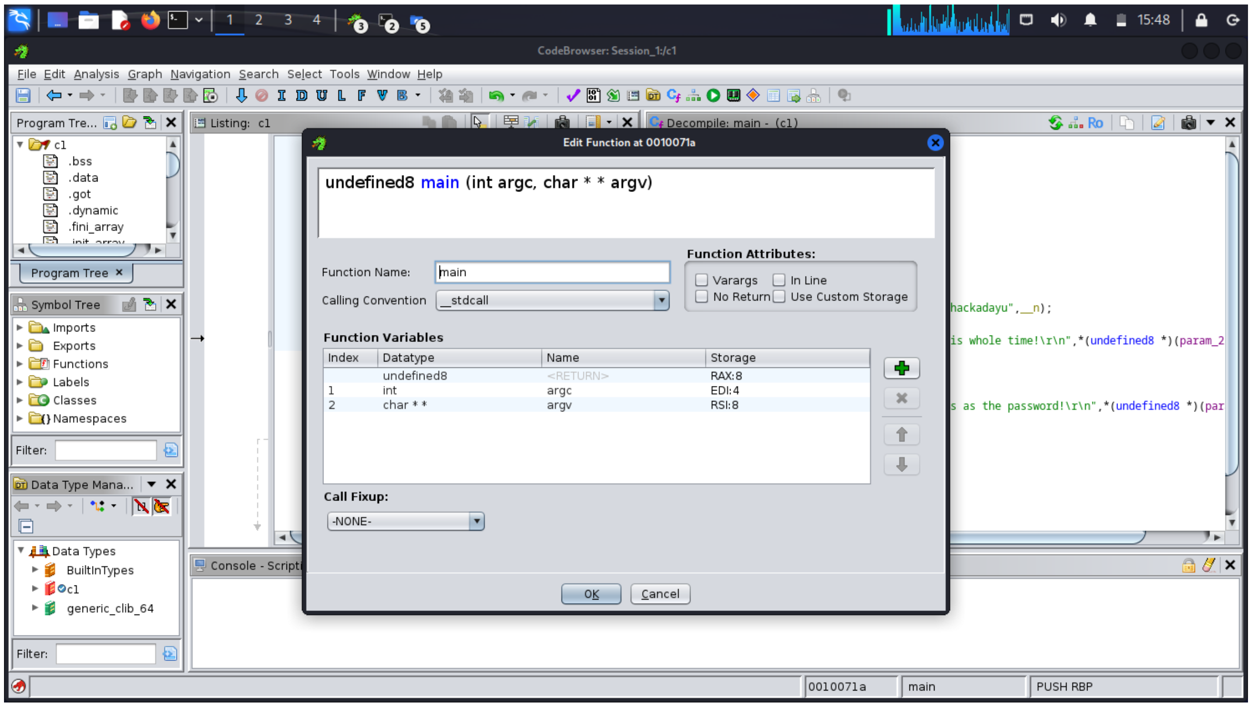
- Put file in a new project and start analyser.
- In analyser, in c decompiler section observe the code for the main function.
- Change the parameter as shown in second figure given below.
- Look for some hint like in first crackme its easy direct password hackdayu is given.



```
0010078b 48 8d 3d     LEA     RDI,[s_Wrong_answer,_we'd_never_use_%s_a_00100...
         ee 00 00 00
00100792 b8 00 00     MOV     EAX,0x0
         00 00
00100797 e8 54 fe     CALL    <EXTERNAL>::printf
         ff ff
0010079c b8 ff ff     MOV     EAX,0xffffffff
         ff ff
001007a1 eb 24        JMP     LAB_001007c7

                  LAB_001007a3                              XREF[1]:     0010
001007a3 48 8b 45 e0  MOV     RAX,qword ptr [RBP + local_28]
001007a7 48 83 c0 08  ADD     RAX,0x8
001007ab 48 8b 00     MOV     RAX,qword ptr [RAX]
001007ae 48 89 c6     MOV     RSI,RAX
001007b1 48 8d 3d     LEA     RDI,[s_Correct!_The_password_was_%s_thi_001008...
         00 01 00 00
001007b8 b8 00 00     MOV     EAX,0x0
         00 00
001007bd e8 2e fe     CALL    <EXTERNAL>::printf
         ff ff
001007c2 b8 00 00     MOV     EAX,0x0
         00 00
```

```
{
  int iVar1;
  undefined8 uVar2;
  size_t __n;

  if (param_1 == 2) {
    __n = strlen("hackadayu");
    iVar1 = strncmp(*(char **)(param_2 + 8),"ha
    if (iVar1 == 0) {
      printf("Correct! The password was %s this
      uVar2 = 0;
    }
    else {
      printf("Wrong answer, we\'d never use %s
      uVar2 = 0xffffffff;
    }
  }
  else {
    puts("Please supply the password!\r");
    uVar2 = 0xffffffff;
  }
  return uVar2;
}
```

Fig 8: View of a file inside Ghidra



Edit Function at 0010071a

undefined8 main (int argc, char * * argv)

Function Name: main

Calling Convention: __stdcall

Function Attributes: Varargs [ ] In Line [ ] No Return [ ] Use Custom Storage [ ]

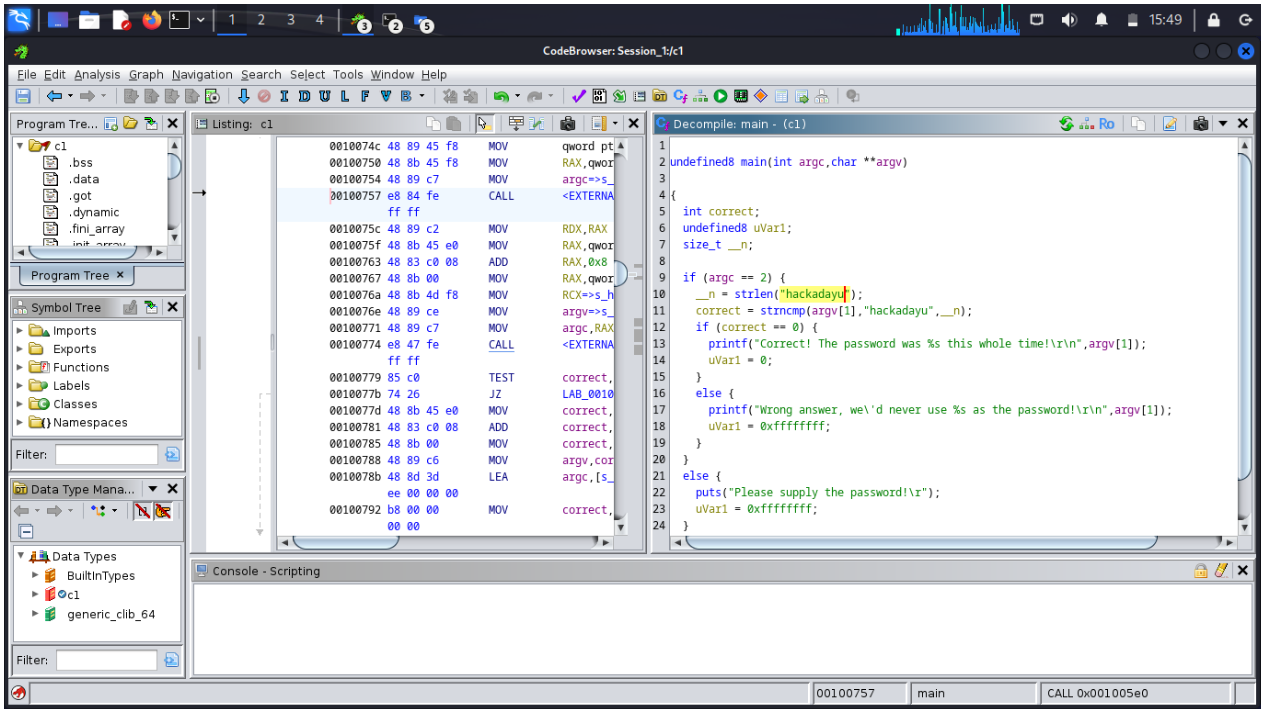
Function Variables

| Index | Datatype | Name | Storage |
|---|---|---|---|
| | undefined8 | <RETURN> | RAX:8 |
| 1 | int | argc | EDI:4 |
| 2 | char * * | argv | RSI:8 |

Call Fixup: -NONE-

```
hackadayu",__n);
is whole time!\r\n",*(undefined8 *)(param_2
s as the password!\r\n",*(undefined8 *)(par
```

Fig 9: Changing of parameters of main function


```
0010074c 48 89 45 f8    MOV     qword pt
00100750 48 8b 45 f8    MOV     RAX,qwor
00100754 48 89 c7       MOV     argc=>s_
00100757 e8 84 fe       CALL    <EXTERNA
         ff ff
0010075c 48 89 c2       MOV     RDX,RAX
0010075f 48 8b 45 e0    MOV     RAX,qwor
00100763 48 83 c0 08    ADD     RAX,0x8
00100767 48 8b 00       MOV     RAX,qwor
0010076a 48 8b 4d f8    MOV     RCX=>s_h
0010076e 48 89 ce       MOV     argv=>s_
00100771 48 89 c7       MOV     argc,RAX
00100774 e8 47 fe       CALL    <EXTERNA
         ff ff
00100779 85 c0          TEST    correct,
0010077b 74 26          JZ      LAB_0010
0010077d 48 8b 45 e0    MOV     correct,
00100781 48 83 c0 08    ADD     correct,
00100785 48 8b 00       MOV     correct,
00100788 48 89 c6       MOV     argv,cor
0010078b 48 8d 3d       LEA     argc,[s_
         ee 00 00 00
00100792 b8 00 00       MOV     correct,
         00 00
```

```

undefined8 main(int argc,char **argv)

{
  int correct;
  undefined8 uVar1;
  size_t __n;

  if (argc == 2) {
    __n = strlen("hackadayu");
    correct = strncmp(argv[1],"hackadayu",__n);
    if (correct == 0) {
      printf("Correct! The password was %s this whole time!\r\n",argv[1]);
      uVar1 = 0;
    }
    else {
      printf("Wrong answer, we\'d never use %s as the password!\r\n",argv[1]);
      uVar1 = 0xffffffff;
    }
  }
  else {
    puts("Please supply the password!\r");
    uVar1 = 0xffffffff;
  }
```

Fig 10: Cracked point hackdayu

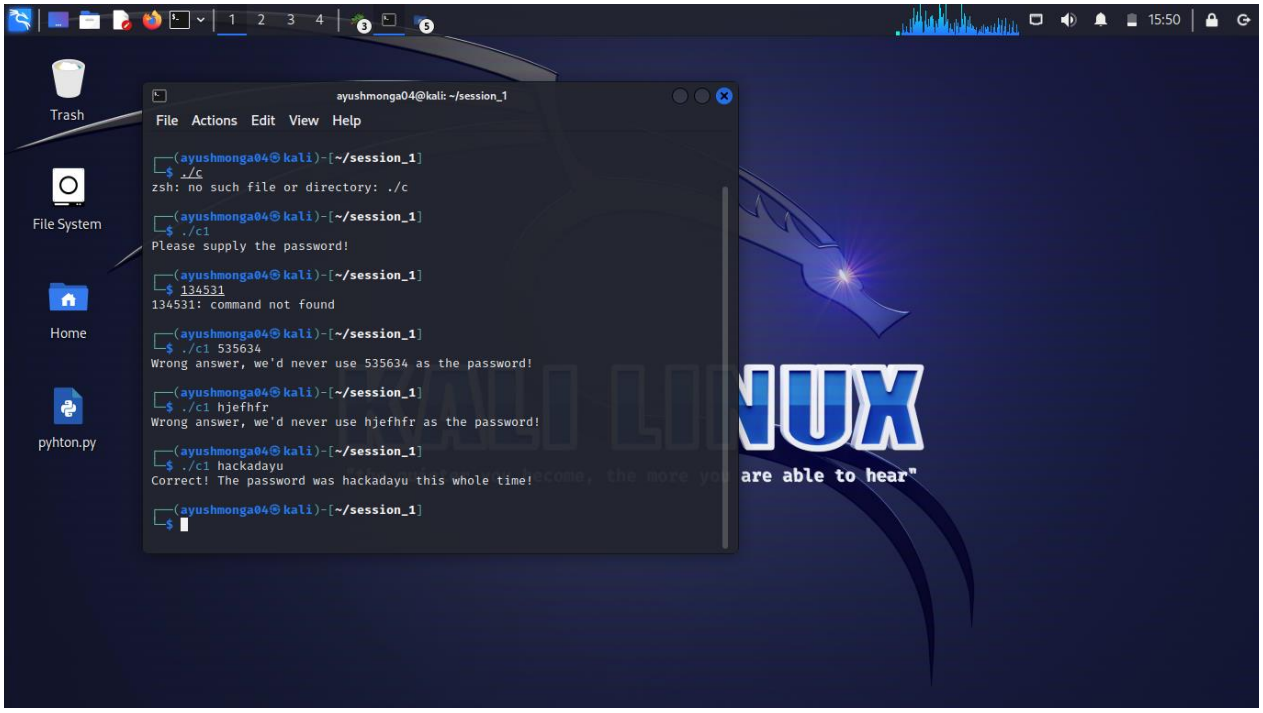
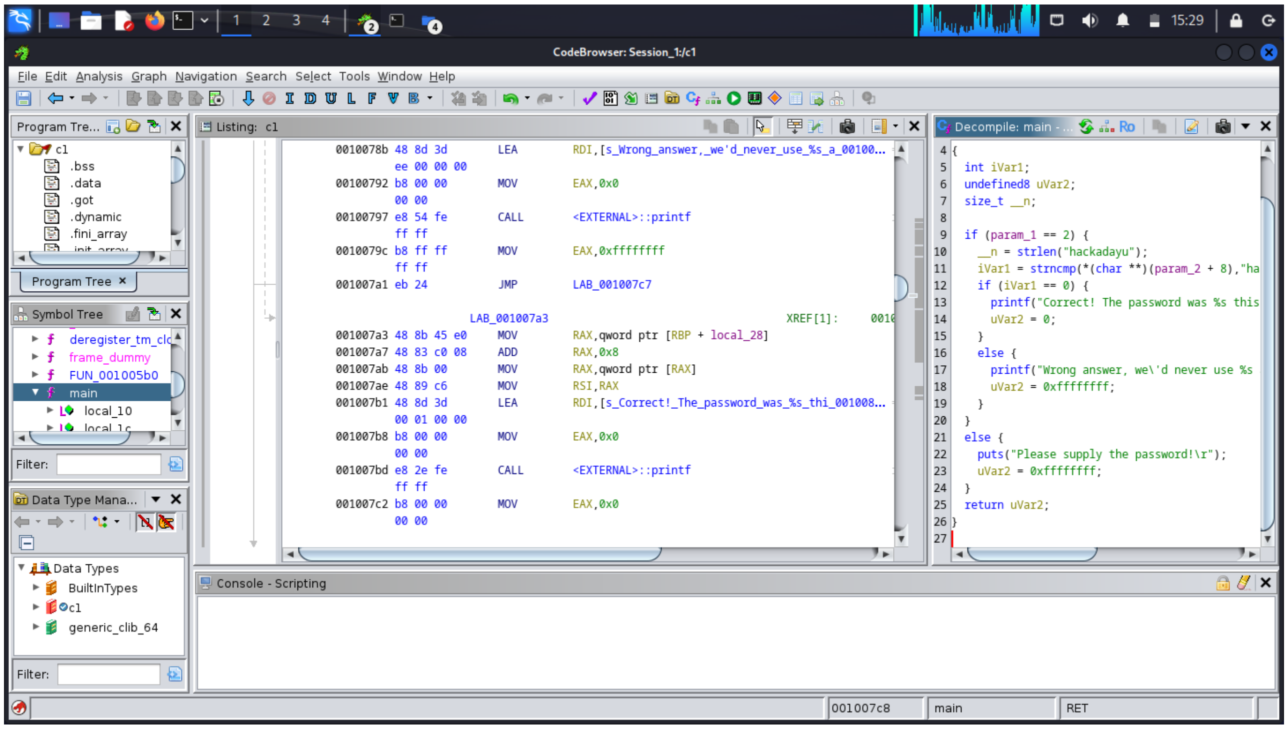

```
┌──(ayushmonga04㉿kali)-[~/session_1]
└─$ ./c
zsh: no such file or directory: ./c

┌──(ayushmonga04㉿kali)-[~/session_1]
└─$ ./c1
Please supply the password!

┌──(ayushmonga04㉿kali)-[~/session_1]
└─$ 134531
134531: command not found

┌──(ayushmonga04㉿kali)-[~/session_1]
└─$ ./c1 535634
Wrong answer, we'd never use 535634 as the password!

┌──(ayushmonga04㉿kali)-[~/session_1]
└─$ ./c1 hjefhfr
Wrong answer, we'd never use hjefhfr as the password!

┌──(ayushmonga04㉿kali)-[~/session_1]
└─$ ./c1 hackadayu
Correct! The password was hackadayu this whole time!

┌──(ayushmonga04㉿kali)-[~/session_1]
└─$ 
```

Fig 11: Running the cracked security key successfully

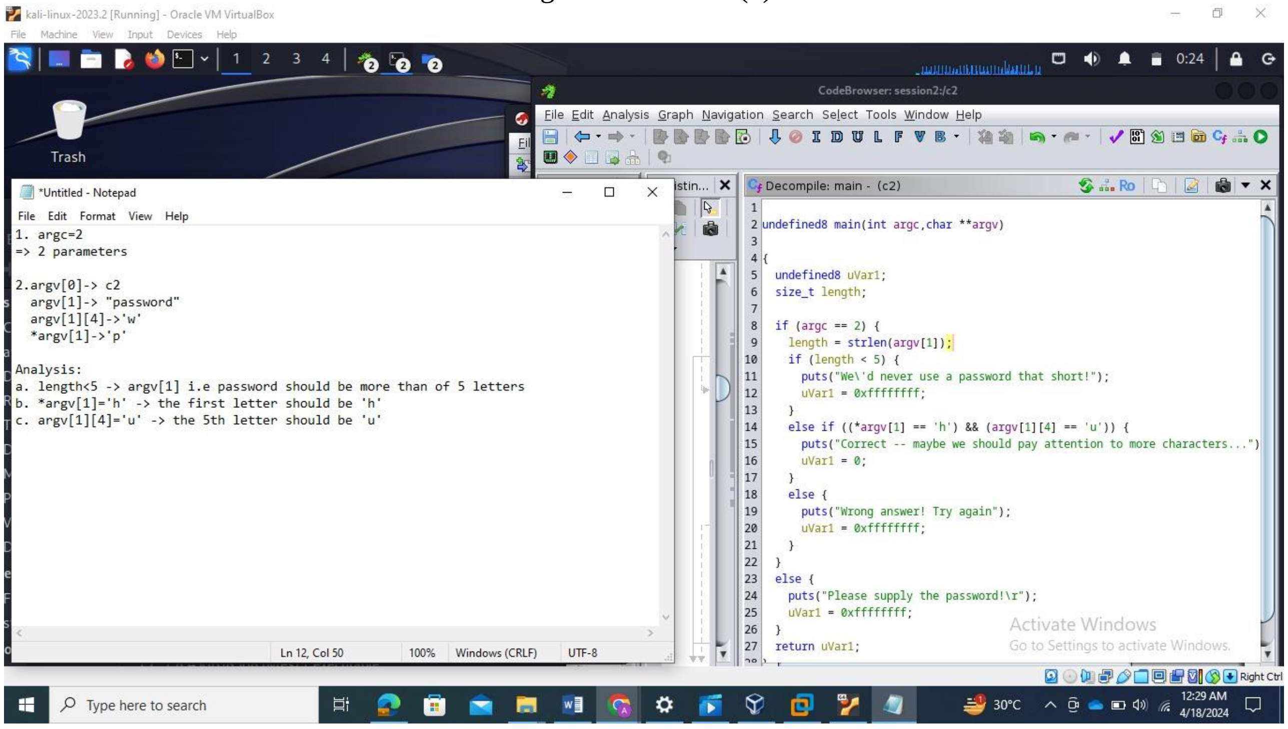
Now lets look at some more crackme which are a bit more complex to solve. We have also shown how we are cracking the crackme on notepad side by side so that you could understand it in the figures given below.



```
0010078b 48 8d 3d      LEA     RDI,[s_Wrong_answer,_we'd_never_use_%s_a_00100...
         ee 00 00 00
00100792 b8 00 00      MOV     EAX,0x0
         00 00
00100797 e8 54 fe      CALL    <EXTERNAL>::printf
         ff ff
0010079c b8 ff ff      MOV     EAX,0xffffffff
         ff ff
001007a1 eb 24         JMP     LAB_001007c7

                LAB_001007a3                            XREF[1]:     0010
001007a3 48 8b 45 e0   MOV     RAX,qword ptr [RBP + local_28]
001007a7 48 83 c0 08   ADD     RAX,0x8
001007ab 48 8b 00      MOV     RAX,qword ptr [RAX]
001007ae 48 89 c6      MOV     RSI,RAX
001007b1 48 8d 3d      LEA     RDI,[s_Correct!_The_password_was_%s_thi_001008...
         00 01 00 00
001007b8 b8 00 00      MOV     EAX,0x0
         00 00
001007bd e8 2e fe      CALL    <EXTERNAL>::printf
         ff ff
001007c2 b8 00 00      MOV     EAX,0x0
         00 00
```

```
{
  int iVar1;
  undefined8 uVar2;
  size_t __n;

  if (param_1 == 2) {
    __n = strlen("hackadayu");
    iVar1 = strncmp(*(char **)(param_2 + 8),"ha
    if (iVar1 == 0) {
      printf("Correct! The password was %s this
      uVar2 = 0;
    }
    else {
      printf("Wrong answer, we\'d never use %s
      uVar2 = 0xffffffff;
    }
  }
  else {
    puts("Please supply the password!\r");
    uVar2 = 0xffffffff;
  }
  return uVar2;
}
```

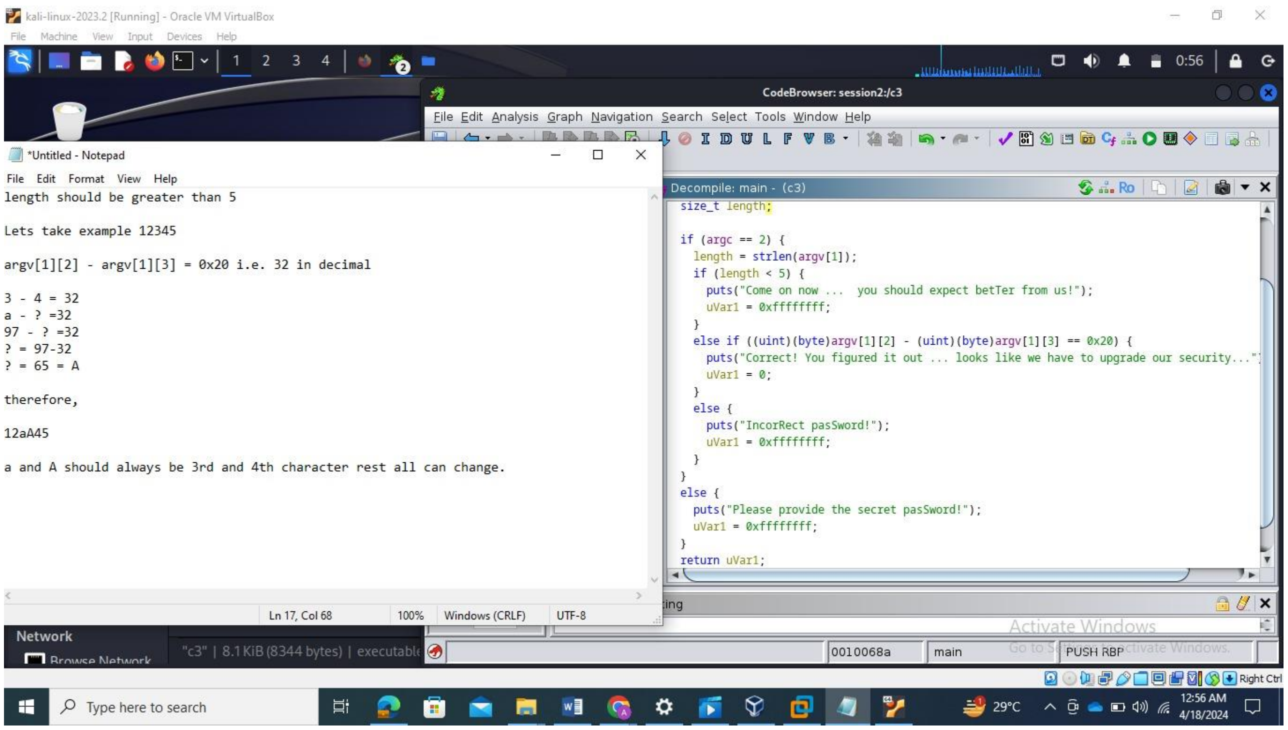
Fig 12: CrackMe 2(1)



```
1. argc=2
=> 2 parameters

2.argv[0]-> c2
  argv[1]-> "password"
  argv[1][4]->'w'
  *argv[1]->'p'

Analysis:
a. length<5 -> argv[1] i.e password should be more than of 5 letters
b. *argv[1]='h' -> the first letter should be 'h'
c. argv[1][4]='u' -> the 5th letter should be 'u'
```

```

undefined8 main(int argc,char **argv)

{
  undefined8 uVar1;
  size_t length;

  if (argc == 2) {
    length = strlen(argv[1]);
    if (length < 5) {
      puts("We\'d never use a password that short!");
      uVar1 = 0xffffffff;
    }
    else if ((*argv[1] == 'h') && (argv[1][4] == 'u')) {
      puts("Correct -- maybe we should pay attention to more characters...")
      uVar1 = 0;
    }
    else {
      puts("Wrong answer! Try again");
      uVar1 = 0xffffffff;
    }
  }
  else {
    puts("Please supply the password!\r");
    uVar1 = 0xffffffff;
  }
  return uVar1;
```

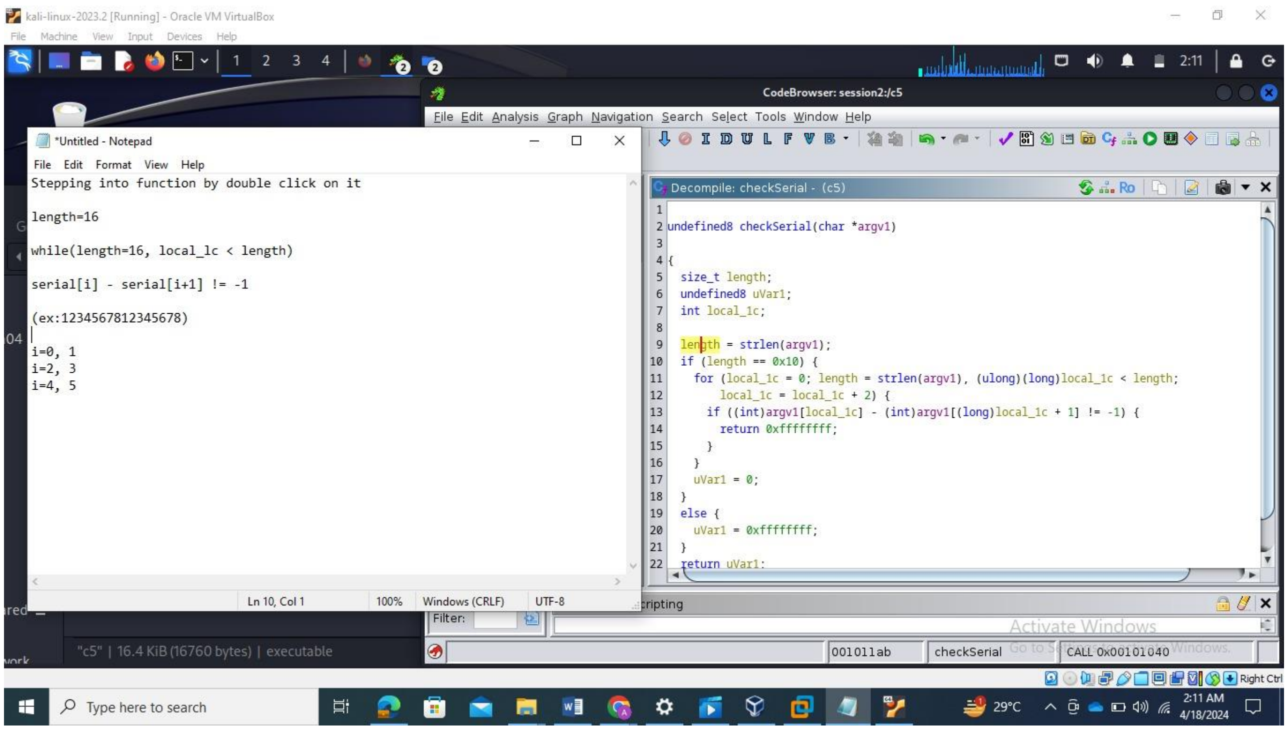
Fig 13: CrackMe 2(2)



```
length should be greater than 5

Lets take example 12345

argv[1][2] - argv[1][3] = 0x20 i.e. 32 in decimal

3 - 4 = 32
a - ? =32
97 - ? =32
? = 97-32
? = 65 = A

therefore,

12aA45

a and A should always be 3rd and 4th character rest all can change.
```

```
size_t length;

if (argc == 2) {
  length = strlen(argv[1]);
  if (length < 5) {
    puts("Come on now ...  you should expect betTer from us!");
    uVar1 = 0xffffffff;
  }
  else if ((uint)(byte)argv[1][2] - (uint)(byte)argv[1][3] == 0x20) {
    puts("Correct! You figured it out ... looks like we have to upgrade our security..."
    uVar1 = 0;
  }
  else {
    puts("IncorRect pasSword!");
    uVar1 = 0xffffffff;
  }
}
else {
  puts("Please provide the secret pasSword!");
  uVar1 = 0xffffffff;
}
return uVar1;
```

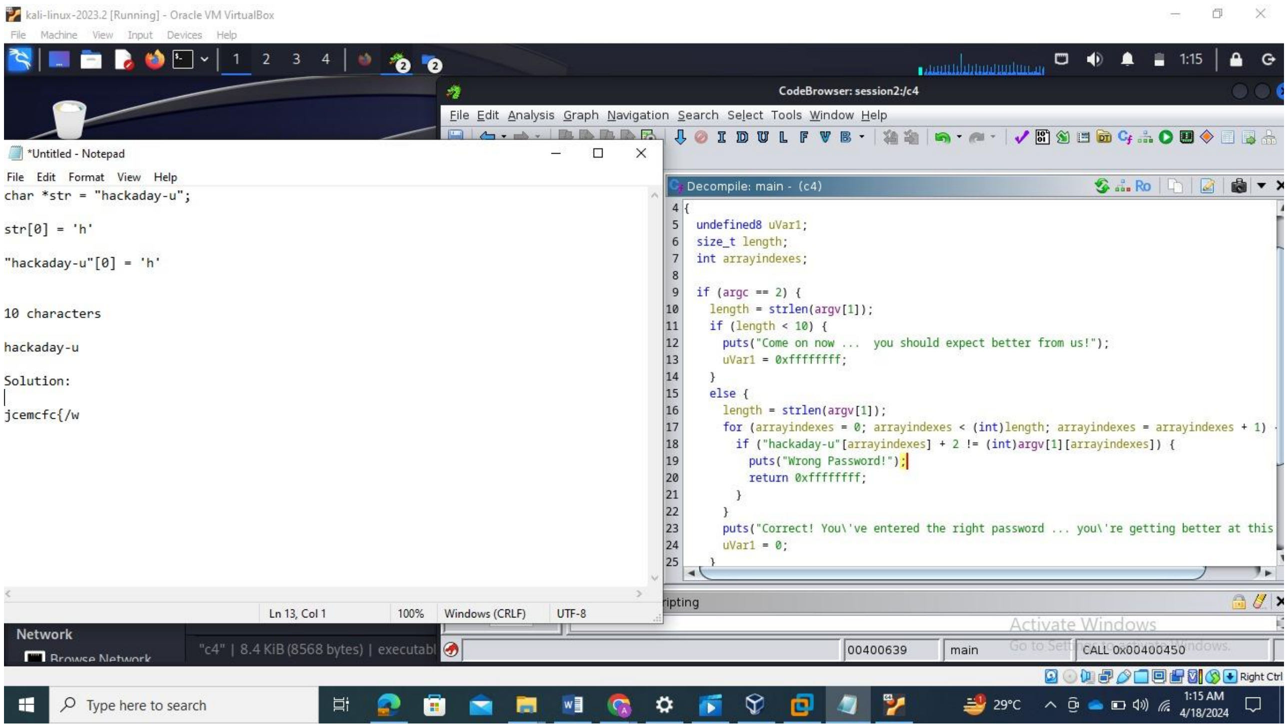
Fig 14: CrackMe 3



```
Stepping into function by double click on it

length=16

while(length=16, local_lc < length)

serial[i] - serial[i+1] != -1

(ex:1234567812345678)

i=0, 1
i=2, 3
i=4, 5
```

```
undefined8 checkSerial(char *argv1)

{
  size_t length;
  undefined8 uVar1;
  int local_1c;

  length = strlen(argv1);
  if (length == 0x10) {
    for (local_1c = 0; length = strlen(argv1), (ulong)(long)local_1c < length;
        local_1c = local_1c + 2) {
      if ((int)argv1[local_1c] - (int)argv1[(long)local_1c + 1] != -1) {
        return 0xffffffff;
      }
    }
    uVar1 = 0;
  }
  else {
    uVar1 = 0xffffffff;
  }
  return uVar1;
```

Fig 15: CrackMe 4



```
char *str = "hackaday-u";

str[0] = 'h'

"hackaday-u"[0] = 'h'


10 characters

hackaday-u

Solution:

jcemcfc{/w
```

```
{
  undefined8 uVar1;
  size_t length;
  int arrayindexes;

  if (argc == 2) {
    length = strlen(argv[1]);
    if (length < 10) {
      puts("Come on now ...  you should expect better from us!");
      uVar1 = 0xffffffff;
    }
    else {
      length = strlen(argv[1]);
      for (arrayindexes = 0; arrayindexes < (int)length; arrayindexes = arrayindexes + 1)
        if ("hackaday-u"[arrayindexes] + 2 != (int)argv[1][arrayindexes]) {
          puts("Wrong Password!");
          return 0xffffffff;
        }
      }
      puts("Correct! You\'ve entered the right password ... you\'re getting better at this
      uVar1 = 0;
    }
```

Fig 16: CrackMe 5

## 5.4 Malware Analysis using Two custom Ghidra Scripts :

Ghidra scripting allows users to automate tasks, extend functionality, and customize the analysis process within the Ghidra framework. Users can write scripts in languages like Java or Python to interact with Ghidra's API, accessing and manipulating various aspects of the disassembled code, symbols, functions, and more.

With Ghidra scripting, users can:

1. Automate repetitive tasks like identifying patterns in the disassembled code or searching for specific functions.

2. Create custom tools to streamline analysis workflows, tailored to specific needs or project requirements.

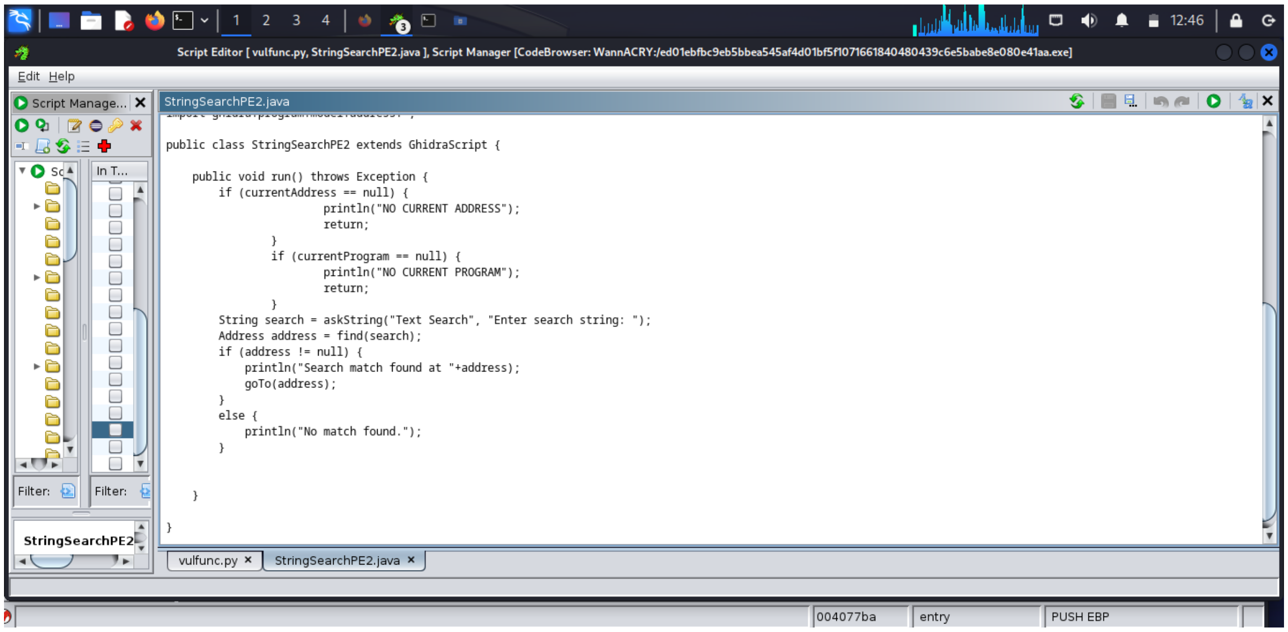
3. Enhance analysis by integrating additional functionality or features not available by default in Ghidra.

4. Share scripts within the Ghidra community, fostering collaboration and knowledge-sharing.

By leveraging the power of scripting, users can make their reverse engineering processes more efficient, effective, and adaptable to different tasks and challenges within Ghidra.

We have used two custom Ghidra scripts for String Serching and Vulnerable Function Searching in Malware we have used two malwares Wannacry and Notpetya to observe its accuracy.

Following two figures show the code of String search and Vulnerable Function analysis that we have used for Ghidra script creation.



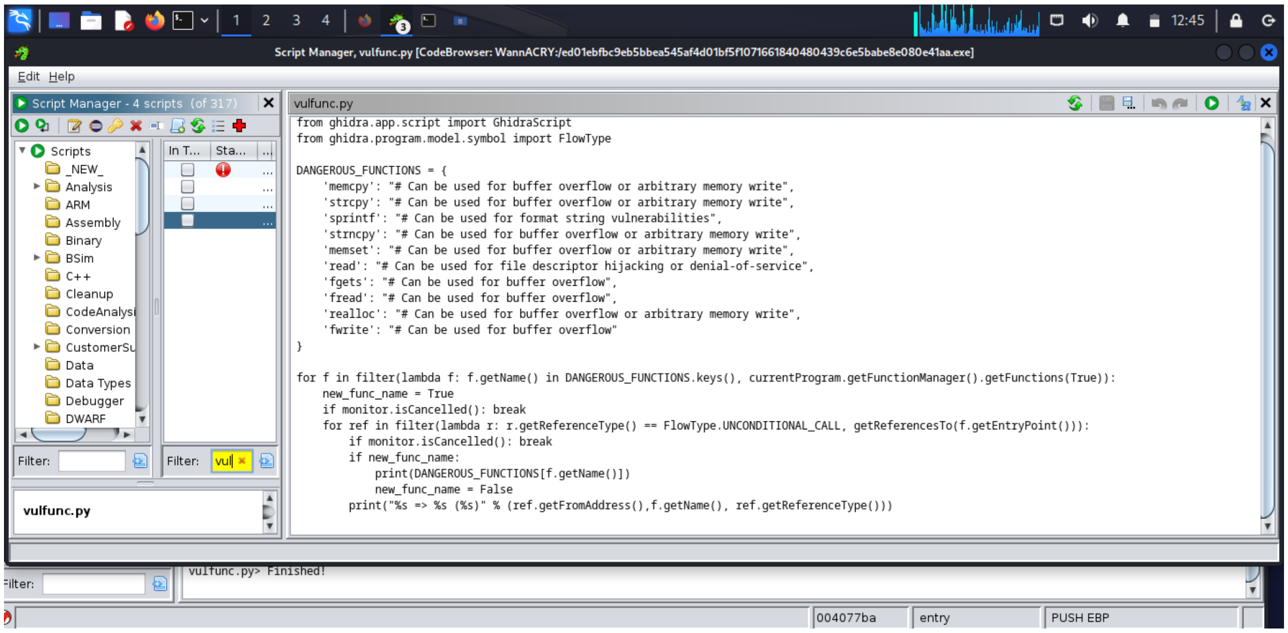
Script Editor [ vulfunc.py, StringSearchPE2.java ], Script Manager [CodeBrowser: WannACRY:/ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa.exe]

StringSearchPE2.java

```java
public class StringSearchPE2 extends GhidraScript {

    public void run() throws Exception {
        if (currentAddress == null) {
                        println("NO CURRENT ADDRESS");
                        return;
                }
                if (currentProgram == null) {
                        println("NO CURRENT PROGRAM");
                        return;
                }
        String search = askString("Text Search", "Enter search string: ");
        Address address = find(search);
        if (address != null) {
            println("Search match found at "+address);
            goTo(address);
        }
        else {
            println("No match found.");
        }


    }

}
```

004077ba entry PUSH EBP

Fig 17: Code for String Search Ghidra Script



Script Manager, vulfunc.py [CodeBrowser: WannACRY:/ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa.exe]

vulfunc.py

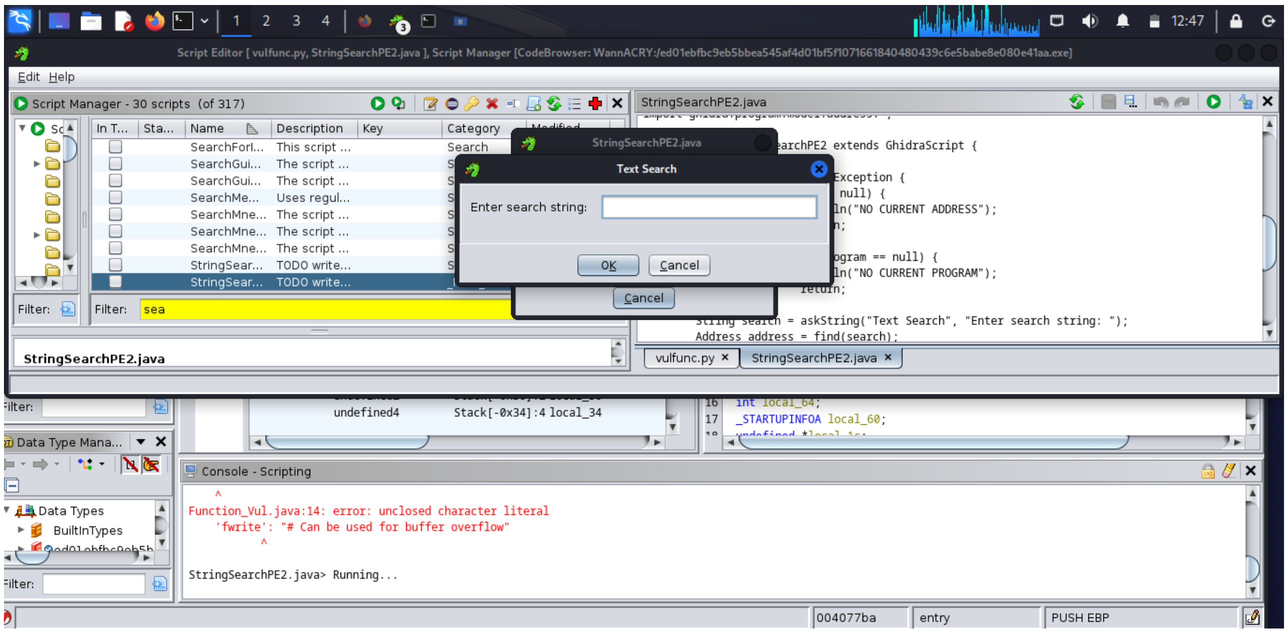
```python
from ghidra.app.script import GhidraScript
from ghidra.program.model.symbol import FlowType

DANGEROUS_FUNCTIONS = {
    'memcpy': "# Can be used for buffer overflow or arbitrary memory write",
    'strcpy': "# Can be used for buffer overflow or arbitrary memory write",
    'sprintf': "# Can be used for format string vulnerabilities",
    'strncpy': "# Can be used for buffer overflow or arbitrary memory write",
    'memset': "# Can be used for buffer overflow or arbitrary memory write",
    'read': "# Can be used for file descriptor hijacking or denial-of-service",
    'fgets': "# Can be used for buffer overflow",
    'fread': "# Can be used for buffer overflow",
    'realloc': "# Can be used for buffer overflow or arbitrary memory write",
    'fwrite': "# Can be used for buffer overflow"
}

for f in filter(lambda f: f.getName() in DANGEROUS_FUNCTIONS.keys(), currentProgram.getFunctionManager().getFunctions(True)):
    new_func_name = True
    if monitor.isCancelled(): break
    for ref in filter(lambda r: r.getReferenceType() == FlowType.UNCONDITIONAL_CALL, getReferencesTo(f.getEntryPoint())):
        if monitor.isCancelled(): break
        if new_func_name:
            print(DANGEROUS_FUNCTIONS[f.getName()])
            new_func_name = False
        print("%s => %s (%s)" % (ref.getFromAddress(),f.getName(), ref.getReferenceType()))
```

```
vulfunc.py> Finished!
```

004077ba entry PUSH EBP

Fig 18: Code for Vulnerable Function analysis Ghidra Script

Now the following figure show the interface created and output the console shows



Script Editor [ vulfunc.py, StringSearchPE2.java ], Script Manager [CodeBrowser: WannACRY:/ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa.exe]

Script Manager - 30 scripts (of 317)

| In T... | Sta... | Name | Description | Key | Category |
|---|---|---|---|---|---|
| | | SearchForI... | This script ... | | Search |
| | | SearchGui... | The script ... | | S |
| | | SearchGui... | The script ... | | S |
| | | SearchMe... | Uses regul... | | S |
| | | SearchMne... | The script ... | | S |
| | | SearchMne... | The script ... | | S |
| | | SearchMne... | The script ... | | S |
| | | StringSear... | TODO write... | | S |
| | | StringSear... | TODO write... | | |

Filter: sea

StringSearchPE2.java

Text Search — Enter search string: [ ] OK Cancel

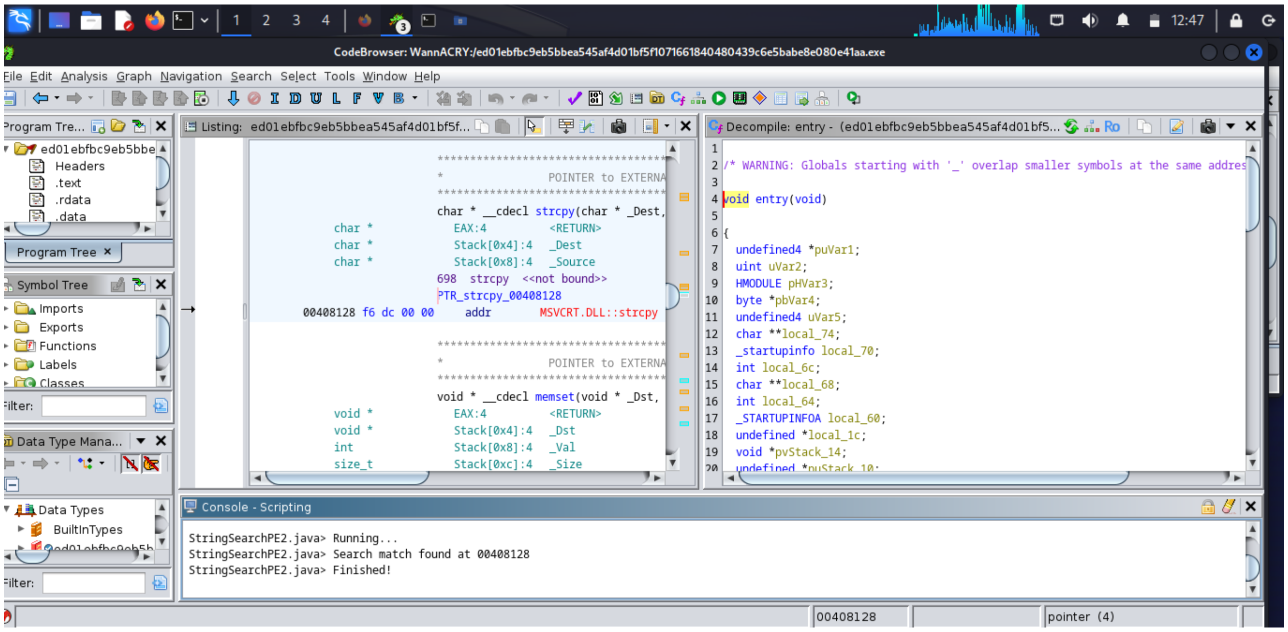
```
earchPE2 extends GhidraScript {

Exception {
 null) {
ln("NO CURRENT ADDRESS");
n;

ogram == null) {
ln("NO CURRENT PROGRAM");
        return;

    String search = askString("Text Search", "Enter search string: ");
    Address address = find(search);
```

Console - Scripting

```
    ^
Function_Vul.java:14: error: unclosed character literal
    'fwrite': "# Can be used for buffer overflow"
          ^

StringSearchPE2.java> Running...
```

004077ba | entry | PUSH EBP

Fig 19: String Search Interface



CodeBrowser: WannACRY:/ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa.exe

```
                     *************************************
                     *              POINTER to EXTERNA
                     *************************************
                     char * __cdecl strcpy(char * _Dest,
     char *             EAX:4         <RETURN>
     char *             Stack[0x4]:4  _Dest
     char *             Stack[0x8]:4  _Source
                     698  strcpy  <<not bound>>
                     PTR_strcpy_00408128
   00408128 f6 dc 00 00     addr        MSVCRT.DLL::strcpy

                     *************************************
                     *              POINTER to EXTERNA
                     *************************************
                     void * __cdecl memset(void * _Dst,
     void *             EAX:4         <RETURN>
     void *             Stack[0x4]:4  _Dst
     int                Stack[0x8]:4  _Val
     size_t             Stack[0xc]:4  _Size
```

Decompile: entry - (ed01ebfbc9eb5bbea545af4d01bf5...

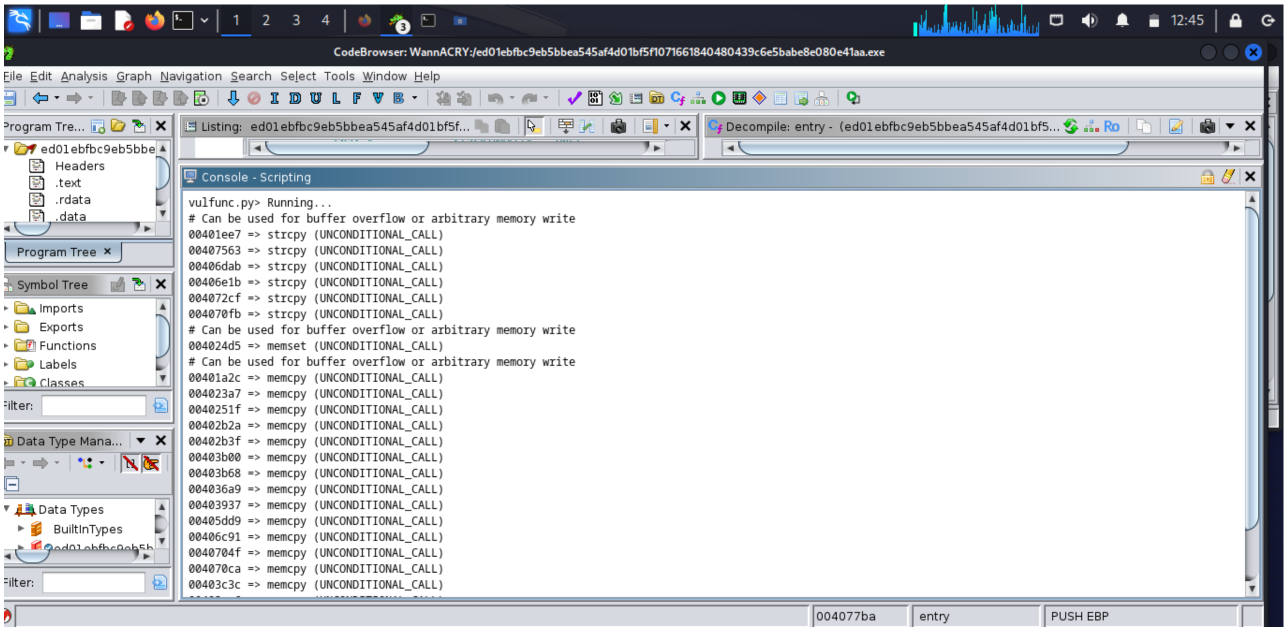
```

/* WARNING: Globals starting with '_' overlap smaller symbols at the same addres

void entry(void)

{
  undefined4 *puVar1;
  uint uVar2;
  HMODULE pHVar3;
  byte *pbVar4;
  undefined4 uVar5;
  char **local_74;
  _startupinfo local_70;
  int local_6c;
  char **local_68;
  int local_64;
  _STARTUPINFOA local_60;
  undefined *local_1c;
  void *pvStack_14;
```

Console - Scripting

```
StringSearchPE2.java> Running...
StringSearchPE2.java> Search match found at 00408128
StringSearchPE2.java> Finished!
```

00408128 | | pointer (4)

Fig 20: String Search Console Run


CodeBrowser: WannACRY:/ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa.exe
File Edit Analysis Graph Navigation Search Select Tools Window Help
Program Tree
Headers
.text
.rdata
.data
Symbol Tree
Imports
Exports
Functions
Labels
Classes
Filter:
Data Type Mana...
Data Types
BuiltInTypes
Listing: ed01ebfbc9eb5bbea545af4d01bf5f...
Decompile: entry - (ed01ebfbc9eb5bbea545af4d01bf5...
Console - Scripting
004077ba
entry
PUSH EBP

```
vulfunc.py> Running...
# Can be used for buffer overflow or arbitrary memory write
00401ee7 => strcpy (UNCONDITIONAL_CALL)
00407563 => strcpy (UNCONDITIONAL_CALL)
00406dab => strcpy (UNCONDITIONAL_CALL)
00406e1b => strcpy (UNCONDITIONAL_CALL)
004072cf => strcpy (UNCONDITIONAL_CALL)
004070fb => strcpy (UNCONDITIONAL_CALL)
# Can be used for buffer overflow or arbitrary memory write
004024d5 => memset (UNCONDITIONAL_CALL)
# Can be used for buffer overflow or arbitrary memory write
00401a2c => memcpy (UNCONDITIONAL_CALL)
004023a7 => memcpy (UNCONDITIONAL_CALL)
0040251f => memcpy (UNCONDITIONAL_CALL)
00402b2a => memcpy (UNCONDITIONAL_CALL)
00402b3f => memcpy (UNCONDITIONAL_CALL)
00403b00 => memcpy (UNCONDITIONAL_CALL)
00403b68 => memcpy (UNCONDITIONAL_CALL)
004036a9 => memcpy (UNCONDITIONAL_CALL)
00403937 => memcpy (UNCONDITIONAL_CALL)
00405dd9 => memcpy (UNCONDITIONAL_CALL)
00406c91 => memcpy (UNCONDITIONAL_CALL)
0040704f => memcpy (UNCONDITIONAL_CALL)
004070ca => memcpy (UNCONDITIONAL_CALL)
00403c3c => memcpy (UNCONDITIONAL_CALL)
```

Fig 21: Vulnerable Function Analysis Console Run

# CHAPTER-6: PROJECT OUTCOME AND APPLICABILITY

## 6.1 PROJECT OUTCOME

This chapter describes the results and relevance of our malware analysis effort. This section provides a complete review of the project's findings, including effect assessment, threat intelligence, and mitigation techniques. Using the Ghidra reverse-engineering tool, we discovered a method for deconstructing and reconstructing this communication. This Reverse-engineering exercises will be shared with collaborators and partners, including professionals, faculty, and students, to establish a common ground for studying and researching malware, analysis tools, and techniques to harden systems, recover after compromise, and detect and protect systems in the future.

### ANALYSIS RESULTS

- Classification: The malware was identified as a version of a recognized threat family, providing context for the way it acts and possible damage.

- Identifying strange Activity: We discovered strange functions, strings, and encryption algorithms employed by the malware.

- Analyzing Malware Behavior: Because Ghidra is more of a decompiler, it helped the generation of human-readable code from the disassembled instructions, allowing for a better understanding of the malware's behavior.

## IMPACT ASSESMENT

Our investigation of the malware sample with Ghidra raised serious issues. Using Ghidra's static analysis capabilities, we analyzed the malware code to discover its inner workings, identifying suspicious strings and API calls that indicated a potential data leakage. The malware's obfuscation techniques indicated an aim to bypass traditional security measures, emphasizing the importance of proactive protection. Ghidra's decompiler function allowed for a more in-depth knowledge of the malware's logic, while research of Windows APIs revealed its destructive purpose and possible impact, especially in instances such as ransomware, where targeted file encryption was clear. Overall, the investigation highlighted the importance of strong security measures to address the sophisticated threats posed by the malware.

### Mitigation Strategies

Based on our analysis results, we recommend the following mitigation strategies:

- Network Traffic Monitoring: Constantly monitor network traffic for inbound and outbound intrusion attempts in order to detect and prevent malware, Denial-of-Service attacks, botnets, and other potential threats. Properly configured firewalls and threat intelligence systems can proactively detect and stop harmful activities.

- Application Control: Use application control to prohibit the execution of unauthorized or malicious programs such as.exe files, DLLs, scripts, and installers. This helps prevent the unwanted execution of software, regardless of the source.

- Educate the users: Provide employees with frequent training on cybersecurity best practices, common malware attack vectors, and the need of reporting unusual activity. A well-informed workforce provides a strong shield against social engineering attempts.

## 6.2 APPLICABILITY

In this section, we investigate how the project findings can be implemented in a variety of scenarios within the field of cybersecurity and threat management.

- Reverse Engineering: Ghidra is used to investigate previously released software and attempt to determine how it works, even if the original code is not available. This is useful for understanding the behavior of software when no source code is supplied.

- Software Vulnerabilities: Using Ghidra to analyze software vulnerabilities allows security researchers to create exploits that obtain unauthorized access or control over a system.

- Incident Response: When a security breach happens, Ghidra can be used to investigate any malware discovered on the systems that were compromised. This helps to identify the extent of the breach and how to respond effectively.

# CHAPTER 7: CONCLUSION

## 7.1 Conclusion

The report on the "Malware Analysis using Ghidra" project provides an overview of malware, its types, objectives, methodology, and results. Finally, we discuss the future of malware analysis in cyber security.

## 7.2 Project Contribution

Our contributions to the field of cybersecurity by this project are: □ □ □

- We successfully identified and classified malware, gaining insights on its threat level and possible impact.
- Our static analysis uncovered the malware's real-time behavior, revealing its impact and intent on our system.□
- Encouraged additional study and collaboration in the field of malware analysis, utilizing Ghidra to boost cybersecurity efforts.
- Developed an organized approach for evaluating malware samples using Ghidra, including approaches for studying code, recognizing suspicious activities, and documentation of findings.

## 7.3 Future Scope

Malware analysis projects using Ghidra are extremely beneficial for real-world cybersecurity. Ghidra functions similarly to a detective tool, assisting professionals in understanding how malicious software operates. Using Ghidra to gain insight into malware code allows professionals to determine what the infection is attempting to achieve, which is critical for preventing cyber attacks in the real world. Ghidra excels at detecting unusual code, such as secret messages or clever ways for malware to hide. This enables cybersecurity specialists to identify possible threats and safeguard systems from attacks. It's like having a specific instrument to detect concealed risks in the digital world.

## 7.4 References

1. Hands-on Cybersecurity Studies: Uncovering and Decoding Malware Communications—

2. Jaime C Acosta,; Daniel E Krych,; Malware Analysis with Ghidra

3. Chronis Anastasios,; Dr. Panayiotis Yannakopoulos,; Malware Analysis and Reverse Engineering

4. The BlackBerry Cylance Threat Research Team,: Code Analysis With Ghidra: An Introduction

5. Ankit Singhal,; Saathwick Venkataramalingam,:Malware Analysis and Reverse Engineering: Unraveling the Digital Threat Landscape

6. Ruchi Tuli,;ANALYZING NETWORK PERFORMANCE PARAMETERS USING WIRESHARK

7. 23. Singhal, A., Chopra, A. (2021). DISTRIBUTED ENCRYPTION AND DECRYPTION STANDARDS - A CONTEMPORARY DISTRIBUTED CRYPTOGRAPHIC ALGORITHM. International Journal of Advance and Innovative Research

8. 15. Zhang, B. (2021). Research summary of anti-debugging technology. Journal of Physics: Conference Series

9. Singhal, A., Kharb, L. (2023). Need of hour: Hybrid encryption and decryption standards (heads) algorithm for Data Security. Studies in Autonomic, Data-Driven and Industrial Computing