K.J. Somaiya College of Engineering

2020 - 2021

A Report on the Security Tool Analysis and implementation of-

HASHCAT

Prepared by:

1711008 - Sagar Dama

1711010 - Heeral Dedhia

1711011 - Arnab Dey

1711017 - Harsh Gokhru

(Batch – A1)

Branch - LY Computer

A report submitted in partial fulfillment of the syllabus of ‘Cryptography and System Security’ laid down by K. J. Somaiya College of Engineering, Vidyavihar, Mumbai - 400077 (An Autonomous College Affiliated to University of Mumbai).

K.J. Somaiya College of Engineering

2020 - 2021

TABLE OF CONTENTS

Github Repo - https://github.com/kjsomaiya/css-assignment-1-coding-hashcat/

Abstract

In cryptanalysis and computer security, password cracking is the process of recovering passwords from data that has been stored in or transmitted by a computer system. A common approach (brute-force attack) is to repeatedly try guesses for the password and to check them against an available cryptographic hash of the password.

The purpose of password cracking might be to help a user recover a forgotten password (installing an entirely new password is less of a security risk, but it involves System Administration privileges), to gain unauthorized access to a system, or to act as a preventive measure whereby system administrators check for easily crackable passwords.

Passwords that are difficult to remember will reduce the security of a system because (a) users might need to write down or electronically store the password using an insecure method, (b) users will need frequent password resets, and (c) users are more likely to re-use the same password.

In "The Memorability and Security of Passwords", Jeff Yan et al. examines the effect of advice given to users about a good choice of password. They found that passwords based on thinking of a phrase and taking the first letter of each word are just as memorable as naively selected passwords, and just as hard to crack as randomly generated passwords.

https://hashcat.net/hashcat/

Introduction

Hashcat is a password recovery tool. It had a proprietary code base until 2015 but was then released as open source software. Versions are available for Linux, OS X, and Windows. Examples of hashcat-supported hashing algorithms are LM hashes, MD4, MD5, SHA-family and Unix Crypt formats as well as algorithms used in MySQL and Cisco PIX.
Hashcat has been publicly noticed because of its optimizations; partly based on flaws in other software discovered by the creator of hashcat. Previously, two variants of hashcat existed:

- hashcat - CPU-based password recovery tool
- oclHashcat/cudaHashcat - GPU-accelerated tool (OpenCL or CUDA)

With the release of hashcat v3.00, the GPU and CPU tools were merged into a single tool called hashcat. The CPU-only version became hashcat-legacy.[4] Both CPU and GPU now require OpenCL. Many of the algorithms supported by hashcat-legacy (such as MD5, SHA1, and others) can be cracked in a shorter time with the GPU-based hashcat. However, not all algorithms can be accelerated by GPUs.

Weak Hash

The weak hash detection is a special feature of hashcat. The goal of this feature is to notice if there is a hash whose plaintext is empty, that means a 0-length password. Typically when you just hit enter. We call it a weak-hash check even if it should have been called a weak-password check but there are simply too many weak-passwords.

However, if your hash list contains millions of salts we have to run a kernel for each salt. If you want to check for empty passwords for that many salts you will have a very long initialization/startup time by running hashcat. To work around this problem there is a parameter called "--weak-hash-threshold". With it, you can set a maximum number of salts for which weak hashes should be checked on start. The default is set to 100, that means if you use a hash list with 101 unique salts it will not try to do a weak-hash check at all. Note we are talking about unique salts, not unique hashes. Cracking unsalted hashes results in 1 unique salt (an empty one). That means if you set it to 0 you are disabling it completely, also for the unsalted hashes.

Identifying Hash Type

Some hashes have signatures that give a strong indication of which algorithm was used, such as “1” for md5crypt. Usually, you can rely on this information; however, this method of identification is not bullet-proof! For example, consider an algorithm such as crypt(sha256(pass), “1”).

For hashes that have no signature, it is virtually impossible to distinguish which algorithm was used. A string of 32 hex characters could be LM, NTLM, MD4, MD5, double MD5, triple md5, md5(sha512(pass)), so on and so forth. There is literally an infinite number of possibilities for what the algorithm may be!

Tools that claim to be able to identify hashes simply use regular expressions to match the hash against common patterns. This method is extremely unreliable and often yields incorrect results. It is best to avoid using such tools.

A much better way to identify the hash algorithm would be to understand the origin of the hashes (e.g. operating system, COTS application, web application, etc.) and make an educated guess at what the hash algorithm might be. Or better yet, use the source.

Decryption Algorithms

The principle of hashing is not to be reversible, there is no decryption algorithm, that's why it is used for storing passwords: it is stored encrypted and not unhashable. The hash functions apply millions of non-reversible operations so that the input data can not be retrieved. Hash functions are created to not be decrypted, their algorithms are public. The only way to decrypt a hash is to know the input data.

Breaking any encryption system can be done with unlimited time and unlimited computing power, both of which do not exist. Anything less than that unlimited power and time will require chance and good investigative skills. Several methods to break encryption include dictionary attacks, brute-force attacks, and rainbow tables.

A dictionary attack tries variations of words found in dictionaries. The speed at which passwords are tried depends upon the computing power. Millions of passwords can be tried each second using a basic computer system with higher end systems able to try even more passwords per second. Considering that a password may not even be in a recognizable word format, a dictionary attack will be ineffective in some instances. Rainbow tables are tables of reversed hashes used to crack password hashes. Computer systems requiring passwords typically store the passwords as a hash value of the user's password. When a computer user enters a password, the system hashes the password and compares it to the stored hash. If the hashes match, the user is given access.

Rainbow Tables

A rainbow table is a precomputed compilation of plaintexts and matching ciphertexts (typically passwords and their matching hashes). Rainbow tables greatly speed up many types of password cracking attacks, often taking minutes to crack where other methods (such as dictionary, hybrid, and brute-force password cracking attempts) may take much longer.

Though rainbow tables act as a database, they are more complex under the hood, relying on a time/memory trade-off to represent and recover passwords and hashes. Most rainbows tables can crack most, but not all, possible hashes.

HASHCAT Software

<u>Using HASHCAT</u>

If your operating system or linux distribution does have some pre-build installation package for hashcat, you may be able to install it using those facilities. For example, you can use the following under Kali Linux:

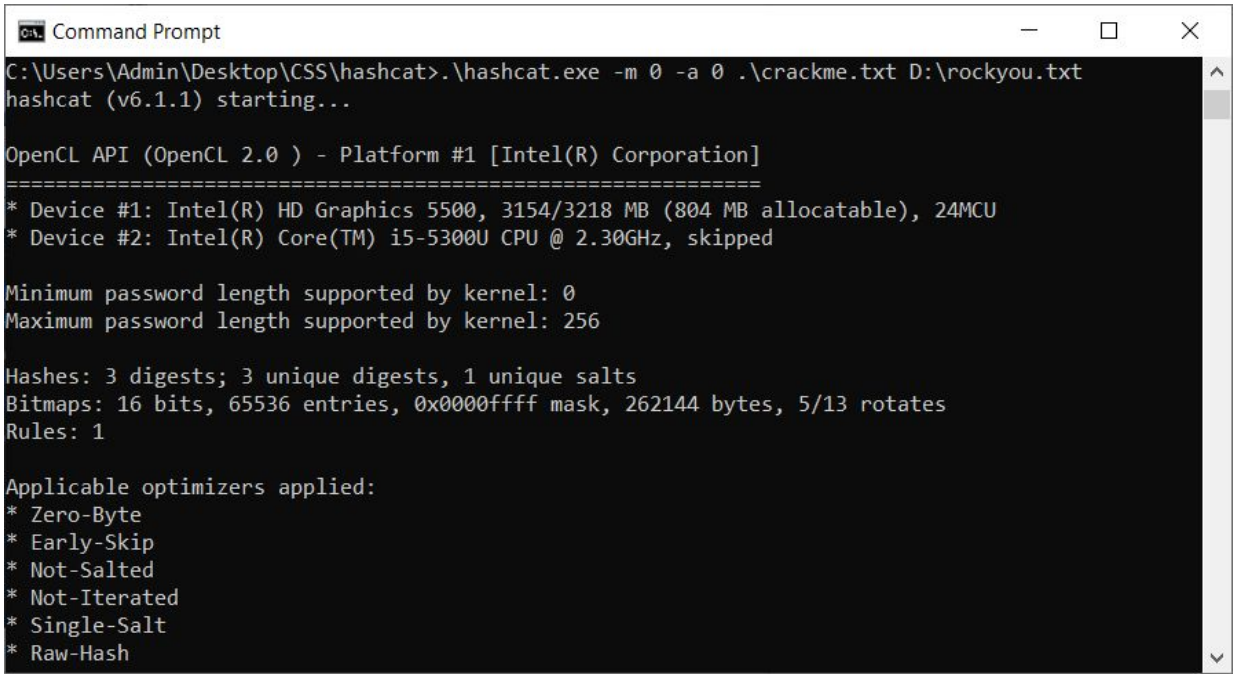
```
$ sudo apt-get update && sudo apt-get install hashcat
```

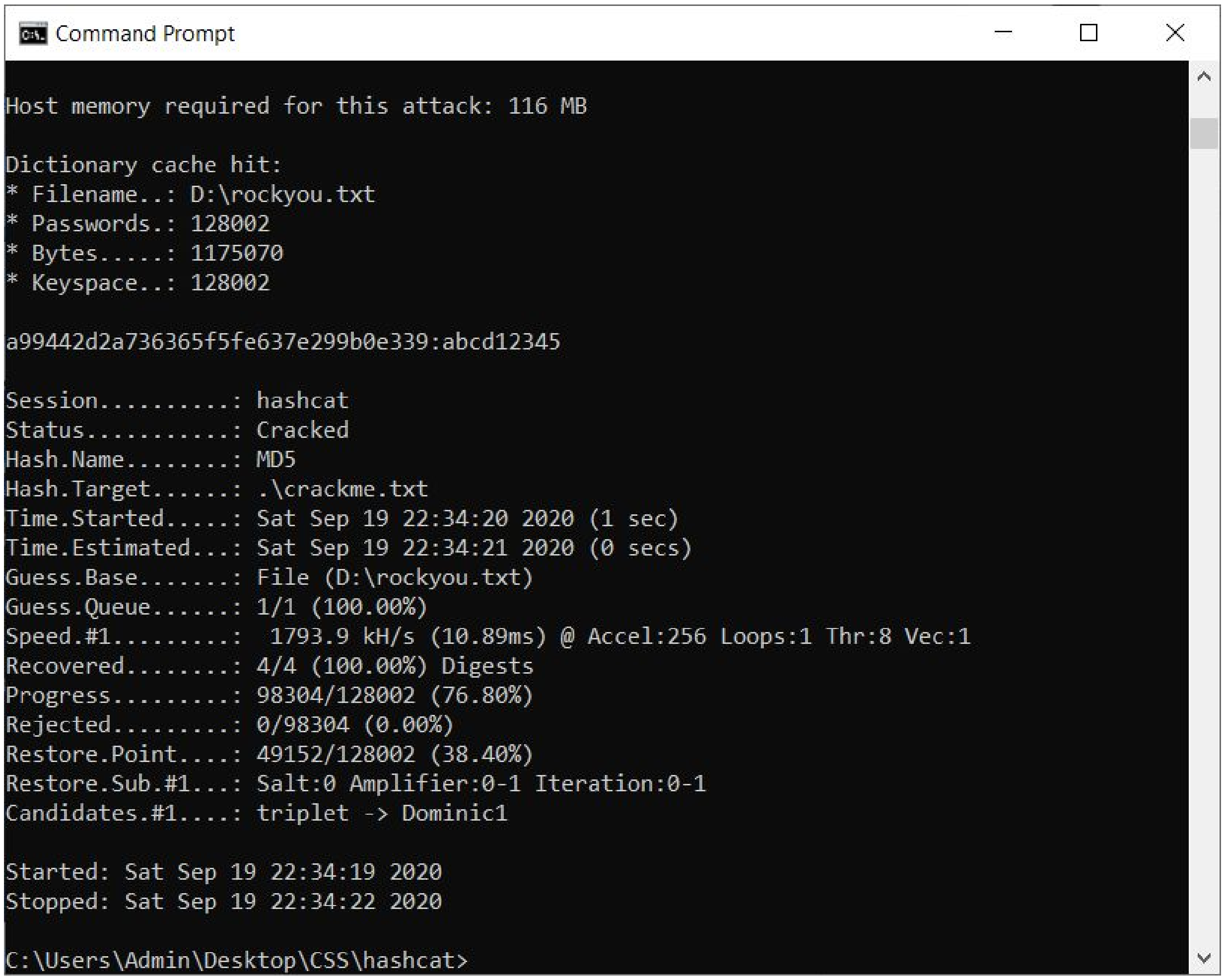
and update it with:

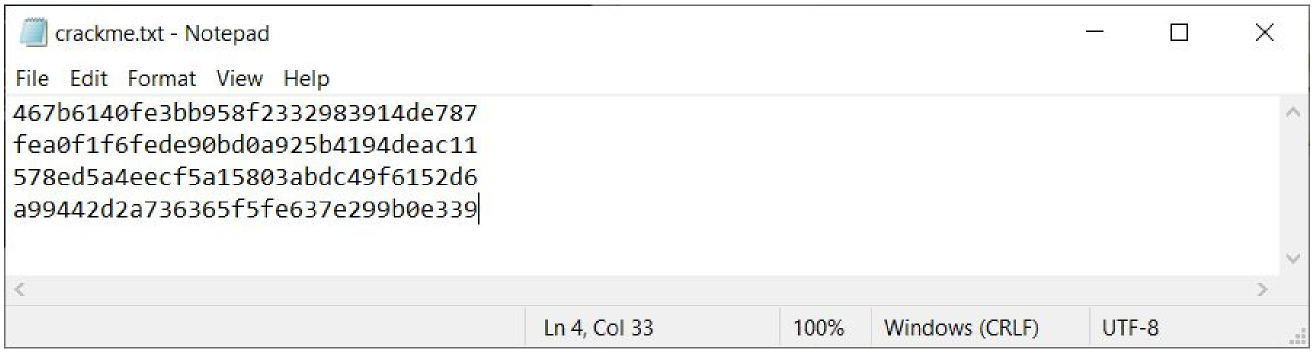
```
$ sudo apt-get update && sudo apt-get upgrade
```

Even if this is supported by some distributions, we do not directly support this here since it depends on the package maintainers to update the packages, install the correct dependencies (some packages may add wrappers, etc), and use reasonable paths.

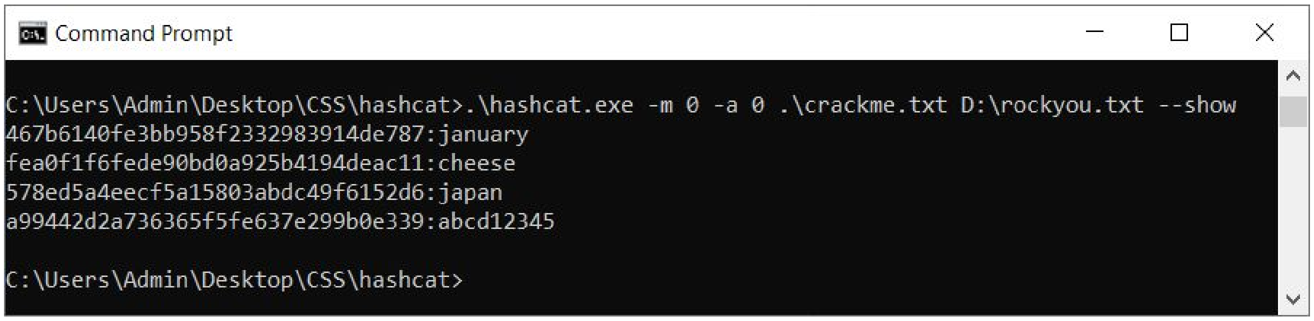
In case something isn't working with the packages you download via your package manager, we encourage you to just download the hashcat archive directly, enter the folder, and run hashcat. This is the preferred and only supported method to “install” hashcat.


```
C:\Users\Admin\Desktop\CSS\hashcat>.\hashcat.exe -m 0 -a 0 .\crackme.txt D:\rockyou.txt
hashcat (v6.1.1) starting...

OpenCL API (OpenCL 2.0 ) - Platform #1 [Intel(R) Corporation]
=============================================================
* Device #1: Intel(R) HD Graphics 5500, 3154/3218 MB (804 MB allocatable), 24MCU
* Device #2: Intel(R) Core(TM) i5-5300U CPU @ 2.30GHz, skipped

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 256

Hashes: 3 digests; 3 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Applicable optimizers applied:
* Zero-Byte
* Early-Skip
* Not-Salted
* Not-Iterated
* Single-Salt
* Raw-Hash
```


```
Host memory required for this attack: 116 MB

Dictionary cache hit:
* Filename..: D:\rockyou.txt
* Passwords.: 128002
* Bytes.....: 1175070
* Keyspace..: 128002

a99442d2a736365f5fe637e299b0e339:abcd12345

Session..........: hashcat
Status...........: Cracked
Hash.Name........: MD5
Hash.Target......: .\crackme.txt
Time.Started.....: Sat Sep 19 22:34:20 2020 (1 sec)
Time.Estimated...: Sat Sep 19 22:34:21 2020 (0 secs)
Guess.Base.......: File (D:\rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:  1793.9 kH/s (10.89ms) @ Accel:256 Loops:1 Thr:8 Vec:1
Recovered........: 4/4 (100.00%) Digests
Progress.........: 98304/128002 (76.80%)
Rejected.........: 0/98304 (0.00%)
Restore.Point....: 49152/128002 (38.40%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidates.#1....: triplet -> Dominic1

Started: Sat Sep 19 22:34:19 2020
Stopped: Sat Sep 19 22:34:22 2020

C:\Users\Admin\Desktop\CSS\hashcat>
```


crackme.txt - Notepad

```
467b6140fe3bb958f2332983914de787
fea0f1f6fede90bd0a925b4194deac11
578ed5a4eecf5a15803abdc49f6152d6
a99442d2a736365f5fe637e299b0e339
```


Command Prompt

```
C:\Users\Admin\Desktop\CSS\hashcat>.\hashcat.exe -m 0 -a 0 .\crackme.txt D:\rockyou.txt --show
467b6140fe3bb958f2332983914de787:january
fea0f1f6fede90bd0a925b4194deac11:cheese
578ed5a4eecf5a15803abdc49f6152d6:japan
a99442d2a736365f5fe637e299b0e339:abcd12345

C:\Users\Admin\Desktop\CSS\hashcat>
```

Implementation

We have implemented a simple application for Hashcat using the Flask framework. The hash text decryption algorithm was implemented in Python.

Tech Stack used: Python, HTML, CSS, Flask

Files Description:

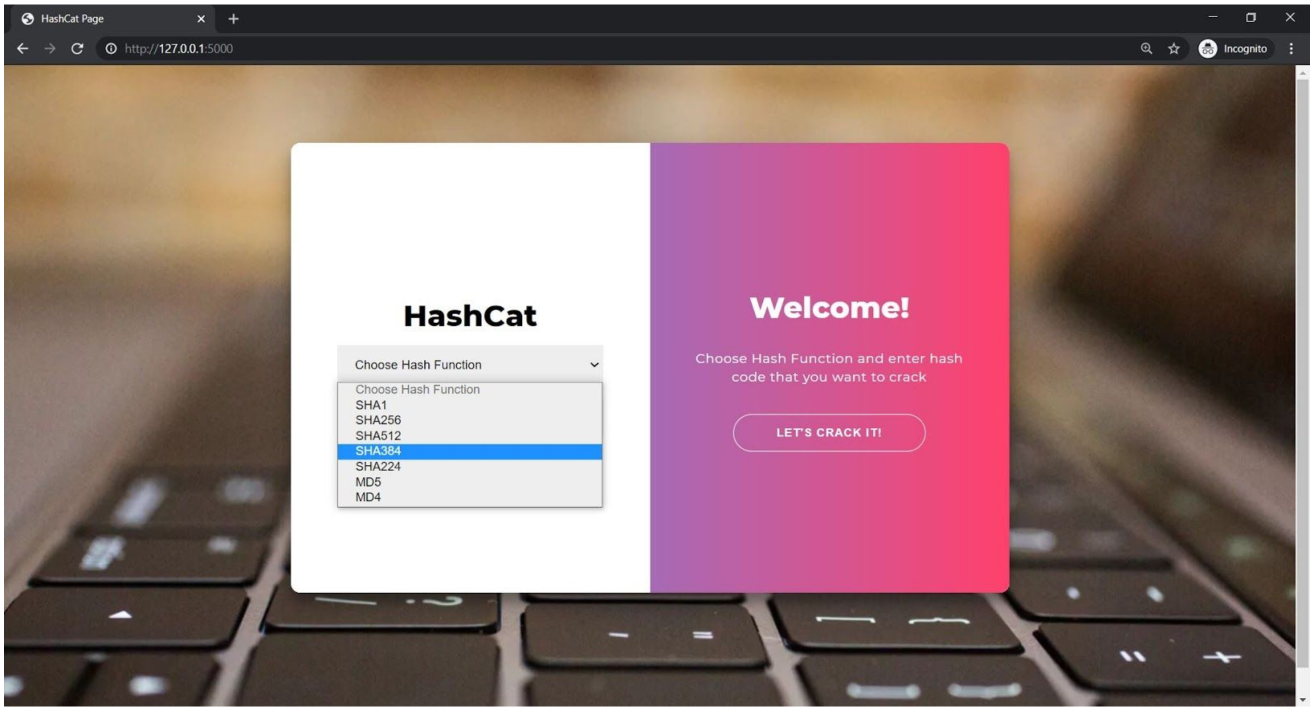
1. HashCat.css - CSS code for frontend
2. HashCat.html - HTML code for frontend
3. Hashapp.py - Flask program module

To run the application, go to the folder with the files and open the command prompt directed to the current directory. Then give the following commands -

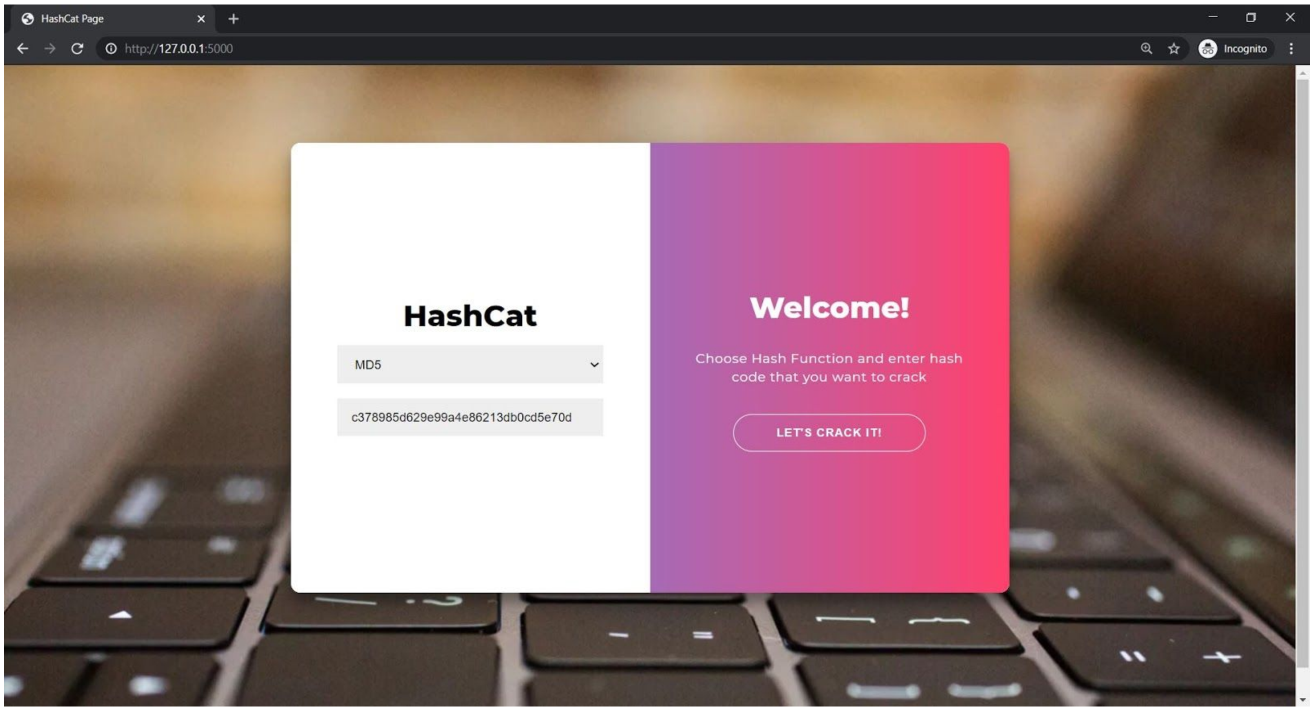
```
set FLASK_APP=hashapp
flask run
```

Output - Go to http://127.0.0.1:5000/ to open the page

Select HASH Function



Enter HASH text to be decrypted

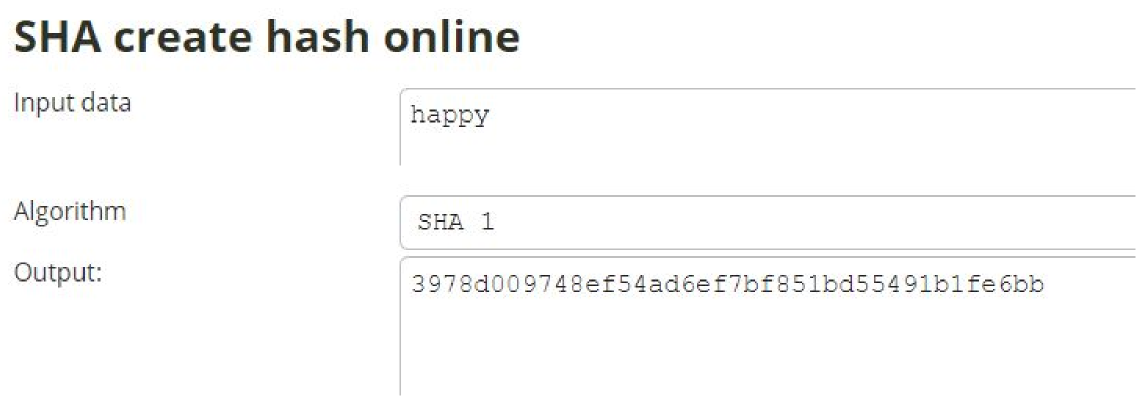


MD5

c378985d629e99a4e86213db0cd5e70d

SHA1 Algorithm

In cryptography, SHA-1 (Secure Hash Algorithm 1) is a cryptographic hash function that takes an input and produces a **160-bit (20-byte) hash value** known as a message digest – typically rendered as a hexadecimal number, **40 digits** long.

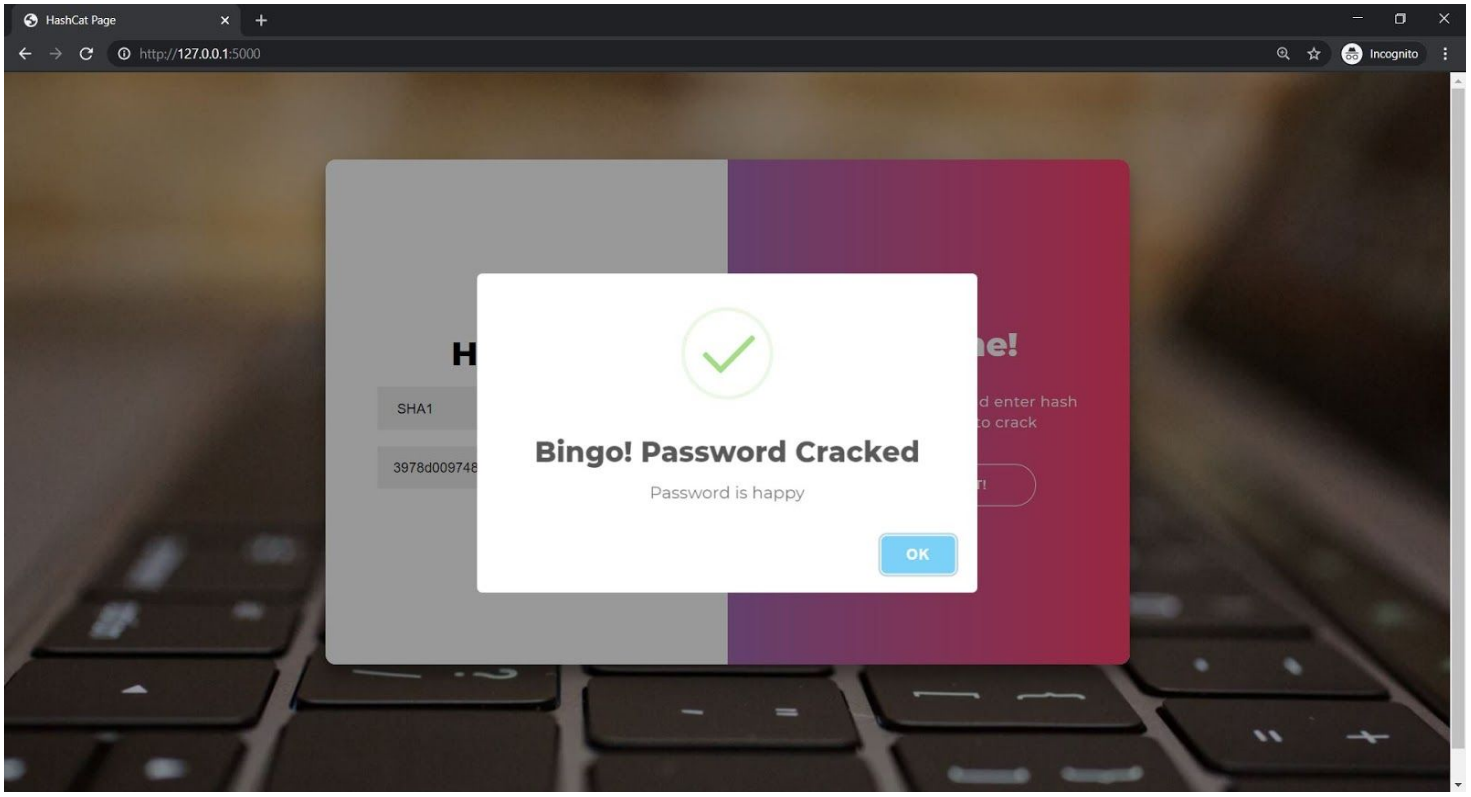


SHA create hash online

Input data: happy

Algorithm: SHA 1

Output: 3978d009748ef54ad6ef7bf851bd55491b1fe6bb

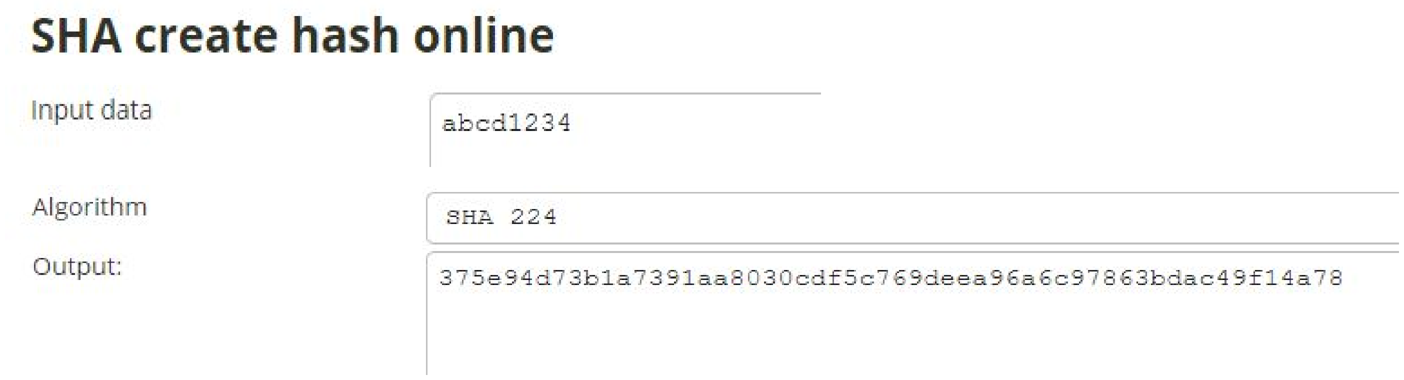


Bingo! Password Cracked

Password is happy

SHA224 Algorithm

SHA-224 belongs to the SHA-2 family of cryptographic hashes. It produces the 224 bit digest of a message. Computation of a SHA-224 hash value is two steps. First, the SHA-256 hash value is computed, except that a different initial value is used. Second, the resulting 256-bit hash value is truncated to 224 bits.

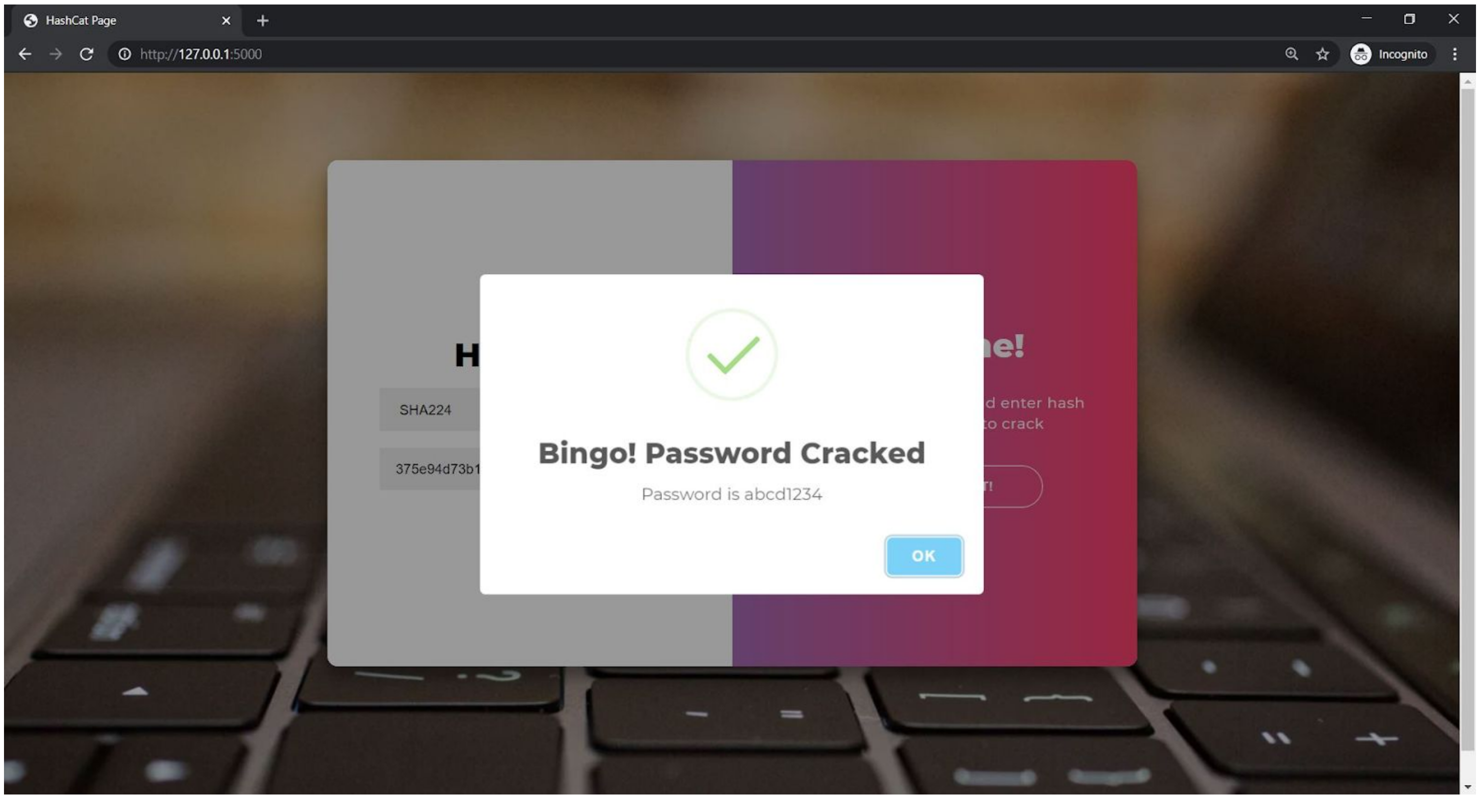


SHA create hash online

Input data: abcd1234

Algorithm: SHA 224

Output: 375e94d73b1a7391aa8030cdf5c769deea96a6c97863bdac49f14a78

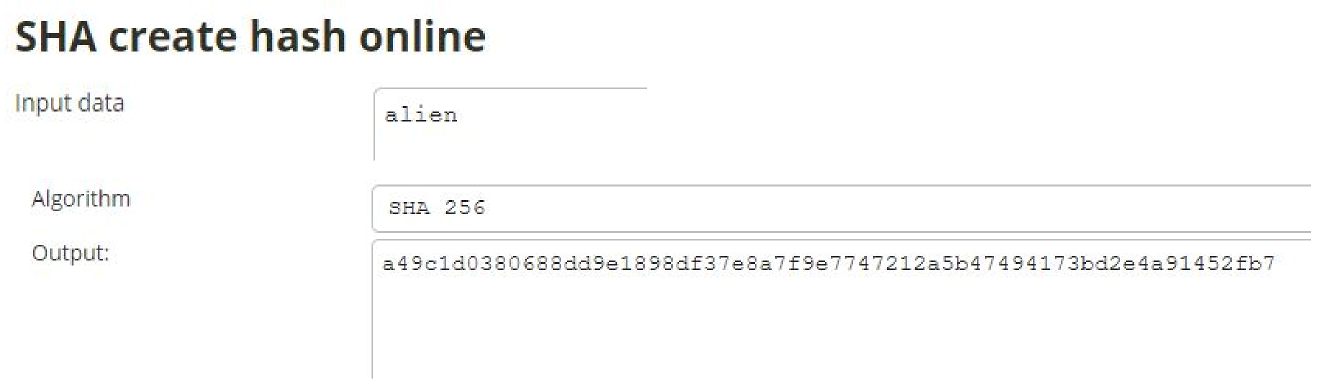


Bingo! Password Cracked

Password is abcd1234

SHA256 Algorithm

SHA-256 (secure hash **algorithm**, FIPS 182-2) is a cryptographic hash function with digest length of **256** bits. It is a keyless hash function; that is, an MDC (Manipulation Detection Code). A message is processed by blocks of 512 = 16 × 32 bits, each block requiring 64 rounds.

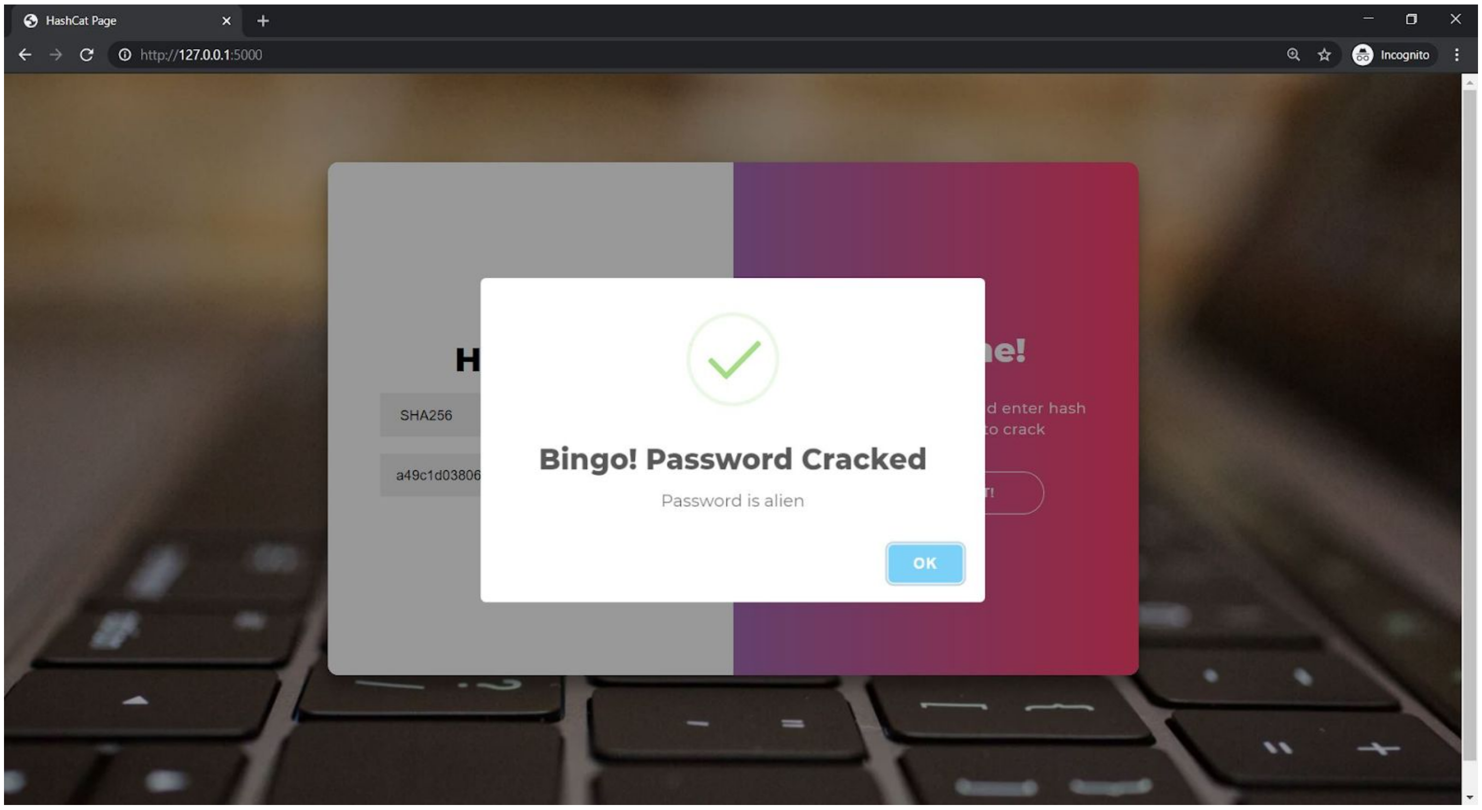


SHA create hash online

Input data: alien

Algorithm: SHA 256

Output: a49c1d0380688dd9e1898df37e8a7f9e7747212a5b47494173bd2e4a91452fb7

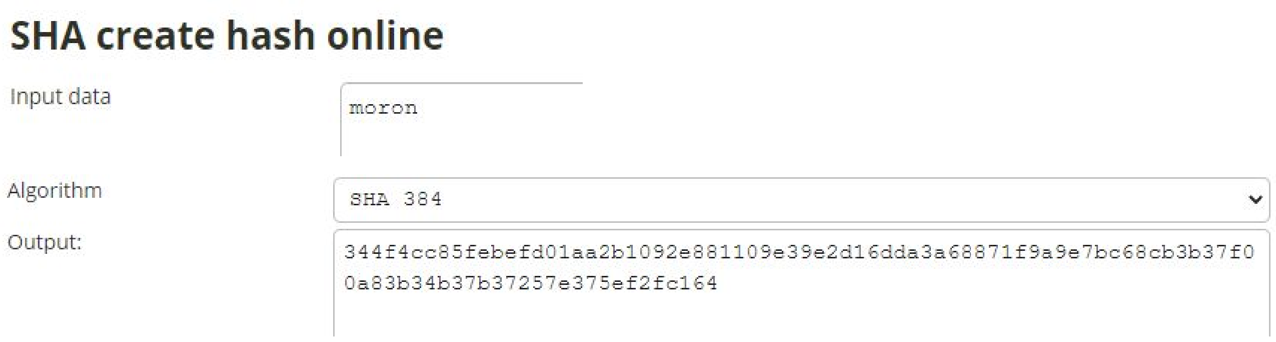


Bingo! Password Cracked

Password is alien

SHA384 Algorithm

SHA-384 belongs to the SHA-2 family of cryptographic hashes. It produces the **384 bit** digest of a message. SHA-384 is roughly 50% faster than SHA-224 and SHA-256 on 64-bit machines, even if its digest is longer. The speed-up is due to the internal computation being performed with 64-bit words, whereas the other two hash functions employ **32-bit words.**

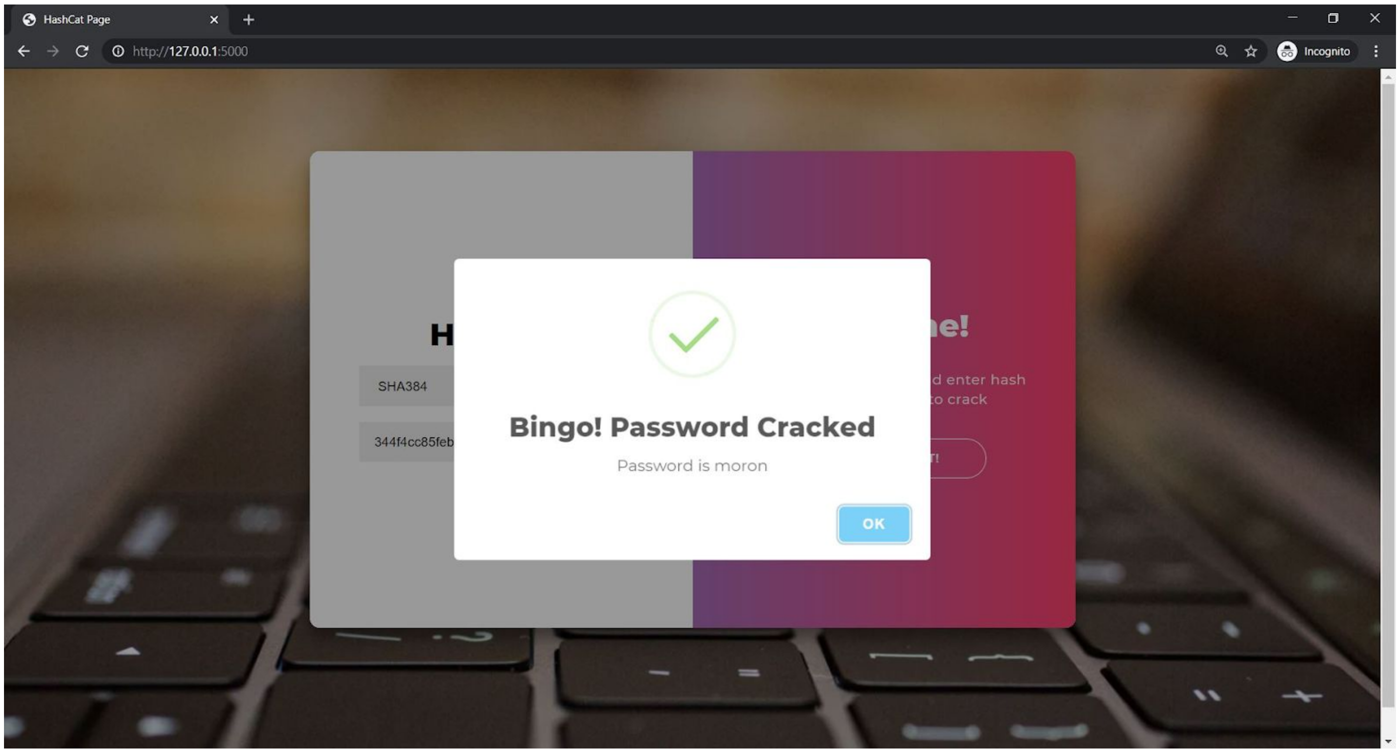


SHA create hash online

Input data: moron

Algorithm: SHA 384

Output:
```
344f4cc85febefd01aa2b1092e881109e39e2d16dda3a68871f9a9e7bc68cb3b37f0
0a83b34b37b37257e375ef2fc164
```

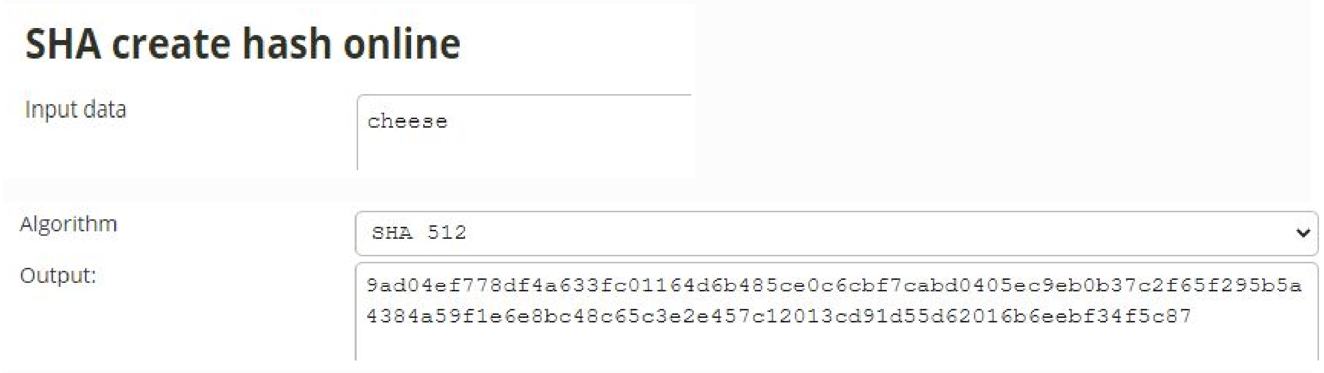


Bingo! Password Cracked

Password is moron

SHA512 Algorithm

SHA-512 and its two truncated variants (SHA-512/224 and SHA-512/256) belong to the SHA-2 family of cryptographic hashes. The speed-up is due to the internal computation being performed with 64-bit words, whereas the other two hash functions employ 32-bit words.



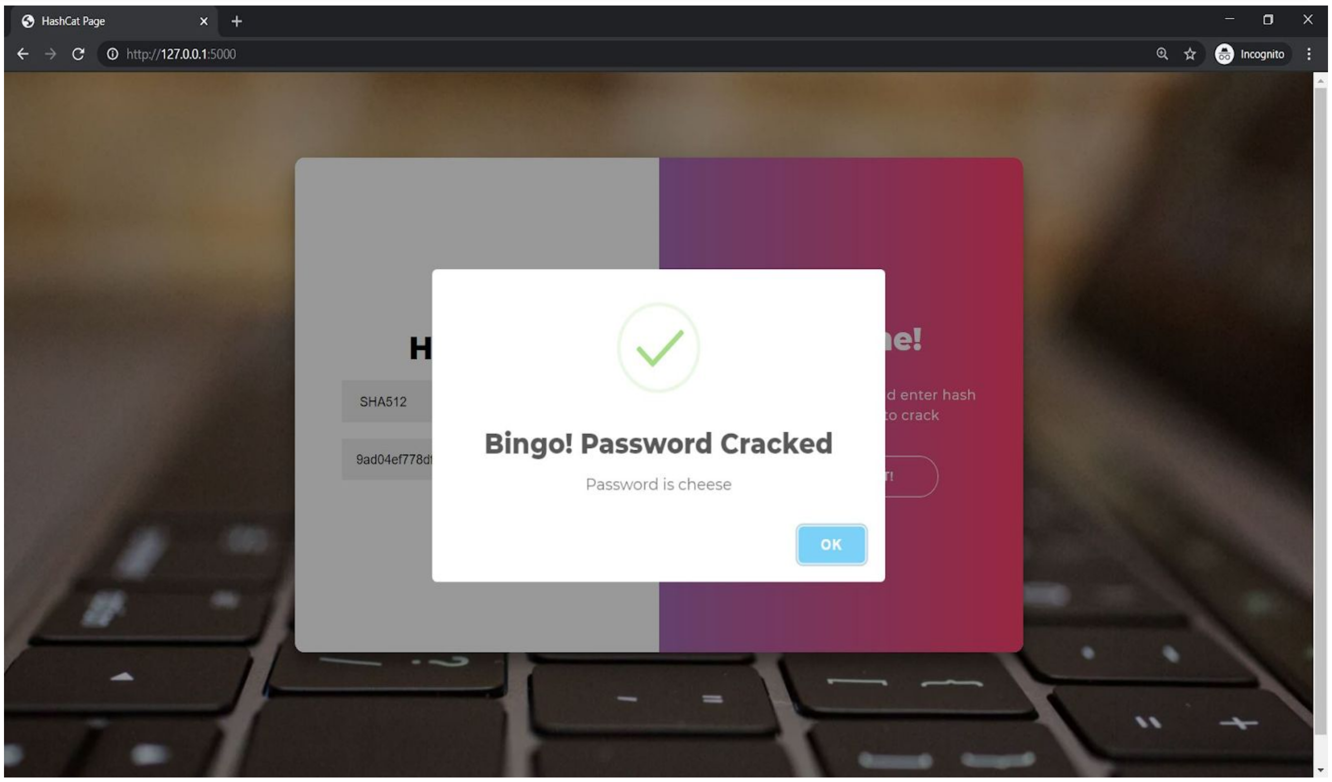
SHA create hash online

Input data: cheese

Algorithm: SHA 512

Output:

```
9ad04ef778df4a633fc01164d6b485ce0c6cbf7cabd0405ec9eb0b37c2f65f295b5a
4384a59f1e6e8bc48c65c3e2e457c12013cd91d55d62016b6eebf34f5c87
```

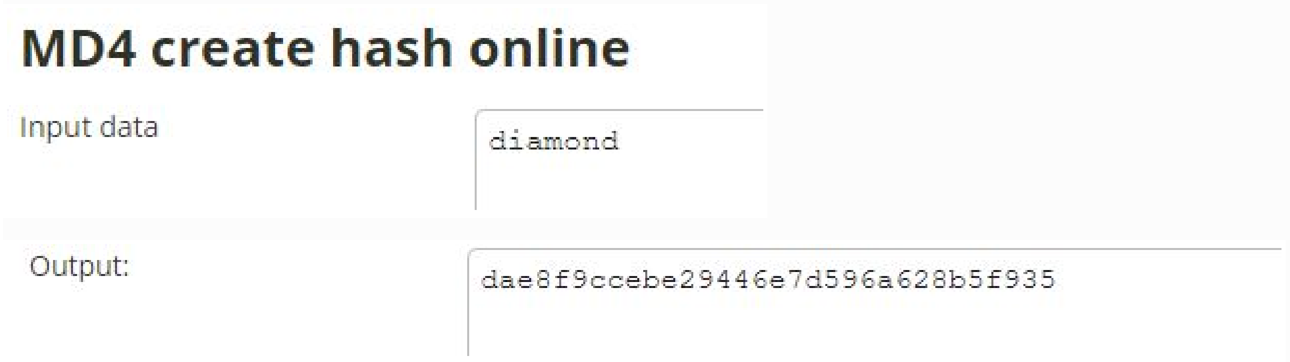


Bingo! Password Cracked

Password is cheese

MD4 Algorithm

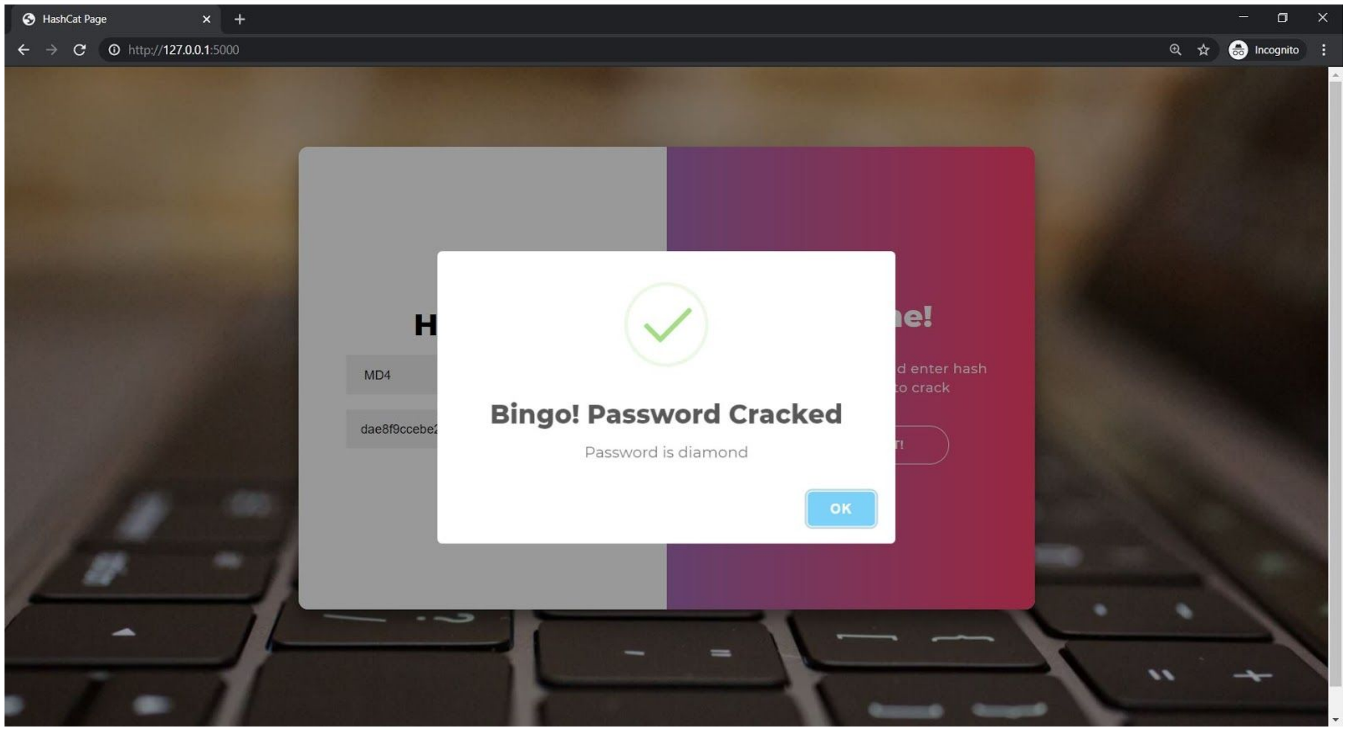
MD4 is a message digest algorithm (the fourth in a series) designed by Professor Ronald Rivest of MIT in 1990. It implements a cryptographic hash function for use in message integrity checks. The digest length is 128 bits. The algorithm has influenced later designs, such as the MD5, SHA and RIPEMD algorithms.



MD4 create hash online

Input data: diamond

Output: dae8f9ccebe29446e7d596a628b5f935

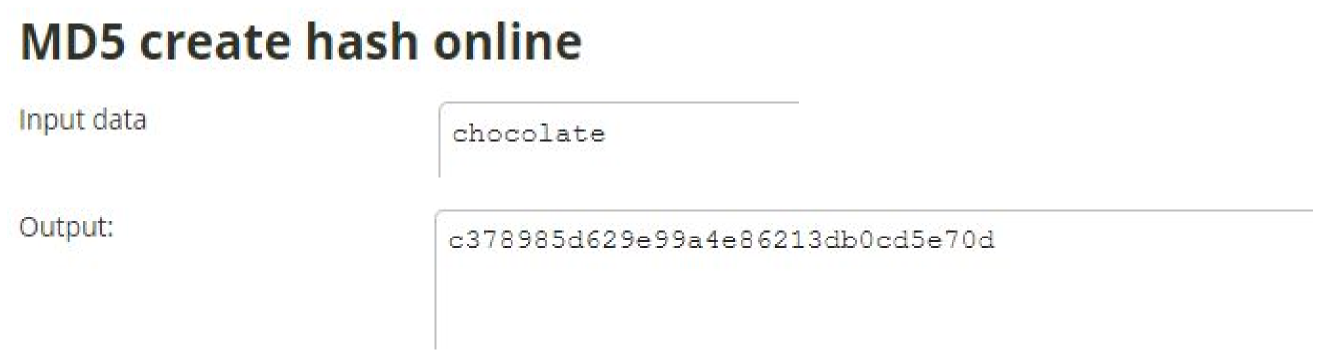


Bingo! Password Cracked

Password is diamond

MD5 Algorithm

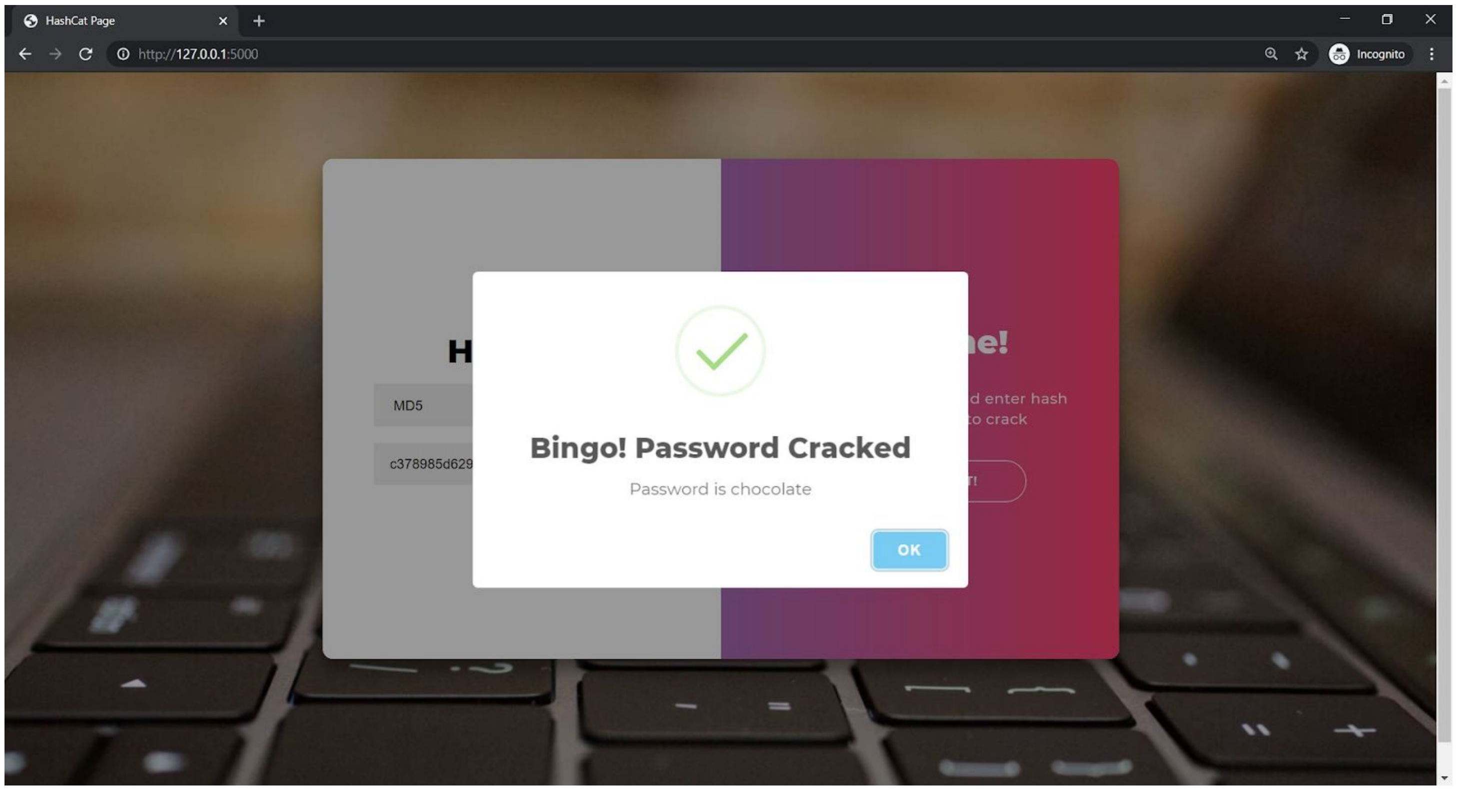
The MD5 hashing algorithm is a one-way cryptographic function that accepts a message of any length as input and returns as output a fixed-length digest value to be used for authenticating the original message.The MD5 hash function was originally designed for use as a secure cryptographic hash algorithm for authenticating digital signatures.



MD5 create hash online

Input data: chocolate

Output: c378985d629e99a4e86213db0cd5e70d



Bingo! Password Cracked

Password is chocolate

Conclusion

In conclusion, Hashcat, should be used when there is strong underlying hardware and a strong password to crack.Advantageous here is the support of graphics cards, their parallel usability and if necessary the distribution over the network.Furthermore, hashcat have strengths and weaknesses in some hashes, like as has been shown, it is not enough to have the strongest hardware, it is more important to adjust the software to the hardware in order to use it optimally. Incorrect implementation of the software can lead to extreme differences in performance. Moreover, for hashcat the binaries should be self-compiled and pre-compiled versions should be avoided.

Future Work & Scope of Improvement

There are currently two versions of the tool available: Hashcat and oclHashcat, one to use CPU and one to use GPU. We can expect the final version of Hashcat and the most important news will be the fusion of the two tools. Not only that, but you'll also be able to utilize both CPU and GPU, even in parallel, and it will support exotic hardware like FPGA and DSP.

References

[1] Binnie, Chris. (2016) - Password Cracking with Hashcat. 10.1002/9781119283096.ch9.
https://www.researchgate.net/publication/316361921_Password_Cracking_with_Hashcat/

[2] Hashcat Password Cracking (Linux)
https://medium.com/armourinfosec/hashcat-password-cracking-linux-97f91cdf0e47

[3] Hashcat Tutorial for beginners
https://resources.infosecinstitute.com/hashcat-tutorial-beginners/#gref

[4] What Should I Know About Hashcat?
https://www.alvareztg.com/what-should-i-know-about-hashcat/