---------------------------------------------------------------

Password: “neeQAbVpNVwwBDNGpjnoBosAHmo” **Fixed across runs**

Approach:

1. strings -a -n 4 xxx\_1 > 1.out
   1. Find out “Congratulations!” and also locate a string, which really looks like a password, above “Sorry! Not correct!”. So I try that long and complicated string, it works. And the reason I choose to use strings, not mystrings, is that strings can print out more information (include strings with tabs/spaces)
   2. Here’s how they are arranged in 1.out:

Line #

1183 neeQAbVpNVwwBDNGpjnoBosAHmo <= password!

1184 Sorry! Not correct!

1185 Congratulations!

1186 Unlocked with passphrase %s

1. Alright, another approach:
   1. disass main => b \*0x0804830c (break point before repz cmps BYTE PTR ds:[esi],BYTE PTR es:[edi]) => info reg => x/s 0x80b388c (esi)=> “neeQAbVpNVwwBDNGpjnoBosAHmo”
   2. repz cmps compares two strings to see if they are equal, that’s why I set a break point before this to see what’s the string compared to my input

---------------------------------------------------------------

Password: “24.120.27.50 58867 22”

**Not fixed / Subject to the change of IP address**

Approach:

1. strings -a -n 4 xxx > 2.out
   1. Locate a string: “do this project on thoth, cmon, stop being afraid of thoth” I try to enter this string as the passphrase. Cannot unlock the program
2. gdb xxx => b main. Not much helpful, so quit
3. gdb xxx => disass main
   1. gives many “procedures” eg. 0x0804861b <+0>: push ebp
   2. after two “returns,” I see call 0x80486e6 strncmp@plt
   3. So I start to focus on this call, reason is same in 1.2.b:
      1. disas 0x8048428 to see what happens in strncmp
      2. b \*0x080486c4 set a break point before strncmp
      3. r run the program
      4. info reg check the contents of the registers
      5. I try to enter the values of these regs, but none of them works
      6. I try to inspect if some strings store in these regs
      7. x/s 0xffffd0cc (gives what I just input), but x/s 0xffffd4b2 gives a string “24.120.27.50 58867 22” which contains the client’s IP address, might be a password. I don’t know what the second number means (58867 may be the time? And 22 means the port #?)
      8. That is, register ebx stores the correct password, and it **changes** when I run the program again (changes from 58867 to 59947). **Besides,** as long as the IP address is correct, the program can be unlocked successfully, which means we don’t have to give the last two numbers correctly
      9. I input this string, “24.120.27.50 58867 22”, as the passphrase, it works. And when I go back to Philly, IP address **changes**, this time I **only enter the new ip address** which still can unlock the program. This proves my idea in viii.
4. So the passphrase is: a correct format of the current IP address, eg. 24.120.27.50

**Notes:** after I use objdump -d Mintel xxx, I see a func called getenv which, after I searched I little bit, can return associated environment values to the string. Then I add a break point before getenv, and print out $ebx, it gives “SSH\_CLIENT”. So my conclusion is that “SSH\_CLIENT” passed as an argument to get **the current IP address which determines the correct passphrase**. And, below is the reason I realize something is stored in ebx as param in getenv

(gdb) **disas 0x80483d8**

Dump of assembler code for function getenv@plt:

=> 0x080483d8 <+0>: jmp DWORD PTR ds:**0x80499e8**

0x080483de <+6>: push 0x10

0x080483e3 <+11>: jmp 0x80483a8

End of assembler dump.

(gdb) **disas 0x80499e8**

Dump of assembler code for function getenv@got.plt:

0x080499e8 <+0>: fiadd WORD PTR [**ebx**-0x7c11f7fc]

---------------------------------------------------------------

Password: (123}12312312312

**NOT fixed / Has certain rules**

Approach:

1. Try to use strings to print out some strings, like how I did for program 1 and 2. But there’s nothing really useful
2. gdb xxx => disass main doesn’t work
3. objdump -x xxx not really helpful
4. objdump -d Mintel xxx
   1. it shows some procedures(?) and .text, which I think are the keys to unlock the program. So I start to focus on these parts
   2. I see many getchar called, so I guess this program gets one input character each time and compares it to something. I also try to unlock the program, and I find out that the input must be at least **15** length long (eg. 123456789012345)
   3. I locate this block of instructions, since cmp is the straightforward sign of password input checking:

**8048462:**  8b 45 f4 mov -0xc(%ebp),%eax

8048465: 83 e8 01 sub $0x1,%eax

8048468: 0f b6 44 05 df movzbl -0x21(%ebp,%eax,1),%eax

804846d: 0f be c0 movsbl %al,%eax

8048470: 83 f8 5b **cmp $0x5b,%eax**

8048473: 74 1e je 8048493 <puts@plt+0x13f>

8048475: 83 f8 5b **cmp $0x5b,%eax**

8048478: 7f 0a jg 8048484 <puts@plt+0x130>

804847a: 83 e8 28 **sub $0x28,%eax**

804847d: 83 f8 01 **cmp $0x1,%eax**

8048480: 77 16 ja 8048498 <puts@plt+0x144>

8048482: eb 0f jmp 8048493 <puts@plt+0x13f>

8048484: 83 f8 7b **cmp $0x7b,%eax**

8048487: 74 0a je 8048493 <puts@plt+0x13f>

8048489: 83 f8 7d **cmp $0x7d,%eax**

804848c: 74 05 je 8048493 <puts@plt+0x13f>

804848e: 83 f8 5d  **cmp $0x5d,%eax**

8048491: 75 05 jne 8048498 <puts@plt+0x144>

8048493: 83 45 f0 01 addl $0x1,-0x10(%ebp)

8048497: 90 nop

8048498: 83 45 f4 01 addl $0x1,-0xc(%ebp)

804849c: 83 7d f4 10 cmpl $0x10,-0xc(%ebp) **// if read less than 16 chars, jump to 8048462, go over again**

80484a0: 7e c0 jle 8048462 <puts@plt+0x10e>

80484a2: 83 7d f0 02 cmpl $0x2,-0x10(%ebp) **// after reading the first 16 chars, if not 2 brackets, jump to …**

80484a6: 75 16 jne 80484be <puts@plt+0x16a>

* 1. These 5b,7d,… make me think about ASCII chart, so I look them up and find out that they are all delimiters, or brackets. Thus I try to input different strings as the passphrase and get:

|  |  |
| --- | --- |
| Correct | Not Correct |
| [[00000000000000 | (00000000000000 14 0s |
| ((00000000000000) 14 0s | ({{{{{{{{{{{{{{) 14 {s |
| (4454y2g14uqn67) | (000000000000000) 15 0s |
| [00000000000000] 14 0s | {{81321398()0012 |
| {00000000000000} 14 0s |  |
| (00000000000000] 14 0s |  |
| ( ) 14 spaces |  |
| **{1231231231)1324** |  |
| Correct **((qwertyuiopasdf**}{}()i3123fnisnfiienq123n | |

And I find out that the unlock info only shows the first 16 chars (eg. “Unlocked with passphrase [[12345678901234”), so combined with the above attempts, the final conclusion is:

As long as the first 16 chars contain **exactly 2 brackets**, including ‘(’,‘)’,‘[’,‘]’,‘{’,and‘}’, the program can be unlocked. The two brackets can be unpaired, and their positions in the first 16 chars input has no fixed rule