Reverse Engineering Challenges solved step-by-step to demonstrate the many uses of reverse engineering

Joel Maldonado Rivera
Advisor: Jeffrey Duffany
Computer Science Department

Abstract

The field of reverse engineering has seen many different applications such as analysis of computer viruses and malware such as trojans, worms, viruses, ransomware, and so on. Other uses involve analyzing legacy code to possibly recreate the code in a more modern program and can even be used to test the quality of software. These are just some of the uses of reverse engineering, which should be discussed and be more well known throughout people who practice coding. There are many different approaches to reverse engineering since one can use tools such as IDA, CFF Explorer, Ghidra, Hopper, GDB, and many others in order to examine the programs. Being able to properly understand how to use these tools will help in the proper understanding of what the code is doing and how it's behaving which will be better demonstrated by solving reverse engineering challenges and explaining the methodology behind how they were solved.

Introduction

Reverse engineering is an area that has seen many different implementations and approaches which has helped tackle different problems such as working with legacy code, analyzing malware, and has also been used to improve existing software. Despite the wide range of implementations that reverse engineering offers it is seldom taught as a core class in many programs which is why bringing more awareness to it and being able to demonstrate to other computer scientist or individuals the possibilities and uses it brings is what this project hopes to demonstrate.

GDB will be the tool used to solve six challenges from the crackmes.one website.

- GDB which is also known as the GNU debugger, can be used in programming languages such as C, C++, Go, among others. With GDB you can disassemble specific parts of code one wants to analyze into assembly language, and this can be used to interpret the code and reverse engineer it.

- Assembly language is a low-level programming language which was created with the purpose to be able to directly communicate with a computer's hardware and is also readable by humans. It is used to translate high-level programming languages, such as python, to machine language, in other words to 1's and 0's which represent on or off electrical states to the machine.

- The crackmes.one page is a website that allows for any reverse engineer to upload their challenges in order to help others practice and learn reverse engineering skills, techniques and to improve what they know. The website also allows you to submit your solution and offers a variety of challenges which have varying difficulty and ways to be solved.

Hopefully, this will be able to help demonstrate the importance of reverse engineering and the different applications it can be used for. Which is the reason why it should be better known by the computer science students in order for them to be able to capitalize on the benefits of knowing reverse engineering concepts and being able to apply them for their use. Students especially interested in cybersecurity may want to practice their reverse engineering skills.

Problem

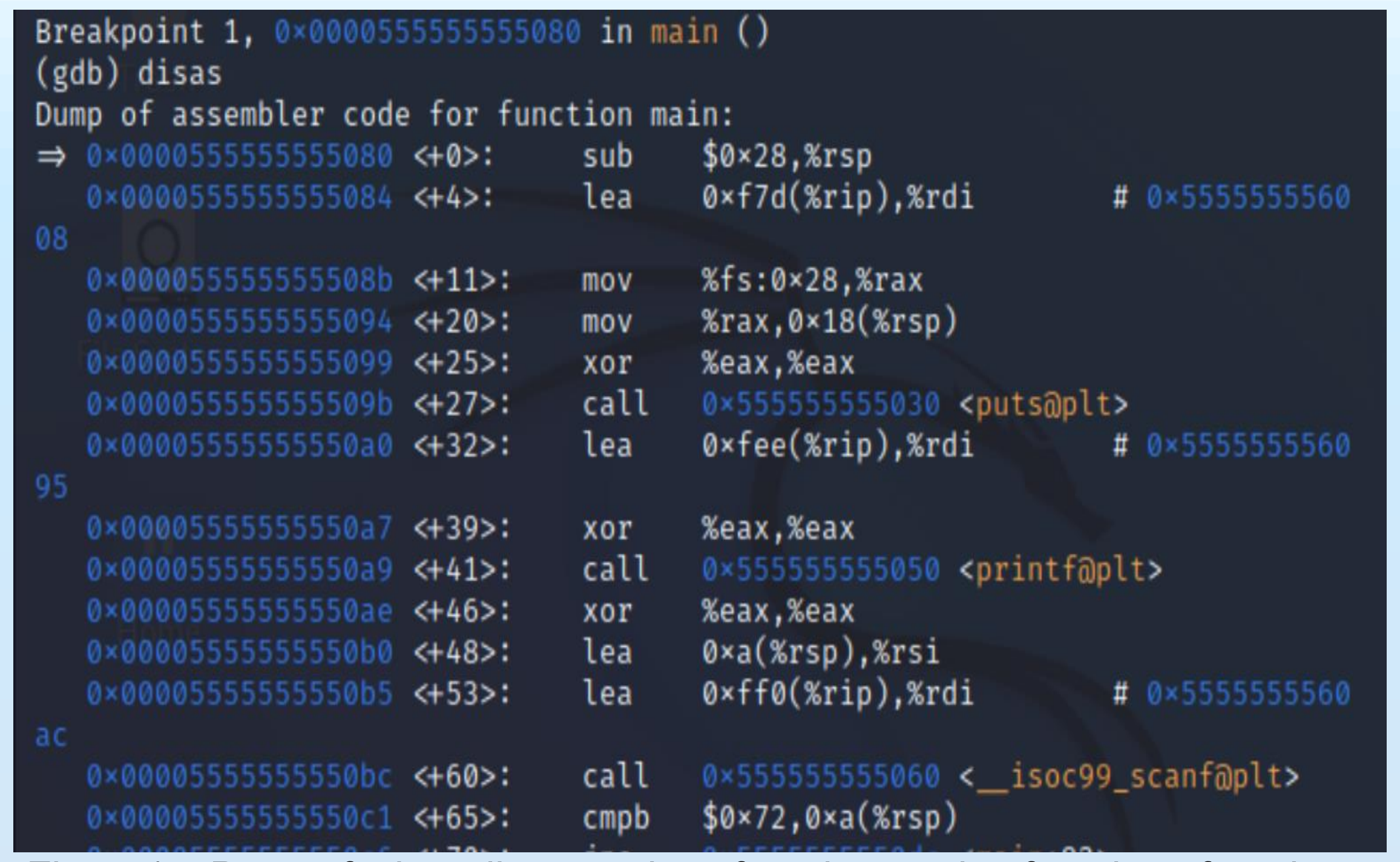
This project aimed to solve six challenges from the crackmes.one page which are all referenced. All of the challenges are unique and are a good starting place for those who have limited to no experience in reverse engineering since they help the user start getting used to using GDB and to the assembly language. These problems have a writeups that were made as part of the Design Project as well. Each write up explains each challenge and the assembly language generated by these in more depth and in a step-by-step manner in order to allow people who are interested in learning to have a guide

Methodology

In order to solve problems that were chosen from the crackmes.one site, they were first analyzed by using commands such as the file command in order to get more details about what type of executable file was being worked on and then proceeding to use GDB in order to take a much more in-depth analysis of the executable file. GDB has many different commands that have different uses in helping better understand the code such as:

- **disassemble(disas)** which allows the user to disassemble a specific function or a function fragment thus allowing a closer look at the assembly code to that specific area.
- **breakpoint** which allows the user to specify a breakpoint which will make the execution of the program halt once reached.
- **next instruction(ni)** which allows the user to go to the following instruction in the assembly code therefore giving the user a chance to verify changes that occurred.
- the **examine(x/..)** instruction which allows the examination of the provided memory but also allows one to place a flag in order to indicate how the values should be represented before it is displayed.
- **info functions** in order to see debugging symbols that can be accessed in order to help with the analysis of the program.

These challenges are not stripped executables, which allows for the use of info functions command to find debugging symbols and functions to set the breakpoints. For every challenge, these commands can be used in order to view functions of interest and set breakpoints for them to then go and disassemble them in order to analyze what the code is doing. By understanding the assembly and verifying areas in memory with the examine command the solutions of the challenges were determined.



```
Breakpoint 1, 0×0000555555555080 in main ()
(gdb) disas
Dump of assembler code for function main:
⇒ 0×0000555555555080 <+0>:     sub    $0×28,%rsp
  0×0000555555555084 <+4>:     lea    0×f7d(%rip),%rdi        # 0×5555555560
08
  0×000055555555508b <+11>:    mov    %fs:0×28,%rax
  0×0000555555555094 <+20>:    mov    %rax,0×18(%rsp)
  0×0000555555555099 <+25>:    xor    %eax,%eax
  0×000055555555509b <+27>:    call   0×555555555030 <puts@plt>
  0×00005555555550a0 <+32>:    lea    0×fee(%rip),%rdi        # 0×5555555560
95
  0×00005555555550a7 <+39>:    xor    %eax,%eax
  0×00005555555550a9 <+41>:    call   0×555555555050 <printf@plt>
  0×00005555555550ae <+46>:    xor    %eax,%eax
  0×00005555555550b0 <+48>:    lea    0×a(%rsp),%rsi
  0×00005555555550b5 <+53>:    lea    0×ff0(%rip),%rdi        # 0×5555555560
ac
  0×00005555555550bc <+60>:    call   0×555555555060 <__isoc99_scanf@plt>
  0×00005555555550c1 <+65>:    cmpb   $0×72,0×a(%rsp)
```

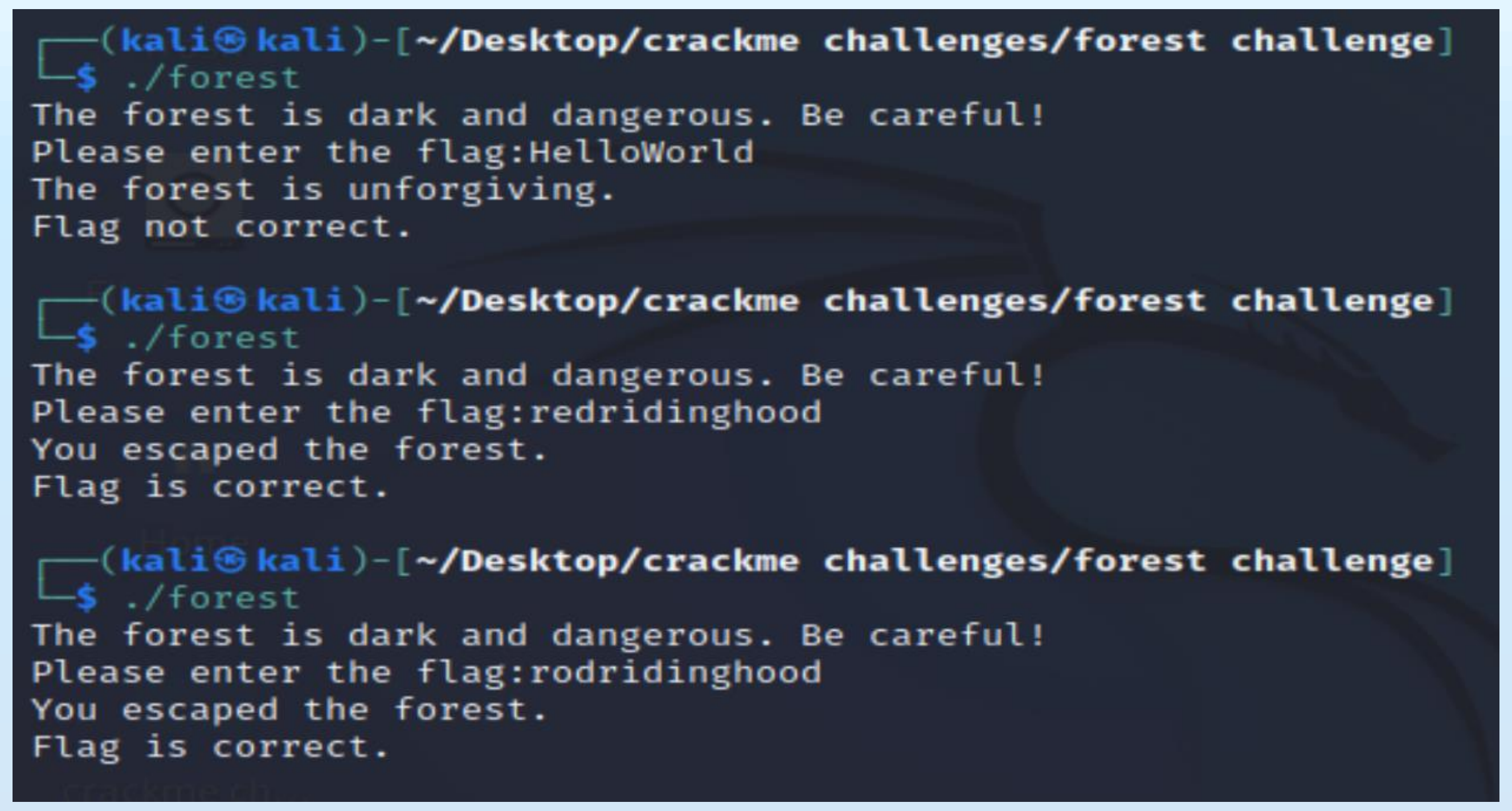
Figure1: Part of the disassembly for the main function for the , MKesenheimer's Forest challenge

Results and Discussion

Using the methodology that was described all the challenges that were chosen were solved by using GDB and a guide was made in order to show the solutions and how each challenge's solution was reached. Reverse engineering has various applications which not many computer scientists are aware of such as using it to test the quality of software, verifying legacy code in order to create a newer version of said code, malware analysis, and so on. The step-by-step guide will hopefully help those who are interested understand the basics of assembly and reverse engineering concepts which in turn may provide them with better insight on the topic of reverse engineering and the various uses it has.

Challenge	Solution
MKesenheimer's Forest	'redridinghood'
Bueb810'sgugus the first challenge	'gu!gu?s'
NomanProdhan's license checker 0x02	"NomanProdhan" "KS-LICENSE-KEY-2021-REV-2"
eventhorizon02's Super Easy	be any number that is at least 3 digits long and is a prime number
SilentWRaith's lockcode	sum of the characters of the strings must be equal to 2977 for the password to be accepted. Example:'HelloThereGeneralKenobiSkywaHH'.
ezman's easy keyg3nme	1223

Table1: Shows the name of each of the crackmes challenges along with solutions to for each.



```
┌──(kali㉿kali)-[~/Desktop/crackme challenges/forest challenge]
└─$ ./forest
The forest is dark and dangerous. Be careful!
Please enter the flag:HelloWorld
The forest is unforgiving.
Flag not correct.

┌──(kali㉿kali)-[~/Desktop/crackme challenges/forest challenge]
└─$ ./forest
The forest is dark and dangerous. Be careful!
Please enter the flag:redridinghood
You escaped the forest.
Flag is correct.

┌──(kali㉿kali)-[~/Desktop/crackme challenges/forest challenge]
└─$ ./forest
The forest is dark and dangerous. Be careful!
Please enter the flag:rodridinghood
You escaped the forest.
Flag is correct.
```

Figure2: Showing how the Forest challenge was run with an incorrect answer and with two different correct answers.

Conclusions

The challenges that were chosen were all able to be solved by using GDB and interpreting the assembly which was disassembled. There are tools that make this process easier but having a core understanding of the assembly and using GDB set up good fundamentals. A step-by-step documentation on how to solve the challenges was also created hopefully motivating computer science and cybersecurity students gain some basic knowledge of reverse engineering concepts.

Future Work

There are many different proposals that can be considered for future works such as using specific tools such as IDA or Ghirdra to highlight the capabilities of said tool and how they are able to fend off some of the techniques that exists to try to stop the reverse engineering process from being achieved. Reverse engineering can also be utilized for malware analysis, which is more pertinent to cybersecurity, to understand how the malware work and what vulnerabilities are being exploited to be able to patch and counter the use of these. A future project where some diverse types of malwares are analyzed, handled and properly reverse engineered would be ideal, especially to highlight why it's important for cybersecurity.

Acknowledgements

I would like to acknowledge and thank Dr. Jeffrey Duffany for his recommendation and guidance during the process of the Design Project. I'm also thankful for the master's program overall I'm also thankful to the Computer Science Department for the growth they have let me develop.

References

[1]*"Assembly - introduction," Tutorials Point. [Online]. Available: https://www.tutorialspoint.com/assembly_programming/assembly_introduction.htm*
[2]*bueb810, "gugus the first," Crackmes. [Online]. Available: https://crackmes.one/crackme/61e9983133c5d413767ca5ac*.
[3] *"Ethics in computing," Reverse engineering. [Online]. Available: https://ethics.csc.ncsu.edu/intellectual/reverse/study.php*
[4]*eventhorizon02, "super_easy," Crackmes. [Online]. Available: https://crackmes.one/crackme/611e9bfb33c5d45db85dc2d7.*
[5]*ezman, "easy keyg3nme," Crackmes. [Online]. Available: https://crackmes.one/crackme/5da31ebc33c5d46f00e2c661.*
[6]*GDB: The GNU project debugger*. [Online]. Available: https://www.sourceware.org/gdb/ .
[7]*J. Fernando, "Assembly language definition," Investopedia, 21-Sep-2021. [Online]. Available: https://www.investopedia.com/terms/a/assembly-language.asp*.
[8]*MKesenheimer, "Forest," Crackmes. [Online]. Available: https://crackmes.one/crackme/60f31f1d33c5d42814fb3381*.
[9]*NomanProdhan, "License Checker 0x02," Crackmes. [Online]. Available: https://crackmes.one/crackme/61c62bde33c5d413767ca0a0* .
[10] *Oracle VM VirtualBox*. [Online]. Available: https://www.virtualbox.org/.

[11] *R. Singh , "A Review of Reverse Engineering Theories and Tools." [Online]. Available: https://idc-online.com/technical_references/pdfs/mechanical_engineering/A%20Review%20of.pdf*.
[12] S. Megira and F. W. Wibowo, "Malware analysis and detection using reverse engineering technique," *Journal of Physics: Conference Series*, 12-Mar-2019. [Online]. Available: https://www.academia.edu/38540935/Malware_Analysis_and_Detection_Using_Reverse_Engineering_Technique.
[13] *SilentWraith, "lockcode," Crackmes. [Online]. Available: https://crackmes.one/crackme/5fda4fa43c5d41f64dee37b*
[14] "What is Kali Linux?: Kali linux documentation," *Kali Linux*. [Online]. Available: https://www.kali.org/docs/introduction/what-is-kali-linux/.