Hashing Crypto 101 TryHackme Writeup

Shamsher khan · Follow
11 min read · May 9, 2021

Listen Share More

By Shamsher khan This is a Writeup of Tryhackme room "Hashing Crypto 101"

https://tryhackme.com/room/hashingcrypto101

Room link: https://tryhackme.com/room/hashingcrypto101
Note: This room is for Premium Members Only. who purchased THM premium membership.

Before we start, we need to get some jargon out of the way.
Read these, and take in as much as you can. We'll expand on some of them later in the room.

Plaintext — Data before encryption or hashing, often text but not always as it could be a photograph or other file instead.

Encoding — This is NOT a form of encryption, just a form of data representation like base64 or hexadecimal. Immediately reversible.

Hash — A hash is the output of a hash function. Hashing can also be used as a verb, “to hash”, meaning to produce the hash value of some data.

Brute force — Attacking cryptography by trying every different password or every different key

Cryptanalysis — Attacking cryptography by finding a weakness in the underlying maths

This room will likely involve some research. Get good at using search engines, it’s crucial to infosec.

Read the words, and understand the meanings!
Question 1. Is base64 encryption or encoding?

> *Answer: encoding*

What’s a hash function?

Hash functions are quite different from encryption. There is no key, and it’s meant to be impossible (or very very difficult) to go from the output back to the input.

A hash function takes some input data of any size, and creates a summary or “digest” of that data. The output is a fixed size. It’s hard to predict what the output will be for any input and vice versa. Good hashing algorithms will be (relatively) fast to compute, and slow to reverse (Go from output and determine input). Any small change in the input data (even a single bit) should cause a large change in the output.

The output of a hash function is normally raw bytes, which are then encoded. Common encodings for this are base 64 or hexadecimal. Decoding these won’t give you anything useful.

Why should I care?

Hashing is used very often in cyber security. When you logged into TryHackMe, that used hashing to verify your password. When you logged into your computer, that also used hashing to verify your password. You interact indirectly with hashing more than you would think, mostly in the context of passwords.

What’s a hash collision?

A hash collision is when 2 different inputs give the same output. Hash functions are designed to avoid this as best as they can, especially being able to engineer (create intentionally) a collision. Due to the pigeonhole effect, collisions are not avoidable. The pigeonhole effect is basically, there are a set number of different output values for the hash function, but you can give it any size input. As there are more inputs than outputs, some of the inputs must give the same output. If you have 128 pigeons and 96 pigeonholes, some of the pigeons are going to have to share.

MD5 and SHA1 have been attacked, and made technically insecure due to engineering hash collisions. However, no attack has yet given a collision in both algorithms at the same time so if you use the MD5 hash AND the SHA1 hash to compare, you will see they're different. The MD5 collision example is available from https://www.mscs.dal.ca/~selinger/md5collision/ and details of the SHA1 Collision are available from https://shattered.io/. Due to these, you shouldn't trust either algorithm for hashing passwords or data.

Question 2. What is the output size in bytes of the MD5 hash function?

MD5 processes a variable-length message into a fixed-length output of 128 bits.

128 bit= 16 bytes

> *Answer: 16*

Question 3. Can you avoid hash collisions? (Yea/Nay)

> *Answer: Nay*

Question 4. If you have an 8 bit hash output, how many possible hashes are there?

> *Answer: 256*

What can we do with hashing?

Hashing is used for 2 main purposes in Cyber Security. To verify integrity of data (More on that later), or for verifying passwords.

Hashing for password verification

Most webapps need to verify a user's password at some point. Storing these passwords in plain text would be bad. You've probably seen news stories about companies that have had their database leaked. Knowing some people, they use the same password for everything including their banking, so leaking these would be really really bad.

Quite a few data breaches have leaked plaintext passwords. You're probably familiar with "rockyou.txt" on Kali as a password word list. This came from a company that made widgets for MySpace. They stored their passwords in plaintext and the company had a data breach. The txt file contains over 14 million passwords (although some are *unlikely* to have been user passwords. Sort by length if you want to see what I mean).

Adobe had a notable data breach that was slightly different. The passwords were encrypted, rather than hashed and the encryption that was used was not secure. This meant that the plaintext could be relatively quickly retrieved. If you want to read more about this breach, this post from Sophos is excellent: https://nakedsecurity.sophos.com/2013/11/04/anatomy-of-a-password-disaster-adobes-giant-sized-cryptographic-blunder/

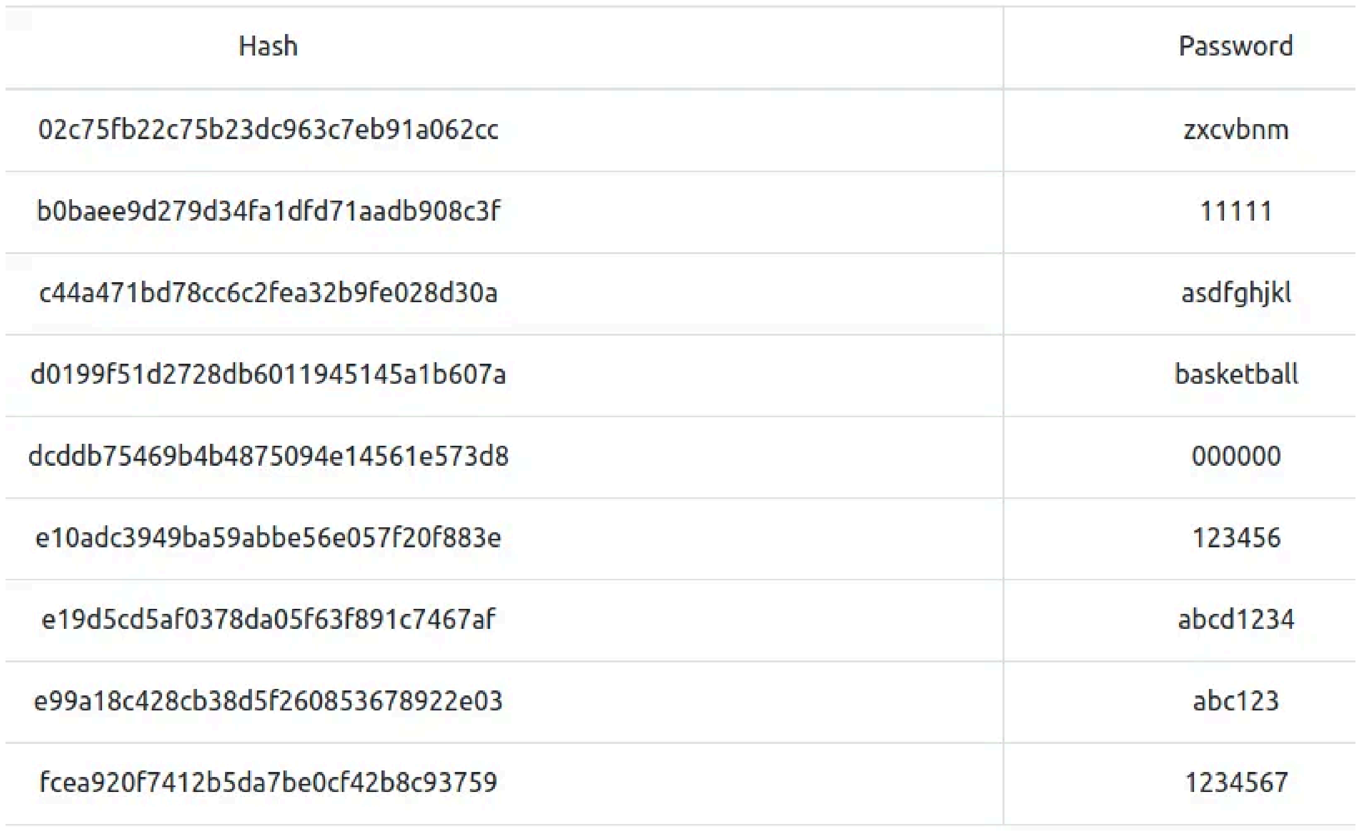
Linkedin also had a data breach. Linkedin used SHA1 for password verification, which is quite quick to compute using GPUs.

You can't encrypt the passwords, as the key has to be stored somewhere. If someone gets the key, they can just decrypt the passwords.

This is where hashing comes in. What if, instead of storing the password, you just stored the hash of the password? This means you never have to store the user's password, and if your database was leaked then an attacker would have to crack each password to find out what the password was. That sounds fairly useful.

There's just one problem with this. What if two users have the same password? As a hash function will always turn the same input into the same output, you will store the same password hash for each user. That means if someone cracks that hash, they get into more than one account. It also means that someone can create a "Rainbow table" to break the hashes.

A rainbow table is a lookup table of hashes to plaintexts, so you can quickly find out what password a user had just from the hash. A rainbow table trades time taken to crack a hash for hard disk space, but they do take time to create.
Here's a quick example so you can try and understand what they're like.



Hash	Password
02c75fb22c75b23dc963c7eb91a062cc	zxcvbnm
b0baee9d279d34fa1dfd71aadb908c3f	11111
c44a471bd78cc6c2fea32b9fe028d30a	asdfghjkl
d0199f51d2728db6011945145a1b607a	basketball
dcddb75469b4b4875094e14561e573d8	000000
e10adc3949ba59abbe56e057f20f883e	123456
e19d5cd5af0378da05f63f891c7467af	abcd1234
e99a18c428cb38d5f260853678922e03	abc123
fcea920f7412b5da7be0cf42b8c93759	1234567

https://tryhackme.com/room/hashingcrypto101

Websites like **Crackstation** internally use HUGE rainbow tables to provide fast password cracking for hashes without salts. Doing a lookup in a sorted list of hashes is really quite fast, much much faster than trying to crack the hash.

Protecting against rainbow tables

To protect against rainbow tables, we add a salt to the passwords. The salt is randomly generated and stored in the database, unique to each user. In theory, you could use the same salt for all users but that means that duplicate passwords would still have the same hash, and a rainbow table could still be created specific passwords with that salt.

The salt is added to either the start or the end of the password before it's hashed, and this means that every user will have a different password hash even if they have the same password. Hash functions like bcrypt and sha512crypt handle this automatically. Salts don't need to be kept private.

Question 1. Crack the hash "d0199f51d2728db6011945145a1b607a" using the rainbow table manually.

You can use this ***website*** to crack this hash value

> *Answer: basketball*

Question 2. Crack the hash "5b31f93c09ad1d065c0491b764d04933" using online tools

You can use this ***website*** to crack this hash value.

> *Answer: tryhackme*

Question 3. Should you encrypt passwords? Yea/Nay

> *Answer: Nay*

Recognising password hashes

Automated hash recognition tools such as https://pypi.org/project/hashID/ exist, but they are unreliable for many formats. For hashes that have a prefix, the tools are reliable. Use a healthy combination of context and tools. If you found the hash in a web application database, it's more likely to be md5 than NTLM. Automated hash recognition tools often get these hash types mixed up, which highlights the importance of learning yourself.

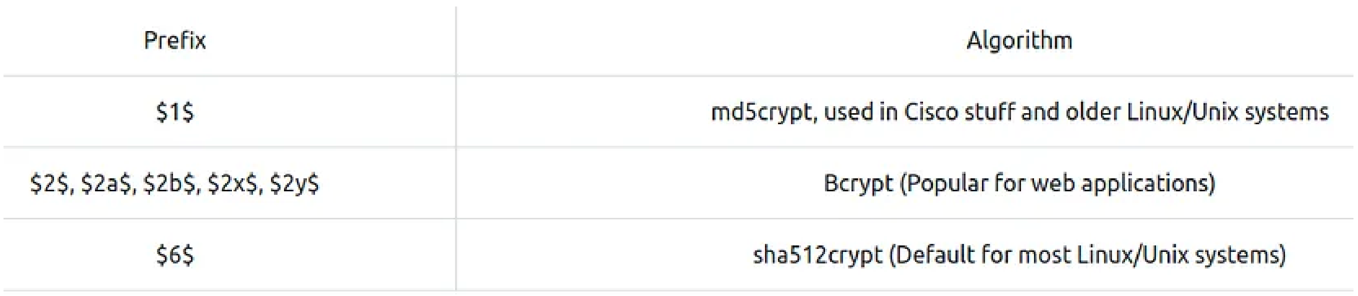
Unix style password hashes are very easy to recognise, as they have a prefix. The prefix tells you the hashing algorithm used to generate the hash. The standard format is **$format$rounds$salt$hash**.

Windows passwords are hashed using NTLM, which is a variant of md4. They're visually identical to md4 and md5 hashes, so it's very important to use context to work out the hash type.

On Linux, password hashes are stored in /etc/shadow. This file is normally only readable by root. They used to be stored in /etc/passwd, and were readable by everyone.

On Windows, password hashes are stored in the SAM. Windows tries to prevent normal users from dumping them, but tools like mimikatz exist for this. Importantly, the hashes found there are split into NT hashes and LM hashes.

Here's a quick table of the most Unix style password prefixes that you'll see.

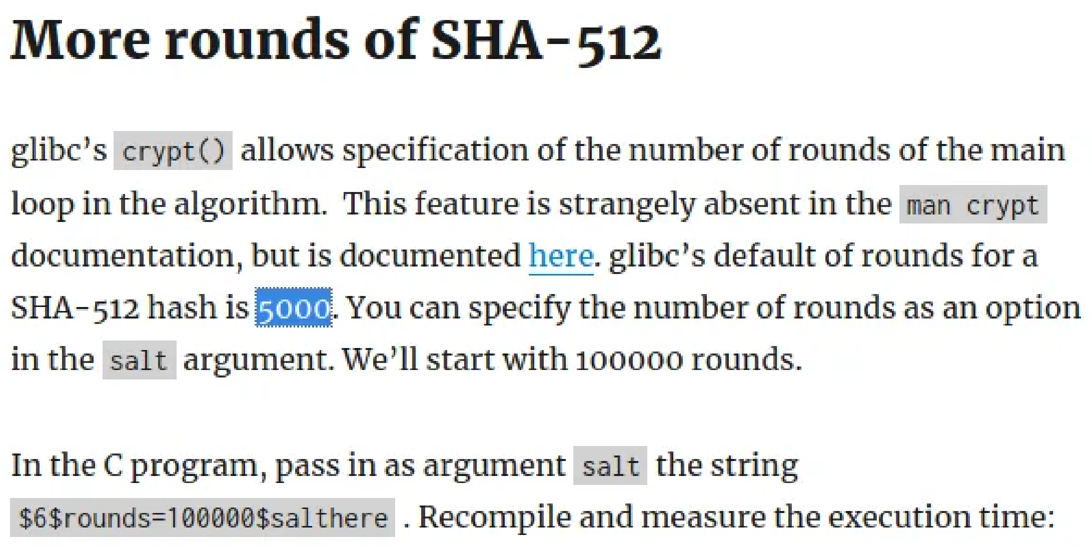


Prefix	Algorithm
1	md5crypt, used in Cisco stuff and older Linux/Unix systems
2, $2a$, $2b$, $2x$, $2y$	Bcrypt (Popular for web applications)
6	sha512crypt (Default for most Linux/Unix systems)

A great place to find more hash formats and password prefixes is the hashcat example page, available here: https://hashcat.net/wiki/doku.php?id=example_hashes.
For other hash types, you'll normally need to go by length, encoding or some research into the application that generated them. Never underestimate the power of research.

Question 1. How many rounds does sha512crypt ($6$) use by default?

You can read this ***website*** for this question



More rounds of SHA-512

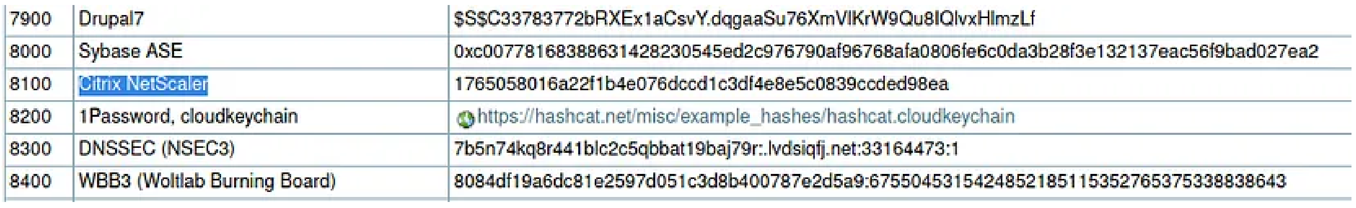
glibc's `crypt()` allows specification of the number of rounds of the main loop in the algorithm. This feature is strangely absent in the `man crypt` documentation, but is documented here. glibc's default of rounds for a SHA-512 hash is 5000. You can specify the number of rounds as an option in the `salt` argument. We'll start with 100000 rounds.

In the C program, pass in as argument `salt` the string `$6$rounds=100000$salthere` . Recompile and measure the execution time:

> *Answer: 5000*

Question 2. What's the hashcat example hash (from the website) for Citrix Netscaler hashes?

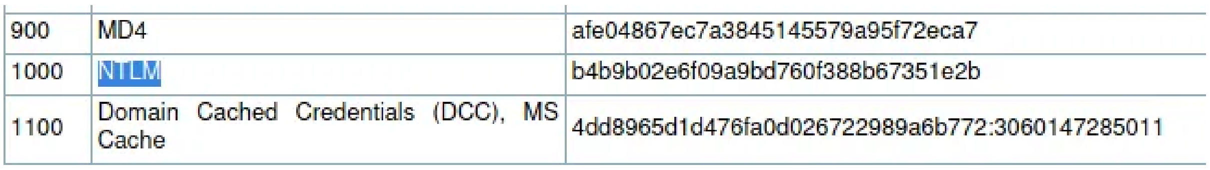
You can find the answer in this ***website***.



7900	Drupal7	SC33783772bRXEx1aCsvY.dqgaaSu76XmVlKrW9Qu8IQlvxHlmzLf
8000	Sybase ASE	0xc00778168388631428230545ed2c976790af96768afa0806fe6c0da3b28f3e132137eac56f9bad027ea2
8100	Citrix NetScaler	1765058016a22f1b4e076dccd1c3df4e8e5c0839ccded98ea
8200	1Password, cloudkeychain	https://hashcat.net/misc/example_hashes/hashcat.cloudkeychain
8300	DNSSEC (NSEC3)	7b5n74kq8r441blc2c5qbbat19baj79r:.lvdsiqfj.net:33164473:1
8400	WBB3 (Woltlab Burning Board)	8084df19a6dc81e2597d051c3d8b400787e2d5a9:6755045315424852185115352765375338838643

> *Answer: 1765058016a22f1b4e076dccd1c3df4e8e5c0839ccded98ea*

Question 3. How long is a Windows NTLM hash, in characters?



900	MD4	afe04867ec7a3845145579a95f72eca7
1000	NTLM	b4b9b02e6f09a9bd760f388b67351e2b
1100	Domain Cached Credentials (DCC), MS Cache	4dd8965d1d476fa0d026722989a6b772:3060147285011

You can find the answer in this ***website***.

> *Answer: 32*

Password Cracking

We've already mentioned rainbow tables as a method to crack hashes that don't have a salt, but what if there's a salt involved?

You can't "decrypt" password hashes. They're not encrypted. You have to crack the hashes by hashing a large number of different inputs (often rockyou, these are the possible passwords), potentially adding the salt if there is one and comparing it to the target hash. Once it matches, you know what the password was. Tools like Hashcat and John the Ripper are normally used for this.

Why crack on GPUs?

Graphics cards have thousands of cores. Although they can't do the same sort of work that a CPU can, they are very good at some of the maths involved in hash functions. This means you can use a graphics card to crack most hash types much more quickly. Some hashing algorithms, notably bcrypt, are designed so that hashing on a GPU is about the same speed as hashing on a CPU which helps them resist cracking.

Cracking on VMs?

It's worth mentioning that virtual machines normally don't have access to the host's graphics card(s) (You can set this up, but it's a lot of work). If you want to run hashcat, it's best to run it on your host (Windows builds are available on the website, run it from powershell). You can get Hashcat working with OpenCL in a VM, but the speeds will likely be much worse than cracking on your host. John the ripper uses CPU by default and as such, works in a VM out of the box although you may get better speeds running it on the host OS as it will have more threads and no overhead from running in a VM.

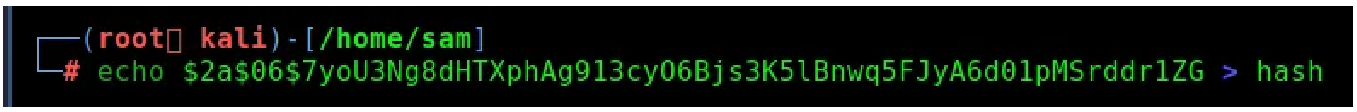
NEVER (I repeat, NEVER!) use — force for hashcat. It can lead to false positives (wrong passwords being given to you) and false negatives (skips over the correct hash).

UPDATE: As of Kali 2020.2, hashcat 6.0 will run on the CPU without — force. I still recommend cracking on your host OS if you have a GPU, as it will be much much faster.

Time to get cracking!

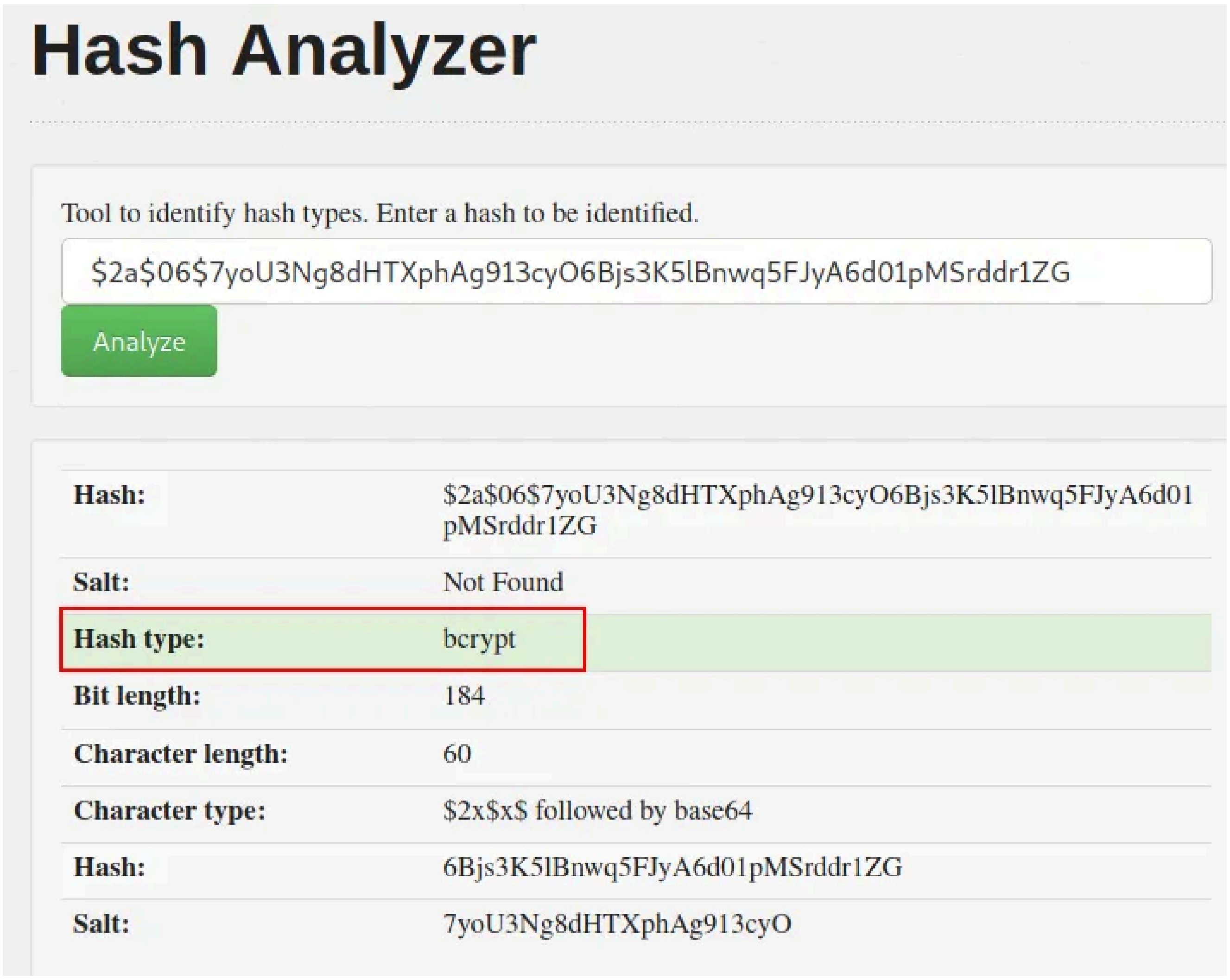
I'll provide the hashes. Crack them. You can choose how. You'll need to use online tools, Hashcat, and/or John the Ripper. Remember the restrictions on online rainbow tables. Don't be afraid to use the hints. Rockyou or online tools should be enough to find all of these.

Question 1. Crack this hash:
$2a$06$7yoU3Ng8dHTXphAg913cyO6Bjs3K5lBnwq5FJyA6d01pMSrddr1ZG



```
┌──(root㉿ kali)-[/home/sam]
└─# echo $2a$06$7yoU3Ng8dHTXphAg913cyO6Bjs3K5lBnwq5FJyA6d01pMSrddr1ZG > hash
```

Then, I analyzed this hash value.



Hash Analyzer

Tool to identify hash types. Enter a hash to be identified.

$2a$06$7yoU3Ng8dHTXphAg913cyO6Bjs3K5lBnwq5FJyA6d01pMSrddr1ZG

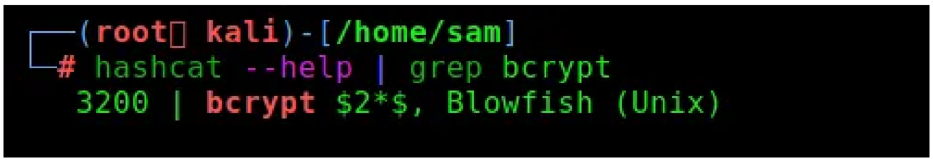
Analyze

Hash:	$2a$06$7yoU3Ng8dHTXphAg913cyO6Bjs3K5lBnwq5FJyA6d01pMSrddr1ZG
Salt:	Not Found
Hash type:	bcrypt
Bit length:	184
Character length:	60
Character type:	$2x$x$ followed by base64
Hash:	6Bjs3K5lBnwq5FJyA6d01pMSrddr1ZG
Salt:	7yoU3Ng8dHTXphAg913cyO

https://www.tunnelsup.com/hash-analyzer/

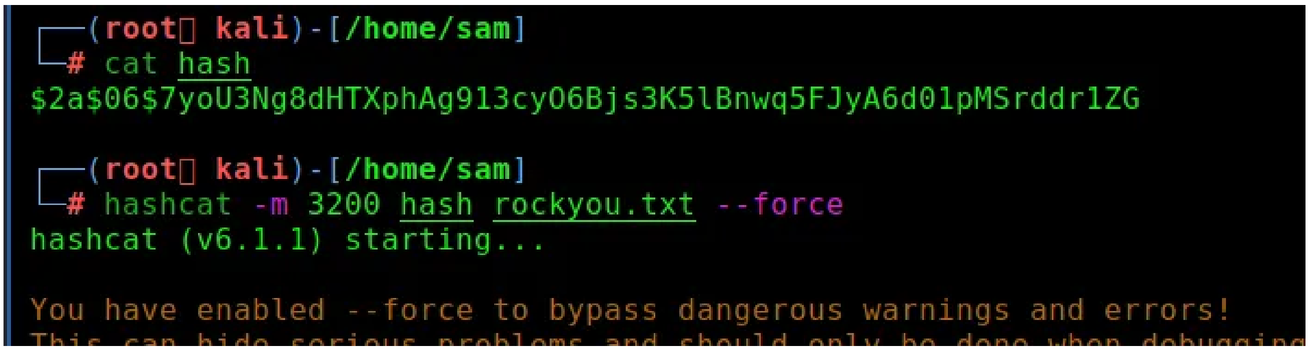
Then I used "hashcat" in Kali Linux.

In hashcat tool, bcrypt hash code is 3200. You can see this hash code with "hashcat –help" command.



```
┌──(root㉿ kali)-[/home/sam]
└─# hashcat --help | grep bcrypt
  3200 | bcrypt $2*$, Blowfish (Unix)
```

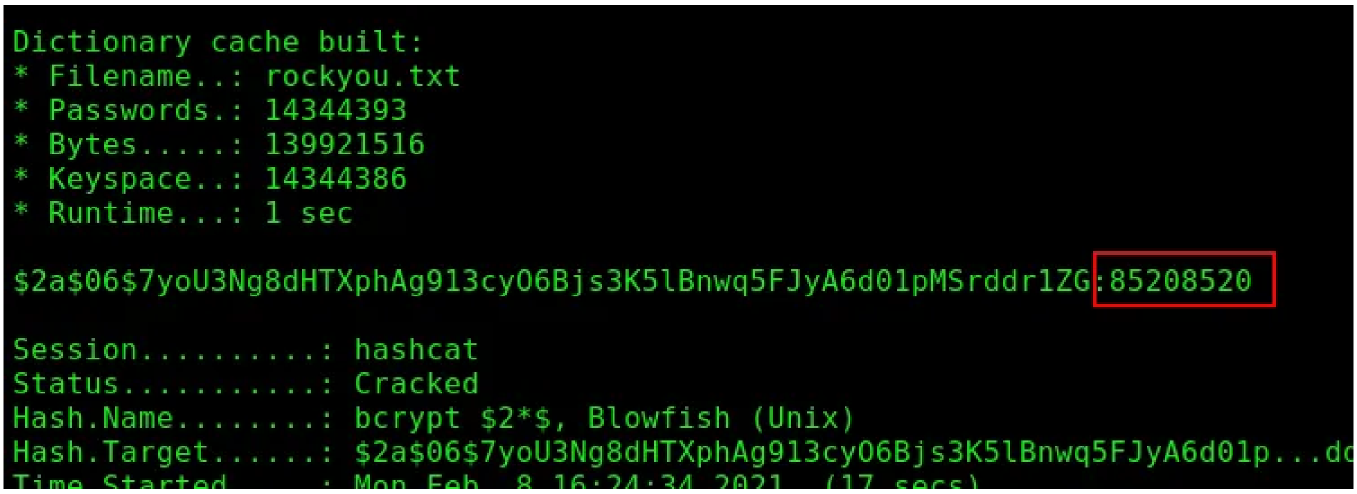
Then I used this command and "rockyou.txt" file for worldlist.



```
┌──(root□ kali)-[/home/sam]
└─# cat hash
$2a$06$7yoU3Ng8dHTXphAg913cy06Bjs3K5lBnwq5FJyA6d01pMSrddr1ZG

┌──(root□ kali)-[/home/sam]
└─# hashcat -m 3200 hash rockyou.txt --force
hashcat (v6.1.1) starting...

You have enabled --force to bypass dangerous warnings and errors!
This can hide serious problems and should only be done when debugging
```



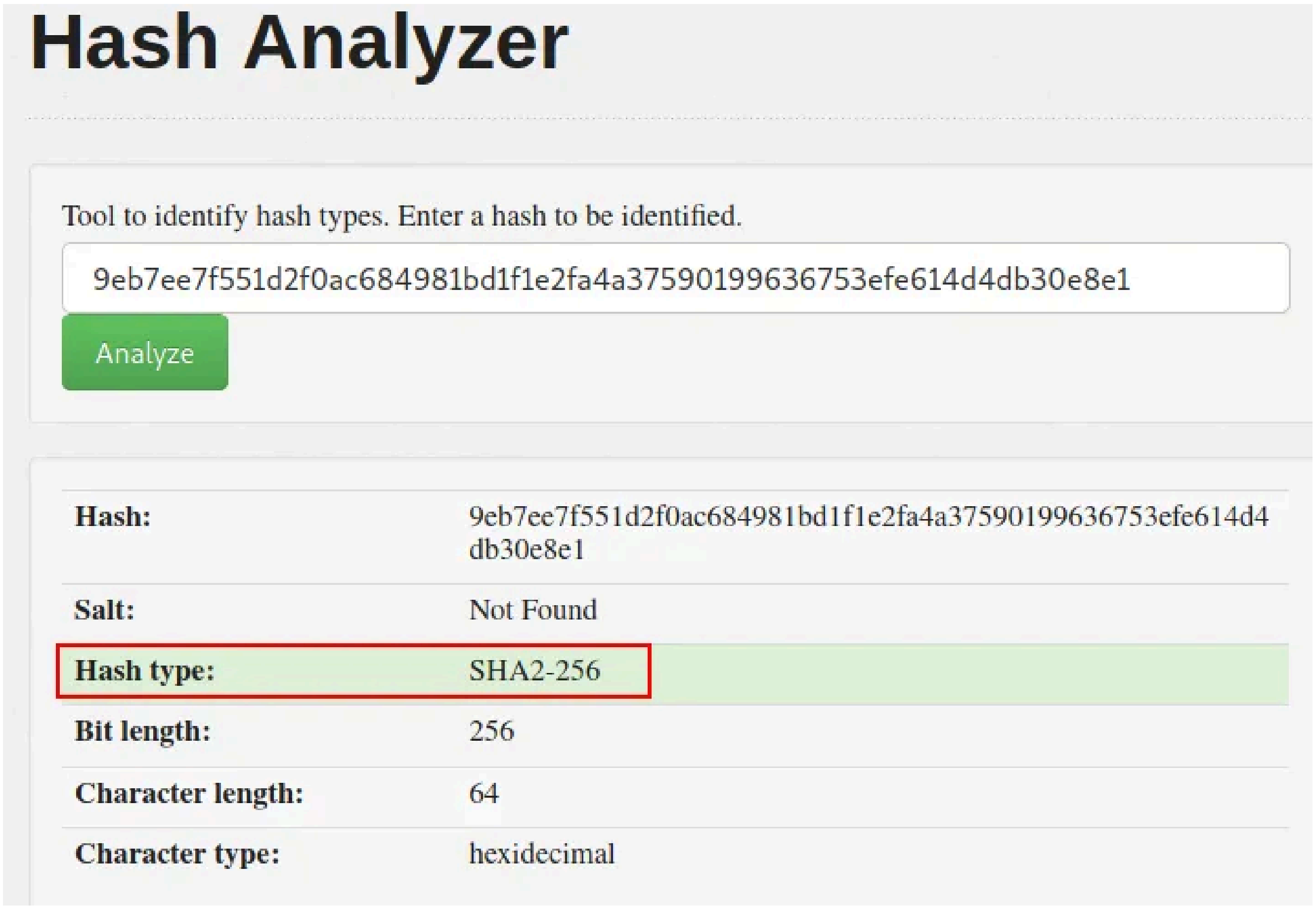
```
Dictionary cache built:
* Filename..: rockyou.txt
* Passwords.: 14344393
* Bytes.....: 139921516
* Keyspace..: 14344386
* Runtime...: 1 sec

$2a$06$7yoU3Ng8dHTXphAg913cy06Bjs3K5lBnwq5FJyA6d01pMSrddr1ZG:85208520

Session..........: hashcat
Status...........: Cracked
Hash.Name........: bcrypt $2*$, Blowfish (Unix)
Hash.Target......: $2a$06$7yoU3Ng8dHTXphAg913cy06Bjs3K5lBnwq5FJyA6d01p...dc
Time.Started.....: Mon Feb  8 16:24:34 2021  (17 secs)
```

> ***Answer:** 85208520*

Question 2. Crack this hash: 9eb7ee7f551d2f0ac684981bd1f1e2fa4a37590199636753efe614d4db30e8e1



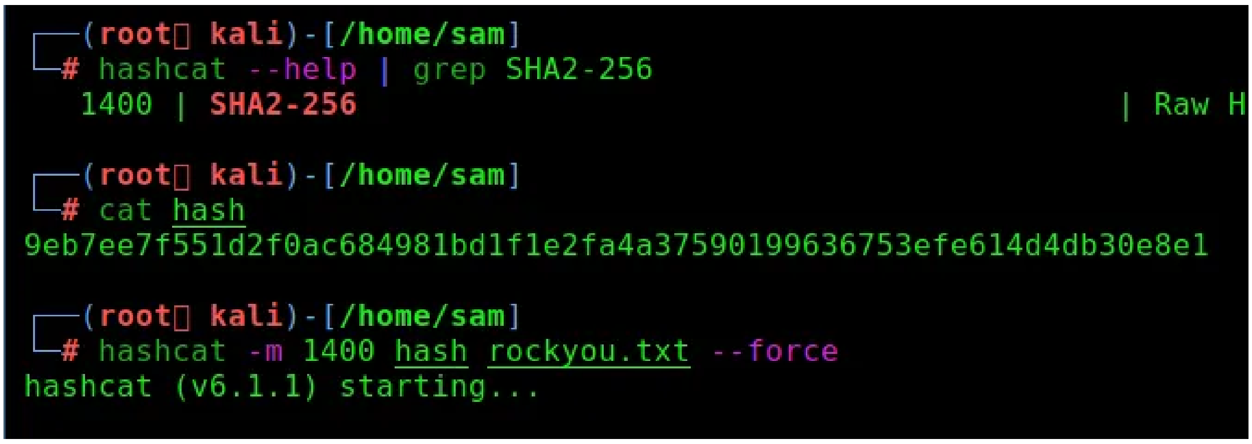
Hash Analyzer

Tool to identify hash types. Enter a hash to be identified.

9eb7ee7f551d2f0ac684981bd1f1e2fa4a37590199636753efe614d4db30e8e1

Analyze

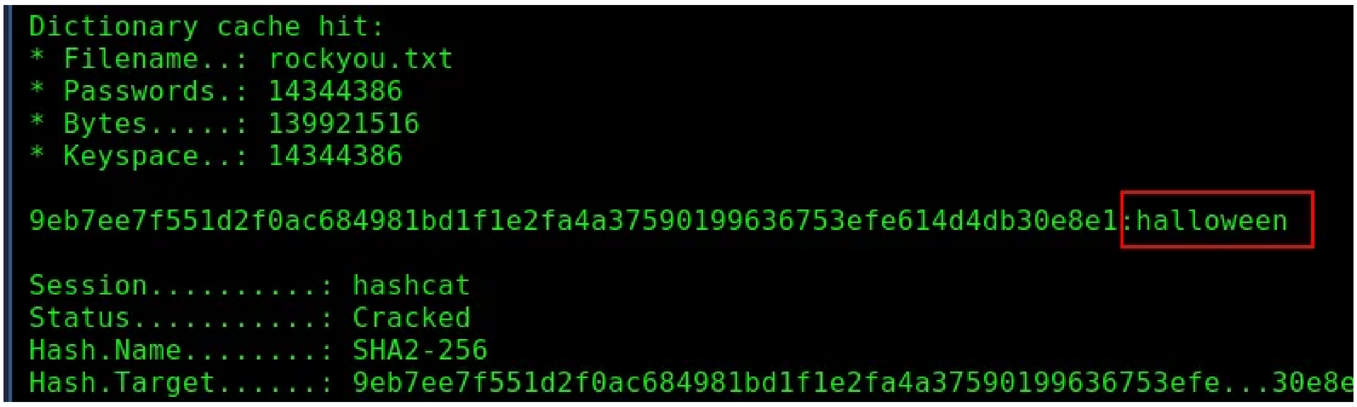
Hash:	9eb7ee7f551d2f0ac684981bd1f1e2fa4a37590199636753efe614d4db30e8e1
Salt:	Not Found
Hash type:	SHA2-256
Bit length:	256
Character length:	64
Character type:	hexidecimal



```
┌──(root💀 kali)-[/home/sam]
└─# hashcat --help | grep SHA2-256
   1400 | SHA2-256                                           | Raw H

┌──(root💀 kali)-[/home/sam]
└─# cat hash
9eb7ee7f551d2f0ac684981bd1f1e2fa4a37590199636753efe614d4db30e8e1

┌──(root💀 kali)-[/home/sam]
└─# hashcat -m 1400 hash rockyou.txt --force
hashcat (v6.1.1) starting...
```



```
Dictionary cache hit:
* Filename..: rockyou.txt
* Passwords.: 14344386
* Bytes.....: 139921516
* Keyspace..: 14344386

9eb7ee7f551d2f0ac684981bd1f1e2fa4a37590199636753efe614d4db30e8e1:halloween

Session..........: hashcat
Status...........: Cracked
Hash.Name........: SHA2-256
Hash.Target......: 9eb7ee7f551d2f0ac684981bd1f1e2fa4a37590199636753efe...30e8e
```

> *Answer: halloween*

Question 3. Crack this hash: $6$GQXVvW4EuM$ehD6jWiMsfNorxy5SINsgdlxmAEl3.yif0/c3NqzGLa0P.S7KRDYjycw5bnYkF5ZtB8wQy8KnskuWQS3Yr1wQ0

same method

> *Answer: spaceman*

Question 4. Bored of this yet? Crack this hash: b6b0d451bbf6fed658659a9e7e5598fe

You can use this *website* to crack this hash value.

> *Answer: funforyou*

Hashing for integrity checking

Integrity Checking

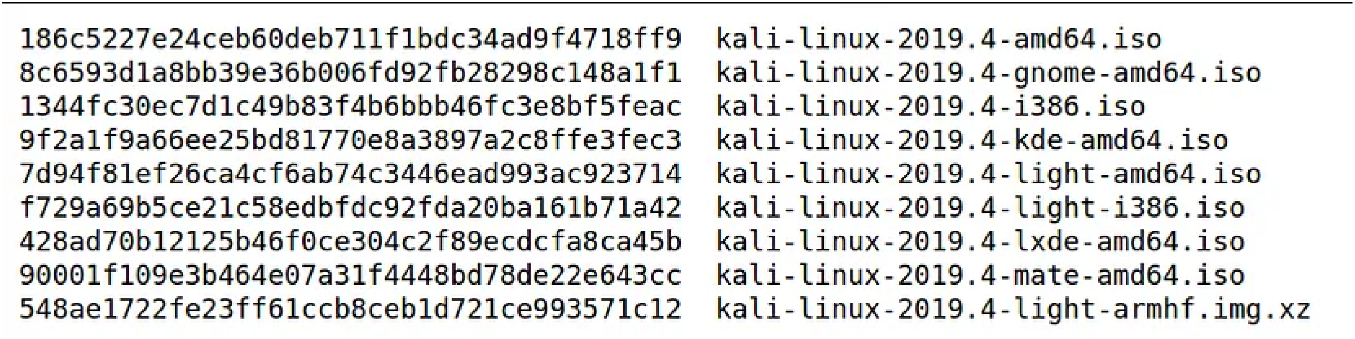
Hashing can be used to check that files haven't been changed. If you put the same data in, you always get the same data out. If even a single bit changes, the hash will change a lot. This means you can use it to check that files haven't been modified or to make sure that they have downloaded correctly. You can also use hashing to find duplicate files, if two pictures have the same hash then they are the same picture.

HMACs

HMAC is a method of using a cryptographic hashing function to verify the authenticity and integrity of data. The TryHackMe VPN uses HMAC-SHA512 for message authentication, which you can see in the terminal output. A HMAC can be used to ensure that the person who created the HMAC is who they say they are (authenticity), and that the message hasn't been modified or corrupted (integrity). They use a secret key, and a hashing algorithm in order to produce a hash.

What's the SHA1 sum for the amd64 Kali 2019.4 ISO? http://old.kali.org/kali-images/kali-2019.4/

http://old.kali.org/kali-images/kali-2019.4/SHA1SUMS



```
186c5227e24ceb60deb711f1bdc34ad9f4718ff9  kali-linux-2019.4-amd64.iso
8c6593d1a8bb39e36b006fd92fb28298c148a1f1  kali-linux-2019.4-gnome-amd64.iso
1344fc30ec7d1c49b83f4b6bbb46fc3e8bf5feac  kali-linux-2019.4-i386.iso
9f2a1f9a66ee25bd81770e8a3897a2c8ffe3fec3  kali-linux-2019.4-kde-amd64.iso
7d94f81ef26ca4cf6ab74c3446ead993ac923714  kali-linux-2019.4-light-amd64.iso
f729a69b5ce21c58edbfdc92fda20ba161b71a42  kali-linux-2019.4-light-i386.iso
428ad70b12125b46f0ce304c2f89ecdcfa8ca45b  kali-linux-2019.4-lxde-amd64.iso
90001f109e3b464e07a31f4448bd78de22e643cc  kali-linux-2019.4-mate-amd64.iso
548ae1722fe23ff61ccb8ceb1d721ce993571c12  kali-linux-2019.4-light-armhf.img.xz
```

http://old.kali.org/kali-images/kali-2019.4/SHA1SUMS

OR

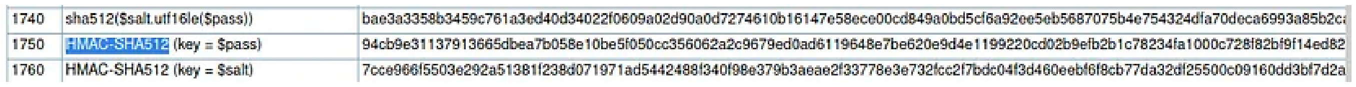
After download kali linux

```
% openssl sha1 kali-linux-2019.4-amd64.iso
SHA1 (kali-linux-2019.4-amd64.iso) = 186c5227e24ceb60deb711f1bdc34ad9f4718ff9
```

> *Answer: 186c5227e24ceb60deb711f1bdc34ad9f4718ff9*

What's the hashcat mode number for HMAC-SHA512 (key = $pass)?

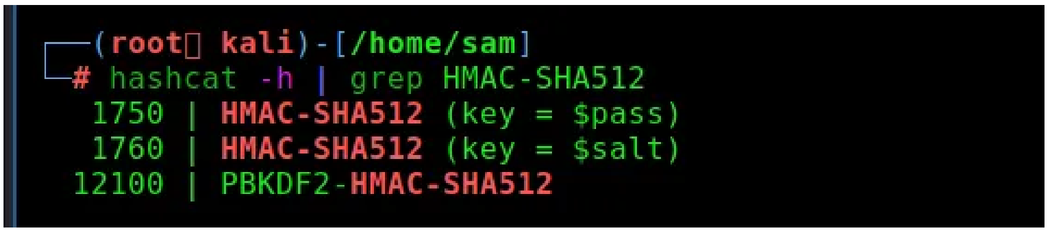
You can see this code in this ***website***.



1740	sha512($salt.utf16le($pass))	bae3a3358b3459c761a3ed40d34022f0609a02d90a0d7274610b16147e58ece00cd849a0bd5cf6a92ee5eb5687075b4e754324dfa70deca6993a85b2c
1750	HMAC-SHA512 (key = $pass)	94cb9e31137913665dbea7b058e10be5f050cc356062a2c9679ed0ad6119648e7be620e9d4e1199220cd02b9efb2b1c78234fa1000c728f82bf9f14ed82
1760	HMAC-SHA512 (key = $salt)	7cce966f5503e292a51381f238d071971ad5442488f340f98e379b3aeae2f33778e3e732fcc2f7bdc04f3d460eebf6f8cb77da32df25500c09160dd3bf7d2a

https://hashcat.net/wiki/doku.php?id=example_hashes

OR



```
┌──(root㉿ kali)-[/home/sam]
└─# hashcat -h | grep HMAC-SHA512
  1750 | HMAC-SHA512 (key = $pass)
  1760 | HMAC-SHA512 (key = $salt)
 12100 | PBKDF2-HMAC-SHA512
```

> *Answer: 1750*

You can find me on:
LinkedIn:- https://www.linkedin.com/in/shamsher-khan-651a35162/
Twitter:- https://twitter.com/shamsherkhannn
Tryhackme:- https://tryhackme.com/p/Shamsher

For more walkthroughs stay tuned…
Before you go…

Visit my other walkthrough's:-

and thank you for taking the time to read my walkthrough.
If you found it helpful, please hit the 👏 button 👏 (up to 40x) and share it to help others with similar interests! + Feedback is always welcome!

Hashing Cryptography Tryhackme Tryhackme Walkthrough Tryhackme Writeup

Follow

Written by Shamsher khan

335 Followers · 5 Following

Web Application Pen-tester || CTF Player || Security Analyst || Freelance Cyber Security Trainer

Responses (1)

What are your thoughts?

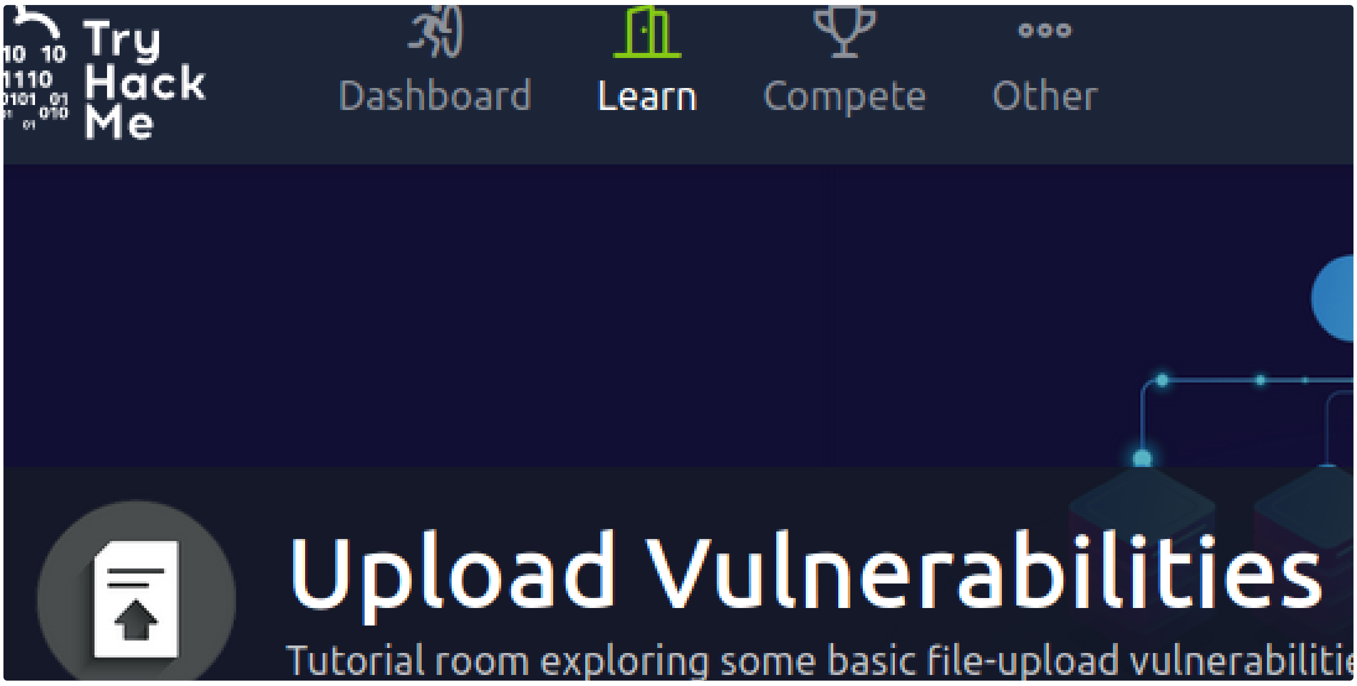
Respond

Muhmmad Aslam
about 2 years ago

Thank you!

Reply

More from Shamsher khan

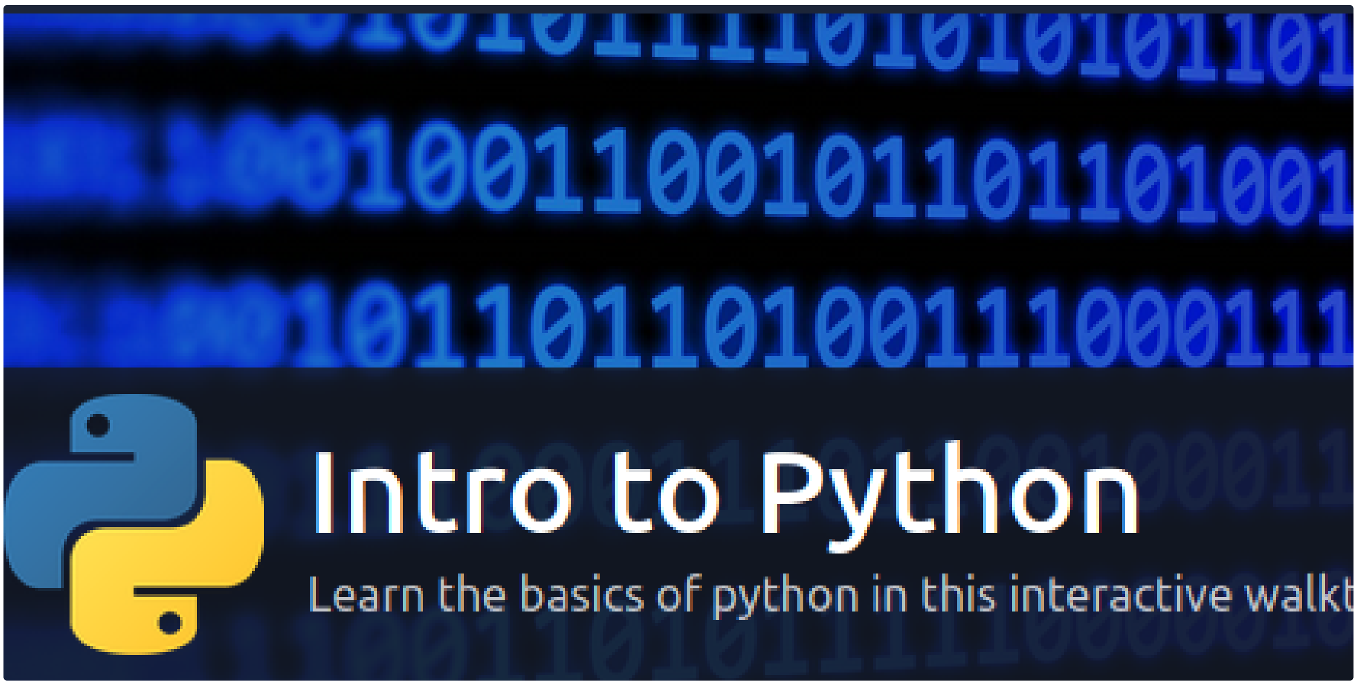

Try Hack Me
Dashboard
Learn
Compete
Other
Upload Vulnerabilities
Tutorial room exploring some basic file-upload vulnerabilitie

In InfoSec Write-ups by Shamsher khan

Upload Vulnerabilities TryHackme Writeup

This is a Writeup of Tryhackme room “Upload Vulnerabilities”

May 5, 2021 372 3



Shamsher khan

Intro to Python TryHackme

By Shamsher khna This is a Writeup of Tryhackme room “Intro to Python”

May 22, 2021 349 16

Shamsher khan

Linux Strength Training Tryhackme Writeup

By Shamsher khan This is a Writeup of Tryhackme room “Linux Strength Training”

May 8, 2021 15

730
Linux PrivEsc
Practice your Linux Privilege Escalation skills
Credentials: user:password321

In InfoSec Write-ups by Shamsher khan

Linux PrivEsc Tryhackme Writeup

By Shamsher khan This is a Writeup of Tryhackme room "JLinux PrivEsc"

Apr 20, 2021 105

See all from Shamsher khan

Recommended from Medium

Open in app ↗

In T3CH by Axoloth

TryHackMe | Training Impact on Teams | WriteUp

Discover the impact of training on teams and organisations

Nov 5, 2024 60

embossdotar

TryHackMe—Hashing Basics—Writeup

Key points: Hashing | Hashing functions | File integrity checking | Hashcat | John the Ripper | Rainbow table | Password cracking. Hashing...

Oct 23, 2024 2

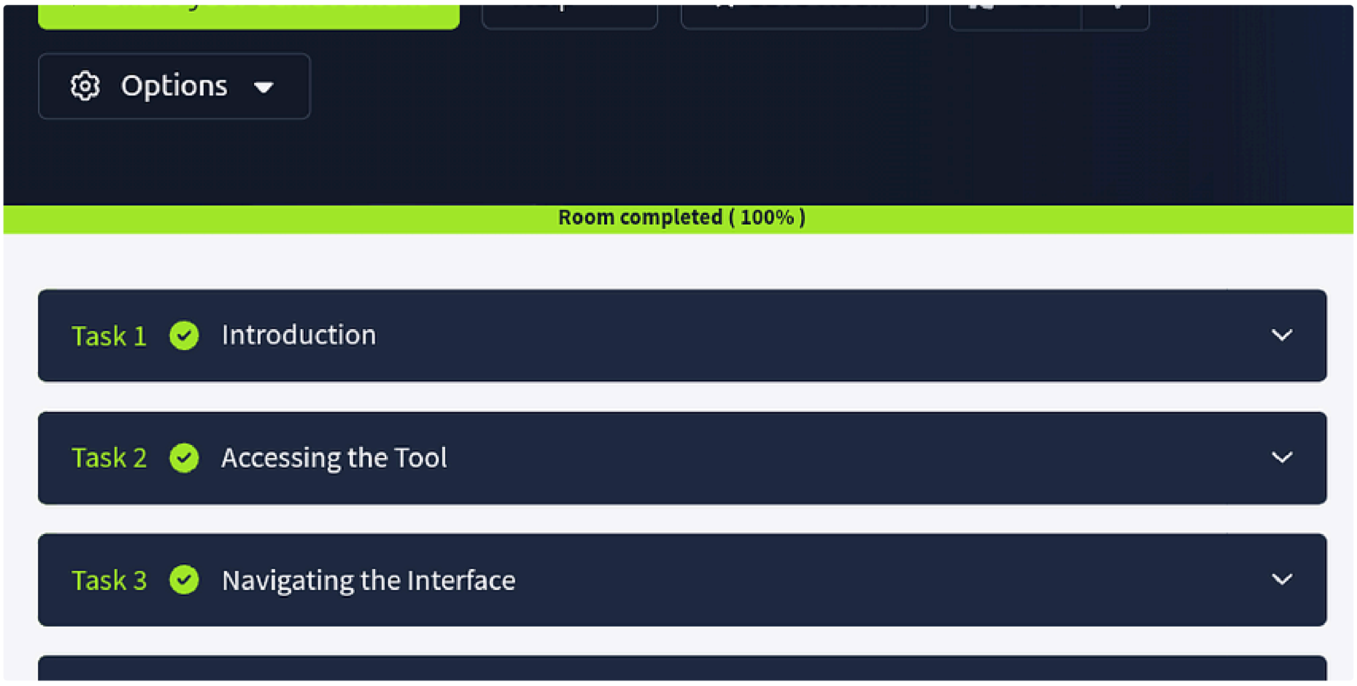
Lists

General Coding Knowledge

20 stories · 1841 saves

Staff picks

791 stories · 1541 saves

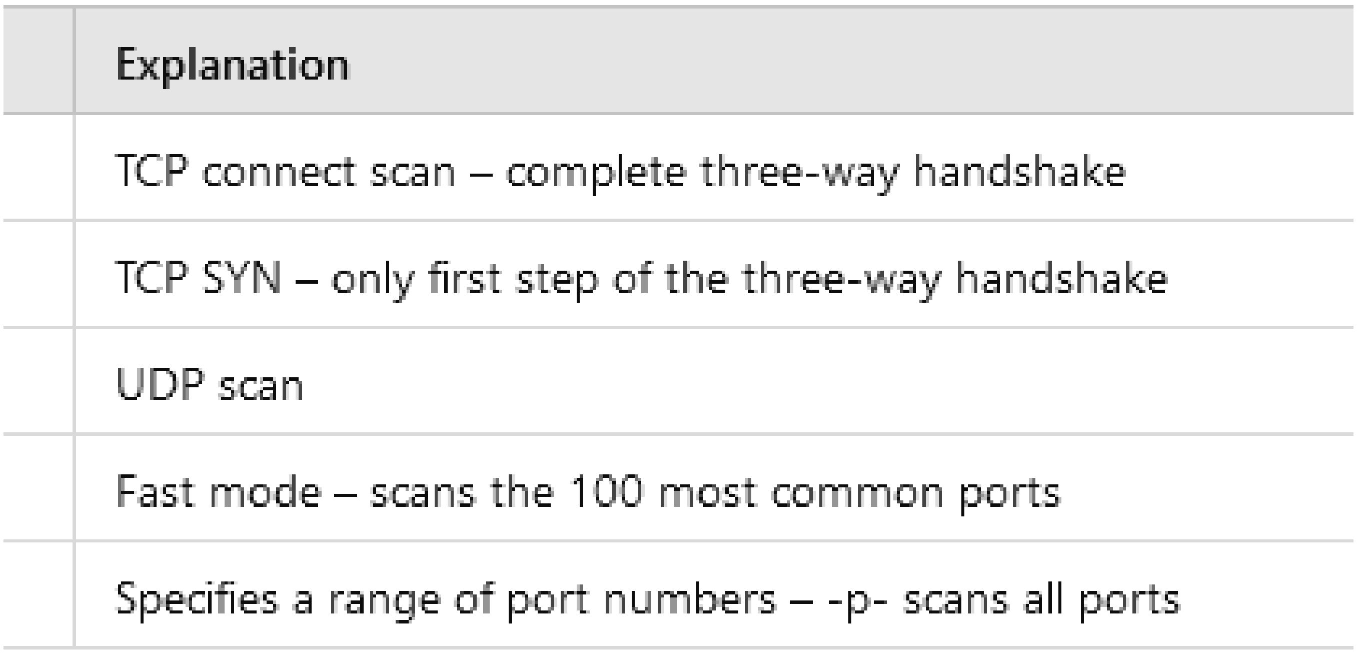


Jawstar

CyberChef: The Basics Tryhackme Write up

Tryhackme

Nov 7, 2024 8



	Explanation
	TCP connect scan – complete three-way handshake
	TCP SYN – only first step of the three-way handshake
	UDP scan
	Fast mode – scans the 100 most common ports
	Specifies a range of port numbers – -p- scans all ports

rutbar

TryHackMe—Nmap: The Basics | Cyber Security 101 (THM)

Host Discovery: Who Is Online Scenario: This task focuses on using Nmap to discover live hosts.

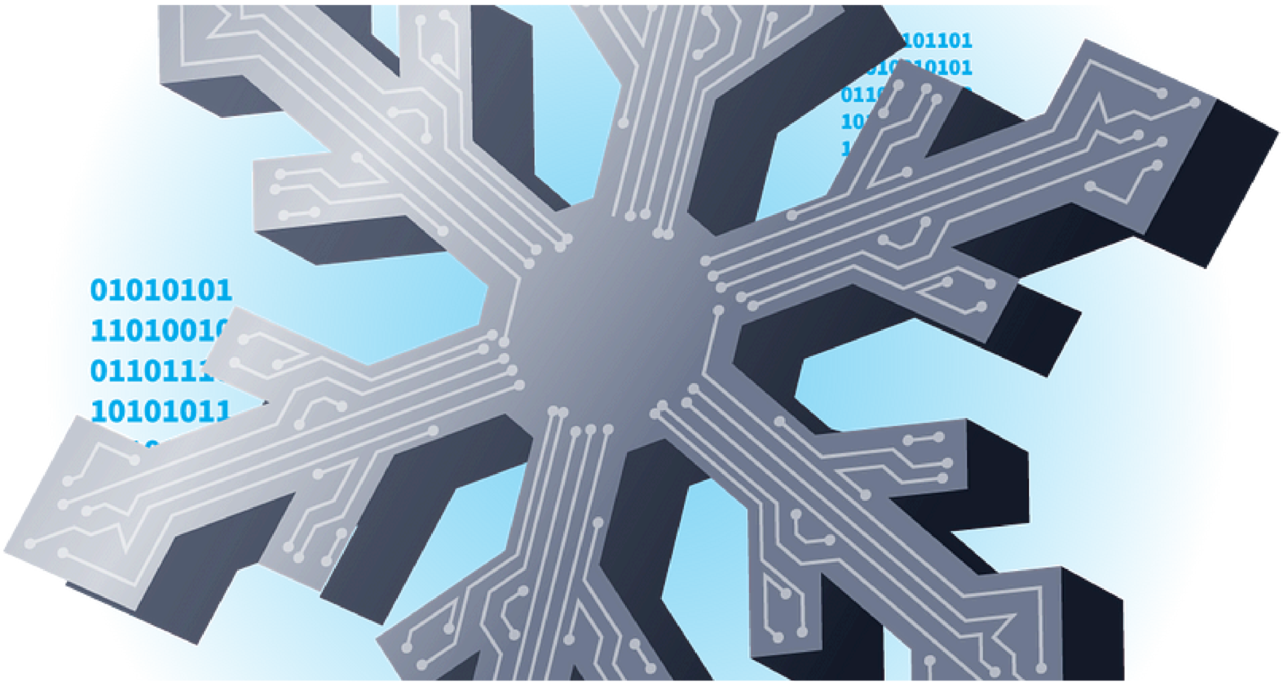
Oct 23, 2024 1 1

rutbar

TryHackMe—Hashing Basics | Cyber Security 101 (THM)

Hash Functions

Oct 26, 2024 1



Z3pH7

TryHackMe—Hashing Basics | Cyber Security 101 (THM)

Hey everyone! TryHackMe just announced the NEW Cyber Security 101 learning path, and there are tons of giveaways this time! This article...

Oct 29, 2024 101 1

See more recommendations