# Challenge Writeups

**Offensive Security Spring 2024**

Lindsay Von Tish

lmv9443@nyu.edu

Table of Contents

[Challenge Writeups 1](#_Toc159503300)

[Challenge Details 2](#_Toc159503301)

[Bridge of Death 2](#_Toc159503302)

[Overview 2](#_Toc159503303)

[Details 2](#_Toc159503304)

[Attempt 10](#_Toc159503305)

[Dora 11](#_Toc159503306)

[Overview 11](#_Toc159503307)

[Details 11](#_Toc159503308)

[Attempt 15](#_Toc159503309)

[Appendix A: Student Information 16](#_Toc159503310)

[Appendix B: Tools 16](#_Toc159503311)

[Appendix C: Postage 17](#_Toc159503312)

[Overview 17](#_Toc159503313)

[Details 17](#_Toc159503314)

[Challenge Attempt 19](#_Toc159503315)

[Appendix D: BoD\_Remote.py 23](#_Toc159503316)

[Appendix E: Dora\_BF.py 25](#_Toc159503317)

Assignment 0: Challenges

## Are You Alive

### Overview

|  |  |  |
| --- | --- | --- |
| Are You Alive? | | |
| **1 Point** | **Flag Value** | flag{youve\_been\_lied\_to\_about\_how\_your\_computer\_works} |
| **Location** | nc offsec-chalbroker.osiris.cyber.nyu.edu 7332 |
| **Lore** | Dan Brown Multiverse |
| **Filename** | Flag.txt |

### Details

The challenge begins with a prompt asking the user to download a text file titled flag.txt, as shown in the following figure:

A screenshot of a computer

Description automatically generated  
**Download Prompt**

The text file contains the challenge flag, as shown below:

|  |
| --- |
| ┌──(kali㉿kali)-[~/Downloads]  └─$ ls -latr  total 12  -rw-r--r-- 1 kali kali 54 Jan 31 14:09 flag.txt  ...omitted for brevity...  ┌──(kali㉿kali)-[~/Downloads]  └─$ cat flag.txt  flag{youve\_been\_lied\_to\_about\_how\_your\_computer\_works} |

**File Contents**

#### **Steps to Reproduce**

1. Download file
2. View file contents  
   Ex:  
   cat flag.txt

## Doors of Durin

### Overview

|  |  |  |
| --- | --- | --- |
| Doors of Durin | | |
| **10**0 **Points** | **Flag Value** | flag{the\_dwarves\_dug\_too\_deep} |
| **Location** | nc offsec-chalbroker.osiris.cyber.nyu.edu 1235 |
| **Lore** | J. R. R. Tolkein |
| **Filename** | N/A |

### Details

The challenge begins with a prompt directing students to connect to offsec-chalbroker.osiris.cyber.nyu.edu at port 1235.

A screenshot of a computer

Description automatically generated  
**Challenge Prompt**

Connecting to the service reveals a text-based adventure game playing through a Lord of the Rings scene. The game prompts the user to answer the riddle, "Speak friend and enter." The game text is shown in the following figure:

|  |
| --- |
| ─$ nc offsec-chalbroker.osiris.cyber.nyu.edu 1235  You and your party of a Wizard, a Dwarf, and Elf, 2 Men, and 3 other Hobbits stand around the Doors of Durin, the entrance to the Dwarven Mines of Moria.  A door blocks your way into the Mines, the only remaining path you have to get to the forest Lothlórien, where the Lady Galadriel is sure to offer you sanctuary from the dark forces pursuing you.  The Wizard looks at the Doors, and reads:  "Ennyn Durin Aran Moria. Pedo mellon a Minno. Im Narvi hain echant. Celebrimbor o Eregion tethant. I thiw hin."  You ask, "What does it mean?"  "Oh, it is a simple riddle," says the Wizard.  "The Doors of Durin, Lord of Moria. Speak friend and enter. I Narvi made them. Celebrimbor of Hollin drew these signs."  You think for a moment. "Speak friend and enter." What could it mean?  Suddenly, the answer comes to you!  You shout: |

**Game Text**

The Tolkien elvish word for friend is "Mellon," and after a few spelling errors, the user successfully input the password and gained access to the challenge flag, as shown below:

|  |
| --- |
| Suddenly, the answer comes to you!  You shout: mellon  The Doors open! As you delve into the Mines, you hear a whisper on the wind:  The flag is: flag{the\_dwarves\_dug\_too\_deep} |

**Password Correct**

An attacker would not need to use an exploit to attack this system. Any attacker with working knowledge of Lord of the Rings would be able to guess the password and obtain the challenge flag.

#### **Steps to Reproduce**

1. Spend your childhood in Alaska with no TV and only books for company.
2. Connect to the system with the following command:  
   nc offsec-chalbroker.osiris.cyber.nyu.edu 1235
3. Input the password

## Mathwhiz

### Overview

|  |  |  |
| --- | --- | --- |
| Mathwhiz | | |
| **20**0 **Points** | **Flag Value** | flag{you\_sure\_are\_a\_math\_genius} |
| **Location** | nc offsec-chalbroker.osiris.cyber.nyu.edu 1236 |
| **Lore** | Math |
| **Filename** | N/A |

### Details

This challenge begins with a prompt similar to the one provided with the Doors of Durin challenge. The URL is the same as the Doors of Durin challenge, but the port number is different. After connecting to the service running on port 1236, the student faces a math challenge. To complete the challenge, the user must successfully answer 100 math questions in a row. The questions increase in difficulty by switching between decimal numbers, text values, and numbers encoded in hexadecimal and binary. The connection closes if the user inputs the wrong answer or an answer that is not a number.

To solve this challenge, the user's code must account for all the different formats in which the math problems are presented. The following method is a part of a program that successfully defeated the challenge.

|  |
| --- |
| def main():  # Start remote connection  URL = "offsec-chalbroker.osiris.cyber.nyu.edu"  PORT = 1236  conn = start(URL, PORT)  log = open("MathWhiz.txt", "a")  # Get the greeting, which is 183 char/183 bytes long  g = conn.recvn(183)  log.write("Greeting: " + str(g) + "\n")  # Get the math problem next  n = QandA(conn, log)  log.close() |

**Main Method**

After connecting to the remote service and receiving a greeting, the program enters the QandA method, shown below, which takes in new math problems, passes them to the solver method, and then responds with the correct answer.

|  |
| --- |
| def QandA(conn, log):  n = 0  while n < 100:  log.write("Question " + str(n) + "\n")  b = conn.recvline()  ans = doSomeMath(b, log)  #conn.pack(ans)  conn.sendline(ans)  b = conn.recvline()  log.write(str(b) + "\n")  n+=1  b = conn.recvall()  print((str(b) + "\n"))  log.write(str(b) + "\n") |

**Q and A Method**

The solver method, depicted in the following figure, translates a math problem into a format that the Python eval function can accept. The solver checks each number to see if it is written in text, in which case it calls a translator method to translate the written number into an integer. Because the eval function accepts encoded input, the only condition that the method must check for is text.

|  |
| --- |
| def doSomeMath(byteString, log):  # Translate into readable problem  byteString = byteString[:-4]  problem = str(byteString, encoding='utf-8')  log.write(problem)  mathList = problem.split()  i = 0  while i < len(mathList):  # Check to see if the number is made of text or not  if re.search("^\D+$", mathList[i]):  mathList[i] = textToIntStr(mathList[i])  i += 2  mathString = " ".join(mathList)  # Perform calculation and return answer  str\_ans = str(eval(mathString))  byte\_ans = str\_ans.encode()  log.write(" = " + str\_ans+ "\n")  return byte\_ans |

**Solver Method**

After checking and translating each number, the solver calculates the answer using the eval function.

The text numbers are formatted as [Digit]-[Digit]-[Digit]. The translator method takes the text number and iterates through each digit to translate it to the decimal format before returning a numerical string. The method is shown below:

|  |
| --- |
| def textToIntStr(t):  tStr = t.split("-")  a = tStr  n = 0  for num in tStr:  a[n] = str(getValue(num))  n+=1  ans = "".join(a)  return ans |

**Solver Method**

The complete code is available in [Appendix B](#_Appendix_B:_Mathwhiz).

# Assignment 1: Reverse Engineering Part One

# Assignment 2

## Bridge of Death

### Overview

|  |  |  |
| --- | --- | --- |
| Bridge of Death | | |
| **150 Points** | Flag Value | flag{@\_W1tch\_W3'v3\_G0t\_@\_W1tch!!!!!!!!!} |
| Location | offsec-chalbroker.osiris.cyber.nyu.edu 8005 |
| Lore | Monty Python and the Holy Grail |

### Details

The program asks the user a series of questions that anyone familiar with Monty Python and the Holy Grail should recognize. On the first run of bridge\_of\_death, I fell to my death.

|  |
| --- |
| gdb ./bridge\_of\_death  ...omitted for brevity...  Reading symbols from ./bridge\_of\_death...  (No debugging symbols found in ./bridge\_of\_death)  (gdb) r  Starting program: /home/kali/Desktop/2-Week/bridge\_of\_death  What is your name?  Juneau  What is your quest?  Pass this class!  kek  kek  kek  kek  Auuuuuuuugh!  [Inferior 1 (process 5082) exited normally]  (gdb) q |

I Fall to my Death

The bridge\_of\_death main() method, shown below after decompilation with *Ghidra*, calls three different "question" methods. Each method returns a boolean value that the method uses to determine whether the program continues or throws the user into the Gorge of Eternal Peril.

|  |
| --- |
| undefined8 main(EVP\_PKEY\_CTX \*param\_1)  {  int iVar1;    init(param\_1);  puts(  "Stop! Who would cross the Bridge of Death must answer me these questions three, ere the other side he see.\n\nWhat is your name?"  );  iVar1 = question1();  if (iVar1 == 0) {  throw\_into\_gorge\_of\_eternal\_peril();  }  puts("What is your quest?");  iVar1 = question2();  if (iVar1 != 0) {  throw\_into\_gorge\_of\_eternal\_peril();  }  puts("What is the air-speed velocity of an unladen swallow?");  iVar1 = question3();  if (iVar1 != 0) {  throw\_into\_gorge\_of\_eternal\_peril();  }  puts("Right. Off you go.");  print\_flag();  return 0;  } |

Main Method

#### Question 1: What is your name?

The bridge\_of\_death binary stores the answer to question 1 in plaintext.

|  |
| --- |
| strings bridge\_of\_death| grep -i "Lancelot"  My name is Sir Lancelot of Camelot. |

Strings

The question1() method compares the user-entered text to the string "My name is Sir Lancelot of Camelot."

|  |
| --- |
| void question1(void)  {  long in\_FS\_OFFSET;  char guess [136];  long local\_10;    local\_10 = \*(long \*)(in\_FS\_OFFSET + 0x28);  fgets(guess,0x80,stdin);  strcmp("My name is Sir Lancelot of Camelot.",guess);  if (local\_10 != \*(long \*)(in\_FS\_OFFSET + 0x28)) {  /\* WARNING: Subroutine does not return \*/  \_\_stack\_chk\_fail();  }  return;  } |

Question 1

However, subsequent testing showed that question1() would return true regardless of the name entered.

|  |
| --- |
| nc offsec-chalbroker.osiris.cyber.nyu.edu 8005  Stop! Who would cross the Bridge of Death must answer me these questions three, ere the other side he see.  What is your name?  My name is Sir Lancelot of Camelot.  What is your quest?  ...omitted for brevity...  nc offsec-chalbroker.osiris.cyber.nyu.edu 8005  Stop! Who would cross the Bridge of Death must answer me these questions three, ere the other side he see.  What is your name?  Juneau  What is your quest? |

Name Demonstration

#### Question 2: What is your quest?

The question2() function uses get\_number() to take in two user-entered integers. The get\_number() function will return 0 if the entered value is not a valid integer. It is the same function used in Postage, shown for reference in [Appendix C](#_Appendix_C:_Postage).

|  |
| --- |
| bool question2(void)  {  undefined4 uVar1;  int iVar2;  int iVar3;    uVar1 = get\_number();  iVar2 = get\_number();  iVar3 = func2(uVar1,0,0x14);  return iVar2 != iVar3;  } |

Question 2

After reading in the numbers, the function calls func2() and checks to see if the return value is equal to the second user-entered integer. The method takes three inputs: the user-entered guess (p1) and two other numbers.

|  |
| --- |
| int func2(int p1,int p2,int p3)  {  int v1;  int v2;  puts("kek");  v1 = p2 + (p3 - p2) / 2;  if (p1 < v1) {  v2 = func2(p1,p2,v1 + -1);  v1 = v2 + v1;  }  else if (v1 < p1) {  v2 = func2(p1,v1 + 1,p3);  v1 = v2 + v1;  }  return v1;  } |

Func2

The recursion in func2() is a red herring; the important part of the problem is the math used to calculate v1. If p1 is equal to v1, then func2() skips the recursion entirely and returns the value of v1. The last two parameters func2() receives are hardcoded by question2(), making it easy to solve for the correct value of p1.

|  |
| --- |
| a = Int('a')  b, c = Reals('b, c')  #g = Int('g')  s = Solver()  s.add(b == 0)  s.add(c == 20)  s.add(a == b + (c-b)/2)  print(s.check())  print(s.model())  >> sat  >> [a = 10, c = 20, b, = 0] |

Solver Script

The correct value of p1 is 10, which should be entered for both guesses to answer question 2.

|  |
| --- |
| What is your name?  Juneau  What is your quest?  10  10  kek  What is the air-speed velocity of an unladen swallow?  I have no clue  Auuuuuuuugh! |

Success

#### Question 3

The question3() function is much longer and more complex than the previous questions.

|  |
| --- |
| undefined8 question3(void)  {  long lVar1;  uint guess1;  uint guess2;  undefined8 flag;  long in\_FS\_OFFSET;  int counter;    lVar1 = \*(long \*)(in\_FS\_OFFSET + 0x28);  counter = 1;  do {  if (9 < counter) {  flag = 0;  LAB\_0010159d:  if (lVar1 != \*(long \*)(in\_FS\_OFFSET + 0x28)) {  /\* WARNING: Subroutine does not return \*/  \_\_stack\_chk\_fail();  }  return flag;  }  guess1 = get\_number();  guess2 = get\_number();  if ((0xff < guess1) || (0xff < guess2)) {  flag = 1;  goto LAB\_0010159d;  }  if (counter != (char)forestOfEwing[(ulong)guess2 + (ulong)guess1 \* 0x100]) {  flag = 1;  goto LAB\_0010159d;  }  counter = counter + 1;  } while( true );  } |

Question 3

The important part of the code is in the highlighted if statement.

At each iteration of the loop, question3() compares the number of the current iteration (counter) to a character in the array forestOfEwing. The character's position is determined using two user-entered integers (guess1 and guess2).

The question3() disassembly gives more insight into how bridge\_of\_death calculates the character's location in forestOfEwing.

|  |
| --- |
| 0x000055555555554b <+98>: mov -0x94(%rbp),%edx  0x0000555555555551 <+104>: mov -0x98(%rbp),%eax  0x0000555555555557 <+110>: shl $0x8,%rax  0x000055555555555b <+114>: add %rax,%rdx  0x000055555555555e <+117>: lea 0x2abb(%rip),%rax #0x555555558020 <forestOfEwing>  0x0000555555555565 <+124>: add %rdx,%rax  0x0000555555555568 <+127>: movzbl (%rax),%eax  0x000055555555556b <+130>: movsbl %al,%eax  0x000055555555556e <+133>: cmp %eax,-0x9c(%rbp)  0x0000555555555574 <+139>: je 0x555555555584 <question3+155> |

Question 3 Disassembly

The program calculates the index in the array using the two guesses and then adds it to the address of forestOfEwing. After saving the address in RAX, the program uses it to load the character value into the EAX register before the comparison. Setting the value of RAX to the address of the counter will ensure that the values are equal.

|  |
| --- |
| gdb ./bridge\_of\_death  (gdb) break \*0x0000555555555568  Breakpoint 4 at 0x555555555568  (gdb) c  ...omitted for brevity...  What is the air-speed velocity of an unladen swallow?  1  2  ...omitted for brevity...  Breakpoint 4, 0x0000555555555568 in question3 ()  (gdb) info registers rax  rax 0x555555558122 93824992248098  (gdb) set $rax=$rbp-0x9c  (gdb) info registers rax  rax 0x7fffffffdd24 140737488346404  (gdb) c  Continuing.  1  2  ...omitted for brevity...  Breakpoint 4, 0x0000555555555568 in question3 ()  (gdb) info registers rax  rax 0x555555558122 93824992248098  (gdb) set $rax=$rbp-0x9c  (gdb) info registers rax  rax 0x7fffffffdd24 140737488346404  (gdb) c  Continuing.  Right. Off you go.  ERROR: no flag found. |

Debugging Solution

Unfortunately, this method only works on a local instance of the program. Because of the size limitations on the guesses, the address used must point to a character in forestOfEwing. The program code contains the forestOfEwing array, which is 65535 characters long.

After exporting the forestOfEwing data from the bridge\_of\_death disassembly in *Ghidra*, I put it in a Python list and used a script to search for characters matching each possible counter value.

|  |
| --- |
| def question3(p):  i = 0  index = [[0,0],[0,0],[0,0],[0,0],[0,0],[0,0],[0,0],[0,0],[0,0]]  for value in f0e:  if value == 0x1:  index[0][0] = i//256  index[0][1] = i%256  elif value == 0x2:  index[1][0] = i//256  index[1][1] = i%256  elif value == 0x3:  index[2][0] = i//256  index[2][1] = i%256  elif value == 0x4:  index[3][0] = i//256  index[3][1] = i%256  elif value == 0x5:  index[4][0] = i//256  index[4][1] = i%256  elif value == 0x6:  index[5][0] = i//256  index[5][1] = i%256  elif value == 0x7:  index[6][0] = i//256  index[6][1] = i%256  elif value == 0x8:  index[7][0] = i//256  index[7][1] = i%256  elif value == 0x9:  index[8][0] = i//256  index[8][1] = i%256  i +=1  i = 0  while i < 9:  msg = str(index[i][0]).encode()  p.sendline(msg)  msg = str(index[i][1]).encode()  p.sendline(msg)  i += 1  return index |

Solver Script

The script uses the location of each matching value to calculate the integer values for both guesses.

|  |
| --- |
| python3 Q3-search.py  [[64, 234], [4, 44], [132, 146], [14, 148], [41, 138], [170, 133], [173, 99], [12, 9], [73, 199]] |

Results

|  |
| --- |
| gdb ./bridge\_of\_death  GNU gdb (Debian 13.2-1) 13.2  ...omitted for brevity...  Stop! Who would cross the Bridge of Death must answer me these questions three, ere the other side he see.  What is your name?  l  What is your quest?  10  10  kek  What is the air-speed velocity of an unladen swallow?  64  234  4  44  132  146  14  148  41  138  170  133  173  99  12  9  73  199  Right. Off you go.  ERROR: no flag found. |

Local Success

### Attempt

After verifying the results locally, I attempted the remote challenge. The solver script is available in [Appendix D](#_Appendix_D:_BoD_Remote.py).

|  |
| --- |
| python3 BoD\_Remote.py  [+] Opening connection to offsec-chalbroker.osiris.cyber.nyu.edu on port 8005: Done  Answering Question 1  Juneau  Answering Question 2  ['10', '10']  Answering Question 3  [[64, 234], [4, 44], [132, 146], [14, 148], [41, 138], [170, 133], [173, 99], [12, 9], [73, 199]]  b"@\_W1tch\_W3'v3\_G0t\_@\_W1tch!!!!!!!!!}\n"  [\*] Closed connection to offsec-chalbroker.osiris.cyber.nyu.edu port 8005 |

Results

## Dora

### Overview

|  |  |  |
| --- | --- | --- |
| Bridge of Death | | |
| **150 Points** | Flag Value | flag{mmaped\_some\_fresh\_pages} |
| Location | offsec-chalbroker.osiris.cyber.nyu.edu 1250` |
| Lore*-a* | Dora the Explorer |

### Details

The first run of Dora made it clear that the wrong input would cause a segmentation error.

|  |
| --- |
| gdb ./dora  What's the key?  13  Program received signal SIGILL, Illegal instruction.  0x00007ffff7fc2000 in ?? ()  (gdb) q  gdb ./dora  What's the key?  111  Program received signal SIGSEGV, Segmentation fault.  0x00007ffff7fc2001 in ?? ()  (gdb) q |

First Run

The decompiled Dora main() method is complex but provides valuable insight into the program's operations.

|  |
| --- |
| undefined8 main(EVP\_PKEY\_CTX \*param\_1)  {  undefined8 \*puVar1;  long lVar2;  undefined8 uVar3;  ulong counter;    init(param\_1);  puVar1 = (undefined8 \*)mmap((void \*)0x0,0x1000,7,0x22,-1,0);  uVar3 = DAT\_00104028;  \*puVar1 = read\_flag;  puVar1[1] = uVar3;  ...omitted for brevity...  puVar1[8] = DAT\_00104060;  puVar1[9] = uVar3;  puts("What\'s the key?");  lVar2 = get\_number();  if ((lVar2 < 0) || (0xff < lVar2)) {  puts("That key is out of range :( Try again?");  uVar3 = 1;  }  else {  for (counter = 0; counter < 0x50; counter = counter + 1) {  \*(byte \*)(counter + (long)puVar1) = \*(byte \*)(counter + (long)puVar1) ^ (byte)lVar2;  }  (\*(code \*)puVar1)();  uVar3 = 0;  }  return uVar3;  } |

Main Method

The program creates a memory map (mmap) in the calling process’s virtual address space. Then, it adds data to the mmap from different locations in the program memory before taking in and validating a user-input integer.

A screenshot of a computer code

Description automatically generated  
Stored Data

After this setup, the program iterates through the saved data, performing an exclusive or (XOR) operation on each byte and the user-entered guess and saving the output.

The main() method disassembly reveals the registers containing the guess and mmap byte, which are stored in ESI and EDX, respectively.

|  |
| --- |
| 0x000055555555538d <+275>: mov -0x28(%rbp),%rcx  0x0000555555555391 <+279>: mov -0x30(%rbp),%rax  0x0000555555555395 <+283>: add %rcx,%rax  0x0000555555555398 <+286>: xor %esi,%edx  0x000055555555539a <+288>: mov %dl,(%rax)  0x000055555555539c <+290>: addq $0x1,-0x30(%rbp) |

Disassembly

Using a debugger, we can see the values in each register before the XOR operation, revealing the mmap data character by character. Initially, the mmap data appears to match the data stored in read\_flag.

|  |
| --- |
| (gdb) break \*0x0000555555555398  Breakpoint 2 at 0x555555555398  ...omitted for brevity...  (gdb) c  Continuing.  What's the key?  23  Breakpoint 2, 0x0000555555555398 in main ()  (gdb) info registers edx  edx 0x97 151  (gdb) info registers esi  esi 0x17 23  (gdb) c  Continuing.  Breakpoint 3, 0x000055555555539a in main ()  (gdb) info registers edx  edx 0x80 128  (gdb) c  Continuing.  Breakpoint 2, 0x0000555555555398 in main ()  (gdb) info registers edx  edx 0x46 70  (gdb) c  Continuing.  Breakpoint 3, 0x000055555555539a in main ()  (gdb) info registers edx  edx 0x51 81  (gdb) c  Continuing.  Breakpoint 2, 0x0000555555555398 in main ()  (gdb) info registers edx  edx 0x23 35  (gdb) c  Continuing.  Breakpoint 3, 0x000055555555539a in main ()  (gdb) info registers edx  edx 0x34 52  (gdb) q |

Disassembly

To get more information about the stored data, I wrote a script to extract every value from EDX.

|  |
| --- |
| def main():  # Start gdb session  p = process('/bin/bash')  p.sendline('gdb ./dora -q')  p.sendline('break \_start')  p.recv() # GDB response with one line indicating that the breakpoint is set  p.sendline('r')  p.sendline('break \*0x0000555555555398')  p.recv()  p.sendline('clear \_start')  p.recv()  p.sendline('c')  p.recvuntil(b'What\'s the key?')  p.sendline(b'23')  p.sendline('c')  data = []  for i in range(80):  p.recvuntil(b'Breakpoint 2')  p.recvline()  p.sendline('info registers edx')  c = cleanLine(p.recvline())  r = re.split("\s+", c)  data.append(r[2])  p.sendline('c')  print(data) |

Extraction Script

|  |
| --- |
| ['0x97', '0x46', '0x23', '0x34', '0x4d', '0x8a', '0xc4', '0x7e', '0x7c', '0x7c', '0x7c', '0x73', '0x79', '0x97', '0x47', '0x22', '0x34', '0xf5', '0xbb', '0xc6', '0x83', '0x7c', '0x7c', '0x7c', '0xc4', '0x7c', '0x7c', '0x7c', '0x7c', '0x73', '0x79', '0xc3', '0x7d', '0x7c', '0x7c', '0x7c', '0xc6', '0x83', '0x7c', '0x7c', '0x7c', '0xc4', '0x7d', '0x7c', '0x7c', '0x7c', '0x73', '0x79', '0xc3', '0x7c', '0x7c', '0x7c', '0x7c', '0xc4', '0x40', '0x7c', '0x7c', '0x7c', '0x73', '0x79', '0x94', '0xbd', '0x83', '0x83', '0x83', '0x1a', '0x10', '0x1d', '0x1b', '0x52', '0x8', '0x4', '0x8', '0x7c', '0x94', '0xbc', '0x83', '0x83', '0x83', '0x0'] |

Data

After extracting the data, I attempted to brute force the solution by looping through each possible guess and performing a XOR between the guess integer and each byte in the extracted data.

|  |
| --- |
| def bruteForceMagic():  for i in range(256):  data = bytes(c ^ i for c in test\_chars)  print(data)  return 0 |

Brute Force Script

Initially, the results appeared to be nonsense data, but after using *grep* to search for common terms, I discovered the string “flag.txt” in the script output.

|  |
| --- |
| python3 Dora\_BF.py | grep -i "flag"  b'\xcb\x1a\x7fh\x11\xd6\x98" /%\xcb\x1b~h\xa9\xe7\x9a\xdf \x98 /%\x9f! \x9a\xdf \x98! /%\x9f \x98\x1c /%\xc8\xe1\xdf\xdf\xdfFLAG\x0eTXT \xc8\xe0\xdf\xdf\xdf\\'  b'\xeb:\_H1\xf6\xb8\x02\x00\x00\x00\x0f\x05\xeb;^H\x89\xc7\xba\xff\x00\x00\x00\xb8\x00\x00\x00\x00\x0f\x05\xbf\x01\x00\x00\x00\xba\xff\x00\x00\x00\xb8\x01\x00\x00\x00\x0f\x05\xbf\x00\x00\x00\x00\xb8<\x00\x00\x00\x0f\x05\xe8\xc1\xff\xff\xffflag.txt\x00\xe8\xc0\xff\xff\xff|' |

Script Output

An update to the script revealed which input value resulted in the useable data.

|  |
| --- |
| def bruteForceMagic():  for i in range(256):  data = bytes(c ^ i for c in test\_chars)  if 'flag'.encode() in data:  return i  return 0  >> python3 Dora\_BF.py  >> 124 |

Brute Force Script

The full solver script for this challenge is available in [Appendix E](#_Appendix_E:_Dora_BF.py).

### Attempt

After discovering a possible input value using good old-fashioned brute force, I validated the guess against a local instance of Dora.

|  |
| --- |
| $ gdb ./dora  ...omitted for brevity...  (gdb) r  What's the key?  124  |[Inferior 1 (process 90978) exited normally]  (gdb) q |

Local Success

This value also worked for the remote instance, which revealed the flag.

|  |
| --- |
| nc offsec-chalbroker.osiris.cyber.nyu.edu 1250  What's the key?  124  flag{mmaped\_some\_fresh\_pages} |

Flag

## Appendix A: Tools

|  |  |
| --- | --- |
| Name | URL |
| EDB | <https://www.kali.org/tools/edb-debugger/> |
| GDB | <https://www.gnu.org/software/gdb/gdb.html> |
| Ghidra | <https://ghidra-sre.org/> |
| Netcat | <https://netcat.sourceforge.net/> |
| PwnTools | <https://github.com/Gallopsled/pwntools> |

# 

## Appendix B: Mathwhiz Code Solution

|  |
| --- |
| from pwn import \*  import re  # A function to start the remote connection  # Input: URL string, Port int  # Output: Connection  def start(U, P):  io = remote(U, P)  return io  # Gets the integer value of a text number  # Input: Text string number  # Output: Integer  def getValue(t):  if t == "ONE":  return 1  elif t == "TWO":  return 2  elif t == "THREE":  return 3  elif t == "FOUR":  return 4  elif t == "FIVE":  return 5  elif t == "SIX":  return 6  elif t == "SEVEN":  return 7  elif t == "EIGHT":  return 8  elif t == "NINE":  return 9  elif t == "ZERO":  return 0  # A function to translate a text string into a string of integers  # Input: Text string number  # Output: Itegers in string form  def textToIntStr(t):  tStr = t.split("-")  a = tStr  n = 0  for num in tStr:  a[n] = str(getValue(num))  n+=1  ans = "".join(a)  return ans  # A function to translate the response into a math problem and return the answer  # Input: Byte string representing math problem and the log file  # Output: Byte string representing the answer  def doSomeMath(byteString, log):  # Translate into readable problem  byteString = byteString[:-4]  problem = str(byteString, encoding='utf-8')  log.write(problem)  mathList = problem.split()  i = 0  while i < len(mathList):  # Check to see if the number is made of text or not  if re.search("^\D+$", mathList[i]):  mathList[i] = textToIntStr(mathList[i])  i += 2  mathString = " ".join(mathList)  # Perform calculation and return answer  str\_ans = str(eval(mathString))  byte\_ans = str\_ans.encode()  log.write(" = " + str\_ans+ "\n")  return byte\_ans  # A function to recieve questions and send answers  # Input: Connection  # Output: Number of problems completed  def QandA(conn, log):  n = 0  while n < 100:  log.write("Question " + str(n) + "\n")  b = conn.recvline()  ans = doSomeMath(b, log)  #conn.pack(ans)  conn.sendline(ans)  b = conn.recvline()  log.write(str(b) + "\n")  n+=1  b = conn.recvall()  print((str(b) + "\n"))  log.write(str(b) + "\n")  def main():  # Start remote connection  URL = "offsec-chalbroker.osiris.cyber.nyu.edu"  PORT = 1236  conn = start(URL, PORT)  log = open("MathWhiz.txt", "a")  # Get the greeting, which is 183 char/183 bytes long  g = conn.recvn(183)  log.write("Greeting: " + str(g) + "\n")  # Get the math problem next  n = QandA(conn, log)  log.close()  if \_\_name\_\_=="\_\_main\_\_":  main() |

## Appendix C: Postage

## Appendix D: BoD\_Remote.py

|  |
| --- |
| from pwn import \*  import re  # Array with data from forestsOfEwing  f0e = […omitted for brevity…]  # Host and port for the remote challenge  HOST = 'offsec-chalbroker.osiris.cyber.nyu.edu'  PORT = 8005  # A function to send a name to the remote challenge  # Input: Connection  # Output: Message  def question1(p):  msg = 'Juneau'  p.sendline(msg)  return msg  # A function to send a the answer to question 2  # Input: Connection  # Output: Array with both answers  def question2(p):  msg = '10'  p.sendline(msg.encode())  p.sendline(msg.encode())  ans = [msg, msg]  return ans  # A function to send a the answer to question 3  # Input: Connection  # Output: Array with all nine answers  def question3(p):  i = 0  index = [[0,0],[0,0],[0,0],[0,0],[0,0],[0,0],[0,0],[0,0],[0,0]]  for value in f0e:  if value == 0x1:  index[0][0] = i//256  index[0][1] = i%256  elif value == 0x2:  index[1][0] = i//256  index[1][1] = i%256  elif value == 0x3:  index[2][0] = i//256  index[2][1] = i%256  elif value == 0x4:  index[3][0] = i//256  index[3][1] = i%256  elif value == 0x5:  index[4][0] = i//256  index[4][1] = i%256  elif value == 0x6:  index[5][0] = i//256  index[5][1] = i%256  elif value == 0x7:  index[6][0] = i//256  index[6][1] = i%256  elif value == 0x8:  index[7][0] = i//256  index[7][1] = i%256  elif value == 0x9:  index[8][0] = i//256  index[8][1] = i%256  i +=1  i = 0  while i < 9:  msg = str(index[i][0]).encode()  p.sendline(msg)  msg = str(index[i][1]).encode()  p.sendline(msg)  i += 1  return index  def main():  p = remote(HOST, PORT)  p.recvuntil(b'What is your name?')  print("Answering Question 1")  print(question1(p))  p.recvuntil(b'What is your quest?')  print("Answering Question 2")  print(question2(p))  p.recvuntil(b'What is the air-speed velocity of an unladen swallow?')  print("Answering Question 3")  print(question3(p))  p.recvuntil(b'flag{')  print(p.recvline())  p.close()  return 0    if \_\_name\_\_ == "\_\_main\_\_":  main() |

## Appendix E: Dora\_BF.py

|  |
| --- |
| # Characters taken from the Dora memory map  test\_chars = bytes([...omitted for brevity...])  # A function to xor all 255 characters with every byte in the array to see if we get usable data  # Input: N/A  # Output: The value that creates data with the word "flag" in it  def bruteForceMagic():  for i in range(256):  data = bytes(c ^ i for c in test\_chars)  if 'flag'.encode() in data:  return i  return -13  print(bruteForceMagic()) |