

Assignment #4

CPEN 442

Kaibo Ma (32400129)

Department of Electrical and Computer Engineering
University of British Columbia
Vancouver, Canada

I. PROBLEM #1

mypwd:co5916

where co is the salt.

$$10^4 = 10,000$$

$$2^{13} = 8192$$

$$2^{14} = 1684$$

It is about 13~14 bits of entropy. I used john the ripper for this assignment. I was able to retrieve the first password in mere seconds running with 32 different forks. I found that 'co' is the salt.

```
dhcpc-206-87-113-156:Deliverables kaiboma$ john --fork=32 --mask=7d7d7d q1.txt
Warning: detected hash type "Raw-SHA1", but the string is also recognized as "Raw-SHA1-Link
Use the "--format=Raw-SHA1-Linkedin" option to force loading these as that type instead
Warning: detected hash type "Raw-SHA1", but the string is also recognized as "Raw-SHA"
Use the "--format=Raw-SHA" option to force loading these as that type instead
Warning: detected hash type "Raw-SHA1", but the string is also recognized as "ripemd-160"
Use the "--format=ripemd-160" option to force loading these as that type instead
Warning: detected hash type "Raw-SHA1", but the string is also recognized as "Raw-SHA1-ng"
Use the "--format=Raw-SHA1-ng" option to force loading these as that type instead
Warning: detected hash type "Raw-SHA1", but the string is also recognized as "Raw-SHA1-openc
Use the "--format=Raw-SHA1-openc1" option to force loading these as that type instead
Loaded 1 password hash (Raw-SHA1 [SHA1 128/128 SSE3.8x])
No password hashes left to crack (see FAQ)
dhcpc-206-87-113-156:Deliverables kaiboma$ john --show q1.txt
mypwd:co5916
```

Fig 1. Result of JTR for Question 1.

II. PROBLEM #2

mypwd:Dkqk&UvR

where Dk in the beginning is the salt

$$79^6 = 243,087,455,521$$

$$2^{38} = 274,877,906,944$$

It is about 38 bits of entropy. Used the same program as question with 32 different forks and ran in mask mode. This specified the length and key space to narrow down the search. It took about 2 hours and 30 mins to complete on my Macbook. This would have been faster running off a GPU or a faster CPU.

```
3 0g 0:01:54:54 10.89% (ETA: 15:09:20) 0g/s 362818p/s 362818c/s 362818C/s Dk=xf10&. DkDxf10&
7 0g 0:01:54:54 10.86% (ETA: 15:12:05) 0g/s 361880p/s 361880c/s 361880C/s Dk_6fm,2..Dk6fm,2
10 0g 0:01:54:54 10.89% (ETA: 15:09:42) 0g/s 362696p/s 362696c/s 362696C/s DkHp3j#;. DkOp3j#;
18 0g 0:01:54:54 10.90% (ETA: 15:08:28) 0g/s 363119p/s 363119c/s 363119C/s Dk_Pb9KR..Dk3PB9KR
19 0g 0:01:54:54 10.88% (ETA: 15:09:47) 0g/s 362663p/s 362663c/s 362663C/s DkE*8Nu..DkL*8Nu
20 0g 0:01:54:54 10.88% (ETA: 15:10:25) 0g/s 362448p/s 362448c/s 362448C/s DkFmu9eX..DkFmu9eX
29 0g 0:01:54:54 10.86% (ETA: 15:11:46) 0g/s 361985p/s 361985c/s 361985C/s Dk7uPJs..Dk2uPJs
30 0g 0:01:54:54 10.88% (ETA: 15:10:28) 0g/s 362430p/s 362430c/s 362430C/s Dk=JWGV..Dk=JWGV
4 0g 0:01:54:54 10.87% (ETA: 15:11:06) 0g/s 362214p/s 362214c/s 362214C/s Dk=XRg5)..Dk2XRg5)
31 0g 0:01:54:54 10.86% (ETA: 15:11:58) 0g/s 361916p/s 361916c/s 361916C/s Dk15;VDy..Dk15;VDy
16 0g 0:01:54:54 10.87% (ETA: 15:11:12) 0g/s 362179p/s 362179c/s 362179C/s DkY4s+qL..Dk15s+qL
28 0g 0:01:54:54 10.90% (ETA: 15:08:06) 0g/s 363246p/s 363246c/s 363246C/s DkQ2&Mp..DkX2&Mp
2 0g 0:01:54:54 10.88% (ETA: 15:10:41) 0g/s 362355p/s 362355c/s 362355C/s DkIneb;#..DkIneb;#
27 0g 0:01:54:54 10.89% (ETA: 15:09:00) 0g/s 362932p/s 362932c/s 362932C/s Dk5*XRpm..Dk5*XRpm
32 0g 0:01:54:54 10.88% (ETA: 15:10:00) 0g/s 362591p/s 362591c/s 362591C/s Dk2a*Aj..Dk2a*Aj
6 0g 0:01:54:54 10.88% (ETA: 15:10:17) 0g/s 362490p/s 362490c/s 362490C/s Dk6+bo//..Dk6+bo//
15 0g 0:01:54:54 10.87% (ETA: 15:11:01) 0g/s 362240p/s 362240c/s 362240C/s Dk8?I..DkA&F?I
24 0g 0:01:54:54 10.88% (ETA: 15:10:35) 0g/s 362392p/s 362392c/s 362392C/s Dkt1?EYd..Dk1?EYd
5 0g 0:01:54:54 10.87% (ETA: 15:11:21) 0g/s 362125p/s 362125c/s 362125C/s DkR*k12..DkY*k12
12 0g 0:01:54:54 10.86% (ETA: 15:12:20) 0g/s 361788p/s 361788c/s 361788C/s Dkrrq{0..Dkrrq{0
1 0g 0:02:30:00 DONE (2016-10-27 00:04) 0g/s 2552Kp/s 2552Kc/s 2552KC/s Dkv"|"..Dk"|"
Waiting for children to terminate
Dkqk&UvR (mypwd)
18 1g 0:02:38:32 DONE (2016-10-27 00:12) 0 000105g/s 359854p/s 359854c/s 359854C/s Dk1k&UvR..Dk
7 0g 0:02:39:00 DONE (2016-10-27 00:13) 0g/s 359510p/s 359510c/s 359510C/s Dk123<82..DkPz3<82
10 0g 0:02:39:00 DONE (2016-10-27 00:13) 0g/s 359950p/s 359950c/s 359950C/s Dk:G1J//..DkAG1J//
9 0g 0:02:39:00 DONE (2016-10-27 00:13) 0g/s 360661p/s 360661c/s 360661C/s Dk_#(N28..Dk3#(N28
14 0g 0:02:39:00 DONE (2016-10-27 00:13) 0g/s 359428p/s 359428c/s 359428C/s Dk }'PWG..Dk }'PWG
12 0g 0:02:39:00 DONE (2016-10-27 00:13) 0g/s 359241p/s 359241c/s 359241C/s Dk40&H(A)..Dk:0&H(A)
19 0g 0:02:39:00 DONE (2016-10-27 00:13) 0g/s 360102p/s 360102c/s 360102C/s DkC7HfSU..Dk7HfSU
20 0g 0:02:39:00 DONE (2016-10-27 00:13) 0g/s 359451p/s 359451c/s 359451C/s Dk3t-bpX..Dk:t-bpX
23 0g 0:02:39:00 DONE (2016-10-27 00:13) 0g/s 359334p/s 359334c/s 359334C/s Dk_vfIga..Dk_vfIga
5 0g 0:02:39:00 DONE (2016-10-27 00:13) 0g/s 359378p/s 359378c/s 359378C/s Dk+u14>..Dk2u14>
3 0g 0:02:39:00 DONE (2016-10-27 00:13) 0g/s 359990p/s 359990c/s 359990C/s Dkyn5D&..Dk1o5D&
21 0g 0:02:39:00 DONE (2016-10-27 00:13) 0g/s 359433p/s 359433c/s 359433C/s Dk'Twmd[..Dk'Twmd[
26 0g 0:02:39:00 DONE (2016-10-27 00:13) 0g/s 360051p/s 360051c/s 360051C/s Dkssz'z'j..Dkssz'z'j
27 0g 0:02:39:00 DONE (2016-10-27 00:13) 0g/s 360031p/s 360031c/s 360031C/s Dk(UD)[m..Dk(UD)[m
17 0g 0:02:39:00 DONE (2016-10-27 00:13) 0g/s 359196p/s 359196c/s 359196C/s DkACFVyo..DkACFVyo
2 0g 0:02:39:00 DONE (2016-10-27 00:13) 0g/s 359759p/s 359759c/s 359759C/s DkR10&f..DkY10&f
28 0g 0:02:39:00 DONE (2016-10-27 00:13) 0g/s 360295p/s 360295c/s 360295C/s DkH1V[L..DkH1V[L
29 0g 0:02:39:00 DONE (2016-10-27 00:13) 0g/s 359496p/s 359496c/s 359496C/s Dk-MBUs..Dk-MBUs
6 0g 0:02:39:00 DONE (2016-10-27 00:13) 0g/s 359833p/s 359833c/s 359833C/s DkbpE+;..Dk1PE+;
14 0g 0:02:39:00 DONE (2016-10-27 00:13) 0g/s 360073p/s 360073c/s 360073C/s DkGyytd..DkGyytd
26 0g 0:02:39:00 DONE (2016-10-27 00:13) 0g/s 359498p/s 359498c/s 359498C/s DkH1V[L..DkH1V[L
30 0g 0:02:39:00 DONE (2016-10-27 00:13) 0g/s 359621p/s 359621c/s 359621C/s Dk4&5d'..Dk:5d'5v
31 0g 0:02:39:00 DONE (2016-10-27 00:13) 0g/s 359478p/s 359478c/s 359478C/s DkK<5Py..DkK<5Py
22 0g 0:02:39:00 DONE (2016-10-27 00:13) 0g/s 359532p/s 359532c/s 359532C/s DkExhj'..DkLx)hj'
11 0g 0:02:39:00 DONE (2016-10-27 00:13) 0g/s 359909p/s 359909c/s 359909C/s DkW'RL>..Dk'RL>
15 0g 0:02:39:00 DONE (2016-10-27 00:13) 0g/s 359423p/s 359423c/s 359423C/s DkPp-R..DkPp-R
8 0g 0:02:39:00 DONE (2016-10-27 00:13) 0g/s 359159p/s 359159c/s 359159C/s Dk=;55..DkF=;55
4 0g 0:02:39:00 DONE (2016-10-27 00:13) 0g/s 359484p/s 359484c/s 359484C/s Dkji13A)..Dkqf13A)
13 0g 0:02:39:00 DONE (2016-10-27 00:13) 0g/s 359669p/s 359669c/s 359669C/s DkK*10&D..DkR*10&D
25 0g 0:02:39:00 DONE (2016-10-27 00:13) 0g/s 359673p/s 359673c/s 359673C/s DkG1sag..Dk=1sag
32 0g 0:02:39:00 DONE (2016-10-27 00:13) 0g/s 360095p/s 360095c/s 360095C/s Dk1W-Mj..Dk5W-Mj
Session completed
Kaibos-MacBook-Pro:Assignment4 kaiboma$
```

Fig 2. Result of JTR for Question 2.

III. PROBLEM #3

mypwd:5Wo-u3ozhz8q1M^d

$$79^{16} = 2.3016191 \times 10^{30}$$

$$2^{101} = 2.5353012 \times 10^{30}$$

a) It is about 101 bits of entropy. I obtained the password by looking at the main function assembly code. I realized that it checks if your input string is the same length as a hard coded length of 10h which is 16.

```

push    ecx
call    sub_40CD40
add     esp, 4
cmp     eax, 10h
jnz     short loc_405424

```

Fig 3. The string length comparison check

Then the code will check with a hard coded randomized string inside .rdata. It will compare a certain offset and match your input string with that string.

```

mov     ecx, off_42D000
add     ecx, [ebp+var_14]
movsx   edx, byte ptr [ecx+20h]
mov     eax, [ebp+var_18]
add     eax, [ebp+var_14]
movsx   ecx, byte ptr [eax]
cmp     edx, ecx
jz      short loc_405420

```

Fig 4. The offset for comparison string

```

db 't0FZNjaquZjP$;(1GeZZW!c6VHWtqrB5Wo-u3ozhz8q1M^dz791H-)G8XdgHz1+G'
; DATA XREF: data:off_42D000,jc
db '59)b $2$yRyd0fjqVO!Mh(626pUZh^6X07)2)bD*hu1EUD+S(N00$iz!crt5xc6G0'
db 'zacjFUUk*',0

```

Fig 5. The whole comparison string where my password resides.

I used a script I created to help me find my password inside the string (attached findpw.py).

b) I noticed in the assembly that there is a “test” opcode followed by a “jz”. I knew that it would either pass or fail. Hence I made the “jz” opcode to “nop” which would skip the jump and proceed with logging in successfully.

```

loc_405420:
movzx   edx, [ebp+var_D]
test     edx, edx
jz       short loc_405468

push    offset sub_401910
push    offset aH1StudentId324 ; "H1 Student ID 32400129"
push    offset unk_42B028
call    sub_401480
add     esp, 8
mov     ecx, eax
call    sub_402980
push    offset sub_401910
push    offset aYouSuccessful ; "You successfully logged in!!"
call    sub_401480
add     esp, 8
mov     ecx, eax
call    sub_402980
jmp     short loc_40548c

```

Fig 6. The checking condition for pass or fail

This is the .dif file I created with IDA:

```

This difference file has been created by IDA
32400129.program1.exe
0000482E: 74 90

```

IV. PROBLEM #4

mypwd:U3H\$ki

79^6 = 243,087,455,521

2^38 = 274,877,906,944

It is about 38 bits of entropy.

This is the .dif file I created with IDA:

This difference file has been created by IDA

32400129.program2.exe

00004859: 74 90

a) Unlike Question 3 this programs password is encrypted with SHA-1. I noticed from the assembly that it will check byte by byte in a lookup table imported by libeay32.dll. The password starts at the address referenced by ‘byte_41F234’ at address 41F234. Then in a counter loop of 20 times (this gave away that it was SHA1 as it takes 20 bytes). In the following diagram, this is my SHA1 hashed password in the .rdata.

```

.rdata:0041F234 byte_41F234 db 0DCh ; DATA XREF: _main+87r
.rdata:0041F236 db 2Ch ;
.rdata:0041F237 db 1 ;
.rdata:0041F237 db 0D9h ;
.rdata:0041F238 db 64h ;
.rdata:0041F239 db 30h ; 0
.rdata:0041F23A db 11h ;
.rdata:0041F23B db 4Ch ; L
.rdata:0041F23C db 0F5h ;
.rdata:0041F23D db 0A2h ;
.rdata:0041F23E db 0ECh ;
.rdata:0041F23F db 0DBh ;
.rdata:0041F240 db 98h ;
.rdata:0041F241 db 38h ; 8
.rdata:0041F242 db 58h ; X
.rdata:0041F243 db 0F5h ;
.rdata:0041F244 db 8Ch ;

```

Fig 7. The hashed password value.

Once obtaining this value and copying it over to a .txt file I ran john the ripper on 32 forks. As a result, I obtained my password: U3H\$ki which tested perfectly into the program provided. The brute force search took about 5 hours and 25 minutes to complete.

```

24 0g 0:05:24:50 40.55% (ETA: 08:52:16) 0g/s 477988p/s 477988c/s 477988C/s %2'Ne...2'Ne
0 0g 0:05:24:50 40.59% (ETA: 08:51:25) 0g/s 478418p/s 478418c/s 478418C/s To:b)5...[o:b)5
14 0g 0:05:24:51 40.56% (ETA: 08:52:01) 0g/s 478047p/s 478047c/s 478047C/s J.51kG...Q.51kG
26 0g 0:05:24:50 40.59% (ETA: 08:51:26) 0g/s 478406p/s 478406c/s 478406C/s Tbj8HK...[bj8HK
11 0g 0:05:24:50 40.56% (ETA: 08:52:06) 0g/s 478009p/s 478009c/s 478009C/s H*at>...0*at>
3 0g 0:05:24:50 40.66% (ETA: 08:49:59) 0g/s 479278p/s 479278c/s 479278C/s [X]f-'.b[X]f-
12 0g 0:05:24:50 40.55% (ETA: 08:52:09) 0g/s 477981p/s 477981c/s 477981C/s J=4dqA...Q=4dqA
20 0g 0:05:24:50 40.53% (ETA: 08:52:43) 0g/s 477647p/s 477647c/s 477647C/s UNCLTY...UNCLTY
4 0g 0:05:24:50 40.57% (ETA: 08:51:46) 0g/s 478207p/s 478207c/s 478207C/s 2LXQ*...9LXQ*
21 0g 0:05:24:50 40.55% (ETA: 08:52:10) 0g/s 477895p/s 477895c/s 477895C/s t\j\VA...t\j\VA
29 0g 0:05:24:50 40.56% (ETA: 08:51:59) 0g/s 478078p/s 478078c/s 478078C/s y\3t...1\3t
9 0g 0:05:24:50 40.52% (ETA: 08:52:48) 0g/s 477593p/s 477593c/s 477593C/s t'NRz8...t'NRz8
2 0g 0:05:24:50 40.54% (ETA: 08:52:21) 0g/s 477867p/s 477867c/s 477867C/s hqyC0$...onyC0$
16 0g 0:05:24:51 40.59% (ETA: 08:51:26) 0g/s 478405p/s 478405c/s 478405C/s >1:zeM...E1:zeM
23 0g 0:05:24:50 40.52% (ETA: 08:52:44) 0g/s 477635p/s 477635c/s 477635C/s az-[Pb...h:z-[Pb
5 0g 0:05:24:50 40.61% (ETA: 08:51:04) 0g/s 478626p/s 478626c/s 478626C/s j-*A1...j-*A1
U3H$ki (mypwd)

```

Fig 8. The resulting brute force search on the password.

b) Similar to question 3, the same ‘test’ and ‘jz’ functions are called. This allowed me to just take ‘jz’ written in 74h and change to 90h which is ‘nop’.

c) To write a patch file to make our .exe file accept and password we desire. I created a python script that seeks out the correct address offset of the beginning of where the SHA1 password is stored. Then it will take the arguments you run with the python script and hash that value and overwrite the address location that contains the previous values. Image below shows how I run my script with argument ‘qwerty’ that patches the .exe file and use the new ‘qwerty’ password as input to .exe. The .exe accepts the new password:

```

(dhcp-206-87-113-156:src_files kaiboma$ wine 32400129.program2.exe
fixme:module:load_library unsupported flag(s) used (flags: 0x00000800)
fixme:module:load_library unsupported flag(s) used (flags: 0x00000800)
fixme:module:load_library unsupported flag(s) used (flags: 0x00000800)
fixme:module:load_library unsupported flag(s) used (flags: 0x00000800)
fixme:module:load_library unsupported flag(s) used (flags: 0x00000800)
fixme:module:load_library unsupported flag(s) used (flags: 0x00000800)
fixme:module:load_library unsupported flag(s) used (flags: 0x00000800)
fixme:module:load_library unsupported flag(s) used (flags: 0x00000800)
Application #2 RE (CPEN 442 Assignment 4, Task 4).
Please enter password:qwerty
Access denied!

[4]+ Stopped wine 32400129.program2.exe
(dhcp-206-87-113-156:src_files kaiboma$ python q4.py qwerty
Applying patch...
Patched! Your new password is: qwerty
(dhcp-206-87-113-156:src_files kaiboma$ wine 32400129.program2.exe
fixme:module:load_library unsupported flag(s) used (flags: 0x00000800)
fixme:module:load_library unsupported flag(s) used (flags: 0x00000800)
fixme:module:load_library unsupported flag(s) used (flags: 0x00000800)
fixme:module:load_library unsupported flag(s) used (flags: 0x00000800)
fixme:module:load_library unsupported flag(s) used (flags: 0x00000800)
fixme:module:load_library unsupported flag(s) used (flags: 0x00000800)
fixme:module:load_library unsupported flag(s) used (flags: 0x00000800)
fixme:module:load_library unsupported flag(s) used (flags: 0x00000800)
fixme:module:load_library unsupported flag(s) used (flags: 0x00000800)
Application #2 RE (CPEN 442 Assignment 4, Task 4).
Please enter password:qwerty
Hi Student ID 32400129!
You successfully logged in!!!

```

Fig 9. Successful usage of q4.py script that patches the .exe

```

1 import hashlib
2 import array
3 from sys import argv,exit
4
5 def main():
6     if len(argv) < 2:
7         print "Please Enter Desired Password:"
8         print "Usage: %s <password>" % (argv[0])
9         exit(0)
10
11     else:
12         print "Applying patch..."
13     try:
14         f = open('32400129.program2.exe', 'r+b')
15         sha1 = hashlib.sha1(argv[1]).hexdigest()
16         f.seek(0x1E034)
17         f.write(sha1.decode("hex"))
18         f.close()
19         print "Patched! Your new password is:", (argv[1])
20     except Exception, e:
21         print "Error..."
22         exit(1)
23
24 if __name__ == "__main__":
25     main()

```

Fig 10. My python script that patches the program to any password you want.

I have also attached q4.py along with the email.