Abertay University

Hash Cracking

Paul Oates – @abertay.ac.uk

Intro to Security – CMP110

BSc Ethical Hacking Year 1

2020/21

Note that Information contained in this document is for educational purposes.

.

Abstract

Hash cracking is a common method in which Black Hat Hackers can steal a targets data. The aim of my report is to build my own tools as well as using tools Security Researchers have built to gain a fuller understanding of the subject. Hash cracking attacks can be carried out in two main ways; wordlist and brute force. Wordlist is where an attacker can use previous password leaks to crack passwords. The other method, brute force, is where an attacker tries every possible variation of a password. For the purpose of my investigation, I carried out brute force and wordlist attacks in Python, C++ and Hashcat. My findings showed Hashcat to be the fastest at cracking hashes. My studies also identified that due to the rapid advancement of technology in this area we have seen devices such as GPU's being able to crack weak passwords in less than a second. Technology such as cloud computing have only made this even easier, as large expensive GPU's can be rented for a couple of hours for a small fee. I now believe hash cracking is a security risk that current and future software developers and system admins will need to consider when developing and maintaining software. However, I also firmly believe it is the responsibility of the end user to implement good password security practices wherever possible.

.

Contents

1 Introduction 1
1.1 Background 1
1.2 Aim 2
2 Procedure 3
2.1 Overview of Procedure 3
2.2 Procedure part 1 6
3 Results 10
3.1 Results for part 1 10
4 Discussion 12
4.1 General Discussion 12
4.2 Countermeasures 13
4.3 Conclusions 13
4.4 Future Work 13
References 14
Appendices 15
Appendix A 15

.

1 INTRODUCTION

1.1 BACKGROUND

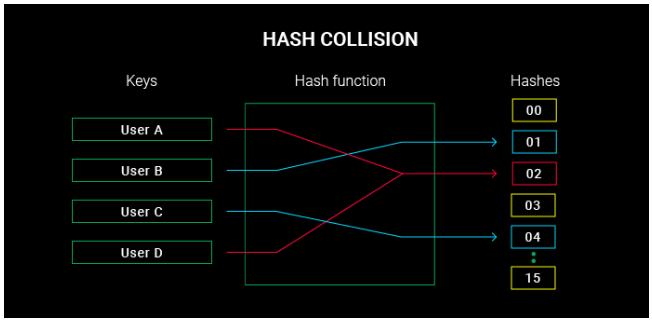
Hashing algorithms are used to obscure data to keep the information hidden and secure. Hashing algorithms have been used since the first computer databases. They involve complex mathematical equations which 'hash' the data making the data difficult to decrypt.

As technology advances hashing algorithms such as MD2, MD4 and MD5 have become more vulnerable to attacks these have been replaced by more secure standards such as SHA256, SHA512. However large portions of the internet still use these outdated standards. Hashing is now used in areas of computing such as computer forensics, to ensure a file has not been tampered with, and Password's to protect the user.

The advancement of technology becomes a problem in itself as passwords are very easy to crack. For example due to the power of modern GPU's such as a Nvidia RTX3090 which can calculate 65079.1 MH/s in MD5 hashes. When comparing to a GPU of two years ago a Nvidia RTX2080TI which can calculate 53975.3 MH/s that's greater than a 20% increase in just two years. Cloud computing software such as Amazon Web Services and Microsoft Azure allow you rent expensive GPU computers for a few hours these can be used to crack hashes.

Another issue with technology is that while password hashing is a key step to protect users, it is not infallible as it hashes in a consistent manner. This means that it is predictable and can be cracked.

Another identified problem is collisions, this is where two completely different phases equal the same hash, due to their only being a limited amount of variation of hashes.



1.2 AIM

- The aim of my project is to investigate wordlist and brute force hash cracking techniques and use these to build my own tools for hash cracking.

- I will then go on to investigate ways these methods can be accelerated on devices such as GPU's and what sort of performance benefit this brings.

- I will then compare the theoretical performance of running hash cracking on AWS and Azure vs. the GPU

2 PROCEDURE

2.1 OVERVIEW OF PROCEDURE

Overview

- I ran a wordlist and brute force attack using Python, C++ and Hashcat
- I also attempted a GPU wordlist attack in CUDA
- The Program output is used to create a performance report
- I calculated the hash rate by dividing the number of hashes by time taken.
- Using this method, I graphed the performance of each attack.

Hardware/Software

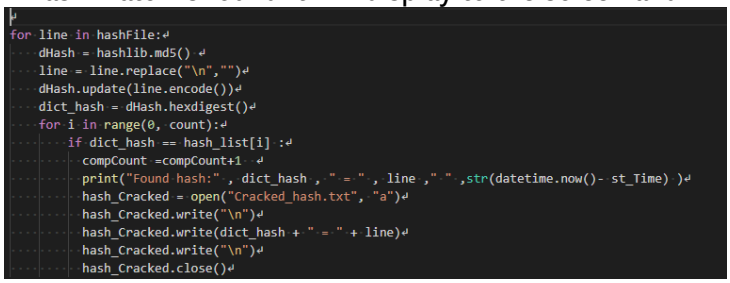
- My machine's Operating System is Windows 10 64bit (10.0.19042 Build 19042)
- Running Intel i7-8565U and a Nvidia MX150
- I installed Nvidia's CUDA driver to run programs on the GPU
- Running Visual Studio 2019 for C++ and Visual Studio Code for python 3.9
- I have installed Hashcat 6.6.1 from GitHub

Screenshots

Fig2.1 shows the main bruteforce function for Python, this receives the value to be hashed encodes it, hashes it and compares it to the user hash list. If hash match is found it will display to the screen and write it to a file.

```
def bruteforce(passVal):
    # print(passVal)
    compCount = 0
    rpword =(passVal.encode())
    brute_Hash = hashlib.md5()
    brute_Hash.update(rpword)
    brute_Hash = brute_Hash.hexdigest()
    for i in range(0, count):
        if brute_Hash == hash_list[i]:
            print("Found hash:" , brute_Hash , " = " , passVal ," " ,str(datetime.now()- st_Time) )
            compCount = compCount + 1
            hash_Cracked = open("cracked_Passwords.txt", "a")
            hash_Cracked.write("\n")
            hash_Cracked.write(hash_list[i] + " = " + passVal )
            hash_Cracked.close()
            if compCount == count:
                break
                #END PROGRAM
    return()
```

Fig2.2 shows main wordlist function for python. This reads in each password in turn from the wordlist file, removes the new line character, encodes as UTF-8 and calculates the hash. It then compares the calculated hash with the user hash list stored in an array. if hash match is found it will display to the screen and write it to a file



```
for line in hashFile:
    dHash = hashlib.md5()
    line = line.replace("\n","")
    dHash.update(line.encode())
    dict_hash = dHash.hexdigest()
    for i in range(0, count):
        if dict_hash == hash_list[i] :
            compCount =compCount+1
            print("Found hash:" , dict_hash , " = " , line ," " ,str(datetime.now()- st_Time) )
            hash_Cracked = open("Cracked_hash.txt", "a")
            hash_Cracked.write("\n")
            hash_Cracked.write(dict_hash + " = " + line)
            hash_Cracked.write("\n")
            hash_Cracked.close()
```

Fig 2.3 main Wordlist function for C++, this reads in all passwords to a memory buffer, removes the new line character and calculates the hash. It then compares the calculated hash with the user hash list provided. if hash match is found it will display the match to the screen and write it to a file

```
const clock_t begin_time = clock();
std::string str;
std::ifstream is("hashval.txt", std::ifstream::binary);
if (is) {
    // get length of file:
    is.seekg(0, is.end);
    int length = is.tellg();
    is.seekg(0, is.beg);
    char* fbuffer = new char[length];

    std::cout << "Reading " << length << " characters... \n";
    // read data as a block:
    is.read(fbuffer, length);
    is.seekg(0, is.beg);


    for (int i = 0; i < length; i = i + 6) {
        char tmpVal[5] = { char(fbuffer[i]),char(fbuffer[i + 1]),char(fbuffer[i + 2]),char(fbuffer[i + 3]), '\0' };
        string stH_word = MD5(tmpVal).toStr();
        for (int ii = 0; ii < hcount; ii++) {
            if (stH_word == hashlist[ii]) {
                cout << "Found Hash " << stH_word << "=" << tmpVal << " Time = " << float((clock() - begin_time) / CLOCKS_PER_SEC) << endl;
                fstream Cracked_hash;
                Cracked_hash.open("cracked_hashes.txt", ios::app);
                Cracked_hash << "hash matched:" << stH_word << "=" << tmpVal << " Time = " << float((clock() - begin_time) / CLOCKS_PER_SEC) << endl;
            }
        }
    }
    is.close();
    delete[] fbuffer;
}
```

Fig 2.4 main Bruteforce function in C++, this generates all possible four character passwords. Each password is hashed in turn and compared to the user hash list provided. if hash match is found it will display the match to the screen and write it to a file

```
// Generate password.
int max_val = 94;
int i;
char pword[94];

i = 33;
for (int n = 0; n < max_val; n++) {
    pword[n] = char(i);
    i++;
}

const clock_t begin_time = clock();
for (int d = 0; d < max_val; d++) {
    for (int c = 0; c < max_val; c++) {
        for (int b = 0; b < max_val; b++) {
            for (int a = 0; a < max_val; a++) {
                char pVal[5] = { char(pword[d]),char(pword[c]),char(pword[b]),char(pword[a]), '\0' };
                //cout << pVal << endl;
                string stH_word = MD5(pVal).toStr();
                for (int ii = 0; ii < hcount; ii++) {
                    if (stH_word == hashlist[ii]) {
                        cout << "Found Hash " << stH_word << "=" << pVal << " Time = " << float((clock() - begin_time) / CLOCKS_PER_SEC) << endl;
                        fstream Cracked_hash;
                        Cracked_hash.open("cracked_hashes.txt", ios::app);
                        Cracked_hash << "hash matched:" << stH_word << "=" << pVal << " Time = " << float((clock() - begin_time) / CLOCKS_PER_SEC) << endl;
                    }
                }
            }
        }
    }
}
```

Fig 2.5 Hashcat 6.6.1 program was obtained from GitHub. The command used (see below) configures Hashcat for MD5 wordlist hash cracking. The program is provided with the user hash list “hashed.hash” and a wordlist dictionary “hashval.dict”. if a hash match is found it will display the match to the screen and write it to a file

Command

.\hashcat.exe -a 0 -m 0 -O .\hashed.hash .\hashval.dict

Output

Session.....: hashcat
Status.....: Exhausted
Hash.Name.....: MD5
Hash.Target.....: .\hashed.hash
Time.Started.....: Mon May 10 11:00:23 2021 (9 secs)
Time.Estimated...: Mon May 10 11:00:32 2021 (0 secs)
Guess.Base.....: File (.\hashval.dict)
Guess.Queue.....: 1/1 (100.00%)
Speed.#1.....: 8241.7 kH/s (1.03ms) @ Accel:64 Loops:1 Thr:1024 Vec:1
Recovered.....: 18/19 (94.74%) Digests
Progress.....: 78074896/78074896 (100.00%)
Rejected.....: 0/78074896 (0.00%)
Restore.Point....: 78074896/78074896 (100.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidates.#1....: ~|V' -> ~~~~
Hardware.Mon.#1..: Temp: 62c Util: 69% Core:1632MHz Mem:3003MHz Bus:4

2.2 PROCEDURE PART 1

Python wordlist

- The python wordlist program runs from Visual Studio Code.
- The program name is “hash_Cracking_wordlist.py”
- The program expects the following files to be stored locally to the Python file:-
    - Dictionary File containing all possible four letter passwords (78074896)
        - “hashval.txt”
    - User Hash List containing the list of user provided hashes
        - “hashed.txt”
- As the program is running it will display status of hashes found in the terminal window
- The program only supports MD5 hash.
- The user can provide their own hashes in the “hashed.txt” file. They must be in MD5 format with a single hash per line.
- I have provided a list of all possible four-letter passwords. The user can add additional passwords to the file “hashval.txt”.

Python Brute Force

- The python wordlist program runs from Visual Studio Code.
- The program name is “hash_Cracking_bruteforce_4letters.py”
- The program expects the following file to be stored locally to the Python file :-
    - User Hash List containing the list of user provided hashes
        - “hashed.txt”
- As the program is running it will display the status of hashes found in the terminal window
- The program only supports MD5 hash.
- The user can provide their own hashes in the “hashed.txt” file. They must be in MD5 format with a single hash per line.
- The program creates and hashes all possible four letter passwords. The program would require to be updated to support any other password length.

C++ Wordlist

- The C++ wordlist program runs on Windows 10 (x64).
- The program name is “C++_wordlist.exe”
- The program expects the following file to be stored in the same directory as the executable:-
    - Dictionary File containing all possible four-letter passwords (78074896)
        - “hashval.txt”
    - User Hash List containing the list of user provided hashes
        - “hashed.txt”
- As the program is running it will display status of hashes found in the terminal window
- The program only supports MD5 hash.
- The user can provide their own hashes in the “hashed.txt” file. They must be in MD5 format with a single hash per line.
- I have provided a list of all possible four-letter passwords. The user can add additional passwords to the file “hashval.txt”.

C++ Bruteforce

- The C++ wordlist program runs on Windows 10 (x64).
- The program name is “C++bruteforce.exe”
- The program expects the following file to be stored in the same directory as the executable:-
    - User Hash List containing the list of user provided hashes
        - “hashed.txt”
- As the program is running it will display the status of hashes found in the terminal window.
- The program only supports MD5 hash.
- The user can provide their own hashes in the “hashed.txt” file. They must be in MD5 format with a single hash per line.
- The program creates and hashes all possible four letter passwords. The program would require to be updated to support any other password length

Hashcat

- I ran Hashcat to show the performance benefits of GPU vs CPU with both a wordlist and brute force attack
- I installed Hashcat 6.6.1 from GitHub at the following link [https://github.com/hashcat/Hashcat][10/05/2021]

Wordlist Hashcat

- Open the ".Potfile" and remove any existing hashes
- The program expects the following file to be stored in the same directory as the executable:-
    - Dictionary File containing all possible four-letter passwords (78074896)
        - "hashval.dict"
    - User Hash List containing the list of user provided hashes
        - "hashed.hash"
- The command executed is.\hashcat.exe -a 0 -m 0 -O .\hashed.hash .\hashval.dict
  Where;-
    - -a is attack mode
    - 0 is wordlist
    - -m is hash type
    - 0 is md5
    - – O is optimised Kernel
    - .\hashed.hash is the user hash file
    - .\hashval.dict is the wordlist dictionary
- The program supports various hash algorithms, but the command needs to be altered.
- The user can provide their own hashes in the "hashed.hash" file. They must be in MD5 format with a single hash per line.
- I have provided a list of all possible four-letter passwords. The user can add additional passwords to the file "hashval.dict".

Bruteforce Hashcat

- Open the ".Potfile" and remove any existing hashes
- The program expects the following file to be stored in the same directory as the executable:-
    - User Hash List containing the list of user provided hashes
        - "hashed.hash"
- **The command executed is**.\hashcat.exe -a 3 -m 0 -O .\hashed.hash ?s?s?s?s **for special charcters**

  Where;-
    - -a is attack mode
    - 3 is bruteforce
    - -m is hash type
    - 0 is md5
    - – O is optimised Kernel
    - .\hashed.hash is the user hash file
    - ?s indicated for the special characters this is repeated 4 times for a password of length four
- **The command executed is**.\hashcat.exe -a 3 -m 0 -O .\hashed.hash
- **Where;-**
    - -a is attack mode
    - 3 is bruteforce
    - -m is hash type
    - 0 is md5
    - – O is optimised Kernel
    - .\hashed.hash is the user hash file
- The program supports various hash algorithms, but the command needs to be altered
- The user can provide their own hashes in the "hashed.hash" file. They must be in MD5 format with a single hash per line.
- The program creates and hashes all possible passwords. The program will only stop if all passwords are matched or program is stopped by the user.

Calculation

- The performance of each method is now calculated.
- This is done by subtracting program finished time from program start time
- This is the number of hashes divided by the time taken then divided by a million to calculate Million Hashes per second (MH/s).

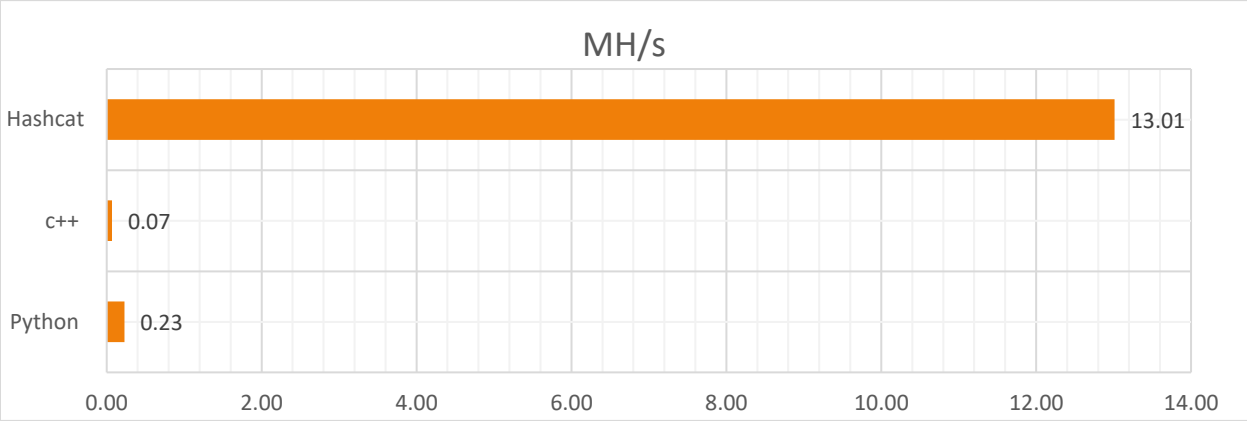
3 RESULTS

3.1 RESULTS FOR PART 1

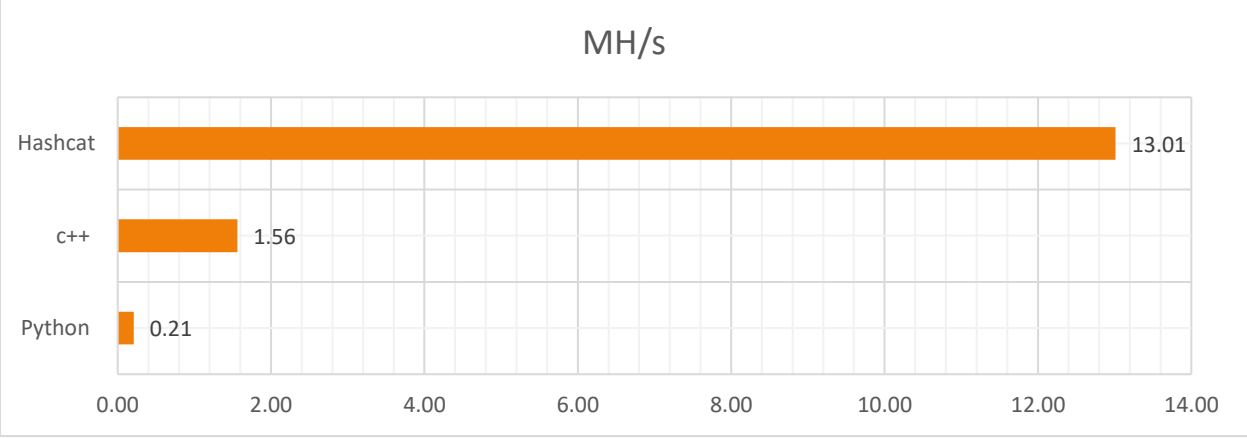
Hash cracking results table

Method	Number of unknown Hashes	Number of Words	Time Taken seconds	Start Time	End	MH/s
Wordlist Python 4 letters	18	78074896	364	11:58:24	12:04:28	0.21
Bruteforce Python 4 letters	18	78074896	339	13:31:46	13:37:25	0.23
c++ bruteforce	18	78074896	1015	00:00:00	00:16:55	0.07692108
c++ wordlist	18	78074896	1000	00:00:00	00:16:40	1.41
Hashcat wordlist	18	78074896	27	11:00:07	11:00:34	2.89
Hashcat bruteforce	18	78074896	6	11:23:40	11:23:46	13.01
RTX3080						65079.00
AWS Server						425000000.00
Azure Server						4852.80

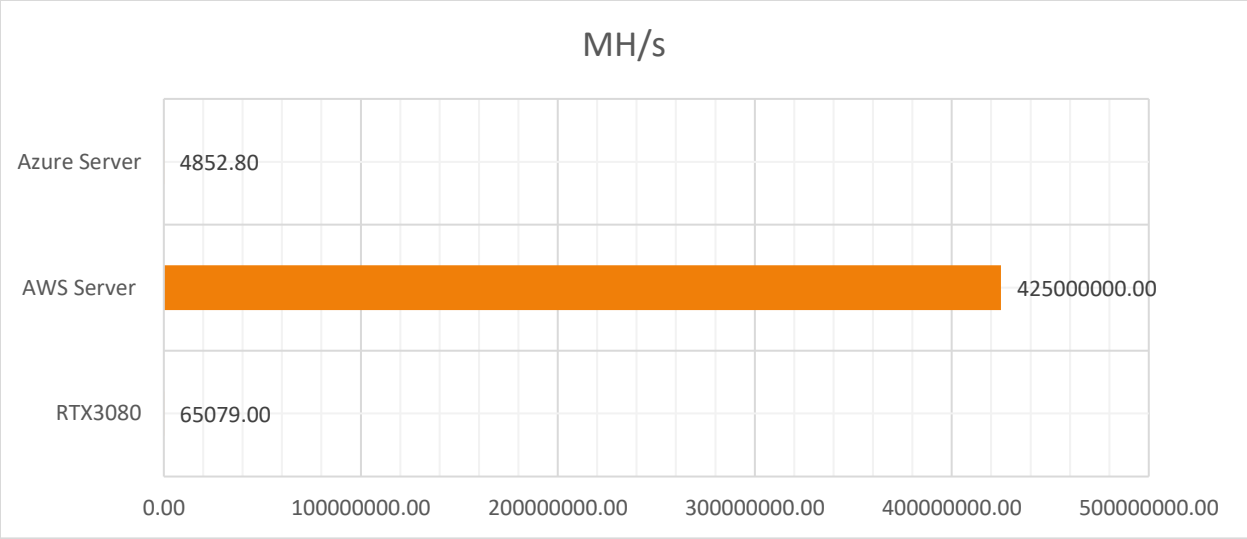
Bruteforce



Wordlist



Other(high end GPU's/severs)


MH/s
Azure Server
4852.80
AWS Server
425000000.00
RTX3080
65079.00
0.00
100000000.00
200000000.00
300000000.00
400000000.00
500000000.00

4 DISCUSSION

4.1 GENERAL DISCUSSION

- I have met the aims of my investigation. I have built my own wordlist and brute force tools using Python and C++. I looked at GPU acceleration in Hashcat and the performance benefit of running AWS and Azure and the cost implications

- As we can see above Hashcat with over 7,000 commits is an efficient tool for hash cracking weak passwords especially with its wide support for AMD, Nvidia and Intel CPU/GPU's. However Strong passwords are still the first line of defence

- The cost of running an AWS server such as the p3.16xlarge is £17.60 per hour this means that the hashed information needs to be important enough to spend this sort of money. However, the AWS server with a hash rate of 425GH/s would be able to crack longer passwords in a much shorter period

- My results demonstrate computers are getting faster and faster at cracking hashes especially in GPU's and cloud computing servers such as AWS.

- With more time I believe I may be able to build hash cracking tools on the GPU using CUDA in C++

- From my reading, wordlists are more effective than brute force at cracking longer passwords

- My knowledge of C++ at this point was not adequate to build a fast hash cracking tool in this language

- Hardware and software available to me at this time may not have been optimal for this type of work

4.2 COUNTERMEASURES

- My results show that with time all hashes can be broken. However, passwords that are complex are not of any interest to a Black Hat Hacker as these passwords will take too long to crack.

- This means that complex passwords such as long anagrams with odd capitalisation and special characters mixed in make good passwords.

- Another counter measure is two step verification this means that even if a password is leaked the hacker would also have to have access to a secondary device to have access to an account

- Modern password managers now alert users to data breaches and websites such as https://haveibeenpwned.com/ can be set up to send emails to alert the user of vulnerable passwords

4.3 CONCLUSIONS

- In conclusion, a determined hacker trying to gain access to an account has many tools available to use against a target.

- A weak password can be cracked more easily than a strong password

- Advancements in hardware is decreasing the time taken to crack passwords

- Openly available wordlists online make end users even more vulnerable to attacks

4.4 FUTURE WORK

- In the future I would like to run some sort of hash cracking software such as Hashcat on an p3.16xlarge to see the speed that people online report it has.
- I would also benefit from gaining further knowledge and understanding of C++ to allow me to optimise my programs
- I would also benefit from gaining further understanding of CUDA to build a GPU hash cracking tool

REFERENCES

[Cybernews][online][11/05/2021]

https://cybernews.com/wp-content/uploads/2020/09/Hasing-VS-encryption-11.png

[Techradar][online][11/05/2021]

https://www.techradar.com/news/the-nvidia-geforce-rtx-3090-is-very-good-at-cracking-passwords-and-thats-bad-news

[onlinehashcrack][online][11/05/2021]

https://www.onlinehashcrack.com/tools-benchmark-hashcat-nvidia-rtx-2080-ti.php

[GitHub][online repository][11/5/2021]

https://github.com/JieweiWei/md5

[GitHub][online repository][11/5/2021]

https://github.com/hashcat/hashcat

[medium][online][11/5/2021]

https://medium.com/@iraklis/running-hashcat-v4-0-0-in-amazons-aws-new-p3-16xlarge-instance-e8fab4541e9b

APPENDICES

APPENDIX A

- I have included the following with my upload :-
  - Executables for my C++ hash cracking tools
  - Python files for my hash cracking tools
  - Wordlist and brute force source code
  - Text files for dictionary wordlist and hashed values