Reverse Engineering

Final Project

Instructor:	G Leaden	**Due:**	Final Start Time
Email:	g.leaden1@marist.edu	**Place:**	Hancock 2023

Goals:

This project is the culmination of all of the topics reviewed in the course. You will take a compiled program of unknown origin and walk through understanding and cracking said program (binary).

Instructions:

1. Download an executable from the website https://crackmes.one/. I reccomend a difficulty level of 2. If you choose to go higher than 2, you will be awarded an additional 5% on your grade for the assignment. If you choose a program that is difficulty 1, you will be deducted 5% on your grade for the assignment.
2. Utilize the skills we have honed through this class to view the machine code of the executable, and understand how you can attain the desired output. I recommend reviewing the list of tools mentioned in Chapter 7 of our book.
3. Take notes through this process, it will greatly help you when completing the next steps.
4. Create a guide on how the program works. Show machine code, explain it, and walk through the process to the correct output. Highlight mistakes made, and how they helped you through the process.
5. Include a preamble explaining the executable, and outlining a specific topic that was either especially compelling to you, or relevant to the reversing of the program.

This project can be submitted as any medium you would like, as long as the instructions are followed and requirements are met. Be creative!

Example Mediums:

- Video
- Poster (Example Included)
- PowerPoint
- Paper
- Mural
- etc.

Submitting:

Email me your final project BEFORE the due date.

Grading Rubric:

Topic Summary	15%
Flow of Visual Guide (does it make sense)	20%
Explanation of Assembly Instructions / Memory Management	20%
Walkthrough and Explination of the Binary	45%

Example:

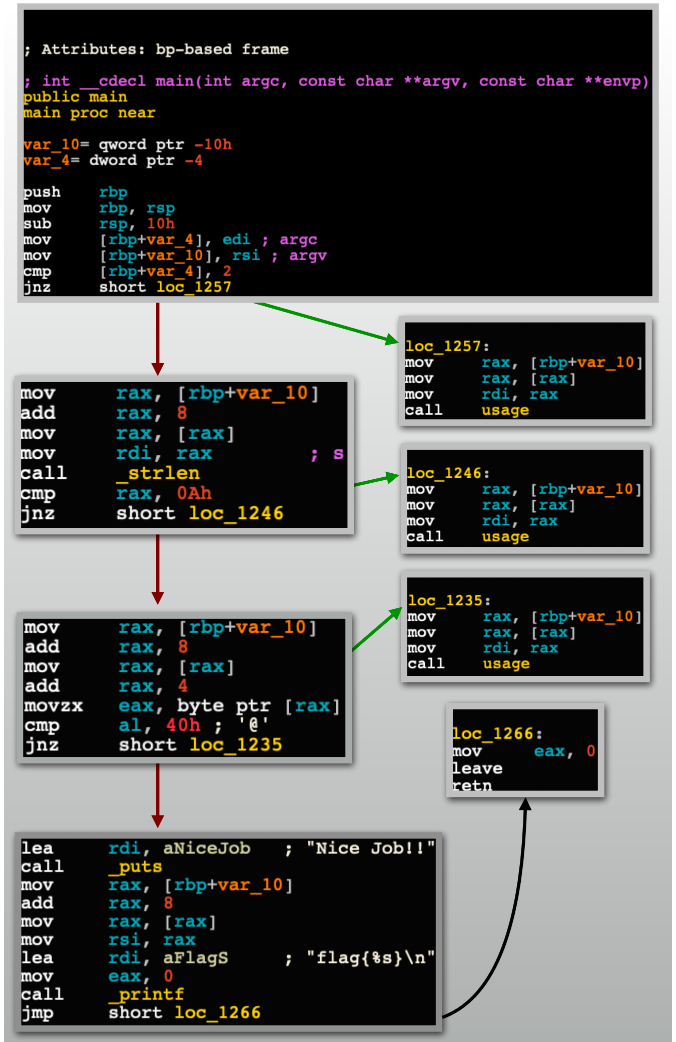
An example is provided on the next page and link to the .keynote source file is available on the website.

Reverse Engineering Final Project

https://crackmes.one/crackme/5b8a37a433c5d45fc286ad83

G Leaden

Professor. G Leaden



```
; Attributes: bp-based frame

; int __cdecl main(int argc, const char **argv, const char **envp)
public main
main proc near

var_10= qword ptr -10h
var_4= dword ptr -4

push    rbp
mov     rbp, rsp
sub     rsp, 10h
mov     [rbp+var_4], edi ; argc
mov     [rbp+var_10], rsi ; argv
cmp     [rbp+var_4], 2
jnz     short loc_1257
```

```
mov     rax, [rbp+var_10]
add     rax, 8
mov     rax, [rax]
mov     rdi, rax            ; s
call    _strlen
cmp     rax, 0Ah
jnz     short loc_1246
```

```
mov     rax, [rbp+var_10]
add     rax, 8
mov     rax, [rax]
add     rax, 4
movzx   eax, byte ptr [rax]
cmp     al, 40h ; '@'
jnz     short loc_1235
```

```
lea     rdi, aNiceJob    ; "Nice Job!!"
call    _puts
mov     rax, [rbp+var_10]
add     rax, 8
mov     rax, [rax]
mov     rsi, rax
lea     rdi, aFlagS      ; "flag{%s}\n"
mov     eax, 0
call    _printf
jmp     short loc_1266
```

```
loc_1257:
mov     rax, [rbp+var_10]
mov     rax, [rax]
mov     rdi, rax
call    usage
```

```
loc_1246:
mov     rax, [rbp+var_10]
mov     rax, [rax]
mov     rdi, rax
call    usage
```

```
loc_1235:
mov     rax, [rbp+var_10]
mov     rax, [rax]
mov     rdi, rax
call    usage
```

```
loc_1266:
mov     eax, 0
leave
retn
```

Visual Guide

Topic Summary

Choose a topic that was either the most compelling for you, or one that relates heavily to the program you are cracking.

Summarize it.

Provide insight into why it was compelling / is relevant.

X64 Instructions, CPU Register Sizing - Add anything relevant to understanding the reversing here

`mov` *D,S*	Move source to destination	`movzx` *D,S*	Move source to destination with zero-extentsion
`add` *D,S*	Add source to destination	`lea` *D,S*	Loads source address into destination register
`call` *Label*	Push return address and jump to label	`leave`	Releases stack space allocated to frame
`cmp` S_2,S_1	Set condition codes according to S_1 - S_2	`push` *S*	Pushes source onto stack
`jnz` *Label*	Jump to label if not equal to 0	`ret`	Returns to calling procedure
`jmp` *Label*	Jump to label		

		AH (8 bits)	AL (8 bits)
RAX (64 bits)	EAX (32 bits)	AX (16 bits)	

Cracking the Code (Understanding the Binary)

The assembly language on the left is the lowest level code that's still "human-readable" before it is converted into 0s and 1s for the hardware to execute. This was obtained by disassembling an executable file. This file was once written in a high level programming language like C or C++ before being compiled and output as assembly language code.

This program is designed to take a password as an argument, and if the password meets the requirements, it will output "Nice Job!!", followed by "flag{<password>}". If the password does not meet the requirements, or the incorrect number of arguments are passed, the program will call a `usage` method that outputs:
"USAGE: ./<filename> <password>
 try again!"

There are three instances where the `usage` method is called. Each occurs after a `cmp` followed by a `jnz`.

1. `2  - [rbp+var_4]`
2. `10 - strlen(rdi)`
3. `@  - al`

Once we have a password that will satisfy each of these compare instructions with a result of 0, we will have achieved our goal. Starting with nothing but an executable file, we are able to identify a pattern that will always result in a successful password. Can you crack the code?