REVERSE ENGINEERING

Amateur Edition

COMP6841: Something Awesome

Nikil Singh

Table of Contents

Overview

General Summary of Project

Reverse engineering was undertaken as the topic of interest for the Something Awesome project in COMP6841. The aim of this is to learn reverse engineering principles by completing challenges from crackmes.one. Through this, skills such as understanding assembly, patching, analysis of programs and writing walkthroughs can be achieved.

Marking Criteria

Pass:

- Reverse engineer a program that has an algorithm and determine the algorithm.
- Would be obtained from crackmes or someone else will make it.
- This would be of level 1.

Credit:

- Reverse engineer programs such as login systems.
- This could be made by someone else or obtained from crackmes.
- In terms of level of difficulty would be level 2 to 3 on crackmes.
- Also, all of the above.

Distinction:

- Successfully reverse engineer programs from crackmes, of level 4 to level 5.
- Also, all of the above.

High Distinction:

- Successfully reverse engineer programs from crackmes that are level 5 and above. (Max on the site is 6 which is subtitled 'insane'.)
- Write a report on various methodologies of reverse engineering. This includes walkthroughs of programs reverse engineered and how it was solved.
- Also, all of the above.

Schedule

- Week 1-3: Determine proposal for something awesome and research the basics of reverse engineering.
- Week 4-5: Begin practice on crackme challenges of level 1-3, and write walkthroughs on them.
- Week 6-7: Possibly get someone to make a login system and attempt to reverse engineer that. Aim to find someone else doing reverse engineering and also make them a login system to reverse engineer. Begin attempting level 4 challenges.
- Week 8-9: Finalise the report regarding reverse engineering and attempt to reverse engineer level 5 challenges.

Reverse Engineering

Overview

Reverse engineering is the process of taking machine code that has been compiled and attempting to convert it into a more readable code, i.e. C Code and pseudocode. Essentially, we seek to understand the functionality of the program, often to extract certain information or bypass certain security features.

When written code (i.e. C) is compiled it is converted into machine code. Machine code or assembly code are formatted to allow for execution by a CPU. Furthermore, the machine code or assembly instruction are often operating specific. i.e. A program compiled on Windows will not be able to run on Linux. Compilation is a one-way process for compiled languages and cannot be exactly converted back. Machine code can be converted back into assembly however, it is more difficult to read and requires practice.

Assembly instructions perform various actions on registers. These include data movement, arithmetic operations and control-flow. Instructions are pieces of memory executed based on its address. Control flow is achieved by jumping or accessing instructions stored at a certain address. Jump conditions are used to determine which branch of instruction to continue towards. Addresses itself can be likened to the indices of an array, where the array is memory and memory addresses are the indices acting as a reference to an instruction.

The language to be used will be C and C++.

Disassemblers are tools which revert machine code into assembly code. Various disassemblers are IDA, Binary Ninja and GNU Debugger (GDB). Binary Ninja was chosen mostly due to its aesthetics in presenting assembly code and it has a free version. However, it can only be operated with in 25 minute sessions.

Decompilers attempt to convert compiled code back into pseudocode for further reconstruction. This gives a general outline of what the original source code looked like.

Assembly Instruction Cheat Sheet

For conditional jump instructions on signed data for arithmetic operations.

Instruction	Description
JE/JZ	Jump Equal or Jump Zero
JNE/JNZ	Jump not Equal or Jump Not Zero
JG/JNLE	Jump Greater or Jump Not Less/Equal
JGE/JNL	Jump Greater/Equal or Jump Not Less
JL/JNGE	Jump Less or Jump Not Greater/Equal
JLE/JNG	Jump Less/Equal or Jump Not Greater

For conditional jump instructions on unsigned data for logical operations.

Instruction	Description
JE/JZ	Jump Equal or Jump Zero
JNE/JNZ	Jump not Equal or Jump Not Zero
JA/JNBE	Jump Above or Jump Not Below/Equal
JAE/JNB	Jump Above/Equal or Jump Not Below
JB/JNAE	Jump Below or Jump Not Above/Equal
JBE/JNA	Jump Below/Equal or Jump Not Above

Register Cheat Sheet

Note: R prefix is for 64-bit, E prefix is for 32-bit, and neither in front is 16-bit.

General Registers

Note: “H” and “L” suffix on 8 bit registers stand for high and low byte.	
RAX, EAX, AX, AH, AL	Called the accumulator register. It is used for I/O port access, arithmetic, interrupt calls, etc …
RBX, EBX, BX, BH, BL	Called the base register. It is used as a base pointer for memory access. Gets some interrupt return values.
RCX, ECX, CX, CH, CL	Called the counter register. It is used as a loop counter and for shifts. Gets some interrupt values.
RDX, EDX, DX, DH, DL	Called the data register. It is used for I/O port access, arithmetic, some interrupt calls.

Segment Registers

Segment registers gold segment address of various items. Only in available in 16 values and can only be set by a general register or special instruction.	
CS	Holds the code segment in which the program is run. Changing its value may make the computer hang.
DS	Holds the data segment that your program accesses. Changing its value might give erroneous data.
ES, FS, GS	These are extra segment registers available for far pointer addressing like video memory and such.
SS	Holds the stack segment the program uses. Sometimes has the same value as DS. Changing its value can give unpredictable results, mostly data related.

Indexes and Pointers

Indexes and pointers and the offset part of and address. They have various uses, but each register has a specific function. They are sometimes used with a segment register to point to far address. Register with an “E” prefix can only be used in protected mode.	
ES: RDI, EDI, DI	Destination index register. Used for string, memory array copying and setting and for far pointer addressing with ES.
DS: RSI, ESI, SI	Source index register. Used for string and memory array copying.
SS: RBP, EBP, BP	Stack base pointer register. Holds the base address of the stack.
SS: RSP, ESP, SP	Stack pointer register. Holds the top address of the stack.
CS: EIP, EIP, IP	Index pointer. Holds the offset of the next instruction. It can only be read.

Function Parameters

RDI	First argument.
RSI	Second argument.
RDX	Third argument.
RCX	Fourth argument.
R8	Fifth argument.
R9	Sixth argument.

Variables

Bit	None	
Byte or Octet	Char	1
WORD	Short	2
DWORD	Integer	4
QWORD	Long	8

Walkthroughs

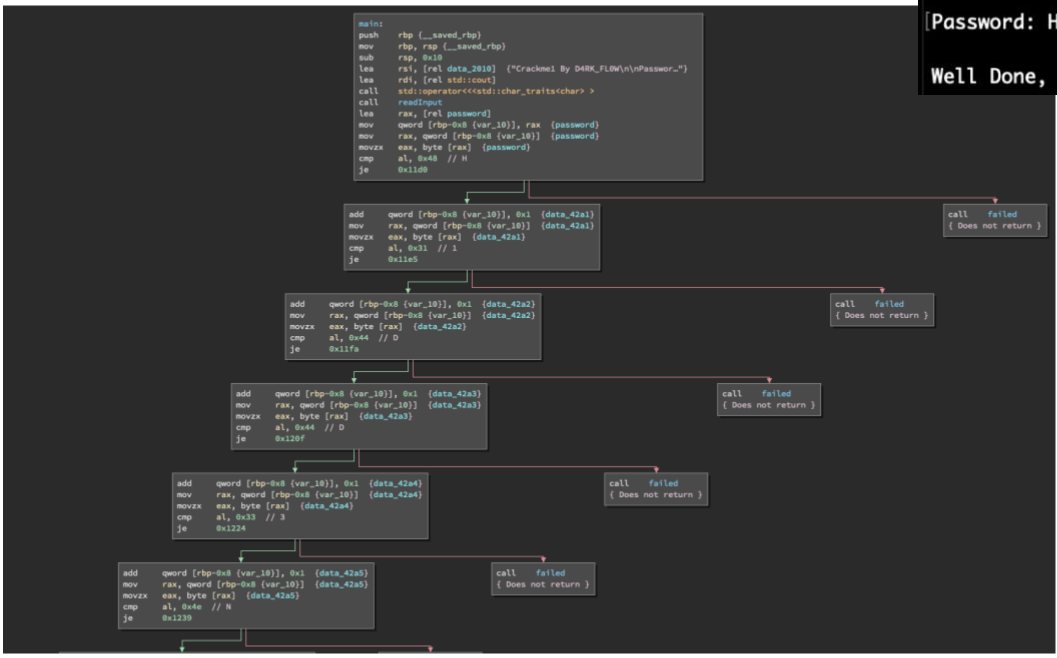
The following can be found on my GITHUB, see appendix A.

Level 1

Crack1 By Dark Flow

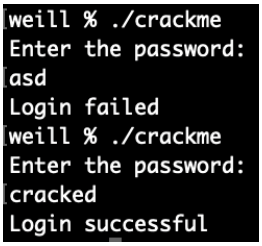
Link: https://www.crackmes.one/crackme/5c8e1a9533c5d4776a837ecf

The first reverse engineering exercise I’ve undertaken does have a walkthrough so I could learn how reverse engineering works. Looking at the disassembled code we can see there are a set of if statements. Those if statements seem to compare an input for a password letter by letter. The password ends up being “H1DD3N”.

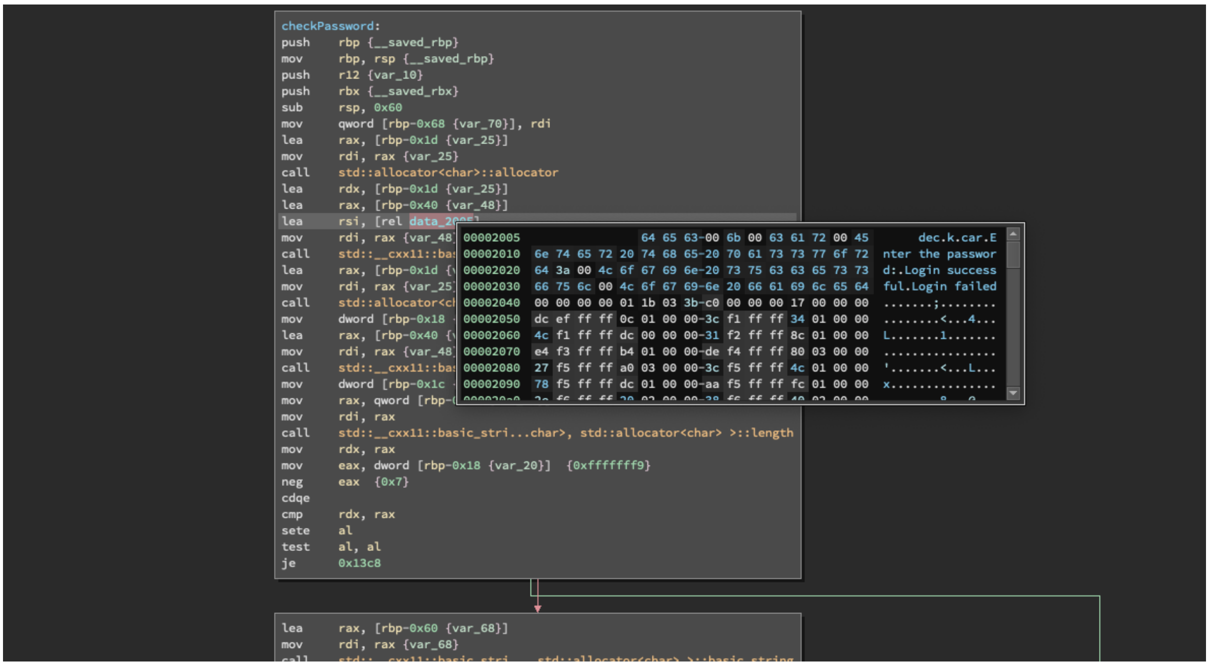


Crackme

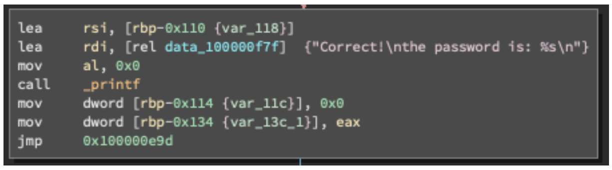
Link: https://crackmes.one/crackme/5c90a72d33c5d4776a837f07

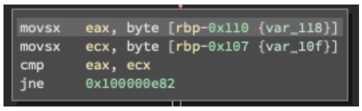


Upon inspection of the disassembled code, it is clear that it checks for a string of length 7. After a lot of researching on what different commands and register mean I found two variables. They contained the letters dec, k, car. I quickly realised that's an anagram for cracked. 'cracked' is also a 7 letter word. I entered the password and I was correct.

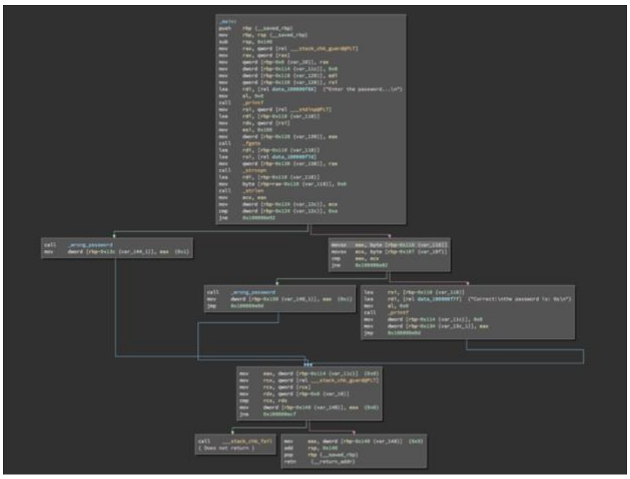


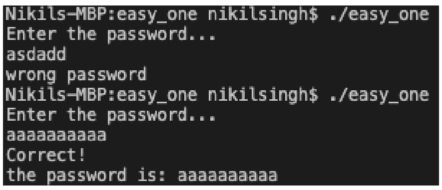
Easy One
Link: https://crackmes.one/crackme/5d443bb533c5d444ad3018b3





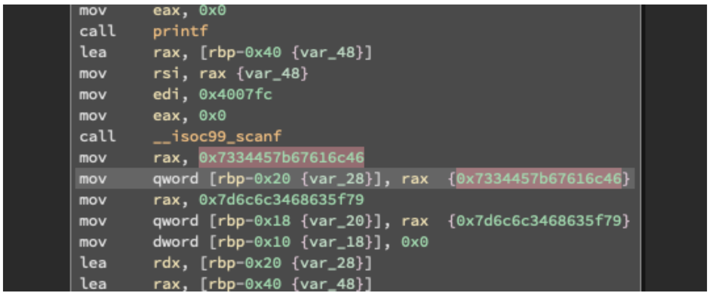
Upon running the program, I discovered I needed to enter a password. Upon disassembling the code, I found the if statement that determined whether the password was correct. Tracing backwards from this I discovered a comparison and that the input string is held in ecx. Next I realised that it is comparing the same string with each other. At this point I became stuck, and got a hint from the solution. The program used fgets and relies on the user buffer overflowing. The overflow data would go into the variable password. Essentially, the program checks whether the first letter in input matches the first letter to enter the buffer overflow. If it passes that, then the password is correct.

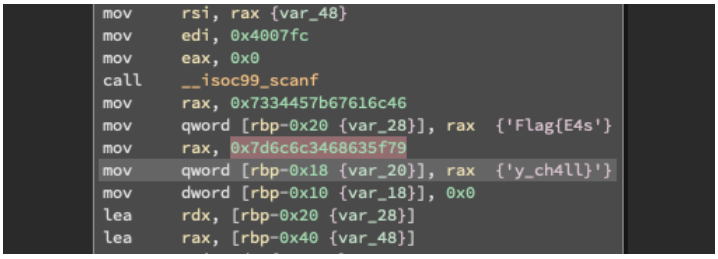


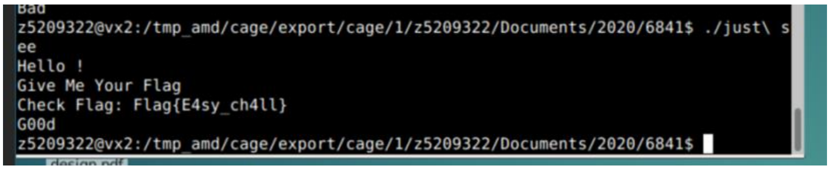


Just See
Link: https://crackmes.one/crackme/5b81014933c5d41f5c6ba944

This was a fairly simple one. I ran the program and it asked for a flag. I then opened the disassembled code and found it was comparing to extremely large addresses. But they were not addresses, but a string when viewed as character constants.

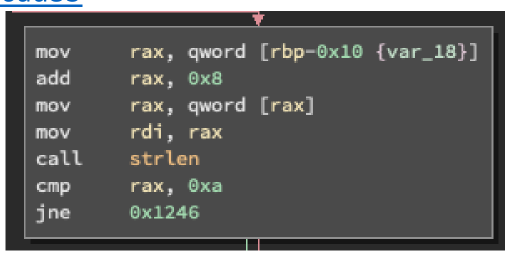


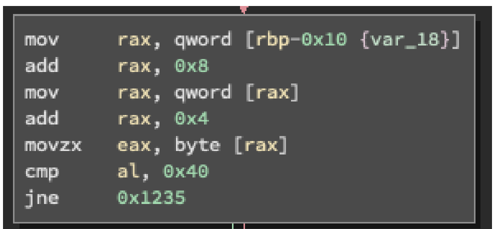




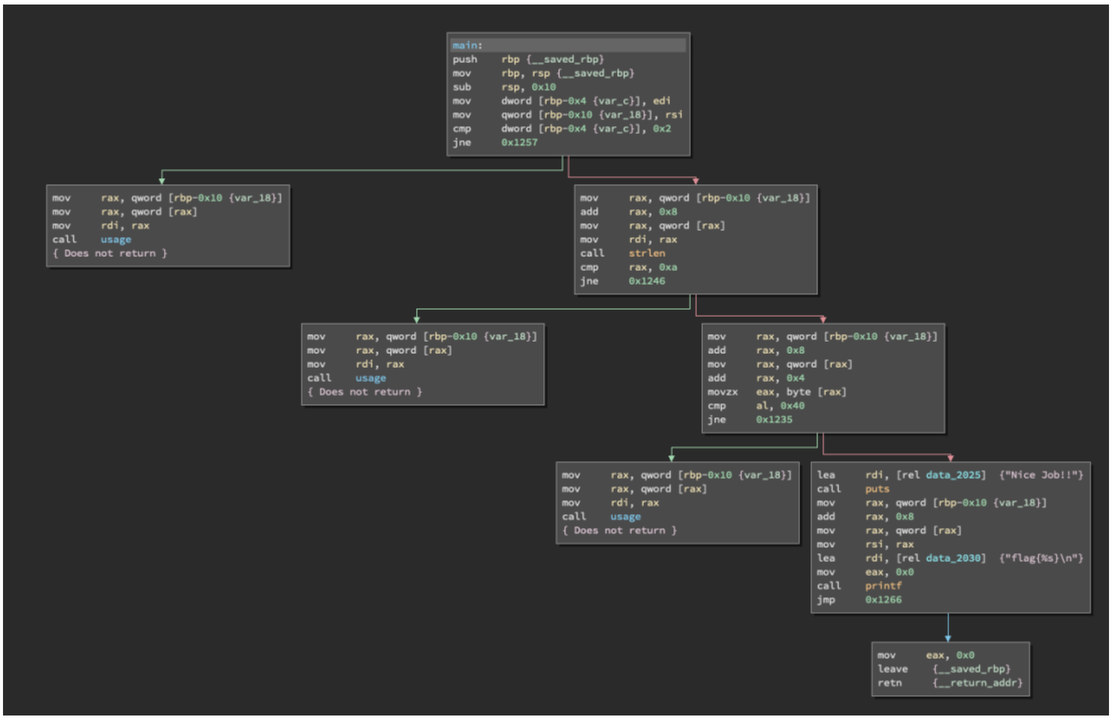
Rev50 Linux 64-Bit
Link: https://crackmes.one/crackme/5b8a37a433c5d45fc286ad83

Upon running the program, the program required me to provide an argument along with ./program. Reading the disassembled code, I also found the agreement has to be of a certain length from strlen. Using disassembled code and ltrace I could not find the required length. After looking for a bit longer I realised the size of string needed to be 10 from 0xa. This is the case since a is 10 in hex. I also found to access the flag section it compared something 0x40 which is @. So, I did a lot of guess and check and found nothing. So, I looked up at the solutions and said the fifth character needs to be @. This could possibly be the case from it moving 0x4 to rax, and then movzx eax, byte [rax].





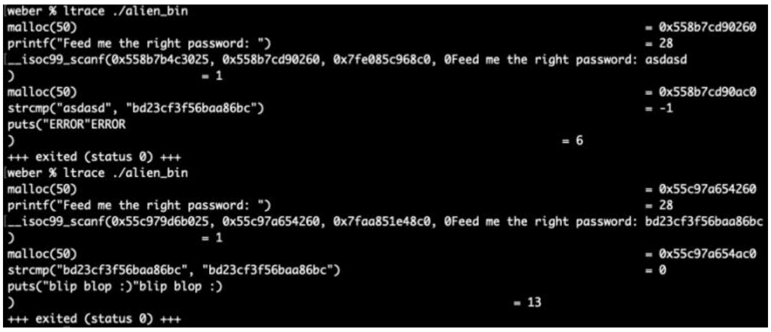
The flag was flag{aaaa@aaaaa} which happened to be my input.



Level 2

Alien Bin

Link: https://crackmes.one/crackme/5d78168833c5d46f00e2c428

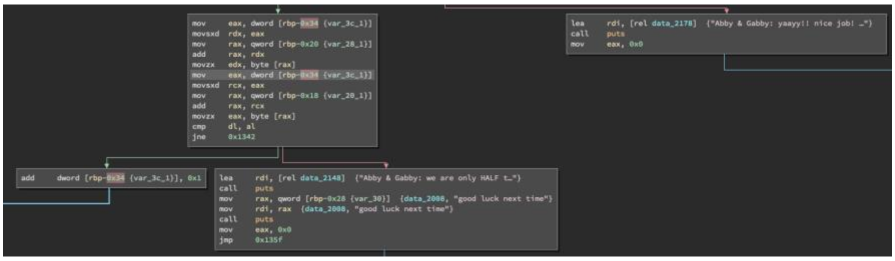


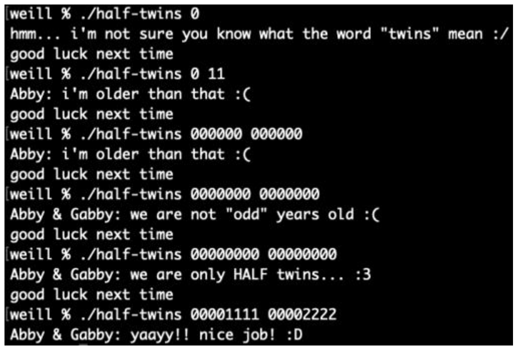
Running this through ltrace, I found a strcmp that compared the input to the following string: "bd23cf3f56baa86bc". I also found a function of the same name in the disassembled code. In the initial run, upon entering random text I got the message "ERROR". I then reran the code and entered that string. The output from that was "blip blop :)".

This was in the level 2 section, but based on how easily I solved it, it should be in level 1.

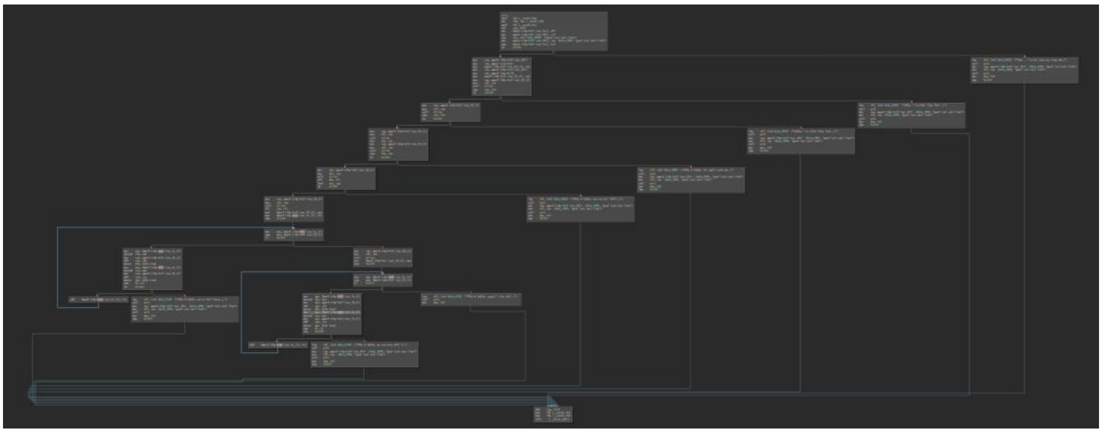
Half Twins

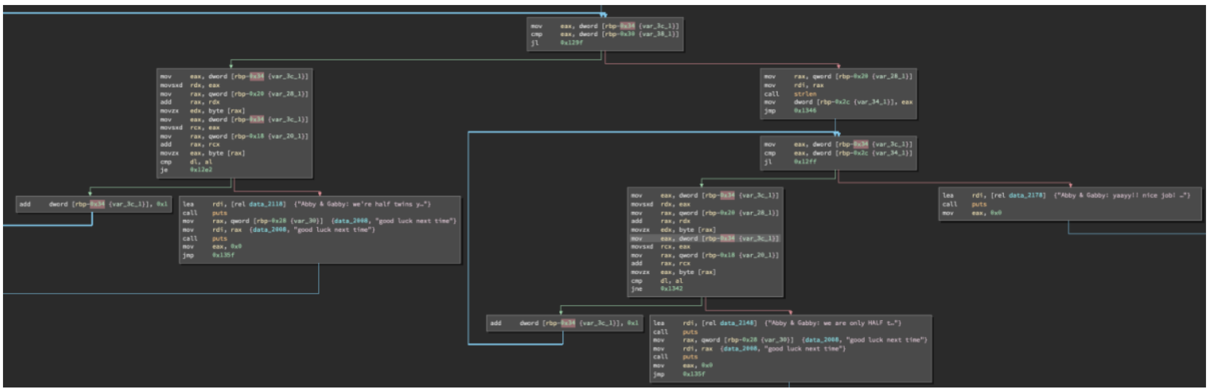
Link: https://crackmes.one/crackme/5dce805c33c5d419aa0131ae





I ran the program without any arguments and it said my input was incorrect. Next, I ran the program with two arguments 0 11 and it came up that abby was older than that. Looking at the disassembled code, strlen was called to measure length of the two inputs. Both inputs need to be of length 6 or greater. I then tried the inputs 000000 000000 and got that abby was older. I tried 0000000 and 0000000 both of which are of length 7, and got that abby and gabby are not odd years old. Next, I tried 00000000 00000000 which are length 8. The response was that abby and gabby were half twins. Looking at the flow of direction of the disassembled code this was the closest conditional statement to getting the correct output. So, if I passed this conditional statement I would have succeeded. After being stuck for a while, I had a look at the solution. I had everything correct. The part I missed was that it looped through and checked the first half of both inputs were the same and the second half were not the same. So a correct input would be “00001111” and “00002222”.

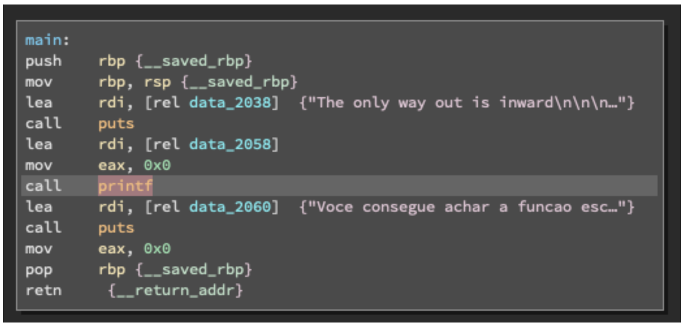




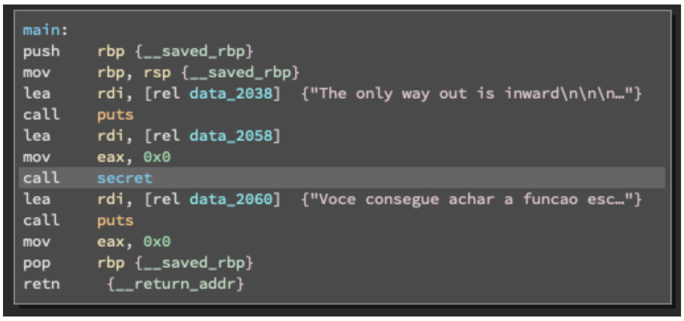
Hidden

Link: https://crackmes.one/crackme/5c11e2f333c5d41e58e0057a

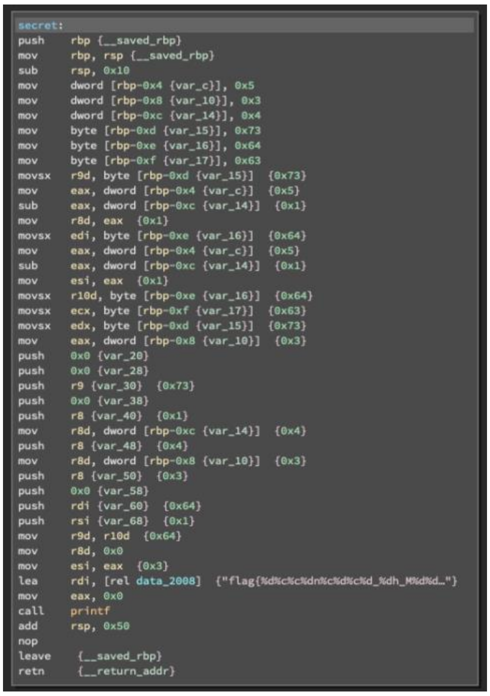
Running the program, it outputted "the only way out is inward". It also outputted "voce consegue achar a funcao escondida" which is Portuguese for "can you find the hidden function". Looking at the disassembled code I found the function called secret, however the function is never called. So, I patched the function call to printf to instead call the function secret. From this I received the flag: flag{3sc0nd1d0_3h_M41s_G0st0S0}.



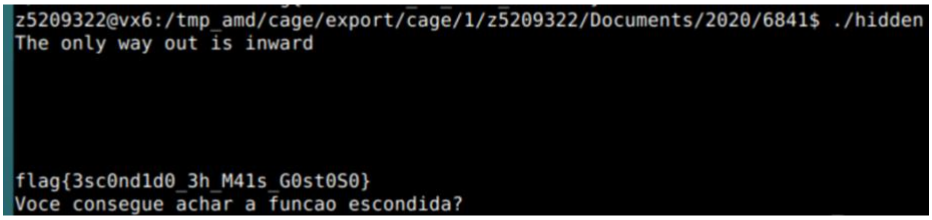
```
main:
push    rbp {__saved_rbp}
mov     rbp, rsp {__saved_rbp}
lea     rdi, [rel data_2038]  {"The only way out is inward\n\n\n…"}
call    puts
lea     rdi, [rel data_2058]
mov     eax, 0x0
call    printf
lea     rdi, [rel data_2060]  {"Voce consegue achar a funcao esc…"}
call    puts
mov     eax, 0x0
pop     rbp {__saved_rbp}
retn     {__return_addr}
```



```
main:
push    rbp {__saved_rbp}
mov     rbp, rsp {__saved_rbp}
lea     rdi, [rel data_2038]  {"The only way out is inward\n\n\n…"}
call    puts
lea     rdi, [rel data_2058]
mov     eax, 0x0
call    secret
lea     rdi, [rel data_2060]  {"Voce consegue achar a funcao esc…"}
call    puts
mov     eax, 0x0
pop     rbp {__saved_rbp}
retn     {__return_addr}
```

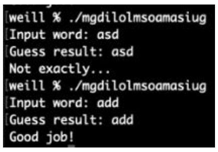


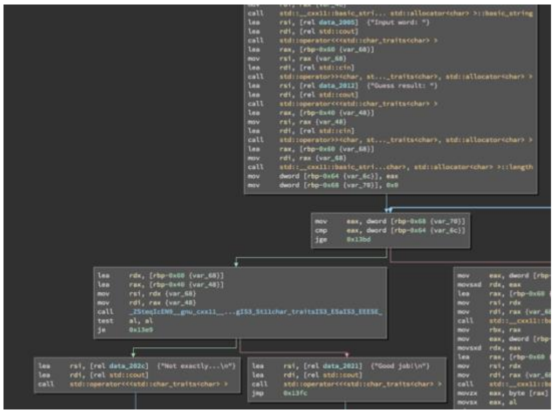
```
secret:
push    rbp {__saved_rbp}
mov     rbp, rsp {__saved_rbp}
sub     rsp, 0x10
mov     dword [rbp-0x4 {var_c}], 0x5
mov     dword [rbp-0x8 {var_10}], 0x3
mov     dword [rbp-0xc {var_14}], 0x4
mov     byte [rbp-0xd {var_15}], 0x73
mov     byte [rbp-0xe {var_16}], 0x64
mov     byte [rbp-0xf {var_17}], 0x63
movsx   r9d, byte [rbp-0xd {var_15}]  {0x73}
mov     eax, dword [rbp-0x4 {var_c}]  {0x5}
sub     eax, dword [rbp-0xc {var_14}]  {0x1}
mov     r8d, eax  {0x1}
movsx   edi, byte [rbp-0xe {var_16}]  {0x64}
mov     eax, dword [rbp-0x4 {var_c}]  {0x5}
sub     eax, dword [rbp-0xc {var_14}]  {0x1}
mov     esi, eax  {0x1}
movsx   r10d, byte [rbp-0xe {var_16}]  {0x64}
movsx   ecx, byte [rbp-0xf {var_17}]  {0x63}
movsx   edx, byte [rbp-0xd {var_15}]  {0x73}
mov     eax, dword [rbp-0x8 {var_10}]  {0x3}
push    0x0 {var_20}
push    0x0 {var_28}
push    r9 {var_30}  {0x73}
push    0x0 {var_38}
push    r8 {var_40}  {0x1}
mov     r8d, dword [rbp-0xc {var_14}]  {0x4}
push    r8 {var_48}  {0x4}
mov     r8d, dword [rbp-0x8 {var_10}]  {0x3}
push    r8 {var_50}  {0x3}
push    0x0 {var_58}
push    rdi {var_60}  {0x64}
push    rsi {var_68}  {0x1}
mov     r9d, r10d  {0x64}
mov     r8d, 0x0
mov     esi, eax  {0x3}
lea     rdi, [rel data_2008]  {"flag{%d%c%c%dn%c%d%c%d_%dh_M%d%d…"}
mov     eax, 0x0
call    printf
add     rsp, 0x50
nop
leave    {__saved_rbp}
retn     {__return_addr}
```



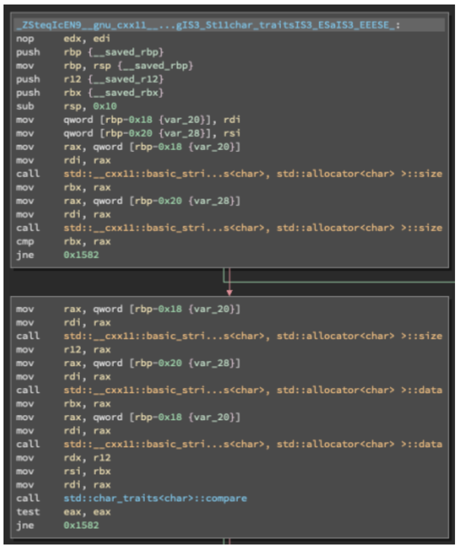
```
z5209322@vx6:/tmp_amd/cage/export/cage/1/z5209322/Documents/2020/6841$ ./hidden
The only way out is inward



flag{3sc0nd1d0_3h_M41s_G0st0S0}
Voce consegue achar a funcao escondida?
```

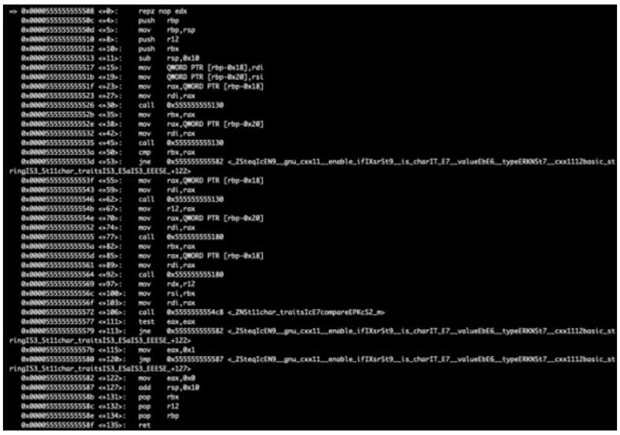
mgdilolmsoamasiug
Link: https://crackmes.one/crackme/5e604d4333c5d4439bb2dd72





Through static analysis of the code, I noticed the program calls a function that has an extremely long name. In this function a std::compare is called to compare the two inputs. I then used gdb to find out what exactly was being compared. A breakpoint was set at main, at the instruction before the function call and right before the compare. The addresses of the latter two breakpoints were obtained by viewing the disassembled code. The two inputs entered were "asd" and "asd". Going to the third breakpoint, rsi contained "add" and rdi contained "asd". Then I reran the program with the inputs "add" and "add". This succeeded. No flag was outputted, but an output of "Good job!" was printed.

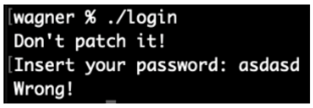




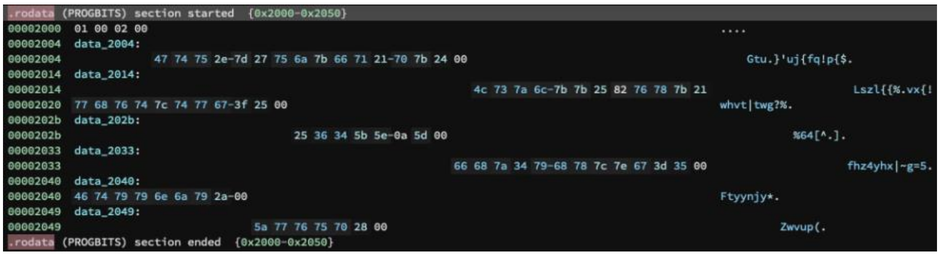
Level 3

Login

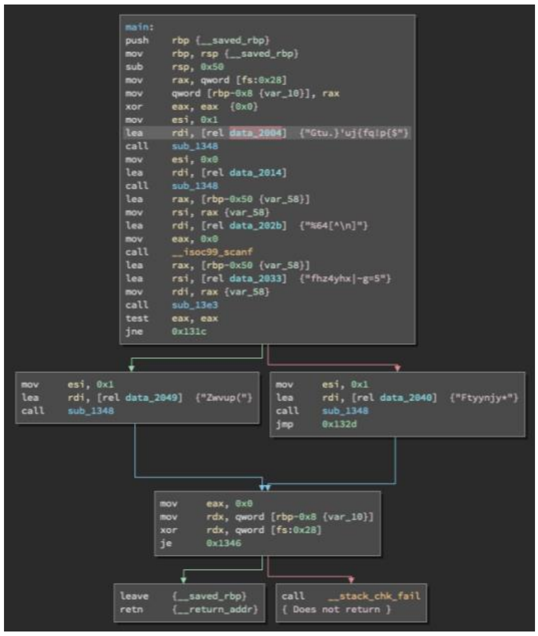
Link: https://crackmes.one/crackme/5db0ef9f33c5d46f00e2c729



While running the program it gave the instructions to not patch. I entered a set of random characters that were incorrect. Upon disassembling the code, I found the following sets of string:



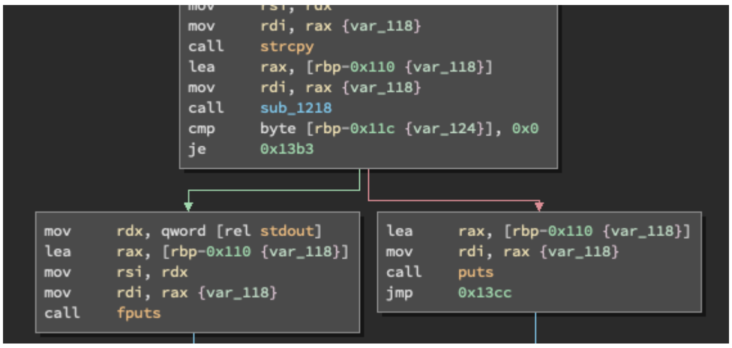
1. "Gtu.}'uj{fq!p{$"
2. "fhz4yhx|~g=5"
3. "Zwvup("
4. "Ftyynjy*"
5. "Lszl{{%\x82vx{!whvt|twg?%"



There is definitely some sort of encryption used here. Furthermore, it seems the messages are decrypted before being sent stdout. So, the message: "Don't patch it!", "Insert your password:" and "Wrong!" are probably encrypted as one of the texts above. This is made evident as these data are sent to either puts or fputs.

From inspection on the lengths of the messages I have, most likely:

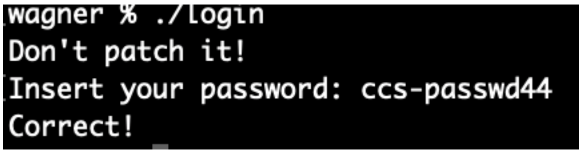
1. "Gtu.}'uj{fq!p{$" => "Don't patch it!"
2. "Zwvup(" => "Wrong!"
3. "Lszl{{%\x82vx{!whvt|twg?%" => "Insert your password:"



Using the ASCII table, I was able to determine the following.

For 1,

G	t	u	.	}	‘	u	j	{	f	q	!	p	{	$
3	5	7	7	9	7	5	9	7	3	9	1	7	7	3
D	o	n	‘	t		p	a	t	c	h		i	t	!



For 2,
Z w v u p (
3 5 7 7 9 7
W r o n g !

Z	w	v	u	p	(
3	5	7	7	9	7
W	r	o	n	g	!

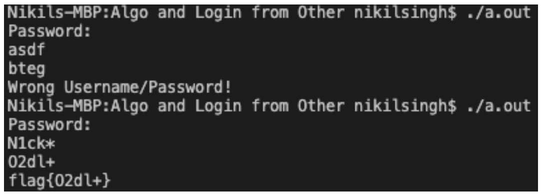
Looking at the shift for the numbers, this seems to be a pattern for the shift in characters. Based on the flow of control in the disassembled code, we see "Zwvup(" exists in one and "Ftyynjy*" in the other. It seems safe to assume the latter represents the branch that is correct. Also "fhz4yhx|~g=5" seems to be used before choosing which branch to go. Based on the decryption above, we get the following:

f	h	z	4	y	h	x	|	~	g	=	5
3	5	7	7	9	7	5	9	7	3	9	1
c	c	s	-	p	a	s	s	w	d	4	4

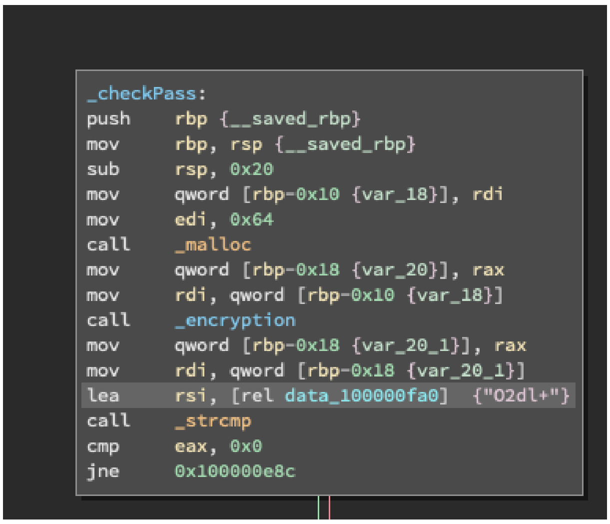
Entering "ccs-passwd44" was successful, with message being sent back being "Correct!".

Program from Colleagues

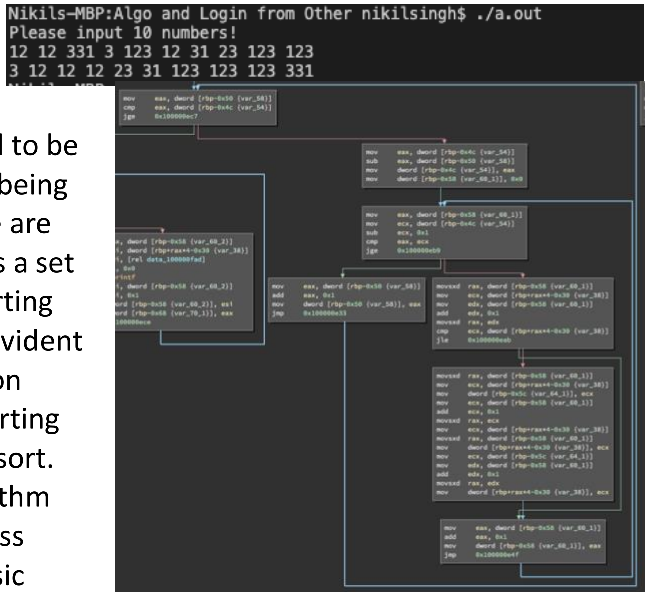
Login System



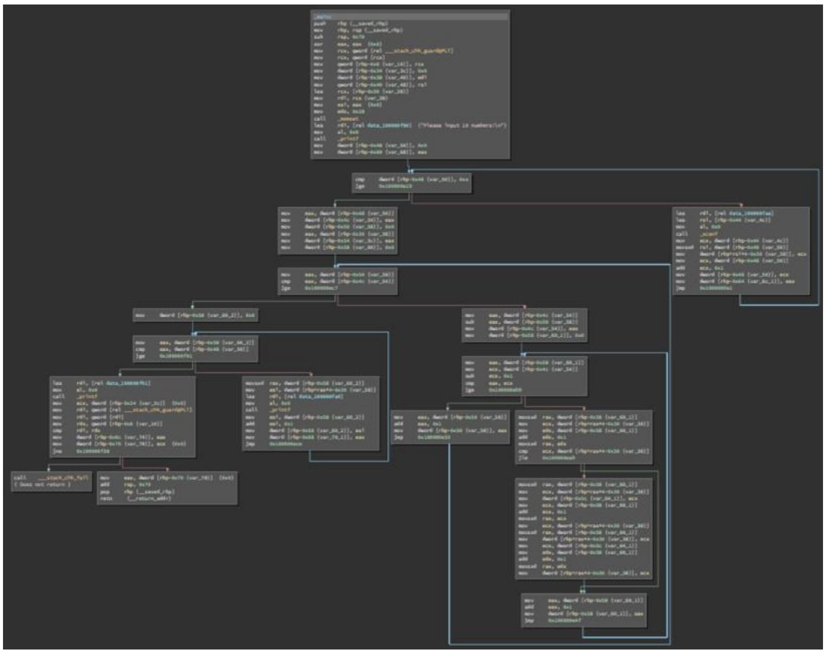
Upon opening it as assembly, I found the string "O2dl+". I also found when running the program, it would output a slightly altered string compared to the input I placed. i.e. “asdf” produced “bteg” There was also a function called encryption. My first guess was it was some sort of Caesar cipher. I then tested more outputs and realised the value of the output was one less on the ASCII table. I then looked up the appropriate ASCII values and found the corresponding values to be "N1ck*" surprisingly. It then also outputted the flag flag{O2dl+}.



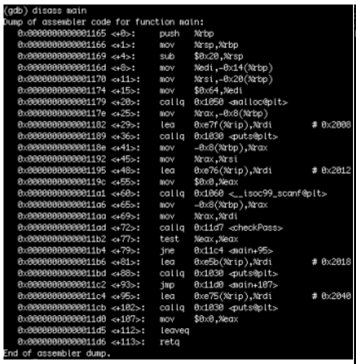
Algorithm

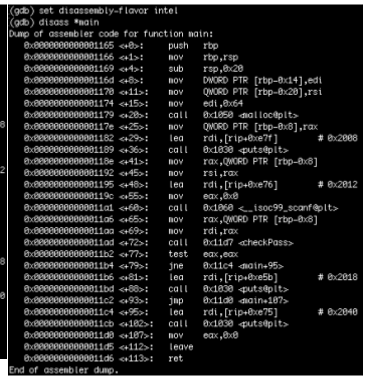


Upon running the actual algorithm, the algorithm seems to be a type of sort. Looking at the disassembled code, there seems to be multiple sets of loops. Some of the loops seemed to be for scanning input and outputting the data after being sorted. Also, it seems like for the algorithm there are two sets of loops. On top of that it seems like it is a set of nested loops. This lead me to believe it is a sorting algorithm of time complexity N^2. This is made evident as two different variables are incremented by 1 on each iteration. This leads limits the number of sorting algorithms to be bubble, selection and insertion sort. Beyond this I cannot tell any further which algorithm was used. If a guess had to be made, I would guess bubble as that tends to be the most common basic sorting algorithm most people would implement.

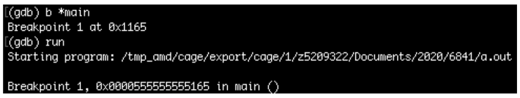


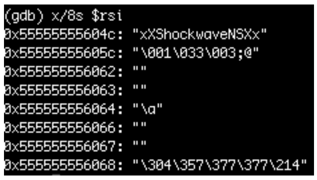
Dynamic Analysis Example

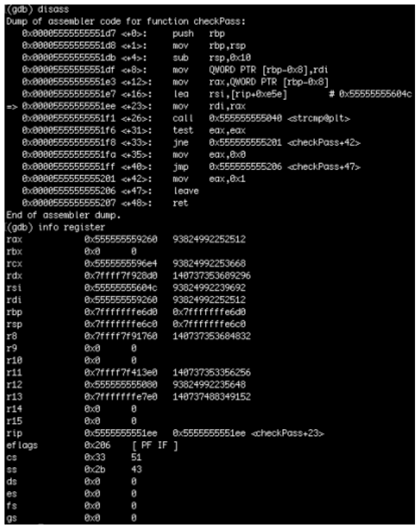


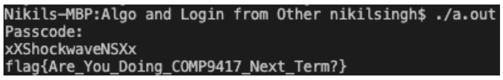


For this exercise gdb was used to run dynamic analysis on the program. First, I displayed the disassembled code in gdb. Since it was unreadable, the disassembly-flavour was set to intel. At that point the disassembled code looked like the assembly displayed in binary ninja. Next a breakpoint was set at main. The program was then run, where it immediately stopped at the breakpoint. I displayed the disassembled code, which showed the functions being called. In that disassembled code, I noticed there was checkPass function. Using si, I went instruction by instruction until I reached the function call to scanf. For that I used ni, and it asked me to enter a passcode. I entered "asd". Next I kept using si until I entered the checkPass function. Inside that function, the disassembled code was displayed. In that strcmp was being called and within rsi data was being loaded. I then used si, until I got to the instruction after rsi was loaded with data. Following this, info registers was used to confirm the data was stored in rsi. Then the instruction x/8s $rsi was used to display the data stored in rsi as string. Here the passcode "xXShockwaveNSXx" was found. Following this, the program was ran as normal and the passcode was entered. From this the flag flag{Are_You_Doing_COMP9417_Next_Term?} was obtained.









Final Thoughts

Overall, this was a fun and interesting topic to cover. Through this I learnt a lot about assembly language enough to the point where I can read it without a guide in most cases. I've gained a lot of experience with static analysis and am now confident enough to do level 2 challenges on crackme without assistance and level 3 challenges with some assistance. Additionally, my skills in cryptography have improved as the more difficult challenges would encrypt the key or password required to solve it. I also learnt the basics of buffer overflows and trying to identify them. Another major skill I learnt was patching the source code. With this I was directly able to get to the function I needed to get to or access function that could not previously be accessed. This was not too useful for the challenges as they discouraged this but its real-world applications are very extensive. ***See appendix B for disclaimer. *** I did cover some of the basics of dynamic analysis using gdb but more time could have been spent using it.

However, there was a larger than expected learning curve in understanding the different responsibilities of the registers and what each instruction does. Following this I also had to learn to navigate the assembly code and determine what was important. This ended up taking more time than initially estimated which placed me behind schedule.

Implementing a login system to try get someone else to reverse engineer is also a fun task. Through this I attempted to learn how to write code that would be difficult to reverse engineer but it did not end up as difficult as expected. In terms of difficulty I would rate the login as level 1 and the algorithm as around level 2. Similarly, solving said challenges from a friend also proved to be fun and a good challenge. There login would probably be rated at level 2 and so would the algorithm. This was a great experience as I got to attempt to make my own challenges and also learnt some basic skills useful in reverse engineering such as cryptography.

In terms of meeting the criteria set out, both were not met. In terms of the schedule, I was not able to keep to it as there was a large learning curve that took up most of the early phases. Although, I began to catch up it was not enough. Additionally, the difficulty of level 3 and above require more time then the 8 weeks provide to accomplish successfully. In terms of selecting what I wished to accomplish, I was over ambitious and not able to finish the set goals.

Advice that would be given to those that wish to undertake a similar project is to immediately begin learning the basic concepts required even before the project has been approved. This will allow one to cover the basic knowledge in reading assembly before doing the challenges on crackme. Furthermore, begin with challenges that have walkthrough solutions so a general idea on the approaches to reverse engineering can be established.

Overall, this was a great project and I would recommend others to do reverse engineering for their something awesome.

Bibliography

1. ‘Reverse Engineering’, *CTF101*, accessed 6 March 2020.
   <https://ctf101.org/reverse-engineering/overview/>

   Used to gather information on the basic concepts of reverse engineering.

2. ‘Assembly Language Tutorial’, *tutorialspoint,* accessed 20 March 2020.
   <https://www.tutorialspoint.com/assembly_programming/assembly_tutorial.pdf>

   Used to gather information on assembly instructions that can be encountered while reverse engineering.

3. Evans, D 2006, ‘x86 Assembly Guide’, *CS216: Program and Data Representation,* accessed 20 March 2020.
   <https://www.cs.virginia.edu/~evans/cs216/guides/x86.html>

   Used to gather information on registers found in assembly.

4. ‘x86 Registers’, *mindsec,* accessed 20 March 2020.
   <http://www.eecg.toronto.edu/~amza/www.mindsec.com/files/x86regs.html>

   Used to gather information on registers found in assembly.

5. 2019, ‘Reverse Engineering a Linux Executable – hello world’, *codementor*, accessed 9 April 2020.
   <https://www.codementor.io/@packt/reverse-engineering-a-linux-executable-hello-world-rjceryk5d>

   Used to get a general guide on how to use gdb to perform reverse engineering and perform a dynamic analysis.

Appendix

A: GITHUB Link

https://github.com/Nikil-Singh/Reverse-Engineering

B: Disclaimer

All programs reversed engineered here shown in the level 1, 2, and 3 section are from crackmes.one. These challenges are designed for the purpose of reverse engineering. The programs reverse engineered in the program from colleagues’ section were made specifically and given from a friend for the purpose of reverse engineering. You should always obtain written permission to reverse engineer any program.

C: Crackmes Website

https://crackmes.one/