# Homework 3 Challenge Writeup

**Reverse Engineering Part 3**

Lindsay Von Tish

lmv9443@nyu.edu

02/21/2024

Table of Contents

[Homework 3 Challenge Writeup 1](#_Toc159858563)

[Challenge Details 2](#_Toc159858564)

[Hand Rolled Cryptex 2](#_Toc159858565)

[Overview 2](#_Toc159858566)

[Details 2](#_Toc159858567)

[Appendix A: Student Information 15](#_Toc159858568)

[Appendix B: Tools 15](#_Toc159858569)

[Appendix C: HRC\_Local\_Debug.py 16](#_Toc159858570)

[Appendix D: HRC\_Remote.py 19](#_Toc159858571)

# Challenge Details

## Hand Rolled Cryptex

### Overview

|  |  |  |
| --- | --- | --- |
| Hand Rolled Cyptex | | |
| **1o0 Points** | Flag Value | flag{str1PP3d\_B1N4R135\_r\_S0o0\_much\_FUN\_408012} |
| Location | nc offsec-chalbroker.osiris.cyber.nyu.edu 7332 |
| Lore | Dan Brown Multiverse |

### Details

The hand\_rolled\_cryptex program asks for two input values at the beginning. Entering an incorrect value will cause the program to stop.

|  |
| --- |
| ./hand\_rolled\_cryptex  I found this weird cryptex...  ...it seems to take some weird series of operations...  ...but all the symbols are obscured...  ...could you crack it for me??  The first round requires two inputs...  > 13  > 22  Oh no! That input broke the vial of vinegar, ruining  the papyrus scroll with the flag! |

First Run

Looking at the program in Ghidra reveals that the binary has been stripped. There are no function names.

|  |  |
| --- | --- |
|  |  |

Function Names in Ghidra

However, BinaryNinja was able to infer where the main method began.

A screen shot of a computer code

Description automatically generated  
Function Names in Binary Ninja

The main method, disassembled using BinaryNinja, is shown below.

A screenshot of a computer program

Description automatically generated  
**Main Method**

Each if statement highlighted in red corresponds with a different question function. If any of the functions return a value less than 0, the program will not output the flag.

#### Debugging

Even though there are no symbols in the binary, we can see that it calls \_\_libc\_start\_main by using strings.

|  |
| --- |
| ┌──(kali㉿kali)-[~/Desktop/3-Week]  └─$ strings hand\_rolled\_cryptex  /lib64/ld-linux-x86-64.so.2  sKkTGT2"/  mgUa  libc.so.6  \_\_stack\_chk\_fail  stdin  stdout  \_\_cxa\_finalize  setvbuf  \_\_libc\_start\_main  GLIBC\_2.4  GLIBC\_2.2.5  \_ITM\_deregisterTMCloneTable  \_\_gmon\_start\_\_ |

\_\_libc\_start\_main

The call to \_\_libc\_start\_main passes the location of the main method as the first parameter in the call. The call is visible in entry and includes the location of the main method.

|  |
| --- |
| void processEntry entry(undefined8 param\_1,undefined8 param\_2)  {  undefined auStack\_8 [8]; \_\_libc\_start\_main(FUN\_00101930,param\_2,&stack0x00000008,FUN\_00101f40,FUN\_00101fb0,param\_1, auStack\_8);  do {  /\* WARNING: Do nothing block with infinite loop \*/  } while( true );  } |

Entry

This information makes it easy to find the main method in Ghidra and is also valuable for finding memory locations for dynamic analysis.

A screenshot of a computer

Description automatically generated  
Main Method Offset

After running the program, find the memory address of the entry point.

|  |
| --- |
| gdb ./hand\_rolled\_cryptex  ...omitted for brevity...  Oh no! That input broke the vial of vinegar, ruining  the papyrus scroll with the flag!  [Inferior 1 (process 3736152) exited normally]  (gdb) info file  Symbols from "/home/kali/Desktop/3-Week/hand\_rolled\_cryptex".  Local exec file:  `/home/kali/Desktop/3-Week/hand\_rolled\_cryptex', file type elf64-x86-64.  Entry point: 0x555555555080 |

Entry Address

Then, set a breakpoint at the entry address and look at the process map to find the start address.

|  |
| --- |
| ┌──(kali㉿kali)-[~/Desktop/3-Week]  └─$ gdb ./hand\_rolled\_cryptex  (gdb) break \*0x555555555080  ...omitted for brevity...  Breakpoint 1, 0x0000555555555080 in ?? ()  (gdb) info proc map  process 100090  Mapped address spaces:  Start Addr End Addr Size Offset Perms objfile  0x555555554000 0x555555555000 0x1000 0x0 r--p |

Start Address

The location of the main method is equal to the Start Address increased by the Byte Source Offset.

|  |
| --- |
| [Start Address] + [Main Method Offst]  555555554000 + 1930 = 5555555559300x0 r--p |

Main Method Math

#### Question 1

The first question takes in two pieces of user-entered data and then opens a file specified in the first user-entered string.

|  |
| --- |
| undefined4 Question1(void)  {  undefined4 uVar1;  int iVar2;  long local\_10;    // Print question 1 message and store answer  uVar1 = get\_Length(&local\_48);  hrc\_write(1,&local\_48,uVar1);  iVar2 = hrc\_read(0,&readData,0x100);  if (iVar2 == 0) {  uVar1 = 0xffffffff;  }  else {  if ((&readData)[iVar2 - 1] == '\n') {  (&readData)[iVar2 - 1] = 0;  }  hrc\_copy(&storedData,&readData,0x20);  hrc\_overwrite(&readDat,0,0x100);  local\_4d = 0x203e200a;  local\_49 = 0;  // Print ">" for next answer  uVar1 = get\_Length(&local\_4d);  hrc\_write(1,&local\_4d,uVar1);  iVar2 = hrc\_read(0,&DAT\_00104040,0x100);  if (iVar2 == 0) {  uVar1 = 0xffffffff;  }  else {  iVar2 = FUN\_001013cf((int)DAT\_00104040);  if (iVar2 == -1) {  uVar1 = 0xffffffff;  }  else {  // Print question 1 message and store answer  DAT\_00104010 = hrc\_open(&DAT\_00104240,iVar2);  hrc\_overwrite(&DAT\_00104240,0,0x20);  hrc\_overwrite(&DAT\_00104040,0,0x100);  uVar1 = DAT\_00104010;  }  }  } |

Question 1

While the first parameter must point to a valid filename, at this point in the reverse engineering process, it seems that the second parameter is just a number.

|  |
| --- |
| ┌──(kali㉿kali)-[~/Desktop/3-Week]  └─$ echo "testtesttest" > test.txt  ┌──(kali㉿kali)-[~/Desktop/3-Week]  └─$ gdb ./hand\_rolled\_cryptex  (gdb) r  Starting program: /home/kali/Desktop/3-Week/hand\_rolled\_cryptex  [Thread debugging using libthread\_db enabled]  Using host libthread\_db library "/lib/x86\_64-linux-gnu/libthread\_db.so.1".  I found this weird cryptex...  ...it seems to take some weird series of operations...  ...but all the symbols are obscured...  ...could you crack it for me??  The first round requires two inputs...  > ./test.txt  > 13  \*The first chamber opened! Ok, the second phase requires a single input... |

Question 1 Success

#### Question 2

Question two takes in one number before performing operations.

A screenshot of a computer program

Description automatically generated  
Question 2

The function reads in a number and then performs bitwise arithmetic before using the final value as the file descriptor in a read call.

|  |
| --- |
| iVar2 = hrc\_read(0,&DAT\_00104040,0x100);  // ...omitted for brevity...  else {  uVar1 = hrc\_read(~DAT\_00104040 ^ 0xc9,&DAT\_00104140,0x100);  hrc\_overwrite(&DAT\_00104040,0,0x100);  } |

Main Functionality

In previous read operations, the file descriptor value was 0. The below solver found the correct input for a result of 0:

|  |
| --- |
| def question2():  s = Solver()  a = BitVec('a',4)  s.add(~(a) ^ 0xc9 == 0)  print(s.check())  print(s.model())    >> sat  >> [a = 6] |

Solver

However, the answer was not correct. After further testing, I set a breakpoint after the call to Question1 to see what value was returned.

|  |
| --- |
| ┌──(kali㉿kali)-[~/Desktop/3-Week]  └─$ gdb ./hand\_rolled\_cryptex  (gdb) break \*0x555555555C08  ...omitted for brevity...  The first round requires two inputs...  > ./test.txt  > 13  Breakpoint 2, 0x0000555555555c08 in ?? ()  (gdb) info registers rax  rax 0x3 3 |

Registers

After rerunning the solver with a target value of 3, I discovered that an input value of 5 was correct, but only if the second input from Question1 was 0.

|  |
| --- |
| ┌──(kali㉿kali)-[~/Desktop/3-Week]  └─$ gdb ./hand\_rolled\_cryptex  (gdb) r  I found this weird cryptex...  ...it seems to take some weird series of operations...  ...but all the symbols are obscured...  ...could you crack it for me??  The first round requires two inputs...  > ./flag.txt  > 0  \*The first chamber opened! Ok, the second phase requires a single input...  > 5  Nice, the second chamber opened! Ok, the final level requires another single input...3 |

Question 2 Success

#### Question 3

On the first attempt, it looked like I had the third question almost correct.

|  |
| --- |
| Nice, the second chamber opened! Ok, the final level requires another single input...  > 13  The final chamber opened, but a flaw in the design  popped a vinegar vial which started to eat away at the papyrus  scroll inside. You hold it up, trying to decipher the text... [Inferior 1 (process 107790) exited normally] |

Attempt

In the program main method, it appears that the function tries to print the flag value after the message.

A computer screen with text and numbers

Description automatically generated  
Main Method

The value highlighted in green is the address of the filename saved during the first question.

A computer screen with numbers and symbols

Description automatically generated  
**Question 1**

Question3 makes some checks based on the input. If the value is less than - or equal to 1, the return value is set to 01.

|  |
| --- |
| guess\_Q3 = (int)DAT\_00104040;  if (guess\_Q3 == 1) {  uVar1 = 0xffffffff;  }  else if (guess\_Q3 < 0) {  uVar1 = 0xffffffff;  } |

Check 1

If the value is equal to 2, the program sets the return value to the ASCII value of the first character in the entered text.

|  |
| --- |
| if (guess\_Q3 == 2) {  local\_90 = get\_Length(&guess\_Q3);  uVar1 = local\_90;  } else { |

Check 2

It appeared that an entered value of 2 would be correct, but when looking at the registers during the comparison, it appears that the EAX register contains 32, the ASCII value of 2, instead of the integer itself.

|  |
| --- |
| ┌──(kali㉿kali)-[~/Desktop/3-Week]  └─$ gdb ./hand\_rolled\_cryptex  (gdb) break \*0x55555555586d  Breakpoint 1 at 0x55555555586d  ...omitted for brevity...  Nice, the second chamber opened! Ok, the final level requires another single input...  > 2  Breakpoint 1, 0x000055555555586d in ?? ()  (gdb) info registers eax  eax 0x32 50  ...omitted for brevity...  [Inferior 1 (process 354433) exited normally]  (gdb) r  ...omitted for brevity...  Nice, the second chamber opened! Ok, the final level requires another single input...  > 1234  Breakpoint 1, 0x000055555555586d in ?? ()  (gdb) info registers eax  eax 0x31 49 |

Registers

The below script fuzzes the Question3 input to determine which encoding of 2 was correct. The runFuzz function sets up a breakpoint and then loops through a list of potential encodings. The complete code for HRC\_Local\_Debug.py is available in Appendix C.

|  |
| --- |
| def runFuzz(p):  fuzz = ['2','32','02','032','002','0032','%2','%32','%02','%032','x2','x32','x02','#x32','&#x32','\2','\32','\02','\\x2','\x32','\x02','0x2','0x32','0x02','\0x2','\0x32','\0x02']  log = open("HRC\_Q3\_dbg.txt", "a")  log.write("Hand Rolled Cryptex Q3 Debug Log:" + "\n")  p.sendline('break \*0x55555555586d')  flag = -1  i = 0  for guess in fuzz:  p.sendline('r')  question1(p)  question2(p)  q3 = FuzzQ3(p, guess)  p.sendline('c')  reg = re.split("\s+", q3)  if(reg[3] == '2'):  print("Correct Answer Found")  log.write("Valid Answer!\n")  log.write("Guess at index " + str(i) + "= "+ guess + "\n" + q3)  p.recvuntil(b'flag')  flag = cleanLine(p.recvline())  else:  p.recvuntil("(gdb)")  i += 1  return flag  def main():  # Start gdb session  p = process('/bin/bash')  p.sendline('gdb ./hand\_rolled\_cryptex -q')  print(runFuzz(p))49 |

runFuzz

With each new guess, it runs hand\_rolled\_cryptex and sends the correct answers until Question3. For the third question, the script calls the FuzzQ3 function to send the guess.

|  |
| --- |
| def FuzzQ3(p, ans):  p.recvuntil(b'>')  p.sendline(ans.encode())  p.recvuntil("Breakpoint")  p.recvline()  p.sendline("info registers eax")  return(cleanLine(p.recvline())) |

FuzzQ3

The runFuzz function checks the EAX value. If it is equal to 2, the guess is correct. In that case, the runFuzz function logs the answer and saves the flag value. Otherwise, the function waits for the program to end before running again.

When run against a local instance of hand\_rolled\_cryptex, the debugging script produces three correct answers and prints the value stored in flag.txt.

|  |
| --- |
| ┌──(kali㉿kali)-[~/Desktop/3-Week]  └─$ python3 HRC\_Local\_Debug.py  [+] Starting local process '/bin/bash': pid 313099  Correct Answer Found  Correct Answer Found  Correct Answer Found  {This is not a real flag}  [\*] Stopped process '/bin/bash' (pid 313099)  ┌──(kali㉿kali)-[~/Desktop/3-Week]  └─$ cat HRC\_Q3\_dbg.txt  Hand Rolled Cryptex Q3 Debug Log:  Valid Answer!  Hand Rolled Cryptex Q3 Debug Log:  Valid Answer!  Guess at index 15=  (gdb) eax 0x2 2  Valid Answer!  Guess at index 17=  (gdb) eax 0x2 2  Valid Answer!  Guess at index 20=  (gdb) eax 0x2 2 |

Output

The correct values are shown below:

|  |
| --- |
| '\2', '\02', '\x02' |

Values

#### Solution

I created a new solver script for the remote challenge. Each question function sends its respective answer, and question3 waits for the flag data. The complete code for HRC\_Remote.py is available in Appendix D.

|  |
| --- |
| def question1(p):  p.recvuntil(b'>')  ans = "./flag.txt"  p.sendline(ans.encode())  p.recvuntil(b'>')  ans = "0"  p.sendline(ans.encode())  return(cleanLine(p.recvline()))  def question2(p):  p.recvuntil(b'>')  ans = "5"  p.sendline(ans.encode())  return(cleanLine(p.recvline()))    def question3(p):  p.recvuntil(b'>')  ans = "\x02"  p.sendline(ans.encode())  p.recvuntil(b'flag')  return(cleanLine(p.recvline())) |

Script

When run, the script successfully retrieves the hand\_rolled\_cryptex flag from the remote server.

|  |
| --- |
| ┌──(kali㉿kali)-[~/Desktop/3-Week]  └─$ python3 HRC\_Remote.py  [+] Opening connection to offsec-chalbroker.osiris.cyber.nyu.edu on port 7332: Done  b'I found this weird cryptex...\n'  \*The first chamber opened! Ok, the second phase requires a single input...  Nice, the second chamber opened! Ok, the final level requires another single input...  {str1PP3d\_B1N4R135\_r\_S0o0\_much\_FUN\_408012}  [\*] Closed connection to offsec-chalbroker.osiris.cyber.nyu.edu port 7332 |

Success

# Appendix A: Student Information

|  |  |
| --- | --- |
| Lindsay Von Tish | |
| Email | [lmv9443@nyu.edu](mailto:lmv9443@nyu.edu) |

# Appendix B: 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 C: HRC\_Local\_Debug.py

|  |
| --- |
| from pwn import \*  import re  #############################################################  # HRC\_Local\_Debug.py #  # Lindsay Von Tish (lmv9443@nyu.edu) #  # Reverse Engineering 3: Hand Rolled Cryptex #  # 02/21/2024 #  #############################################################  # A function to convert encoded input to a string and remove text format characters  # Input: Encoded string  # Output: Unencoded string  def cleanLine(ln):  ansi\_escape = re.compile(r'\x1B(?:[@-Z\\-\_]|\[[0-?]\*[ -/]\*[@-~])')  l = ansi\_escape.sub('', str(ln, encoding='utf-8'))  return l  # A function to send the answer to question 1  # Input: Connection  # Output: Response  def question1(p):  p.recvuntil(b'>')  ans = "./flag.txt"  p.sendline(ans.encode())  p.recvuntil(b'>')  ans = "0"  p.sendline(ans.encode())  return(cleanLine(p.recvline()))  # A function to send a the answer to question 2  # Input: Connection  # Output: Response  def question2(p):  p.recvuntil(b'>')  ans = "5"  p.sendline(ans.encode())  return(cleanLine(p.recvline()))  # A function to send a string to question 3 and get register information  # Input: Connection, potential answer string  # Output: String containing register data  def FuzzQ3(p, ans):  p.recvuntil(b'>')  p.sendline(ans.encode())  p.recvuntil("Breakpoint")  p.recvline()  p.sendline("info registers eax")  return(cleanLine(p.recvline()))  # A function to solve question 3  # Input: Connection  # Output: N/A  def runFuzz(p):  fuzz = ['2','32','02','032','002','0032','%2','%32','%02','%032','x2','x32','x02','#x32','&#x32','\2','\32','\02','\\x2','\x32','\x02','0x2','0x32','0x02','\0x2','\0x32','\0x02']  log = open("HRC\_Q3\_dbg.txt", "a")  log.write("Hand Rolled Cryptex Q3 Debug Log:" + "\n")  p.sendline('break \*0x55555555586d')  flag = -1  i = 0  for guess in fuzz:  p.sendline('r')  question1(p)  question2(p)  q3 = FuzzQ3(p, guess)  p.sendline('c')  reg = re.split("\s+", q3)  if(reg[3] == '2'):  print("Correct Answer Found")  log.write("Valid Answer!\n")  log.write("Guess at index " + str(i) + "= "+ guess + "\n" + q3)  p.recvuntil(b'flag')  flag = cleanLine(p.recvline())  else:  p.recvuntil("(gdb)")  i += 1  return flag  # A function to send a the answer to question 3  # Input: Connection  # Output: Response  def question3(p):  p.recvuntil(b'>')  ans = "\x02"  p.sendline(ans.encode())  p.recvuntil(b'flag')  return(cleanLine(p.recvline()))  # A function to solve question 3  # Input: Connection  # Output: N/A  def runSolve(p):  p.sendline('r')  #print(p.recvline())  print(question1(p))  print(question2(p))  print(question3(p))  # Close remote session  p.close()  return 0  def main():    # Start gdb session  p = process('/bin/bash')  p.sendline('gdb ./hand\_rolled\_cryptex -q')  # Uncomment for if solving or debugging  #runSolve(p)  print(runFuzz(p))  if \_\_name\_\_=="\_\_main\_\_":  main() |

# Appendix D: HRC\_Remote.py

|  |
| --- |
| from pwn import \*  import re  #############################################################  # HRC\_Remote.py #  # Lindsay Von Tish (lmv9443@nyu.edu) #  # Reverse Engineering 3: Hand Rolled Cryptex #  # 02/21/2024 #  #############################################################  # Host and port for the remote challenge  HOST = 'offsec-chalbroker.osiris.cyber.nyu.edu'  PORT = 7332  # A function to convert encoded input to a string and remove text format characters  # Input: Encoded string  # Output: Unencoded string  def cleanLine(ln):  ansi\_escape = re.compile(r'\x1B(?:[@-Z\\-\_]|\[[0-?]\*[ -/]\*[@-~])')  l = ansi\_escape.sub('', str(ln, encoding='utf-8'))  return l  # A function to send the answer to question 1  # Input: Connection  # Output: Response  def question1(p):  p.recvuntil(b'>')  ans = "./flag.txt"  p.sendline(ans.encode())  p.recvuntil(b'>')  ans = "0"  p.sendline(ans.encode())  return(cleanLine(p.recvline()))  # A function to send a the answer to question 2  # Input: Connection  # Output: Response  def question2(p):  p.recvuntil(b'>')  ans = "5"  p.sendline(ans.encode())  return(cleanLine(p.recvline()))  # A function to send a the answer to question 3  # Input: Connection  # Output: Response  def question3(p):  p.recvuntil(b'>')  ans = "\x02"  p.sendline(ans.encode())  p.recvuntil(b'flag')  return(cleanLine(p.recvline()))  def main():  # Start remote session  p = remote(HOST, PORT)  print(p.recvline())  print(question1(p))  print(question2(p))  print(question3(p))  # Close remote session  p.close()  if \_\_name\_\_=="\_\_main\_\_":  main() |