HY455 Cyber Security
Assignment 3
Chris Papastamos (csd4569)

Question 1

In order to identify the hash algorithm used on each of the given hases, I used *hashid* which is a tool used for this exact job. I also used the option -j that show the format code that JTR needs in order to run on these hashes for the next question. This tool cannot pinpoint the exact algorithm used for the hash, but on the contrary it suggests a list of algorithms that could produce a hash like the given.

For the first hash the most possible is some version of the Message Digest (MD) algorithm

```
┌──(kali㉿kali)-[~]
└─$ hashid 491fe2ec1b129dc19210e65931f4c23f -j
Analyzing '491fe2ec1b129dc19210e65931f4c23f'
[+] MD2 [JtR Format: md2]
[+] MD5 [JtR Format: raw-md5]
[+] MD4 [JtR Format: raw-md4]
[+] Double MD5
[+] LM [JtR Format: lm]
[+] RIPEMD-128 [JtR Format: ripemd-128]
[+] Haval-128 [JtR Format: haval-128-4]
[+] Tiger-128
[+] Skein-256(128)
[+] Skein-512(128)
[+] Lotus Notes/Domino 5 [JtR Format: lotus5]
[+] Skype
[+] Snefru-128 [JtR Format: snefru-128]
[+] NTLM [JtR Format: nt]
[+] Domain Cached Credentials [JtR Format: mscach]
[+] Domain Cached Credentials 2 [JtR Format: mscach2]
[+] DNSSEC(NSEC3)
[+] RAdmin v2.x [JtR Format: radmin]
```

Similar to the first one, the second hash seems to be an MD hash as well

```
┌──(kali㉿kali)-[~]
└─$ hashid 1a7f2a5ad77128b2f81feddac78df213 -j
Analyzing '1a7f2a5ad77128b2f81feddac78df213'
[+] MD2 [JtR Format: md2]
[+] MD5 [JtR Format: raw-md5]
[+] MD4 [JtR Format: raw-md4]
[+] Double MD5
[+] LM [JtR Format: lm]
[+] RIPEMD-128 [JtR Format: ripemd-128]
[+] Haval-128 [JtR Format: haval-128-4]
[+] Tiger-128
[+] Skein-256(128)
[+] Skein-512(128)
[+] Lotus Notes/Domino 5 [JtR Format: lotus5]
[+] Skype
[+] Snefru-128 [JtR Format: snefru-128]
[+] NTLM [JtR Format: nt]
[+] Domain Cached Credentials [JtR Format: mscach]
[+] Domain Cached Credentials 2 [JtR Format: mscach2]
[+] DNSSEC(NSEC3)
[+] RAdmin v2.x [JtR Format: radmin]
```

Judging by the different size of the hash it is clear that a different algorithm than the first two is used, *hashid* suggests SHA-1 among other hash types

```
┌──(kali㉿kali)-[~]
└─$ hashid ec65a740f5a00cafe7c7fb6de725fe369c87f0de -j
Analyzing 'ec65a740f5a00cafe7c7fb6de725fe369c87f0de'
[+] SHA-1 [JtR Format: raw-sha1]
[+] Double SHA-1
[+] RIPEMD-160 [JtR Format: ripemd-160]
[+] Haval-160
[+] Tiger-160
[+] HAS-160
[+] LinkedIn [JtR Format: raw-sha1-linkedin]
[+] Skein-256(160)
```

Question 2

For cracking the above hashes I will use the software called John The Ripper (*john* aka JTR). JTR will do a dictionary attack for each hash which will only work if the given format is indeed the hash algorithm used to create these hashes (I am going to specify the format using *–format=[Format from hashid]*). I decided not to provide JTR with a wordlist (dictionary) like *rockyou.txt* as it can use its default one (which seems to get the job done)

First up, trying to crack the first password the MD5 format didn't return any results. That either means that the password is not in the dictionary that JTR used to attack, or that the given format does not match the actual algorithm used.

Here we can observe that when we run JTR again with MD4 as format, we get a result. With this we conclude that the first password is **"awesome"**

```
┌──(kali㉿kali)-[~]
└─$ john --format=raw-md5 hash1.txt
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-MD5 [MD5 128/128 AVX 4×3])
Warning: no OpenMP support for this hash type, consider --fork=4
Proceeding with single, rules:Single
Press 'q' or Ctrl-C to abort, almost any other key for status
Almost done: Processing the remaining buffered candidate passwords, if any.
Proceeding with wordlist:/usr/share/john/password.lst
Proceeding with incremental:ASCII
0g 0:00:00:03  3/3 0g/s 5142Kp/s 5142Kc/s 5142KC/s fkm..0850000023
Session aborted

┌──(kali㉿kali)-[~]
└─$ john --format=raw-md4 hash1.txt
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-MD4 [MD4 128/128 AVX 4×3])
Warning: no OpenMP support for this hash type, consider --fork=4
Proceeding with single, rules:Single
Press 'q' or Ctrl-C to abort, almost any other key for status
Almost done: Processing the remaining buffered candidate passwords, if any.
Proceeding with wordlist:/usr/share/john/password.lst
awesome          (?)
1g 0:00:00:00 DONE 2/3 (2023-04-02 09:46) 100.0g/s 76800p/s 76800c/s 76800C/s
 leslie..bigben
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

For the last two hashes the process was pretty much the same as I got a hit with the first try.

As it turns out, the password for the second hash is **"Princess"** and for the third hash it was **"confused"**.

```
┌──(kali㉿kali)-[~]
└─$ john --format=Raw-SHA1 hash3_rsa.txt
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-SHA1 [SHA1 128/128 AVX 4x])
Warning: no OpenMP support for this hash type, consider --fork=4
Proceeding with single, rules:Single
Press 'q' or Ctrl-C to abort, almost any other key for status
Almost done: Processing the remaining buffered candidate passwords, if any.
Proceeding with wordlist:/usr/share/john/password.lst
Princess         (?)
1g 0:00:00:00 DONE 2/3 (2023-04-01 10:58) 100.0g/s 116800p/s 116800c/s 116800
C/s KILLER..Princess
Use the "--show --format=Raw-SHA1" options to display all of the cracked pass
words reliably
Session completed.

┌──(kali㉿kali)-[~]
└─$ john --format=Raw-MD5 hash2_md5.txt
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-MD5 [MD5 128/128 AVX 4×3])
Warning: no OpenMP support for this hash type, consider --fork=4
Proceeding with single, rules:Single
Press 'q' or Ctrl-C to abort, almost any other key for status
Almost done: Processing the remaining buffered candidate passwords, if any.
Proceeding with wordlist:/usr/share/john/password.lst
confused         (?)
1g 0:00:00:00 DONE 2/3 (2023-04-01 10:59) 100.0g/s 134400p/s 134400c/s 134400
C/s Alexis..december
Use the "--show --format=Raw-MD5" options to display all of the cracked passw
ords reliably
Session completed.
```

We can verify our findings by running a tool like crackstation.net which has a pre-computed hash lookup table for commonly used passwords

Free Password Hash Cracker

Enter up to 20 non-salted hashes, one per line:

```
491fe2ec1b129dc19210e65931f4c23f
1a7f2a5ad77128b2f81feddac78df213
ec65a740f5a00cafe7c7fb6de725fe369c87f0de
```

I'm not a robot — reCAPTCHA — Privacy - Terms

Crack Hashes

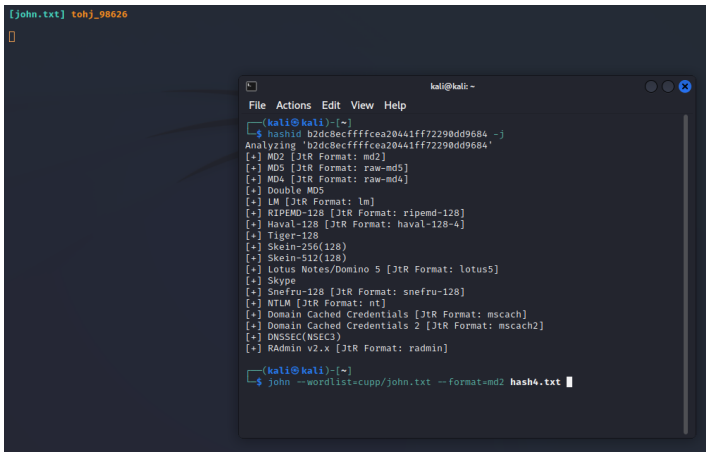
Supports: LM, NTLM, md2, md4, md5, md5(md5_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1(sha1_bin)), QubesV3.1BackupDefaults

Hash	Type	Result
491fe2ec1b129dc19210e65931f4c23f	md4	awesome
1a7f2a5ad77128b2f81feddac78df213	md5	confused
ec65a740f5a00cafe7c7fb6de725fe369c87f0de	sha1	Princess

Color Codes: Green: Exact match, Yellow: Partial match, Red: Not found.

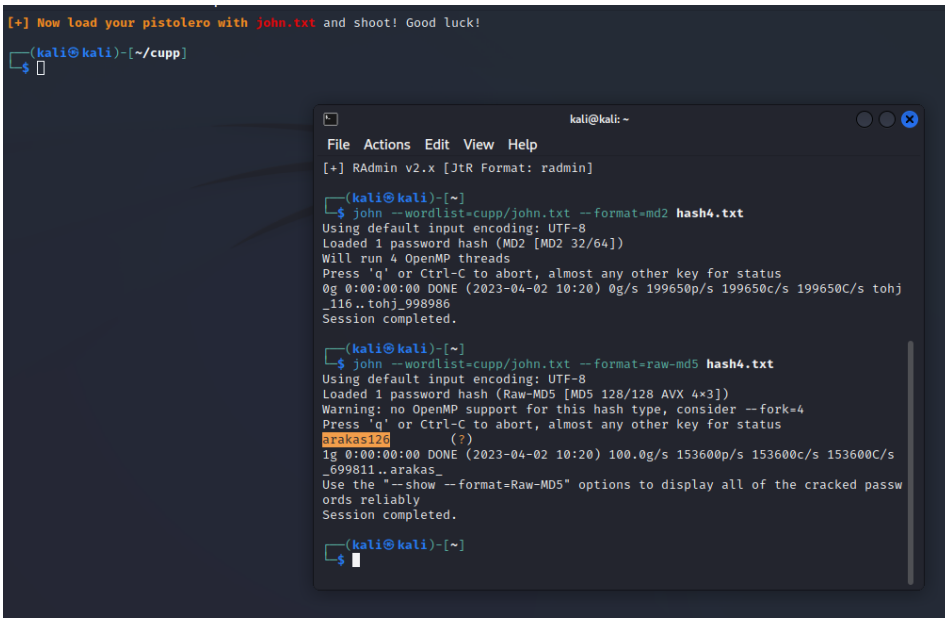
Question 3

For this question I had to get (clone and run) cupp as the question suggests. After running it, a very easy to use user interface appeared in which I entered the information given about our target (Someone could add more information about a target for a bigger wordlist).



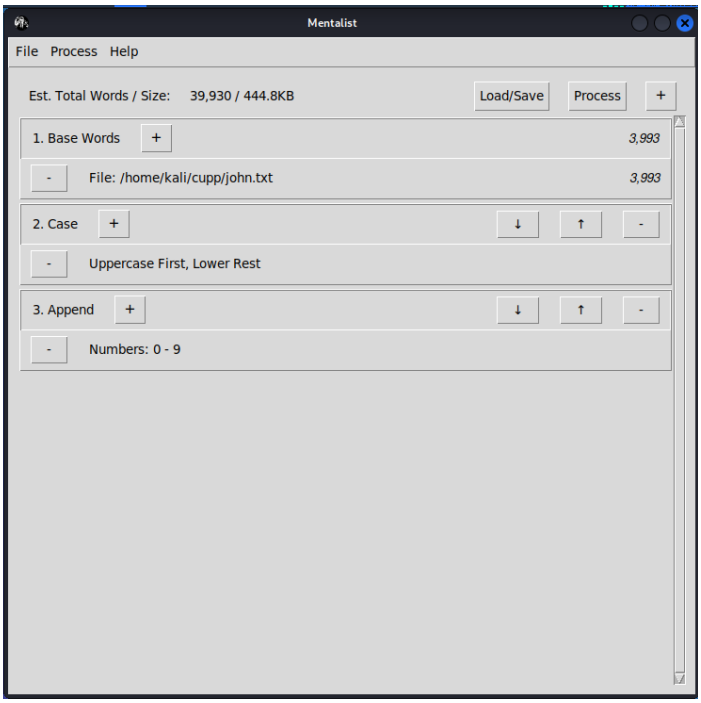
While I was waiting for cupp to finish, I run *hashid* for the given hash and found out it is most probably a Message Digest hash.

After cupp is finished, we can use the generated wordlist to run a dictionary attack (using JTR) for the given hash. The result that came through was **“arakas126”**



Question 4

For this question, I used Mentalist as the question suggested. I had to clone the repo and install it as the wiki instructed. After I launched *mentalist*, all I had to do was to add my basic wordlist from the last question (*john.txt*). I also had to add two processes, one to use uppercase for the first letter and lowercase for the rest of them, and one to append a number from 0 to 9 in the end of each word.



After the new wordlist was created, all I had to do was to run *hashid* to get the hash type of the given hash and when I figured out it was most probably a MD5 algorithm, I ran JTR for the hash and got the password **"Arakas198110"**.

```
┌──(kali㉿kali)-[~/mentalist]
└─$ echo "0f0859c50d27cb7bfd41854245df2b95" > hash5.txt

┌──(kali㉿kali)-[~/mentalist]
└─$ hashid 0f0859c50d27cb7bfd41854245df2b95 -j
Analyzing '0f0859c50d27cb7bfd41854245df2b95'
[+] MD2 [JtR Format: md2]
[+] MD5 [JtR Format: raw-md5]
[+] MD4 [JtR Format: raw-md4]
[+] Double MD5
[+] LM [JtR Format: lm]
[+] RIPEMD-128 [JtR Format: ripemd-128]
[+] Haval-128 [JtR Format: haval-128-4]
[+] Tiger-128
[+] Skein-256(128)
[+] Skein-512(128)
[+] Lotus Notes/Domino 5 [JtR Format: lotus5]
[+] Skype
[+] Snefru-128 [JtR Format: snefru-128]
[+] NTLM [JtR Format: nt]
[+] Domain Cached Credentials [JtR Format: mscach]
[+] Domain Cached Credentials 2 [JtR Format: mscach2]
[+] DNSSEC(NSEC3)
[+] RAdmin v2.x [JtR Format: radmin]

┌──(kali㉿kali)-[~/mentalist]
└─$ cd ..

┌──(kali㉿kali)-[~]
└─$ john --wordlist=mentalist/out.txt --format=raw-md5 hash5.txt
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-MD5 [MD5 128/128 AVX 4×3])
Warning: no OpenMP support for this hash type, consider --fork=4
Press 'q' or Ctrl-C to abort, almost any other key for status
Arakas198110     (?)
1g 0:00:00:00 DONE (2023-04-02 11:02) 100.0g/s 172800p/s 172800c/s 172800C/s Arakas116266..Arakas19907
Use the "--show --format=Raw-MD5" options to display all of the cracked passwords reliably
Session completed.
```

Question 5

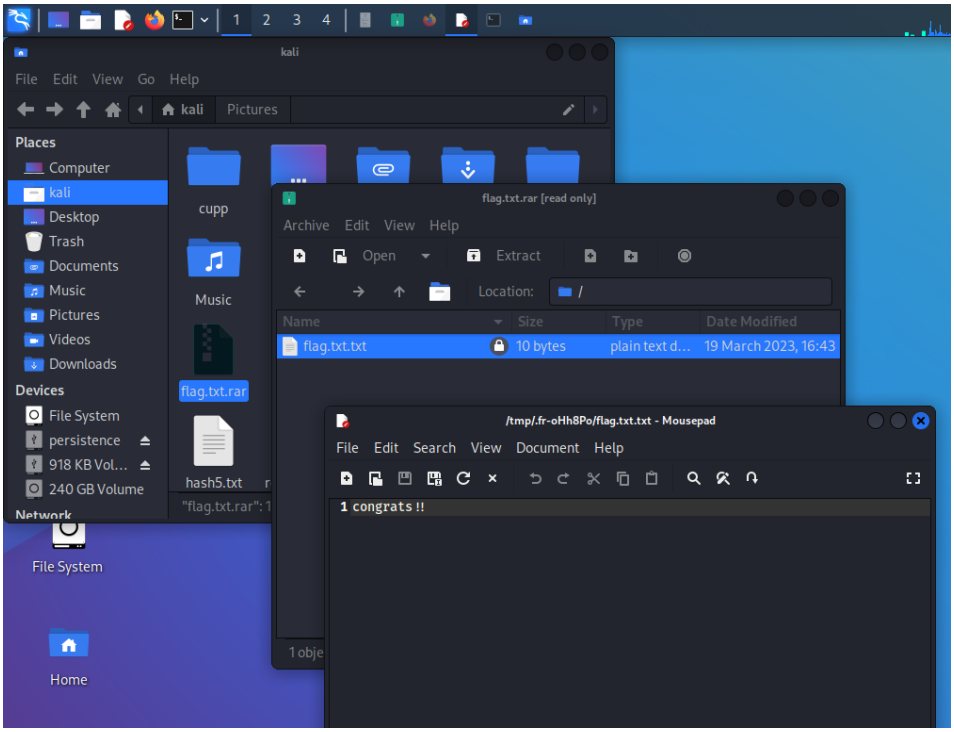
In order to find the password of the .rar file I used *rar2john* in order to produce the hash of the password. After that, all I had to do was to run JTR with the produced hash in order to get the rar password. As it turns out the password was **“secret”**.

```
┌──(kali㉿kali)-[~]
└─$ rar2john flag.txt.rar
flag.txt.rar:$rar5$16$c99736510d9cd79f758f243987021492$15$83b4de6e10e7f2099f0
b293550228732$8$a3965c1ec5543a72

┌──(kali㉿kali)-[~]
└─$ rar2john flag.txt.rar > ziphash.txt

┌──(kali㉿kali)-[~]
└─$ john ziphash.txt
Using default input encoding: UTF-8
Loaded 1 password hash (RAR5 [PBKDF2-SHA256 128/128 AVX 4x])
Cost 1 (iteration count) is 32768 for all loaded hashes
Will run 4 OpenMP threads
Proceeding with single, rules:Single
Press 'q' or Ctrl-C to abort, almost any other key for status
Almost done: Processing the remaining buffered candidate passwords, if any.
Proceeding with wordlist:/usr/share/john/password.lst
secret           (flag.txt.rar)
1g 0:00:02:01 DONE 2/3 (2023-04-02 11:15) 0.008231g/s 185.4p/s 185.4c/s 185.4
C/s 123456..green
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

To verify my findings I extracted the file flag.txt.txt from the .rar file using the password **“secret”** and the txt file contained the flag **“congrats!!”**



Question 6

JTR has the option *–mask* whith which you can specify a mask for the words that it is going to include in the dictionary attack. The mask I am going to use is *–mask=?d?d?d?d?d?d?d?d* which is ordering JTR to search only for 8-digit words. For demonstrating this functonality of JTR I am going to hash the password **“12345678”** using an online MD5 generator. The hash I got and used is **“25d55ad283aa400af464c76d713c07ad”**. Passing this to JTR as well as the mask and the format, results in getting the password we initially hashed using MD5.

```
┌──(kali㉿kali)-[~]
└─$ echo "25d55ad283aa400af464c76d713c07ad" > digit_hash.txt

┌──(kali㉿kali)-[~]
└─$ john --mask=?d?d?d?d?d?d?d?d --format=Raw-MD5 digit_hash.txt
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-MD5 [MD5 128/128 AVX 4×3])
Warning: no OpenMP support for this hash type, consider --fork=4
Press 'q' or Ctrl-C to abort, almost any other key for status
12345678         (?)
1g 0:00:00:02 DONE (2023-04-02 14:42) 0.3546g/s 21229Kp/s 21229Kc/s 21229KC/s
Use the "--show --format=Raw-MD5" options to display all of the cracked passw
Session completed.
```