Cryptography 101

in this module , i have notes and my learning from tryhackme's room Cryptography 101.

Hashing : Crypto 101

Some basic terminology :

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

Hash Functions ??

Hash is quite different from encryption

→ there is no key
→ impossible to go from output back to input

what does hash function do ?

it takes input data of any size and create a summary/digest of the data .

→ output is of fixed size , and hard to predict output from input and vice versa .

what does a good hashing algorithm ?

→ it is fast to compute and slow to reverse .
→ single change in input should cause a large change in output .

in cyber security hashing is most often used in verifying our passwords .

what is hash collision ?

basically when two different input gives same output , it is a hash collision .

collisions are not avoidable , because of pigeonhole principle.

The pigeonhole principle: “**If you put three pigeons in two pigeonholes, at least two of the pigeons end up in the same hole** ,” is an obvious yet fundamental principle of nature as it captures the very essence of counting.

basically there are set of numbers of different output values , but inputs are way too much that we can give , that leads to pigeon-hole effect

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.

Uses of hashing

for password verification :

so web-apps , basically website that need user to be logged in , uses hashing , why?

because if passwords are stored in plain-text on server , if the server gets breached all the passwords are leaked so , we hash password , so that hacker cannot directly just get passwords and use them .

we cannot encrypt passwords , if someone gets the key he can easily decrypt all the passwords .

so if hashed passwords are stored on the database it is safe from being directly usable to anyone until cracked .

rainbow tables are the place where we can find hashes with their plaintext pair . and we can lookup into it to find passwords that are hashed .

so to protect password from these rainbow tables , we can add “salt” to the passwords , the salt is randomly generated and stored in database.

salt is added to the start or the end of the password , before it is hashed , hash functions like bcrypt and sha512crypt handle this task automatically

Recognising Password Hashes .

automatic tools can be used to recognize what type of hash is it but they can be a bit messy and give wrong results .

python tool hashID can be used : https://pypi.org/project/hashID/

so lets see how to identify hashes manually ,

in web apps hashes are mostly NTLM and md5.

unix passwords hashes have these prefix that make it easy to recognize :

The standard format is `$format$rounds$salt$hash`.

windows password are hashed using NTLM , that is a variant of md4 , these are visually identical to md4 and md5 hashes .

on linux password hashes are in /etc/shadow file

in windows passwords are stored in SAM database.

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.

Password Cracking

basically we crack passwords by hashing a different number on input with either salt or salt-less and then compare that output to our hashes and see if it matches , if matched it means we cracked it .

using GPU's for cracking ?

basically GPU's have thousand of cores that are very good for these types of calculations.

cracking on VM's ?

don't ever crack passwords on virtual machines as in virtual machines , graphics card cannot be used and cracking can be extremely slow and may even lead to false positives if --force tag is used with hashcat , always use host machine to crack hashes .

Hashing : Integrity Checking

Integrity Checking With Hashes :

hashing can be used to check if any file has been modified .

if we put same data in then we get the same data out , but if the data is changed the hash changes dramatically .

we can use that to verify if the file has been downloaded correctly or any file has been modified or not .

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 hasnâ€™t been modified or corrupted (integrity). They use a secret key, and a hashing algorithm in order to produce a hash.

John The Ripper : Hash Cracking

John the Ripper is one of the most well known, well-loved and versatile hash cracking tools out there. It combines a fast cracking speed, with an extraordinary range of compatible hash types.

there will be several hashes and task i will solve and show in this document :

so to create a baseline here , the hashes cannot be reversed but can be verified , verified in sense means that we use the same hashing algorithm run it against a bunch of words , preferably known as a wordlist and verify that we got the same hash in question , if the both hashes match , it means we got the password , we will crack different algorithm of hashes now .

for the task shown below i will use rockyou.txt the infamous wordlist from spaceX .

Cracking Basic Hashes

so we will be doing two tasks in every single problem here , that is to

→ first look at the hash and identify the algorithm used for hashing and

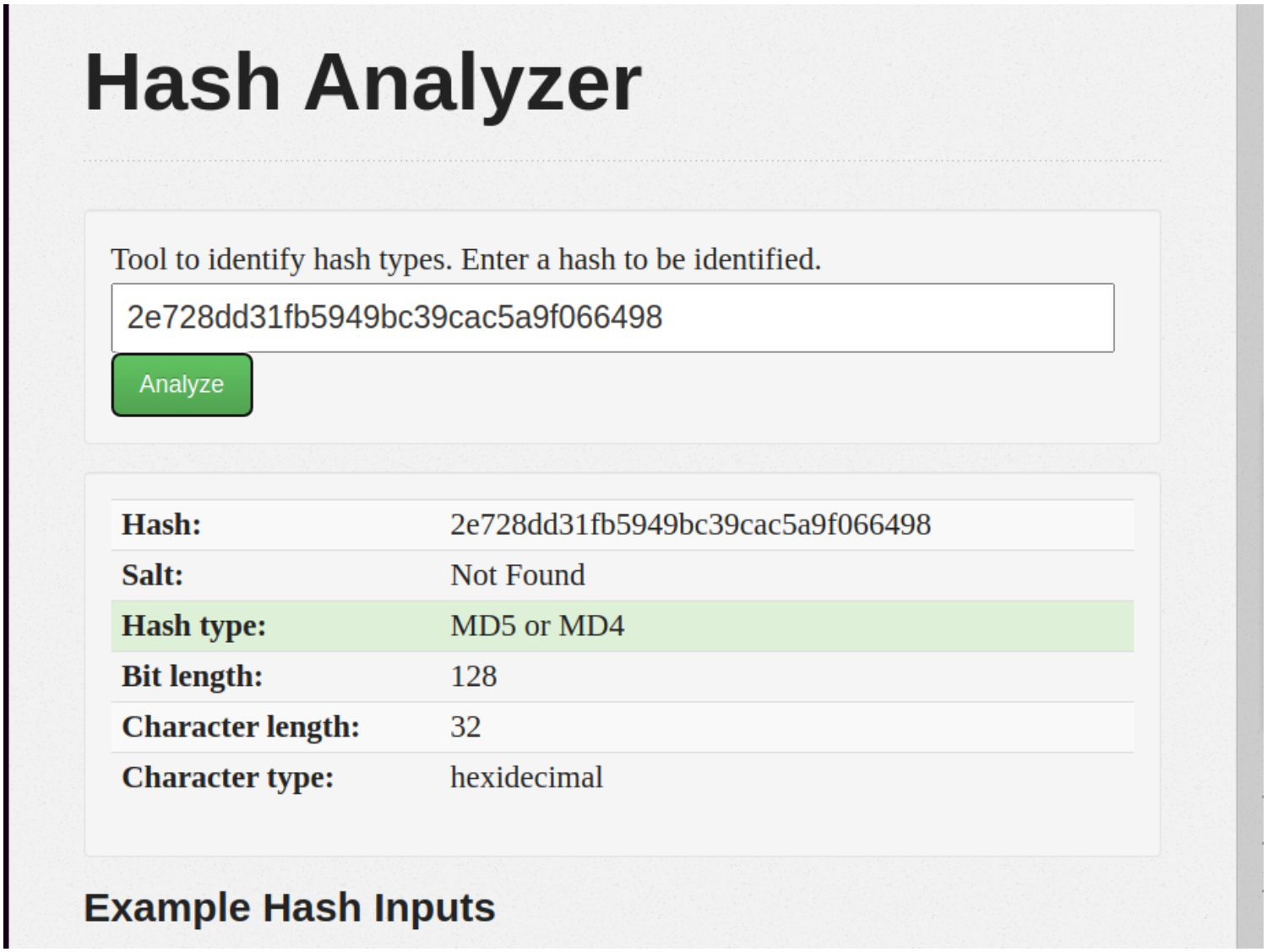
→ then using john to crack the hash with rockyou wordlist.

lets move to Task 1 :

there are four hashes in task 1 , lets identify and crack them one by one

→ hash1.txt :

identifying hash using online tools as hash-id gave a lot of results :



cracking the hash1.txt :

```
┌──(root㉿kali)-[/home/kali/Downloads/first_task_hashes]
└─# john --wordlist=/usr/share/wordlists/rockyou.txt hash1.txt --format=raw-md5
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-MD5 [MD5 256/256 AVX2 8×3])
Warning: no OpenMP support for this hash type, consider --fork=8
Press 'q' or Ctrl-C to abort, almost any other key for status
biscuit          (?)
1g 0:00:00:00 DONE (2022-06-26 04:32) 2.777g/s 7466p/s 7466c/s 7466C/s skyblue..nugget
Use the "--show --format=Raw-MD5" options to display all of the cracked passwords reliably
Session completed.
```

→ hash2.txt :

identifying hash type :

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

1A732667F3917C0F4AA98BB13011B9090C6F8065

Analyze

Hash:	1A732667F3917C0F4AA98BB13011B9090C6F8065
Salt:	Not Found
Hash type:	SHA1 (or SHA 128)
Bit length:	160
Character length:	40
Character type:	hexidecimal

cracking the hash :

```
┌──(root㉿kali)-[/home/kali/Downloads/first_task_hashes]
└─# john --wordlist=/usr/share/wordlists/rockyou.txt hash2.txt --format=raw-sha1
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-SHA1 [SHA1 256/256 AVX2 8x])
Warning: no OpenMP support for this hash type, consider --fork=8
Press 'q' or Ctrl-C to abort, almost any other key for status
kangeroo         (?)
1g 0:00:00:00 DONE (2022-06-26 04:36) 50.00g/s 5857Kp/s 5857Kc/s 5857KC/s karate2..kalvin1
Use the "--show --format=Raw-SHA1" options to display all of the cracked passwords reliably
Session completed.
```

→ hash3.txt

identifying hash type :

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

D7F4D3CCEE7ACD3DD7FAD3AC2BE2AAE9C44F4E9B7FB802D73

Analyze

Hash:	D7F4D3CCEE7ACD3DD7FAD3AC2BE2AAE9C4 4F4E9B7FB802D73136D4C53920140A
Salt:	Not Found
Hash type:	SHA2-256
Bit length:	256
Character length:	64
Character type:	hexidecimal

cracking the hash :

```
┌──(root㉿kali)-[/home/kali/Downloads/first_task_hashes]
└─# john --wordlist=/usr/share/wordlists/rockyou.txt hash3.txt --format=raw-sha256
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-SHA256 [SHA256 256/256 AVX2 8x])
Warning: poor OpenMP scalability for this hash type, consider --fork=8
Will run 8 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
microphone       (?)
1g 0:00:00:00 DONE (2022-06-26 04:41) 50.00g/s 6553Kp/s 6553Kc/s 6553KC/s 123456..kovacs
Use the "--show --format=Raw-SHA256" options to display all of the cracked passwords reliably
Session completed.
```

hash4.txt :

identifying hash type :

```
┌──(root㉿kali)-[/home/kali/Downloads/first_task_hashes]
└─# hashid 'c5a60cc6bbba781c601c5402755ae1044bbf45b78d1183cbf2ca1c865b6c792cf3c6b87791344986c8a832a0f9ca8d0b4afd3d9421a149d57075e1b4e93f90bf' -j
Analyzing 'c5a60cc6bbba781c601c5402755ae1044bbf45b78d1183cbf2ca1c865b6c792cf3c6b87791344986c8a832a0f9ca8d0b4afd3d9421a149d57075e1b4e93f90bf'
[+] SHA-512 [JtR Format: raw-sha512]
[+] Whirlpool [JtR Format: whirlpool]
```

it was not sha-512 but it was whirlpool instead .

cracking the hash :

```
┌──(root㉿kali)-[/home/kali/Downloads/first_task_hashes]
└─# john --wordlist=/usr/share/wordlists/rockyou.txt hash4.txt --format=whirlpool
Using default input encoding: UTF-8
Loaded 1 password hash (whirlpool [WHIRLPOOL 32/64])
Warning: poor OpenMP scalability for this hash type, consider --fork=8
Will run 8 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
colossal         (?)
1g 0:00:00:00 DONE (2022-06-26 04:45) 14.28g/s 9830Kp/s 9830Kc/s 9830KC/s davita1..blah2007
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

Cracking Windows Authentication Hashes

so windows used LM hashing algorithm to hash its user password that are then stored in windows SAM database , we can dump those hashes in a penetration test and pass the hash , or can crack those hashes in john for further lateral and upward privilege escalation , now windows uses NTLM , new technology LM to hash its password , we will crack the given hash :

```
┌──(root㉿kali)-[/home/kali/Downloads]
└─# john --wordlist=/usr/share/wordlists/rockyou.txt ntlm.txt --format=NT
Using default input encoding: UTF-8
Loaded 1 password hash (NT [MD4 256/256 AVX2 8×3])
Warning: no OpenMP support for this hash type, consider --fork=8
Press 'q' or Ctrl-C to abort, almost any other key for status
mushroom         (?)
1g 0:00:00:00 DONE (2022-06-26 04:53) 100.0g/s 307200p/s 307200c/s 307200C/s skater1..dangerous
Use the "--show --format=NT" options to display all of the cracked passwords reliably
Session completed.
```

Cracking /etc/shadow hashes

so in linux password hashes are stored in /etc/shadow file , but to crack these john needs a unshadowed file which is basically the mixture of /etc/passwd (user file) and /etc/shadow (hash file)

/etc/shadow file is only readable by root user (keep that in mind)

now lets see the task file given :

```
┌──(root㉿kali)-[/home/kali/Downloads]
└─# cat etchashes.txt
This is everything I managed to recover from the target machine before my computer crashed … See if you can crack the hash so we can at least salvage a pass
word to try and get back in.

root:x:0:0::/root:/bin/bash
root:$6$Ha.d5nGupBm29pYr$yugXSk24ZljLTAZZagtGwpSQhb3F2DOJtnHrvk7HI2ma4GsuioHp8sm3LJiRJpKfIf7lZQ29qgtH17Q/JDpYM/:18576::::::
```

so we have been given a part of /etc/passwd and other part of /etc/shadow file , we will create 2 separate files from it :

```
┌──(root㉿kali)-[/home/kali/Downloads]
└─# nano passwd.txt

┌──(root㉿kali)-[/home/kali/Downloads]
└─# nano shadow.txt

┌──(root㉿kali)-[/home/kali/Downloads]
└─# cat passwd.txt
root:x:0:0::/root:/bin/bash

┌──(root㉿kali)-[/home/kali/Downloads]
└─# cat shadow.txt
root:$6$Ha.d5nGupBm29pYr$yugXSk24ZljLTAZZagtGwpSQhb3F2DOJtnHrvk7HI2ma4GsuioHp8sm3LJiRJpKfIf7lZQ29qgtH17Q/JDpYM/:18576::::::
```

now we will merge / unshadow both files using unshadow tool :

```
┌──(root㉿kali)-[/home/kali/Downloads]
└─# unshadow passwd.txt shadow.txt > final_hash.txt

┌──(root㉿kali)-[/home/kali/Downloads]
└─# cat final_hash.txt
root:$6$Ha.d5nGupBm29pYr$yugXSk24ZljLTAZZagtGwpSQhb3F2DOJtnHrvk7HI2ma4GsuioHp8sm3LJiRJpKfIf7lZQ29qgtH17Q/JDpYM/:0:0::/root:/bin/bash
```

now we will not need to provide john any format to crack this as it is meant to be cracked only by john but incase it asks use “sha512crypt”

cracking final_hash.txt :

```
┌──(root㉿kali)-[/home/kali/Downloads]
└─# john --wordlist=/usr/share/wordlists/rockyou.txt final_hash.txt
Using default input encoding: UTF-8
Loaded 1 password hash (sha512crypt, crypt(3) $6$ [SHA512 256/256 AVX2 4x])
Cost 1 (iteration count) is 5000 for all loaded hashes
Will run 8 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
1234             (root)
1g 0:00:00:00 DONE (2022-06-26 05:04) 2.083g/s 4266p/s 4266c/s 4266C/s kucing..lovers1
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

Single Crack Mode

so you can also crack password without a wordlist , whatt?

so there is a single crack mode in john that allows it to crack hashes without using a wordlist ,

so what happens here is that , john uses the username and so some word mangling and manipulations in the username and tries it as a password , it is used to crack poor password based off from information about username .

so there are 2 methods in this :

Word Mangling

The best way to show what Single Crack mode is, and what word mangling is, is to actually go through an example:
If we take the username: Markus
Some possible passwords could be:

• Markus1, Markus2, Markus3 (etc.)
• MArkus, MARkus, MARKus (etc.)
• Markus!, Markus$, Markus* (etc.)

This technique is called word mangling. John is building it's own dictionary based on the information that it has been fed and uses a set of rules called "mangling rules" which define how it can mutate the word it started with to generate a wordlist based off of relevant factors for the target you're trying to crack. This is exploiting how poor passwords can be based off of information about the username, or the service they're logging into.

then there is :

GECOS :

basically it takes information from /etc/passwd file separated using columns such as username and home directory name and try word mangling with those to create a dictionary and crack those passwords .

these fields separated by colons are called GECOS fields .

so ,

to use single cracking mode use '--single' mode .

a note on single cracking mode , to use this we must provide a username with the hash separated by a colon ,

for example :

from this :

1efee03cdcb96d90ad48ccc7b8666033

to this :

mike:1efee03cdcb96d90ad48ccc7b8666033

lets see it in action :

there is a hash given to us :

```
┌──(root㉿kali)-[/home/kali/Downloads]
└─# cat hash7.txt
7bf6d9bb82bed1302f331fc6b816aada
```

now in the task username is given as 'Joker'

first edit the file like this : (using nano or vim)

```
┌──(root㉿kali)-[/home/kali/Downloads]
└─# cat hash7.txt
Joker:7bf6d9bb82bed1302f331fc6b816aada
```

then identifying hash type :

Hash Analyzer

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

7bf6d9bb82bed1302f331fc6b816aada

Analyze

Hash:	7bf6d9bb82bed1302f331fc6b816aada
Salt:	Not Found
Hash type:	MD5 or MD4
Bit length:	128
Character length:	32
Character type:	hexidecimal

cracking the hash :

```
┌──(root㉿kali)-[/home/kali/Downloads]
└─# john --single hash7.txt --format=raw-md5
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-MD5 [MD5 256/256 AVX2 8×3])
Warning: no OpenMP support for this hash type, consider --fork=8
Press 'q' or Ctrl-C to abort, almost any other key for status
Warning: Only 2 candidates buffered for the current salt, minimum 24 needed for performance.
Warning: Only 21 candidates buffered for the current salt, minimum 24 needed for performance.
Jok3r            (Joker)
1g 0:00:00:00 DONE (2022-06-26 05:25) 100.0g/s 19600p/s 19600c/s 19600C/s j0ker..J0k3r
Use the "--show --format=Raw-MD5" options to display all of the cracked passwords reliably
Session completed.
```

Custom Rules

okay so if you are using a wordlist , you can add custom rules to the wordlist , so that the attack can be even more precise , for example :

Many organisations will require a certain level of password complexity to try and combat dictionary attacks, meaning that if you create an account somewhere, go to create a password and enter:

polopassword

You may receive a prompt telling you that passwords have to contain at least one of the following:
• Capital letter
• Number
• Symbol

This is good! However, we can exploit the fact that most users will be predictable in the location of these symbols. For the above criteria, many users will use something like the following:

Polopassword1!

to apply these rules to our wordlist we can use regex and stuff to create a rule in john and use it .

we can add a rule in /etc/john/john.conf file . (in kali)

The first line:
[List.Rules:THMRules] - Is used to define the name of your rule, this is what you will use to call your custom rule as a John argument.

We then use a regex style pattern match to define where in the word will be modified, again- we will only cover the basic and most common modifiers here:
Az - Takes the word and appends it with the characters you define

A0 - Takes the word and prepends it with the characters you define

c - Capitalises the character positionally

Lastly, we then need to define what characters should be appended, prepended or otherwise included, we do this by adding character sets in square brackets [ ] in the order they should be used. These directly follow the modifier patterns inside of double quotes " ". Here are some common examples:

[0-9] - Will include numbers 0-9

[0] - Will include only the number 0

[A-z] - Will include both upper and lowercase

[A-Z] - Will include only uppercase letters

[a-z] - Will include only lowercase letters

[a] - Will include only a

[!£$%@] - Will include the symbols !£$%@

Putting this all together, in order to generate a wordlist from the rules that would match the example password "Polopassword1!" (assuming the word polopassword was in our wordlist) we would create a rule entry that looks like this:
[List.Rules:PoloPassword]
cAz"[0-9] [!£$%@]"

In order to:
Capitalise the first letter - c
Append to the end of the word - Az
A number in the range 0-9 - [0-9]
Followed by a symbol that is one of [!£$%@]

Using Custom Rules
We could then call this custom rule as a John argument using the --rule=PoloPassword flag.

As a full command: john --wordlist=[path to wordlist] --rule=PoloPassword [path to file]

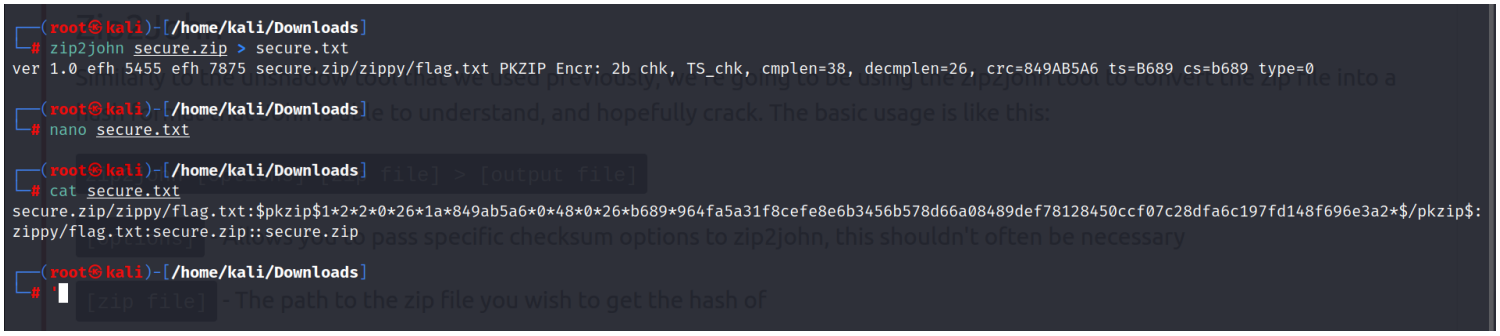
As a note I find it helpful to talk out the patterns if you're writing a rule- as shown above, the same applies to writing RegEx patterns too.

Jumbo John already comes with a large list of custom rules, which contain modifiers for use almost all cases. If you get stuck, try looking at those rules [around line 678] if your syntax isn't working properly.

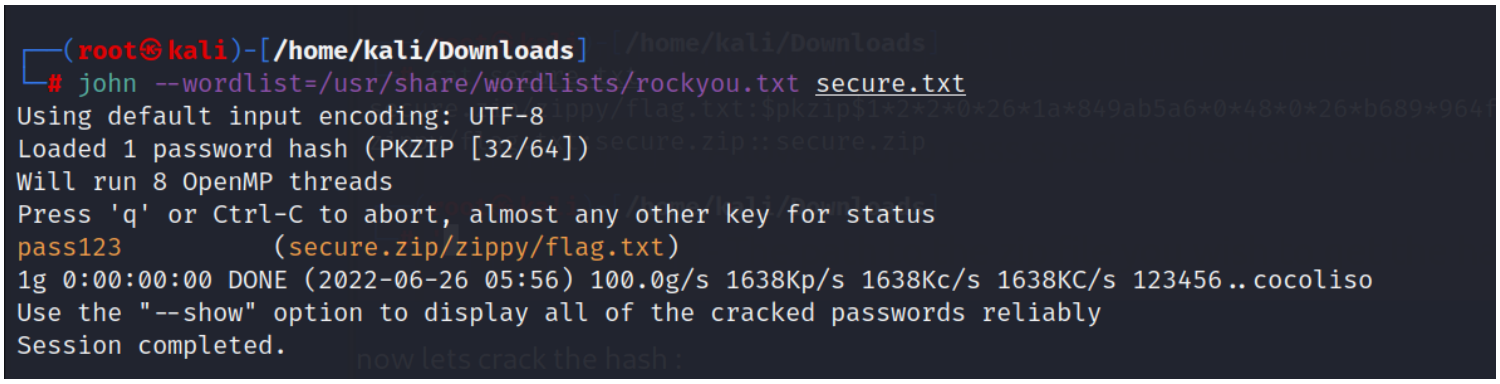
Cracking Password Protected Zip Files

so , we can also crack zip files password using john , similar to unshadow tool , here we will use Zip2John tool to take the hash out of zip and crack that hash .

so in this task we have a secure.zip file , lets extract the hash from it :



now lets crack the hash :



flag :

```
┌──(root㉿kali)-[/home/kali/Downloads/zippy]
└─# cat flag.txt
THM{w3ll_d0n3_h4sh_r0y4l}
```

Cracking Password Protected .rar file

in above task we learned to crack zip files , here it is the same process , but we will use rar2john tool , otherwise its same .

using rar2john tool :

```
┌──(root㉿kali)-[/home/kali/Downloads]
└─# rar2john secure.rar > rar.txt

┌──(root㉿kali)-[/home/kali/Downloads]
└─# cat rar.txt
secure.rar:$rar5$16$b7b0ffc959b2bc55ffb712fc0293159b$15$4f7de6eb8d17078f4b3c0ce650de32ff$8$ebd10bb79dbfb9f8
```

cracking the rar password :

```
┌──(root㉿kali)-[/home/kali/Downloads]
└─# john --wordlist=/usr/share/wordlists/rockyou.txt rar.txt
Using default input encoding: UTF-8
Loaded 1 password hash (RAR5 [PBKDF2-SHA256 256/256 AVX2 8x])
Cost 1 (iteration count) is 32768 for all loaded hashes
Will run 8 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
password         (secure.rar)
1g 0:00:00:00 DONE (2022-06-26 06:01) 4.000g/s 1024p/s 1024c/s 1024C/s 123456..freedom
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

flag :

```
┌──(root㉿kali)-[/home/kali/Downloads]
└─# cat flag.txt
THM{r4r_4rch1ve5_th15_t1m3}
```

Cracking SSH Private Key

while logging into SSH we can either use a password to login or we can use a password ,

but in certain scenarios we might prefer to use password to perform tasks after logging in , so in this task we will crack a private ssh key

first we have to process the key through ssh2john :

```
┌──(root㉿kali)-[/home/kali/Downloads]
└─# locate ssh2john
/usr/share/john/ssh2john.py
/usr/share/john/__pycache__/ssh2john.cpython-39.pyc

┌──(root㉿kali)-[/home/kali/Downloads]
└─# python3 /usr/share/john/ssh2john.py idrsa.id_rsa > ssh.txt

┌──(root㉿kali)-[/home/kali/Downloads]
└─# cat ssh.txt
idrsa.id_rsa:$sshng$1$16$3A98F468854BB3836BF689310D864CE9$1200$08ca19b68bc606b07875701174131b9220d23ef968befc1230eeff0d7c0f904e6734765fe562e8671972e409091f32
c80b754ab248976228a5f2c38e8ac63572d7452e75669aeda932275989ce4c077d43287ed227b8f9053e53f2b1c9bb9dfe876378a32e87e7be4e91a845ae8ee4073bf7ac5aad8414253c97cfb73b0
83107712907da8c704678f46d0b006f7a77b13a04305a988c8e17d83abd2449ed5c3defc8203d7c5f70cef3470b0bbe3fa5a2e957ac55a57ea08b1de4d3fa5436c6160a14b461ac7bc4a3052ddf85
8de657ecb210989507beb96f7219ac3c3790e89f3af71f7f61ebe23570284a482b1504b067fb1e03ed62201c6db71dab65e5f1577751ddb006fe14ceed4525965fce19f8141373094d1aedbb58cb9
03f58f6d80695be0382c31e61baaf366d4f2e722316e91ff4dcb3df15702008b5be3c0b2a81b3f452ef3257c425dd26119324b4de3652e90b91afd87ca2bc41c70abd0d97557d4037952b63c0a0d7
c7ab6ed538c3d76bdf488683213e8d8e897ab51c4990b137d04e5044ccbf8cadbdce9eeec5e50f3d487b1f21e86a2b2785caedbf9503d2d8585b2138d82b35e70d1da03c9c574962cdb6e4d2de761
a594ab8c082d88b43a027649012feb28b6a022c0ab49cf05e8b91e36bda935f188c1bb05925da2168dd15af917ba20a8532010892853da5cb1a8ff80cc5d3aa1dd3fe66543bf14d9b44d082261fd6
1976718bb5eea1d911ddb7fb0cc0505b39cec36ef7bd8e8d9d826eda5f7e1a5a51067ead2f78cf69f85de97be5a8f371174356788554b6bf134072b93bf6728ec26fe19c2485be9e7428208a66cc1
e79329ac16f3034605c63550a424ed8cac39f965b6ffe83240c6709607eaef99b189100ef33e000b4195e07ec5c67bdaf2ca1acbd08327f0c4dcfae322883f7be964cb22393541e883c8c5b748237
a900aab709b6286cea66a214a9fe4e3a1203f999fd995aa049767355e2658828c4a82d58ca15343f0abe6b2779e880ed2682b4730103a84a3410e6c822098d82b04d665b8bf98bc3b69cae0c8d8c9
d140dc99056279d5f330bc439bfdceaf38a56fd1362ce78e96deb49a9f6756ec9b64eeba8f4725ec056ab206e37823d052d539d38016abf792858a169cbbe0f6f0d0049c6d49228833aa8ec10ede0
c183ac737e54346949485e5ffc1bc3105e5686c8b1f6fb8cdb14949aa97b833757d02b970e96cb1281c472a5cb26cfa7cfda0be5bd45cf14d4bc28ccd2be4dd09c6a2ce0cf668035d2aa39a8345ea
154543491436bf8f5e605d86e266d40227f48684e696a225877624ddddf0afe05d0aeec29ad28edb0f8cda0f341ddbbd454bd3c238d1c499effa6bf6f796dae0983182f36fae4781552cb8d9426fc
57132c27a735c5365a5d355a4c4f21d5d7ed2ea11bb2ed8561563906735454 18f47359fd1092767f6dfb3321ee14a0043352fdbaa5cb0e75fde2ec5909c0533251f30bd56ad7e34a8a31b009b53c6
4e9f2de9fd57a0f561564e6a961158cc0b54fcfc43046d9641788ac5e25b89bdb7890c4e6532d1bfabd4d49ae7d3740506c4ecb6bc5cb6a12bc24ed2e0f913338c7dfa08ada66a6128e7de56794d1
d2170d6324af0cd72bc8abcff177f0942d9df5d99d48c8f946fd856d9ccb424966835aa06c94996abcc169aef6f246bbbd7489ec026a
```

cracking the key :

```
┌──(root㉿kali)-[/home/kali/Downloads]
└─# john --wordlist=/usr/share/wordlists/rockyou.txt ssh.txt
Using default input encoding: UTF-8
Loaded 1 password hash (SSH, SSH private key [RSA/DSA/EC/OPENSSH 32/64])
Cost 1 (KDF/cipher [0=MD5/AES 1=MD5/3DES 2=Bcrypt/AES]) is 0 for all loaded hashes
Cost 2 (iteration count) is 1 for all loaded hashes
Will run 8 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
mango            (idrsa.id_rsa)
1g 0:00:00:00 DONE (2022-06-26 06:08) 100.0g/s 435200p/s 435200c/s 435200C/s mango..pearl
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

Encryption : Crypto 101

here we will cover :

- Why cryptography matters for security and CTFs
- The two main classes of cryptography and their uses
- RSA, and some of the uses of RSA
- 2 methods of Key Exchange
- Notes about the future of encryption with the rise of Quantum Computing

Key Terms

Ciphertext - The result of encrypting a plaintext, encrypted data

Cipher - A method of encrypting or decrypting data. Modern ciphers are cryptographic, but there are many non cryptographic ciphers like Caesar.

Plaintext - Data before encryption, often text but not always. Could be a photograph or other file

Encryption - Transforming data into ciphertext, using a cipher.

Encoding - NOT a form of encryption, just a form of data representation like base64. Immediately reversible.

Key - Some information that is needed to correctly decrypt the ciphertext and obtain the plaintext.

Passphrase - Separate to the key, a passphrase is similar to a password and used to protect a key.

Asymmetric encryption - Uses different keys to encrypt and decrypt.

Symmetric encryption - Uses the same key to encrypt and decrypt

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

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

Alice and Bob - Used to represent 2 people who generally want to communicate. They're named Alice and Bob because this gives them the initials A and B. https://en.wikipedia.org/wiki/Alice_and_Bob for more information, as these extend through the alphabet to represent many different people involved in communication.

Why is Encryption important

Encryption is important to protect confidentiality, ensure integrity, ensure authenticity of data .

webserver uses HTTPS over encryption .

SSH uses encryption so that no one can snoop on our session

When you connect to your bank, there‚Äôs a certificate that uses cryptography to prove that it is actually your bank rather than a hacker.

When you download a file, how do you check if it downloaded right? You can use cryptography here to verify a checksum of the data.

Crucial Crypto Maths

In cryptography modulo operator is very crucial , basically when we divide X and Y and there is a remainder , modulus give us that remainder .

example 26 divided by 5 will give a remainder of 1 , and modulo will give the result as 1 as it gives the remainder .

example using python :

```
┌──(root㉿kali)-[/home/kali/Downloads]
└─# python3
Python 3.9.12 (main, Mar 24 2022, 13:02:21)
[GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 30 % 5
0
>>> 25 % 7
4
>>> 118613842 % 9091
3565
```

Types Of Encryption

So, there are two types of encryption .

Symmetric Encryption :

→ here same key is used for encryption as well as decryption
→ examples are AES and DES (broken)
→ smaller keys 128 or 256 bit key for AES and 56 bit for DES

Asymmetric Encryption :

→ there is a pair of keys i.e 2 keys
→ one to encrypt and other pair to decrypt
→ examples can be RSA and Elliptic Curve Cryptography
→ these keys are called public and private keys
→ Data encrypted with the private key can be decrypted with the public key, and vice versa.
→ Private needs to be kept private always
→ asymmetric encryption is slower and keys are also large 2048 and 4096 bit keys

RSA - Rivest Shamir Adleman

on the math side of things , RSA depends on mathematically difficult problem of working out large numbers .

example :

It's very quick to multiply two prime numbers together, say 17*23 = 391, but it's quite difficult to work out what two prime numbers multiply together to make 14351 (113x127 for reference).

on the attacking side of things :

The maths behind RSA seems to come up relatively often in CTFs, normally requiring you to calculate variables or break some encryption based on them. The wikipedia page for RSA seems complicated at first, but will give you almost all of the information you need in order to complete challenges.
There are some excellent tools for defeating RSA challenges in CTFs, and my personal favorite is https://github.com/Ganapati/RsaCtfTool which has worked very well for me.
I've also had some success with https://github.com/ius/rsatool.

The key variables that you need to know about for RSA in CTFs are p, q, m, n, e, d, and c.

p and q are large prime numbers .

n is the product of p and q

public key is referred to as n and e

private key is referred to as n and d

m is used to represent message(plain-text)

and c represents cipher-text (encrypted text)

Using Asymmetric Encryption

so as we know asymmetric keys are big in size , so they are not preffered for faster communications like HTTPS ,

but what we can do is , we can use asymmetric keys to exchange symmetric keys which are fast and can be used for further communication

Digital Signature And Certificates

they are used to prove authenticity of files , to prove who created and modified them .

by using asymmetric cryptography , we can create a signature using private key and verify it using our public key .

Digital signatures and physical signatures have the same value in the UK, legally.

The simplest form of digital signature would be encrypting the document with your private key, and then if someone wanted to verify this signature they would decrypt it with your public key and check if the files match.

Certificates
---

so it is a key use of public key cryptography . linked to digital signatures

certificates are mostly used in HTTPS communication and , it helps our browser to know that we are connecting to original website .

basically all our devices trust Root CA certificate and there is a chain followed by Root CA certificate , as it trusts all other certificates and our devices trust them too .

SSH Authentication

SSH (Secure Shell) is normally authenticated using a username and password .

but , we can also authenticate SSH using private key . this uses a public and private key mechanism to authenticate a user

by default SSH keys are RSA keys , we can use a algorithm of our choice and add a passphrase to encrypt our SSH key using 'ssh-keygen'

how to use that keys ?

these keys are stored in .ssh (hidden directory) inside user's home directory ,

the authorized_keys file is the public key allows to access the server

to use a private key use ‘ssh -i private_key user@host’

and only the owner should be able to read and write ssh private key i.e 'chmod 600 private_key'

we can use ssh to get a better shell , by stealing private key and using them to login .

cracking a ssh private key passphrase :

```
┌──(root㉿kali)-[/home/kali/Downloads]
└─# python3 /usr/share/john/ssh2john.py ssh_private > ssh_private.txt

┌──(root㉿kali)-[/home/kali/Downloads]
└─# john --wordlist=/usr/share/wordlists/rockyou.txt ssh_private.txt
Using default input encoding: UTF-8
Loaded 1 password hash (SSH, SSH private key [RSA/DSA/EC/OPENSSH 32/64])
Cost 1 (KDF/cipher [0=MD5/AES 1=MD5/3DES 2=Bcrypt/AES]) is 0 for all loaded hashes
Cost 2 (iteration count) is 1 for all loaded hashes
Will run 8 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
delicious        (ssh_private)
1g 0:00:00:00 DONE (2022-06-27 04:32) 5.882g/s 23341p/s 23341c/s 23341C/s zamora..tarheels
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

Diffie Hellman Key Exchange

What is Key Exchange?
Key exchange allows 2 people/parties to establish a set of common cryptographic keys without an observer being able to get these keys.

Generally, to establish common symmetric keys.

How does Diffie Hellman Key Exchange work?
Alice and Bob want to talk securely. They want to establish a common key, so they can use symmetric cryptography, but they don’t want to use key exchange with asymmetric cryptography. This is where DH Key Exchange comes in.

Alice and Bob both have secrets that they generate, let’s call these A and B. They also have some common material that’s public, let’s call this C.
We need to make some assumptions. Firstly, whenever we combine secrets/material it’s impossible or very very difficult to separate. Secondly, the order that they're combined in doesn’t matter.

Alice and Bob will combine their secrets with the common material, and form AC and BC. They will then send these to each other, and combine that with their secrets to form two identical keys, both ABC. Now they can use this key to communicate.

PGP, GPG and AES

What is PGP?
PGP stands for Pretty Good Privacy. Itâ€™s a software that implements encryption for encrypting files, performing digital signing and more.

What is GPG?
GnuPG or GPG is an Open Source implementation of PGP from the GNU project. You may need to use GPG to decrypt files in CTFs. With PGP/GPG, private keys can be protected with passphrases in a similar way to SSH private keys. If the key is passphrase protected, you can attempt to crack this passphrase using John The Ripper and gpg2john.

What about AES?
AES, sometimes called Rijndael after its creators, stands for Advanced Encryption Standard. It was a replacement for DES which had short keys and other cryptographic flaws.

The Future : Quantum Computers And Encryption

Quantum computers will soon be a problem for many types of encryption.

Asymmetric and Quantum
While it's unlikely we'll have sufficiently powerful quantum computers until around 2030, once these exist encryption that uses RSA or Elliptical Curve Cryptography will be very fast to break. This is because quantum computers can very efficiently solve the mathematical problems that these algorithms rely on for their strength.

AES/DES and Quantum
AES with 128 bit keys is also likely to be broken by quantum computers in the near future, but 256 bit AES can't be broken as easily. Triple DES is also vulnerable to attacks from quantum computers.

Current Recommendations

The NSA recommends using RSA-3072 or better for asymmetric encryption and AES-256 or better for symmetric encryption. There are several competitions currently running for quantum safe cryptographic algorithms, and it’s likely that we will have a new encryption standard before quantum computers become a threat to RSA and AES.