CS0449

Project 3

Dane Halle

Password 1: **umKmQQaZHSkbOOloGxqjUEJ**

Password 2: **3.141593**

Password(s) 3: M any (see **[1]** below and bottom) but my favorite is “**cs449   cs449**”.

Project 3 Writeup

dmh148_1 Executable:

Writeup: I started with this by running the script provided with mystrings.out and looked through for strings found that were bigger than around 6 characters. In my text editor, it shows a smaller version of the code on the right side and you can see longer lines than the others. The first string I found and tried was:

“QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ
QQQQQj”

I am not sure where this would be in the source code but since it didn’t work, I moved on and continued scrolling down. I then got to the Backtrace part and knew it wouldn’t be past that and saw some strings that were familiar. “Sorry! Not correct!” along with success stings. Right above those was a string that was just what I was looking for, a long string of seemingly random characters. I tried it and succeeded. I tested on another day and it still succeeded showing that the password was not dynamically changed.

dmh148_2 Executable:

Writeup: I first started with the same approach as the first password. The output file was much shorter, and nothing looked like it could be a password, so I tried other things. I knew it required only one password. I used an objdump to get the executable code, so I have a reference. I looked through the code and found a call to an <fgets@plt> so I set a break point at that line and ran through with “si”. However, there was a lot of code between that line and when it accepts a string, or it skips it because of the commands. I then put a break point to the next line of code, and it accepted a string. I input “abcdefg” and saw it was stored inside of the $ebx register. I

followed it through and noticed it got moved to the $eax register. Then I saw the $eax register cleared. Eventually, after checking registers and noticing that a lot of code was dedicated to removing the '\n' on the string, I found the passcode within the $esi register after it was moved from the $ebp+0x8 register. I tested that as it was the first string I found within the program and sure enough it worked. I tested it several more times and it continued to work. This was found on Friday, March 8, 2019. After testing on the next day, I found that this string still worked so there is no dynamic change.

dmh148_3 Executable:

Writeup: I ran this to see if it was the same as the other two executables, it was not. It required different number of passwords each time I ran it. I knew that using mystrings wouldn't work but I tried and found an interesting string of "Xe_3VgfL%" but that did not work. I used an objdump to get the executable code, so I have a reference. Upon initial inspection, I noticed that there wer no real functions but only one large <.text>. I then noticed no <fgets…> calls but quite a lot of <getchar…> calls. While running gdb, I found that shortly after inputting a string, the first character was stored within a void* register after the first run and every run through it would add a bit to a point prior to the register and then store the next character in the next spot directly after the previous. It is basically parsing through the input string to put them in the $ebp register. It included the newline character and after the second input, it put the second string after and so on and until it went through every input and every string. Then every part of the string was put into or compared into the $eax register. There is a lot going on. I can tell that the total count of characters needs to be 16 or more (it would seemingly only accept the first 16). I ran through code and eventually noticed that the point where it checks to see if you should succeed or fail is the "cmp DWORD PTR [ebp-0x10],0xa" line which meant that the $ebp register at -0x10 spot needs to be equal to 10 (0xa). So, I looked for points when that part would increase and saw that under several comparison and jumps statements. After running the code with test characters, I saw that each char would be stored within the $eax register, converted to ascii hex and pulled up an ascii table with hex and decimal and looked to see what characters it was comparing the register to. Those five key characters I found were "c", "s", "4", "9", and "0". Every time one of those were within the passphrase, the $ebp-0x10 register would increase by 1. So, putting it all together now, the passphrase will take in at minimum 16 characters (it will demand more if you refuse to give), it puts each character of your input (including '\n') inside of the $ebp-0x21

register and has an index part in the $ebp-0xc register. The program then compares each char of the passphrase against those 5 characters and if successful, it increases the $ebp-0x10 register by 1. After it finishes comparing, it checks to see if that last register is equal to 10 and if not, fails, but if successful, succeeds. **[1]** This means you can have any combination of the above key chars with any other chars as long as there are exactly 10 key chars within the first 16 characters of the passphrase.

Example Passwords for Executable 3:

“c c c c c c cccc”

“s049 s4c c0s”

“ 490400cs 40 “

Etc etc