CS449 / Project 2 / Part 2
Joshua Rodstein / ps#: 4021607

Passphrase 1: "**VBLunsSYqFylMuPTTgYp**"

My first approach to solving the passphrase for this file was decidedly less/non-scientific. This was the file I used to compare output between the "mystrings" program that I wrote, and the provided "strings" program. Initially the solution came from sifting through the output and looking for suspicious or telling strings. I found the phrase above the output strings for the results of a user guess.

My approach changed after the recitation on the use of GDB to discover passphrases, which I used almost verbatim to discover the 2nd passphrase. I went back and used the same methods to "re-discover" the 1st phrase. I set a breakpoint at main, and dumped the assembly code. Looked for function calls like compare, and discovered at address **0x0804830c** the instruction…

"repz cmpsb %es: (%edi), %ds: (%esi)"

Translation: **repeatWhileEqual compareByteString array_base(string destination), array_base(string source string)**. I then decided to examine the registers %edi and %esi. I found that %esi held the passphrase I discovered initially, and %edi held the phrase I had entered as a guess. If this had been my first approach, at that point I would have entered the contents of %esi as a guess.

While this file was the easiest to solve and did not require much technical effort, using gdb and tracing the assembly code was much quicker and efficient than sifting through a pile of string output.

Passphrase 2: "**jor94**"

As stated in my solution for the first passphrase, I was eventually able to crack the 2nd phrase by following the steps described in the gdb recitation almost verbatim. Prior to that however, I attempted to solve by visual inspection of the string dump. Right off the bat I could see that this method would probably not lead to much of anything useful. The dump was significantly smaller than the first, and there were not really any strings that stood out as being a possible passphrase.

After the lecture on the x86 registers and their purposes, and the gdb recitation, I was able to gain some ground. I opened the file with gdb, set the break point at main, and dumped the assembly code with 'disas'. While looking through the dump of main(), a call to the **strcmp** function caught my eye, so I set a breakpoint at the address right before it, **0x080485d8**. I actually entered the wrong address and ended up 3 instructions prior to the strcmp call. At this address was the instruction "$0x1, %esi. I inspected the $esi register to reveal the string "/jor94_2". I did not think much of this and next'd to the call before strcmp. **"mov %ebx , (%esp)".** I inspected the contents of $ebx, which contained "Joshua_2". Joshua was the guess I had entered. I ran the file from the beginning and entered "AAAA" as a guess. Sure enough at the same instruction, the register $ebx held "AAAA_2". I stepped past the call to strcmp and looked at the instruction "test $eax, $eax". Inspected $eax and saw the string "_2". At this point I was sure I had a pretty good beat on the passphrase. I ran the ELF and entered "jor94", and received the "Congratulations!" message.

CS449 / Project 2 / Part 2
Joshua Rodstein / ps#: 4021607

Passphrase 3: (dynamic)

-10 character minimum length (input past 10 characters is ignored)
-7 characters must be vowels (repeats allowed)
-3 characters must be a printable non-vowel ascii char and/or digit (repeats allowed)

The 3rd executable file was the most challenging. The file had been stripped of its symbol table, which prevented me from simply using "b *main" with gdb to set a breakpoint. I was able to find the entry point by using the "info file" command. I then set the breakpoint at the address **0x80483a0** .

I nexted line by line from the entry point, attempting disas to find a function, and managed to find a breakpoint that would pause the execution of the program after I entered my guess. I used the x/#i $eip command quite a bit to look at assembly when I could not find a function to disas at, and eventually set the correct combination of breakpoints, which allowed me to see that it was looping the same number of times as the length of the string I had entered. I used objdump and found the addresses in the .text section of the code. It was here that I came across a few very interesting and useful instructions.

```
sub     $0x61, $eax
cmp     $0x14, $eax
ja      tolower@plt....
```

I used gdb once again to break at these instructions and inspect the contents of $eax. It held a character, and the command was subtracting the hex value 0x61 ('a' in ascii) from the character stored in $eax, and storing that value back into $eax. Then comparing the value to determine if it was greater than 14. If so, it would jump to the toLower function, which quite obviously converted the character to lower case. The instructions eventually shift the value 0x1 by the result of the subtraction.

The most important line I came across was **"and $0x104111, $eax"** … which performs a bitwise 'and' on 0x104111 and $eax. This resulted in either $eax as 1 or 0. Which affected the execution of following jumps, moves and compares. Comparing the shifting of bits in order make the result of the 'and' instruction 1, the characters would have to correspond with the 1's in the binary bit strings obtained from 0x104111.

0x104111 : 00001 0000 0100 0001 0001 0001

Counting that characters from lower case 'a' in the same way as the bits….

A bcd **E** fgh **I** jklmn **O** pqrst **U**

I wish I could say the rest of my process was just as technical and procedural as the first half. It was not. It was pure chance and a bit of luck in misunderstanding that I initially I miswrote the binary conversion of 0x104111 and ended up with A E I M Q as the required characters. I entered 'aeimaeimam' at one point and received the "congratulations!". I could not produce many consistent results from these characters, so I re traced and found that I had mis-calculated. My final conclusion came from experimentation with repeating and entering different combinations, as well as attempts at tracing and deciphering the instructions.

CS449 / Project 2 / Part 2
Joshua Rodstein / ps#: 4021607