CPEN 442 Assignment 4 Passwords

Agent47 (56406144)

1 Question 1

The first step to solve that question was realize that the size of the hash didn't match any of the known hash sizes. The closest size possible was **SHA1** but I had 2 extra characters.

It6D209CE47FEC653F2947F0E3D09580130C05C3CC

The two first characters aren't hexadecimal so they might be a salt. I ran all the possible pin combination using the salt perpended and after that I was able to find the solution.

The correct pin is: **2160**

The time necessary for finding the solution was:

real	0m0.020s
user	0m0.016s
sys	0m0.004s

2 Question 2

Just like in the previous question the hash was generated using **SHA1** and the first two symbols are the salt. For this question first I had to generate all the possible combinations with the alphabet provided by th question

abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
1234567890
!@#$%^&*()_+-=

For this I used the python library *itertools*. This library has a method called product that will generate all the possible combinations using an specific set of characters.

Because the number of possible combinations is:
$76^6 = 192699928576$ possible passwords

I though that searching for the password going forward and backward at the same time the chances that I would find the solution would be higher. For that I ran 2 different codes where the generated possible passwords were generated starting from 'aaaaaa' to the first terminal and starting from '======' for the second terminal. After around 6 hours processing my computer crashed and I lost the processes that were running. So I had to patch the software in a way that it would not test the previous generated passwords. Even though the program still had to spin lock in the for loop until the counter reaches 5 billion passwords because that was approximately the number of passwords tested before the crash.

I setup the terminals to run the code again and after a few hours the right password was founded.

The correct password for this question is: **NTdLWn2_**

The approximated time necessary for finding the solution was:

real	539m16.173s
user	539m15.059s
sys	0m0.139s

3 Question 3

To solve this assignment I used the software **IDA pro** in order to analyze the executable file.

3.1 a)

At first I was analyzing the code in the text mode. Using the search tool I could find the part of the code that prints the instructions to insert the password. I started analyzing the data from there. Analyzing the assembly code I could recognize that there is an loop that I suppose that is the loop that check for the integrity of the password comparing symbol by symbol the typed password and the hard coded password. I also noticed that there is a hard coded string in the assembly code. The variable can be found at the address

.rdata:004141A0

the content of the hardcoded string is:

t#eKyJ_Zue!VjV4x-5dU)lnbSCISl##vb%Mq#JBrWMytûsgFe-UQHBiWd&_aAOFp

At first I thought that this was the correct password but using this as a password didn't work. I came back to the assembly code and first I saw that before the program starts checking for the integrity of the password it tests the length of the string that I inserted. This test is done by the following code:

```
call _strlen
add esp, 4
cmp eax, 0Ah
```

paying close attention you can see that the last command checks if the string has 0Ah length, that converting to decimal is 10 characters.

At this point I tried to insert the first 10 characters of the string, but it also didn't work.

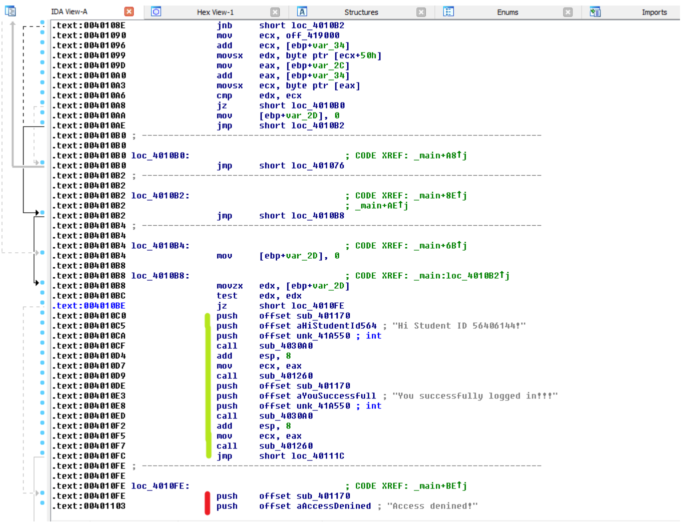
After a more detailed look at the code I also noticed that the inside the loop that checks for the integrity of the password the variable that receives the hard coded string has an offset. This can be seen at the following instructions:

```
mov ecx, off_419000

add ecx, [ebp+var_34]

movsx edx, byte ptr [ecx+50h]
```

At the last line ecx (the variable that receives the hard coded string) is added with an offset of 50h witch is 80 if converted to base 10. considering this offset I could find that the correct password is: **jCng$r4RME**

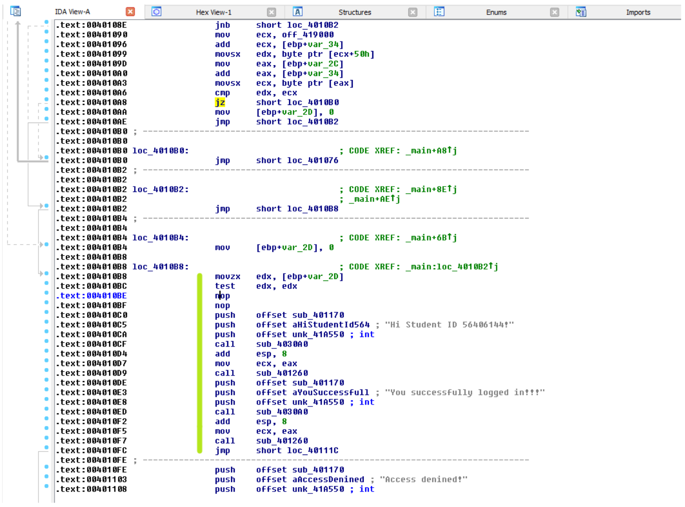
3.2 b)

Looking at the text mode we can see that the ADA pro software has some arrows that shows the instructions flow. Analyzing the execution flow I could see that in the line **.text:004010BE** the program flow is redirected to the area where the program outputs that the password is incorrect (marked in RED). this can be seen in the following image:



```
.text:0040108E                 jnb     short loc_4010B2
.text:00401090                 mov     ecx, off_419000
.text:00401096                 add     ecx, [ebp+var_34]
.text:00401099                 movsx   edx, byte ptr [ecx+50h]
.text:0040109D                 mov     eax, [ebp+var_2C]
.text:004010A0                 add     eax, [ebp+var_34]
.text:004010A3                 movsx   ecx, byte ptr [eax]
.text:004010A6                 cmp     edx, ecx
.text:004010A8                 jz      short loc_4010B0
.text:004010AA                 mov     [ebp+var_2D], 0
.text:004010AE                 jmp     short loc_4010B2
.text:004010B0 ; ---------------------------------------------------------------------------
.text:004010B0
.text:004010B0 loc_4010B0:                             ; CODE XREF: _main+A8↑j
.text:004010B0                 jmp     short loc_401076
.text:004010B2 ; ---------------------------------------------------------------------------
.text:004010B2
.text:004010B2 loc_4010B2:                             ; CODE XREF: _main+8E↑j
.text:004010B2                                         ; _main+AE↑j
.text:004010B2                 jmp     short loc_4010B8
.text:004010B4 ; ---------------------------------------------------------------------------
.text:004010B4
.text:004010B4 loc_4010B4:                             ; CODE XREF: _main+6B↑j
.text:004010B4                 mov     [ebp+var_2D], 0
.text:004010B8
.text:004010B8 loc_4010B8:                             ; CODE XREF: _main:loc_4010B2↑j
.text:004010B8                 movzx   edx, [ebp+var_2D]
.text:004010BC                 test    edx, edx
.text:004010BE                 jz      short loc_4010FE
.text:004010C0                 push    offset sub_401170
.text:004010C5                 push    offset aHiStudentId564 ; "Hi Student ID 56406144!"
.text:004010CA                 push    offset unk_41A550 ; int
.text:004010CF                 call    sub_4030A0
.text:004010D4                 add     esp, 8
.text:004010D7                 mov     ecx, eax
.text:004010D9                 call    sub_401260
.text:004010DE                 push    offset sub_401170
.text:004010E3                 push    offset aYouSuccessfull ; "You successfully logged in!!!"
.text:004010E8                 push    offset unk_41A550 ; int
.text:004010ED                 call    sub_4030A0
.text:004010F2                 add     esp, 8
.text:004010F5                 mov     ecx, eax
.text:004010F7                 call    sub_401260
.text:004010FC                 jmp     short loc_40111C
.text:004010FE ; ---------------------------------------------------------------------------
.text:004010FE
.text:004010FE loc_4010FE:                             ; CODE XREF: _main+BE↑j
.text:004010FE                 push    offset sub_401170
.text:00401103                 push    offset aAccessDenined ; "Access denined!"
```

To do not jump to the area of the code where the program inform that the password is wrong, I changed that line where it checks the program flow for two **nop** instructions. Doing that the program will jump to the section where the correct password would lead. It can be seen in the image bellow:



```
.text:0040108E                 jnb     short loc_4010B2
.text:00401090                 mov     ecx, off_419000
.text:00401096                 add     ecx, [ebp+var_34]
.text:00401099                 movsx   edx, byte ptr [ecx+50h]
.text:0040109D                 mov     eax, [ebp+var_2C]
.text:004010A0                 add     eax, [ebp+var_34]
.text:004010A3                 movsx   ecx, byte ptr [eax]
.text:004010A6                 cmp     edx, ecx
.text:004010A8                 jz      short loc_4010B0
.text:004010AA                 mov     [ebp+var_2D], 0
.text:004010AE                 jmp     short loc_4010B2
.text:004010B0 ; ---------------------------------------------------------------------------
.text:004010B0
.text:004010B0 loc_4010B0:                             ; CODE XREF: _main+A8↑j
.text:004010B0                 jmp     short loc_401076
.text:004010B2 ; ---------------------------------------------------------------------------
.text:004010B2
.text:004010B2 loc_4010B2:                             ; CODE XREF: _main+8E↑j
.text:004010B2                                         ; _main+AE↑j
.text:004010B2                 jmp     short loc_4010B8
.text:004010B4 ; ---------------------------------------------------------------------------
.text:004010B4
.text:004010B4 loc_4010B4:                             ; CODE XREF: _main+6B↑j
.text:004010B4                 mov     [ebp+var_2D], 0
.text:004010B8
.text:004010B8 loc_4010B8:                             ; CODE XREF: _main:loc_4010B2↑j
.text:004010B8                 movzx   edx, [ebp+var_2D]
.text:004010BC                 test    edx, edx
.text:004010BE                 nop
.text:004010BF                 nop
.text:004010C0                 push    offset sub_401170
.text:004010C5                 push    offset aHiStudentId564 ; "Hi Student ID 56406144!"
.text:004010CA                 push    offset unk_41A550 ; int
.text:004010CF                 call    sub_4030A0
.text:004010D4                 add     esp, 8
.text:004010D7                 mov     ecx, eax
.text:004010D9                 call    sub_401260
.text:004010DE                 push    offset sub_401170
.text:004010E3                 push    offset aYouSuccessfull ; "You successfully logged in!!!"
.text:004010E8                 push    offset unk_41A550 ; int
.text:004010ED                 call    sub_4030A0
.text:004010F2                 add     esp, 8
.text:004010F5                 mov     ecx, eax
.text:004010F7                 call    sub_401260
.text:004010FC                 jmp     short loc_40111C
.text:004010FE ; ---------------------------------------------------------------------------
.text:004010FE                 push    offset sub_401170
.text:00401103                 push    offset aAccessDenined ; "Access denined!"
.text:00401108                 push    offset unk_41A550 ; int
```

After the changes being made I exported the .dif file using the ADA software. To apply the changes I executed the C code available at *http://idabook.com/examples/chapter_14/ida_patcher.c*. After compiled this C code will receive as input the .exe file and the .dif file and then merge the files. The .exe file will be modified so I had to make sure that I made a copy of the executable file.

4 Question 4

4.1 a)

Just like in the question 3 the password is hard coded in the program. However in this example the password isn't stored as plain text, what is stored is the hash of the password using SHA1. The hash is stored in the address **.rdata:00414212 byte_414212**. In this case the has is stored in a sequence of hex values. Because the SHA1 only have 40 characters output only the first 20 hex values. For cracking the password in this questions I used the Hashcat software. Because I have access to a graphic card I used cudaHashCat.

The software can be downloaded for free at *http://hashcat.net/oclhashcat/*

To use that software I used the following steps:

Download the program

Unzip the file

Open CMD

Go to the hashcat directory file and run the following command:

.cudaHashcat64.exe -m 100 -a 3 -1 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890!@#$%^&*()_+-=' hash.txt

Here is the output of the hashcat program after cracking that hash 21f0e9b168b2d5870906f1b86693754c0ddae05c:7NeW0i

Status.........: Cracked

Input.Mode.....: Mask (?1?1?1?1?1?1) [6]

Hash.Target....: 21f0e9b168b2d5870906f1b86693754c0ddae05c

Hash.Type......: SHA1

Time.Started...: Mon Nov 02 20:33:38 2015 (7 mins, 28 secs)

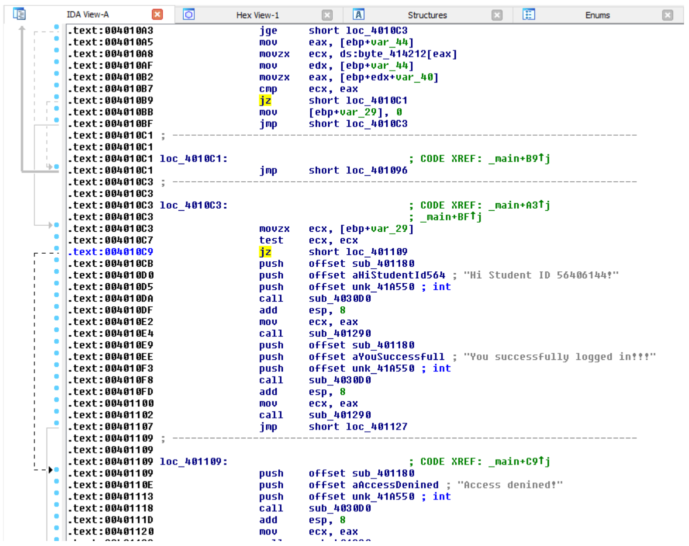
Speed.GPU.#1...: 198.8 MH/s

Recovered......: 1/1 (100.00%) Digests, 1/1 (100.00%) Salts
Progress.......: 89472892928/192699928576 (46.43%)
Rejected.......: 0/89472892928 (0.00%)
Restore.Point..: 15466496/33362176 (46.36%)
HWMon.GPU.#1...: 0% Util, 66c Temp, N/A Fan
Started: Mon Nov 02 20:33:38 2015
Stopped: Mon Nov 02 20:41:11 2015
The password for this file is: **7NeW0i**

4.2 b)

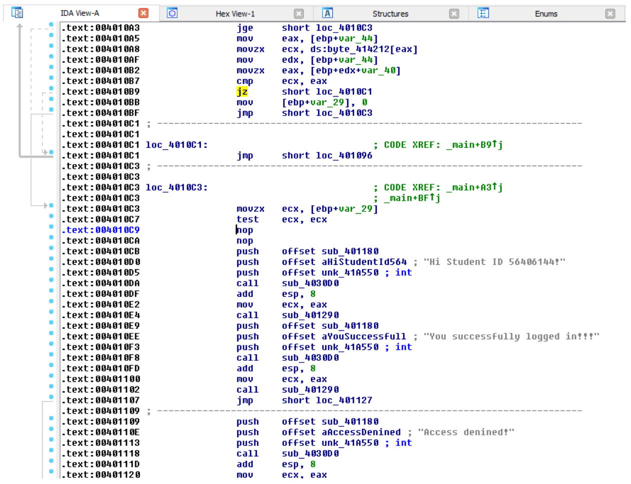
For this part a similar solution was used just like in the **Question 3**

The iamge bellow shows the code without the patch



```
.text:004010A3                 jge     short loc_4010C3
.text:004010A5                 mov     eax, [ebp+var_44]
.text:004010A8                 movzx   ecx, ds:byte_414212[eax]
.text:004010AF                 mov     edx, [ebp+var_44]
.text:004010B2                 movzx   eax, [ebp+edx+var_40]
.text:004010B7                 cmp     ecx, eax
.text:004010B9                 jz      short loc_4010C1
.text:004010BB                 mov     [ebp+var_29], 0
.text:004010BF                 jmp     short loc_4010C3
.text:004010C1 ; ---------------------------------------------------------------------------
.text:004010C1
.text:004010C1 loc_4010C1:                             ; CODE XREF: _main+B9↑j
.text:004010C1                 jmp     short loc_401096
.text:004010C3 ; ---------------------------------------------------------------------------
.text:004010C3
.text:004010C3 loc_4010C3:                             ; CODE XREF: _main+A3↑j
.text:004010C3                                         ; _main+BF↑j
.text:004010C3                 movzx   ecx, [ebp+var_29]
.text:004010C7                 test    ecx, ecx
.text:004010C9                 jz      short loc_401109
.text:004010CB                 push    offset sub_401180
.text:004010D0                 push    offset aHiStudentId564 ; "Hi Student ID 56406144!"
.text:004010D5                 push    offset unk_41A550 ; int
.text:004010DA                 call    sub_4030D0
.text:004010DF                 add     esp, 8
.text:004010E2                 mov     ecx, eax
.text:004010E4                 call    sub_401290
.text:004010E9                 push    offset sub_401180
.text:004010EE                 push    offset aYouSuccessfull ; "You successfully logged in!!!"
.text:004010F3                 push    offset unk_41A550 ; int
.text:004010F8                 call    sub_4030D0
.text:004010FD                 add     esp, 8
.text:00401100                 mov     ecx, eax
.text:00401102                 call    sub_401290
.text:00401107                 jmp     short loc_401127
.text:00401109 ; ---------------------------------------------------------------------------
.text:00401109
.text:00401109 loc_401109:                             ; CODE XREF: _main+C9↑j
.text:00401109                 push    offset sub_401180
.text:0040110E                 push    offset aAccessDenined ; "Access denined!"
.text:00401113                 push    offset unk_41A550 ; int
.text:00401118                 call    sub_4030D0
.text:0040111D                 add     esp, 8
.text:00401120                 mov     ecx, eax
```

Now using the patch to accept any password



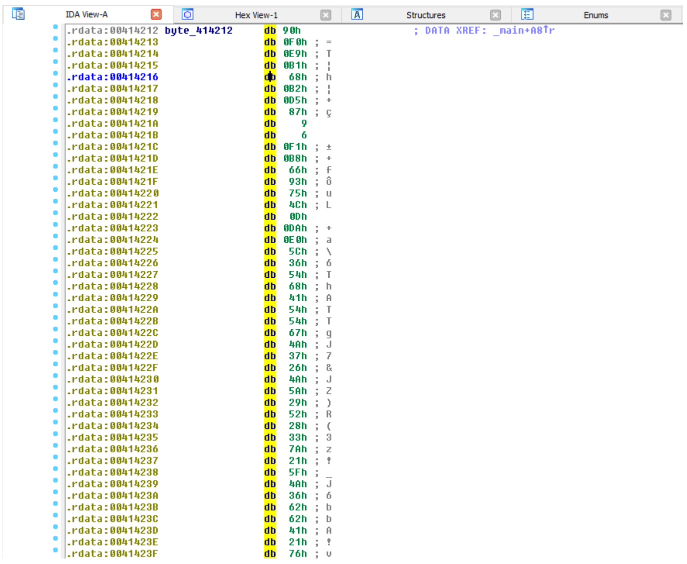
```
.text:004010A3                 jge     short loc_4010C3
.text:004010A5                 mov     eax, [ebp+var_44]
.text:004010A8                 movzx   ecx, ds:byte_414212[eax]
.text:004010AF                 mov     edx, [ebp+var_44]
.text:004010B2                 movzx   eax, [ebp+edx+var_40]
.text:004010B7                 cmp     ecx, eax
.text:004010B9                 jz      short loc_4010C1
.text:004010BB                 mov     [ebp+var_29], 0
.text:004010BF                 jmp     short loc_4010C3
.text:004010C1 ; ---------------------------------------------------------------------------
.text:004010C1
.text:004010C1 loc_4010C1:                             ; CODE XREF: _main+B9↑j
.text:004010C1                 jmp     short loc_401096
.text:004010C3 ; ---------------------------------------------------------------------------
.text:004010C3
.text:004010C3 loc_4010C3:                             ; CODE XREF: _main+A3↑j
.text:004010C3                                         ; _main+BF↑j
.text:004010C3                 movzx   ecx, [ebp+var_29]
.text:004010C7                 test    ecx, ecx
.text:004010C9                 nop
.text:004010CA                 nop
.text:004010CB                 push    offset sub_401180
.text:004010D0                 push    offset aHiStudentId564 ; "Hi Student ID 56406144!"
.text:004010D5                 push    offset unk_41A550 ; int
.text:004010DA                 call    sub_4030D0
.text:004010DF                 add     esp, 8
.text:004010E2                 mov     ecx, eax
.text:004010E4                 call    sub_401290
.text:004010E9                 push    offset sub_401180
.text:004010EE                 push    offset aYouSuccessfull ; "You successfully logged in!!!"
.text:004010F3                 push    offset unk_41A550 ; int
.text:004010F8                 call    sub_4030D0
.text:004010FD                 add     esp, 8
.text:00401100                 mov     ecx, eax
.text:00401102                 call    sub_401290
.text:00401107                 jmp     short loc_401127
.text:00401109 ; ---------------------------------------------------------------------------
.text:00401109                 push    offset sub_401180
.text:0040110E                 push    offset aAccessDenined ; "Access denined!"
.text:00401113                 push    offset unk_41A550 ; int
.text:00401118                 call    sub_4030D0
.text:0040111D                 add     esp, 8
.text:00401120                 mov     ecx, eax
```

After the changes being made I exported the .dif file using the ADA software. To apply the changes I executed the C code available at *http://idabook.com/examples/chapter_14/ida_patcher.c*. After compiled this C code will receive as input the .exe file and the .dif file and then merge the files. The .exe file will be modified so I had to make sure that I made a copy of the executable file.

4.3 c)

To solve that question I had to compute manually the hash of the desired password (for that a python script was used) and than insert that hash into the executable file changing the original hard coded hash value. In that way when the program computes the hash it will match with that was modified.

The lines where the code might be modified are represented in the image below:



```
.rdata:00414212 byte_414212     db  90h                 ; DATA XREF: _main+A8↑r
.rdata:00414213                 db 0F0h ; =
.rdata:00414214                 db 0E9h ; T
.rdata:00414215                 db 0B1h ; ¦
.rdata:00414216                 db  68h ; h
.rdata:00414217                 db 0B2h ; ¦
.rdata:00414218                 db 0D5h ; +
.rdata:00414219                 db  87h ; ç
.rdata:0041421A                 db    9
.rdata:0041421B                 db    6
.rdata:0041421C                 db 0F1h ; ±
.rdata:0041421D                 db 0B8h ; +
.rdata:0041421E                 db  66h ; f
.rdata:0041421F                 db  93h ; ô
.rdata:00414220                 db  75h ; u
.rdata:00414221                 db  4Ch ; L
.rdata:00414222                 db  0Dh
.rdata:00414223                 db 0DAh ; +
.rdata:00414224                 db 0E0h ; a
.rdata:00414225                 db  5Ch ; \
.rdata:00414226                 db  36h ; 6
.rdata:00414227                 db  54h ; T
.rdata:00414228                 db  68h ; h
.rdata:00414229                 db  41h ; A
.rdata:0041422A                 db  54h ; T
.rdata:0041422B                 db  54h ; T
.rdata:0041422C                 db  67h ; g
.rdata:0041422D                 db  4Ah ; J
.rdata:0041422E                 db  37h ; 7
.rdata:0041422F                 db  26h ; &
.rdata:00414230                 db  4Ah ; J
.rdata:00414231                 db  5Ah ; Z
.rdata:00414232                 db  29h ; )
.rdata:00414233                 db  52h ; R
.rdata:00414234                 db  28h ; (
.rdata:00414235                 db  33h ; 3
.rdata:00414236                 db  7Ah ; z
.rdata:00414237                 db  21h ; !
.rdata:00414238                 db  5Fh ; _
.rdata:00414239                 db  4Ah ; J
.rdata:0041423A                 db  36h ; 6
.rdata:0041423B                 db  62h ; b
.rdata:0041423C                 db  62h ; b
.rdata:0041423D                 db  41h ; A
.rdata:0041423E                 db  21h ; !
.rdata:0041423F                 db  76h ; v
```

After modifying the hash code the same methodology of the Question 3 b) and Question 4 b) will be used. First the .dif file was exported using the ADA software. To apply the changes the C code available at *http://idabook.com/examples/chapter_14/ida_patcher.c* was executed. After compiled, this C code will receive as input the .exe file and the .dif file and then merge the files. The .exe file will be modified so I had to make sure that I made a copy of the executable file.

After this process the file will accept only the other the password that generated the hash computed earlier.

5 Appendix

All the codes for this project can be found at *https://github.com/faellacurcio/AssignmentPassword.git*

References

[1] 14.1. hashlib Secure hashes and message digests [Online]. Available: https://docs.python.org/2/library/hashlib.html

[2] string - Generate all possible passwords based on character mapping possible in 3 lines of python? [Online]. Available: http://stackoverflow.com/questions/10082094/generate-all-possible-passwords-based-on-character-mapping-possible-in-3-line

[3] IDA: about [Online]. Available: https://www.hex-rays.com/products/ida/

[4] hashcat - advanced password recovery [Online]. Available: http://hashcat.net/hashcat/

[5] coder 32 edition —X86 Opcode and instruction reference [Online]. Available: http://ref.x86asm.net/coder32.html

[6] [Online]. Available: http://idabook.com/examples/chapter_14/ida_patcher.c

[7] dll used to run program 4 download. Available: https://dl.dropboxusercontent.com/u/14053604/libeay32.dll

[8] [Online]. Available:

Appendix

1 Question 1

```
import hashlib
import binascii

value = 0
while (value <= 9999):
        tested = '0000'+str(value)
        tested = (tested)[len(tested)-4:]
        m = hashlib.sha1()
        m.update(bytes(("It"+tested).encode()))
        hexa = m.hexdigest();
        if(hexa == '6D209CE47FEC653F2947F0E3D09580130C05C3CC'.lower()):
                print ('Found Value!: ', value)
                print('Hash: ', hexa)
                break
        del m
        value +=1
```

2 Question 2

2.1 Code 1

```
import itertools
import hashlib

res = itertools.product('abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVW\
XYZ1234567890!@#$%^&*()_+-=', repeat=6)

counter = 0

for case in res:
        if (counter < 8860000000):
                counter += 1
                if (counter%1000000000 == 0):
                        print ("counter in: ", counter)
        else:
                case = "NT"+''.join(case)
                m = hashlib.sha1()
                m.update(bytes((case).encode()))
                hexa = m.hexdigest();
                if(hexa == '53425620DAF31B5659A7D96C168C4ED82CAAED13'.lower()):
                        print ('Found Value!: ', case)
                        print('Hash: ', hexa)
                        break
                del m
                counter += 1
                if(counter%10000000 == 0):
                        print ("Tested hashes: ",counter," Last password:", case[2:],\
                         "last hash:", hexa)
```

2.2 Code 2

```
import itertools
import hashlib

res = itertools.product('=-+_)(*&^%$#@!0987654321ZYXWVUTSRQPONMLKJIHGFEDCBAzy\
xwvutsrqponmlkjihgfedcba', repeat=6)

counter = 0

for case in res:
        if (counter < 5000000000):
                counter += 1
                if (counter%1000000000 == 0):
                        print ("counter in: ", counter)
        else:
                case = "NT"+''.join(case)
                m = hashlib.sha1()
                m.update(bytes((case).encode()))
                hexa = m.hexdigest();
                if(hexa == '53425620DAF31B5659A7D96C168C4ED82CAAED13'.lower()):
                        print ('Found Value!: ', value)
                        print('Hash: ', hexa)
                        break
                del m
                counter += 1
                if(counter%10000000 == 0):
                        print ("Tested hashes: ",counter," Last password:", case[2:],\
                         "last hash:", hexa)
```