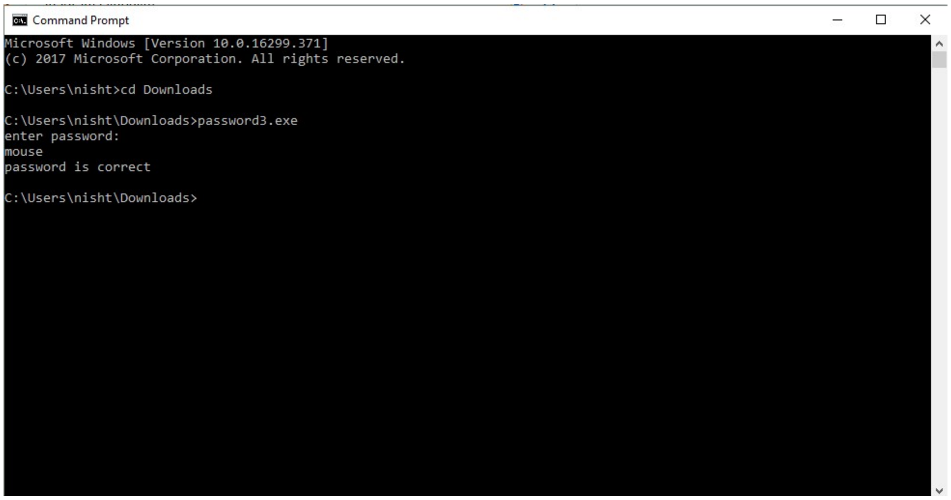
REVERSE ENGINEERING

NISHTHA BHATTACHARJEE
npb258@nyu.edu

Requirements:

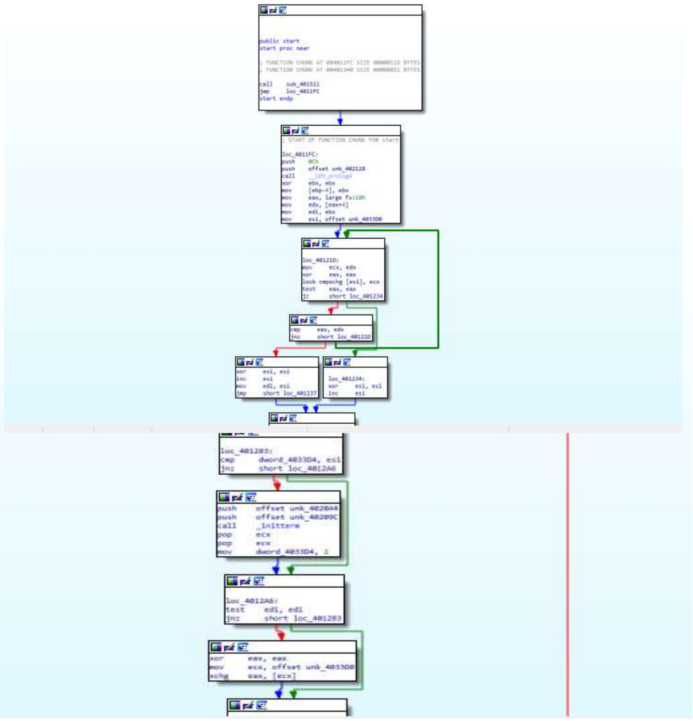
- Successfully completed crack
  - Screenshot of the working solution – 5 pts
- Detailed write-up
  - Includes screenshots of crucial steps – 10 pts
  - Describes thought process – 20 pts
  - Explains crucial code sections needed to “solve” the challenge (or areas of concern that prevented further analysis if not solved) – 20 pts
  - Description of solution, including why it works – 20 pts
- Clear understanding of reverse engineering principles – 15 pts
- All materials submitted (write-up, crack script/patch, screenshots) – 10 pts

Screenshot of the working solution



```
Microsoft Windows [Version 10.0.16299.371]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\nisht>cd Downloads

C:\Users\nisht\Downloads>password3.exe
enter password:
mouse
password is correct

C:\Users\nisht\Downloads>
```

Screenshot of the Crucial Steps



```
public start
start proc near

; FUNCTION CHUNK AT 004011FC SIZE 00000115 BYTES
; FUNCTION CHUNK AT 00401340 SIZE 00000021 BYTES

call    sub_401511
jmp     loc_4011FC
start endp
```

```
; START OF FUNCTION CHUNK FOR start

loc_4011FC:
push    0Ch
push    offset unk_402128
call    __SEH_prolog4
xor     ebx, ebx
mov     [ebp-4], ebx
mov     eax, large fs:18h
mov     edx, [eax+4]
mov     edi, ebx
mov     esi, offset unk_4033D0
```

```
loc_40121D:
mov     ecx, edx
xor     eax, eax
lock cmpxchg [esi], ecx
test    eax, eax
jz      short loc_401234
```

```
cmp     eax, edx
jnz     short loc_40121D
```

```
xor     esi, esi
inc     esi
mov     edi, esi
jmp     short loc_401237
```

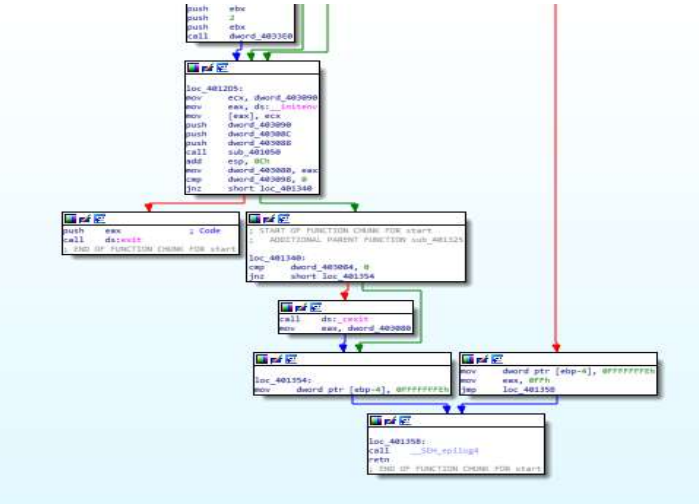
```
loc_401234:
xor     esi, esi
inc     esi
```

```
loc_401283:
cmp     dword_4033D4, esi
jnz     short loc_4012A6
```

```
push    offset unk_4020A4
push    offset unk_40209C
call    _initterm
pop     ecx
pop     ecx
mov     dword_4033D4, 1
```

```
loc_4012A6:
test    edi, edi
jnz     short loc_401283
```

```
xor     eax, eax
mov     ecx, offset unk_4033D0
xchg    eax, [ecx]
```



```
push    ebx
push    2
push    ebx
call    dword_403360
```

```
loc_4012D5:
mov     ecx, dword_403090
mov     eax, ds:__initenv
mov     [eax], ecx
push    dword_403090
push    dword_40308C
push    dword_403088
call    sub_401050
add     esp, 0Ch
mov     dword_403080, eax
cmp     dword_403098, 0
jnz     short loc_401340
```

```
push    eax             ; Code
call    ds:exit
; END OF FUNCTION CHUNK FOR start
```

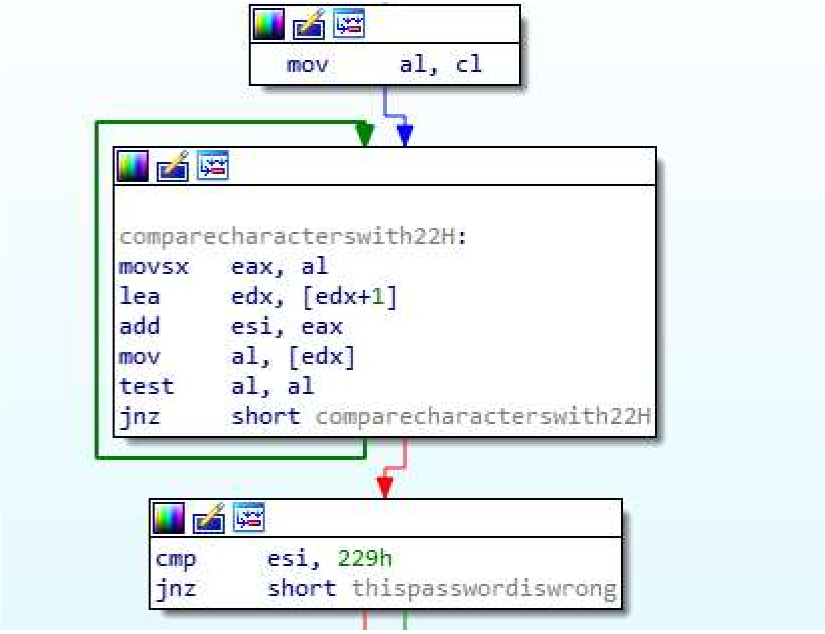
```
; START OF FUNCTION CHUNK FOR start
;   ADDITIONAL PARENT FUNCTION sub_401325

loc_401340:
cmp     dword_403084, 0
jnz     short loc_401354
```

```
call    ds:_cexit
mov     eax, dword_403080
```

```
loc_401354:
mov     dword ptr [ebp-4], 0FFFFFFFEh
```

```
mov     dword ptr [ebp-4], 0FFFFFFFEh
mov     eax, 0FFh
jmp     loc_401358
```

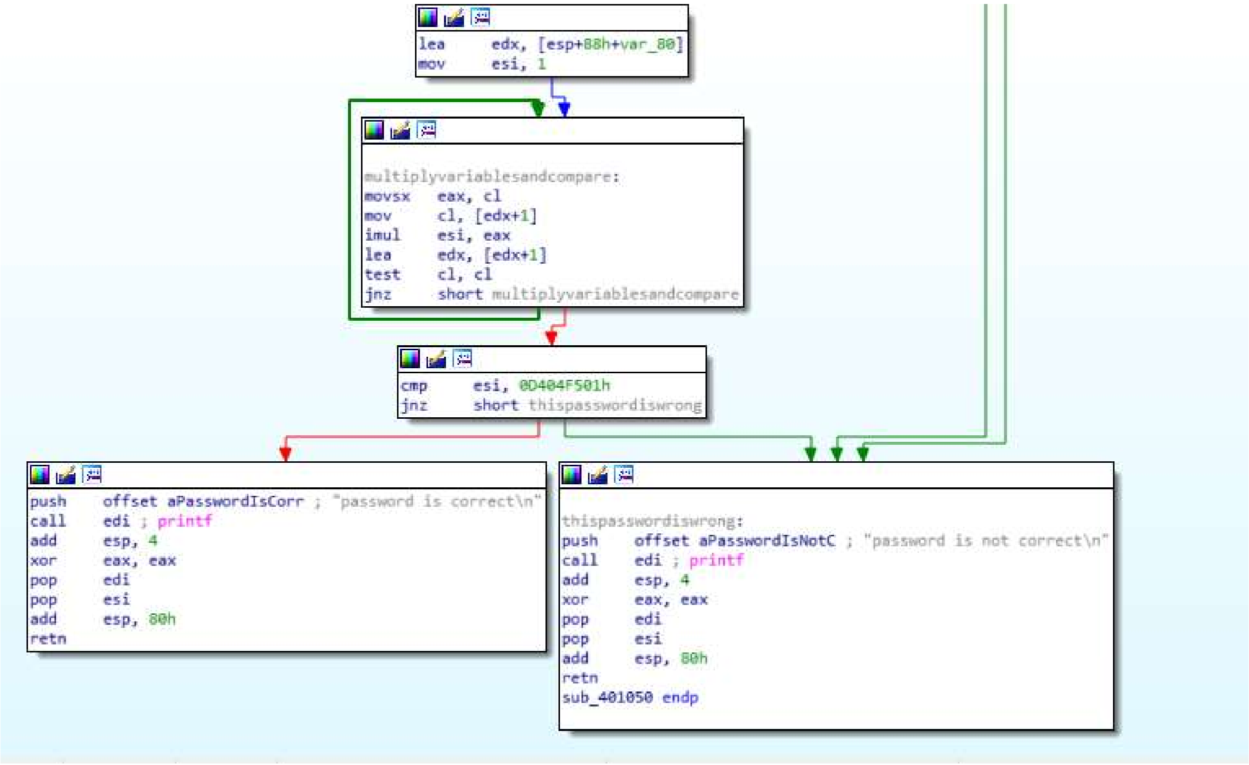
```
loc_401358:
call    __SEH_epilog4
retn
; END OF FUNCTION CHUNK FOR start
```



```
mov     al, cl
```

```
comparecharacterswith22H:
movsx   eax, al
lea     edx, [edx+1]
add     esi, eax
mov     al, [edx]
test    al, al
jnz     short comparecharacterswith22H
```

```
cmp     esi, 229h
jnz     short thispasswordiswrong
```



```
lea     edx, [esp+88h+var_80]
mov     esi, 1
```

```
multiplyvariablesandcompare:
movsx   eax, cl
mov     cl, [edx+1]
imul    esi, eax
lea     edx, [edx+1]
test    cl, cl
jnz     short multiplyvariablesandcompare
```

```
cmp     esi, 0D404F501h
jnz     short thispasswordiswrong
```

```
push    offset aPasswordIsCorr ; "password is correct\n"
call    edi ; printf
add     esp, 4
xor     eax, eax
pop     edi
pop     esi
add     esp, 80h
retn
```

```
thispasswordiswrong:
push    offset aPasswordIsNotC ; "password is not correct\n"
call    edi ; printf
add     esp, 4
xor     eax, eax
pop     edi
pop     esi
add     esp, 80h
retn
sub_401050 endp
```

Thought Process:

As per the assignment guidelines, we had to select a challenge with difficulty level of 2 or above. I selected 39 initially but 40 seemed more challenging to me even though the challenge statement for both were the same. I took the challenge https://challenges.re/40/ and downloaded the windows exe as I will be running the application on a windows machine. I loaded it in IDAPro which I downloaded from https://www.hex-rays.com/products/ida/support/download_freeware.shtml and started analyzing it.

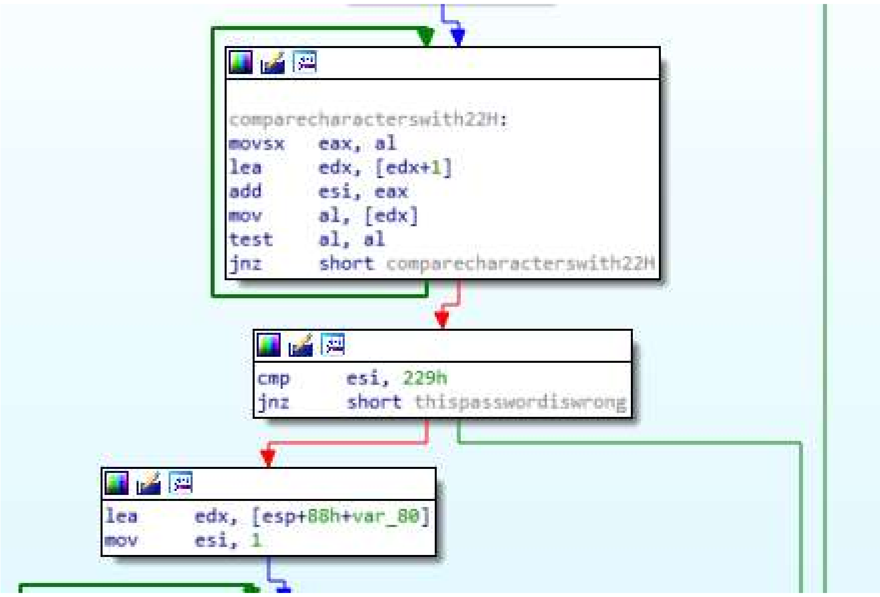
I tried to understand the flow diagram. The colored edges of the flow chart represent the outcome of conditional jump instructions. Green means that the condition is satisfied, red means not satisfied. The password that a user enters is compared with the default password already set up in the database. Therefore, we need to look for commands which compares two values. My focus was thereby shifted on the CMP part.

After reviewing multiple flow diagrams, I came upon two conditions that would help me solve the challenge. Sum & multiplication of each characters in the password combination were 229H and 0D404F501H.
The function names were confusing, and I needed to replace the logical one with some substituted names so that it would be easier for me to debug properly in the IDA platform. Using debuggers, the exact state of the state registers is known. If a user enters a password, there is a function to validate against the user that the password is not null. If it is null, it goes along the red line. If not, there is another function called.

Once I realized the necessary conditions were known, I used some online tools and solutions to find the correct password. We could also solve this solution by using brute force, but we would need at least 5-6 loops and anything beyond On^2 is impractical and redundant in this world.

Crucial Code:

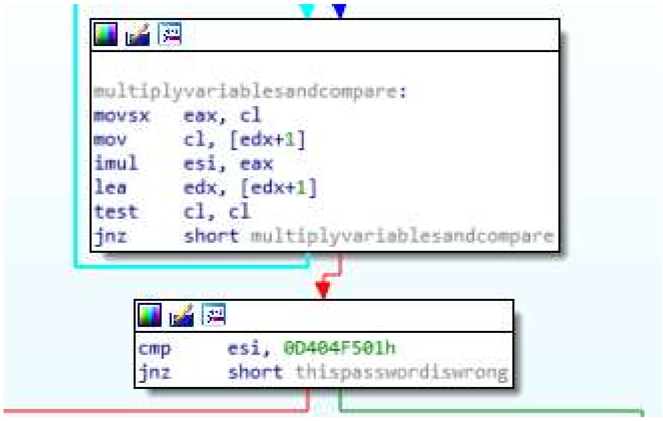
If you notice the flow diagram block, we find that each character is loaded to edx. The contents are added using the add along with esi. One by one all characters are added as in a loop and once the loop exits the sum stored in esi is 229H.



```
comparecharacterswith22H:
movsx   eax, al
lea     edx, [edx+1]
add     esi, eax
mov     al, [edx]
test    al, al
jnz     short comparecharacterswith22H
```

```
cmp     esi, 229h
jnz     short thispasswordiswrong
```

```
lea     edx, [esp+88h+var_80]
mov     esi, 1
```



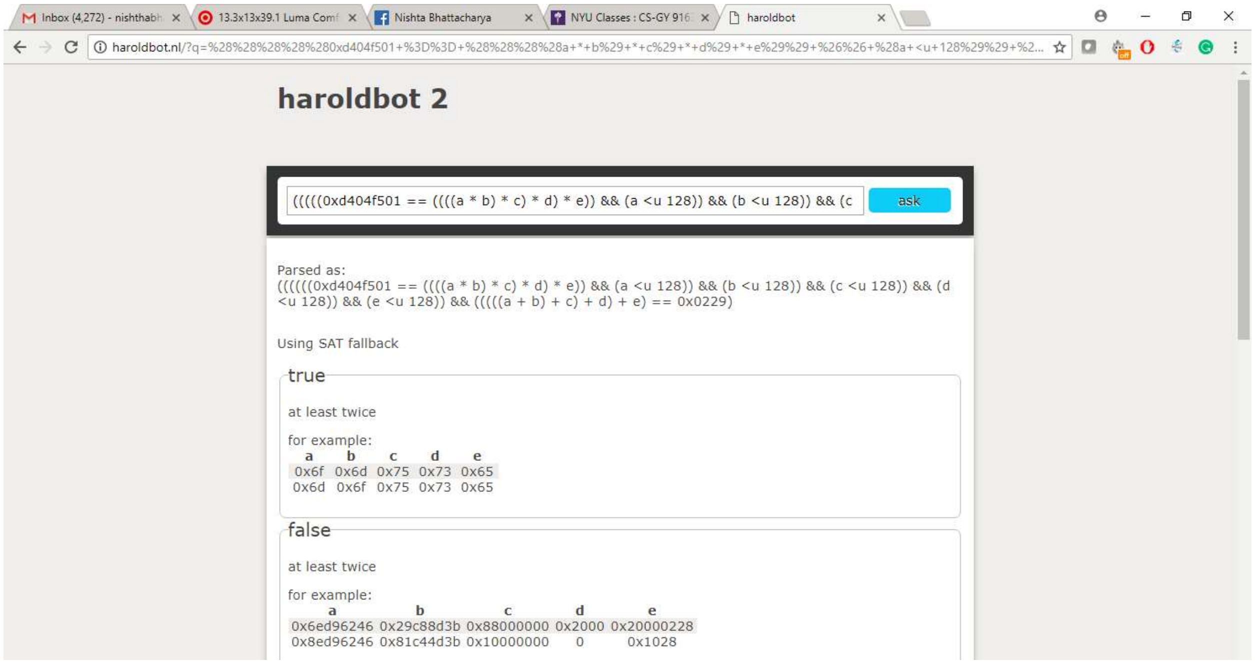
```
multiplyvariablesandcompare:
movsx   eax, cl
mov     cl, [edx+1]
imul    esi, eax
lea     edx, [edx+1]
test    cl, cl
jnz     short multiplyvariablesandcompare
```

```
cmp     esi, 0D404F501h
jnz     short thispasswordiswrong
```

If you go further down the flow diagram, we come across the block shown as above. First, each character is loaded to edx. There is a multiplication between each of the characters in esi and edx. This loop exits when the multiplication is over, and the value is stored in esi which is 0D404f501h.

Description of Solution:

Once I get two conditions there could be million permutations and combinations to get the desired result. However, rather than using the brute force technique, I used an online application called haroldbot. It takes in the expression and then evaluates to give a viable solution. By using hit and trial I found that solution for 1-4 variables are wrong. So, I used a solution with 5 variables.


Inbox (4,272) - nishthabh
13.3x13x39.1 Luma Comf
Nishta Bhattacharya
NYU Classes : CS-GY 916
haroldbot
haroldbot.nl/?q=%28%28%28%28%280xd404f501+%3D%3D+%28%28%28%28a+*+b%29+*+c%29+*+d%29+*+e%29%29+%26%26+%28a+<u+128%29%29+%2...
ask

haroldbot 2

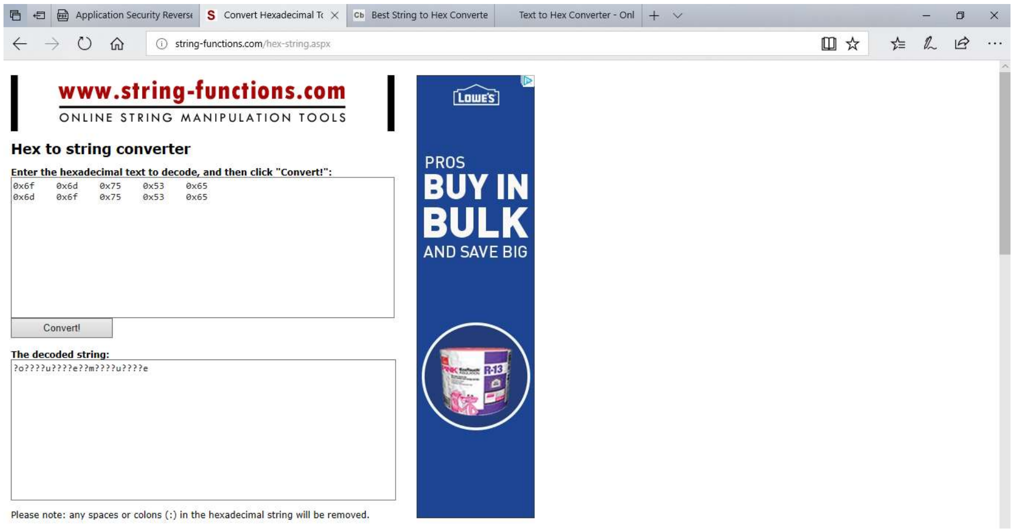
```
(((((0xd404f501 == ((((a * b) * c) * d) * e)) && (a <u 128)) && (b <u 128)) && (c
```

```
Parsed as:
((((((0xd404f501 == ((((a * b) * c) * d) * e)) && (a <u 128)) && (b <u 128)) && (c <u 128)) && (d
<u 128)) && (e <u 128)) && (((((a + b) + c) + d) + e) == 0x0229)

Using SAT fallback

true

at least twice

for example:
  a     b     c     d     e
 0x6f  0x6d  0x75  0x73  0x65
 0x6d  0x6f  0x75  0x73  0x65

false

at least twice

for example:
     a          b          c        d        e
0x6ed96246 0x29c88d3b 0x88000000 0x2000 0x20000228
0x8ed96246 0x81c44d3b 0x10000000   0       0x1028
```

For any variable the maximum length is less than 128-character bytes and hence I evaluated them by using this equation-

(((((0xd404f501 == ((((a * b) * c) * d) * e)) && (a <u 128)) && (b <u 128)) && (c <u 128)) && (d<u 128)) & & (e <u 128) && (((((a+b)+c)+d)+e)==0x0229)

By using this site http://string-functions.com/hex-string.aspx I converted the hex value to a string.


Application Security Reverse
Convert Hexadecimal To
Best String to Hex Converte
Text to Hex Converter - Onl
string-functions.com/hex-string.aspx
www.string-functions.com
ONLINE STRING MANIPULATION TOOLS
LOWE'S
PROS
BUY IN
BULK
AND SAVE BIG
R-13

Hex to string converter

Enter the hexadecimal text to decode, and then click "Convert!":

```
0x6f    0x6d    0x75    0x53    0x65
0x6d    0x6f    0x75    0x53    0x65
```

Convert!

The decoded string:

```
?o????u????e??m????u????e
```

Please note: any spaces or colons (:) in the hexadecimal string will be removed.

Why the solution works?

- The combination I received was ‘mouse’ and it works with any adjacency combination of the string because it gives the same result whether you do the addition, multiplication and compare or solve the multiply, add terms in that order.

Following are the steps on how to solve the problem of reverse engineering-

- Having knowledge of assembly language is helpful. As I have taken up microprocessors course earlier and worked with microcontrollers, this seem to improve my understanding of the workflow. If your knowledge is limited, first please read a documentation or watch some tutorials.
- Understand the requirement of finding the vulnerability. In our case, we had to crack the password. The normal workflow suggests we enter a password which is compared to the one in the database which ensures what we have entered is correct. So, the emphasis is on the ‘comparison part’.
- Once you reach at the crux of your solution, develop what ifs scenario. In our case, we had the sum and multiplication cases. One wouldn’t work without the another and therefore they are commutative.
- After the password you have found, and if it works then we develop test cases and release the patches.