John The Ripper - The Basics

Basic Terms

What Makes Hashes Secure?

Hashing functions are designed as one-way functions

In computer science, P and NP are two classes of problems that help us understand the efficiency of algorithms:

- **P (Polynomial Time)**: Class P covers the problems whose solution can be found in polynomial time. Consider sorting a list in increasing order. The longer the list, the longer it would take to sort; however, the increase in time is not exponential.
- **NP (Non-deterministic Polynomial Time)**: Problems in the class NP are those for which a given solution can be checked quickly, even though finding the solution itself might be hard. In fact, we don’t know if there is a fast algorithm to find the solution in the first place.

Where John Comes in

If you have the hashed version of a password, for example, and you know the hashing algorithm, you can use that hashing algorithm to hash a large number of words

Setting Up Your System

Installation

Installing on linux

Many Linux distributions have John the Ripper available for installation from their official repositories. For instance, on Fedora Linux, you can install John the Ripper with `sudo dnf install john`, while on Ubuntu, you can install it with `sudo apt install john`

Installing on Windows

To install Jumbo John the Ripper on Windows, you need to download and install the zipped binary for either 64-bit systems here or for 32-bit systems here.

Wordlists

There are many different wordlists out there, and a good collection can be found in the SecLists repository.

Kali Linux distributions, the `/usr/share/wordlists` directory contains a series of great wordlists.

Cracking Basic Hashes

John Basic Syntax

The basic syntax of John the Ripper commands is as follows. We will cover the specific options and modifiers used as we use them.

`john [options] [file path]`

- `john` : Invokes the John the Ripper program
- `[options]` : Specifies the options you want to use
- `[file path]` : The file containing the hash you're trying to crack; if it's in the same directory, you won't need to name a path, just the file.

Automatic Cracking

if you can't identify what hash type you're working with and want to try cracking it, it can be a good option! To do this, we use the following syntax:

`john --wordlist=[path to wordlist] [path to file]`

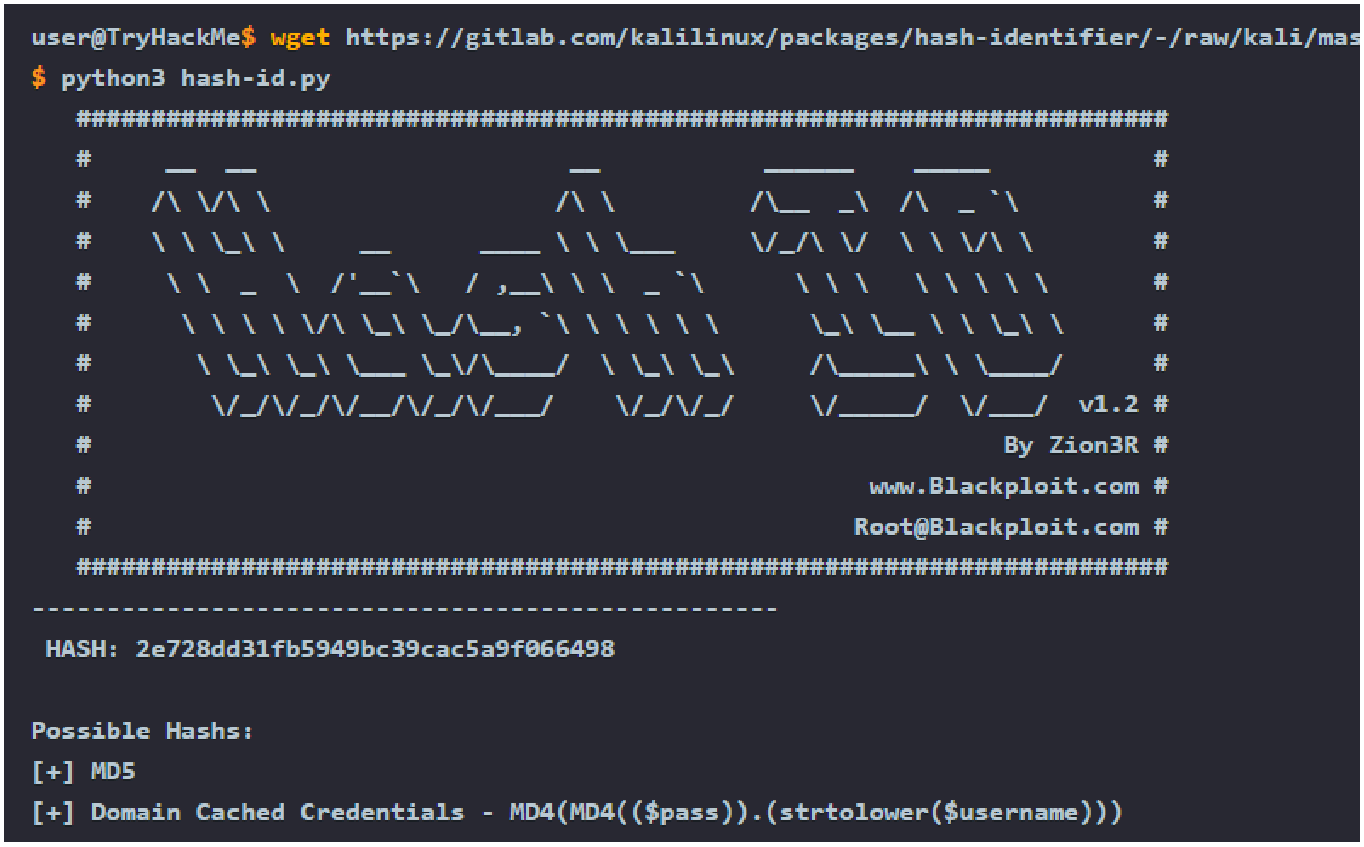
- `--wordlist=` : Specifies using wordlist mode, reading from the file that you supply in the provided path
- `[path to wordlist]` : The path to the wordlist you're using, as described in the previous task

Example Usage:

`john --wordlist=/usr/share/wordlists/rockyou.txt hash_to_crack.txt`

Identifying Hashes

We can use other tools to identify the hash and then set John to a specific format. There are multiple ways to do this, such as using an online hash identifier like this site. I like to use a tool called hash-identifier, a Python tool that is super easy to use and will tell you what different types of hashes the one you enter is likely to be



```
user@TryHackMe$ wget https://gitlab.com/kalilinux/packages/hash-identifier/-/raw/kali/mas
$ python3 hash-id.py
   #########################################################################
   #     __  __                     __           ______    _____           #
   #    /\ \/\ \                   /\ \         /\__  _\  /\  _ `\         #
   #    \ \ \_\ \     __      ____ \ \ \___     \/_/\ \/  \ \ \/\ \        #
   #     \ \  _  \  /'__`\   / ,__\ \ \  _ `\      \ \ \   \ \ \ \ \       #
   #      \ \ \ \ \/\ \_\ \_/\__, `\ \ \ \ \ \      \_\ \__ \ \ \_\ \      #
   #       \ \_\ \_\ \___ \_\/\____/  \ \_\ \_\     /\_____\ \ \____/      #
   #        \/_/\/_/\/__/\/_/\/___/    \/_/\/_/     \/_____/  \/___/  v1.2 #
   #                                                             By Zion3R #
   #                                                    www.Blackploit.com #
   #                                                   Root@Blackploit.com #
   #########################################################################
--------------------------------------------------
 HASH: 2e728dd31fb5949bc39cac5a9f066498

Possible Hashs:
[+] MD5
[+] Domain Cached Credentials - MD4(MD4(($pass)).(strtolower($username)))
```

Format-Specific Cracking

Once you have identified the hash that you're dealing with, you can tell John to use it while cracking the provided hash using the following syntax:

`john --format=[format] --wordlist=[path to wordlist] [path to file]`

- `--format=` : This is the flag to tell John that you're giving it a hash of a specific format and to use the following format to crack it
- `[format]` : The format that the hash is in

Example Usage:

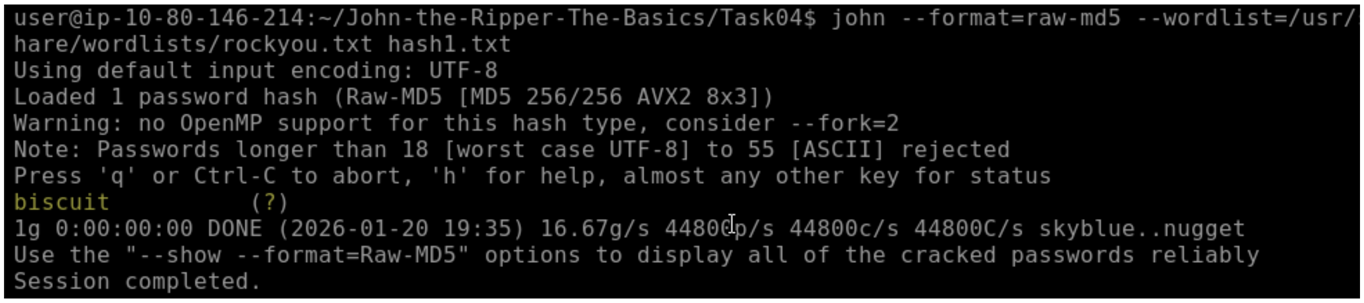
```
john --format=raw-md5 --wordlist=/usr/share/wordlists/rockyou.txt
hash_to_crack.txt
```

A Note on Formats:
if you're dealing with a standard hash type, e.g. md5 as in the example above, you have to prefix it with `raw-` to tell John you're just dealing with a standard hash type

To check if you need to add the prefix or not, you can list all of John's formats using `john --list=formats` and either check manually or grep for your hash type using something like `john --list=formats | grep -iF "md5"`

Practical

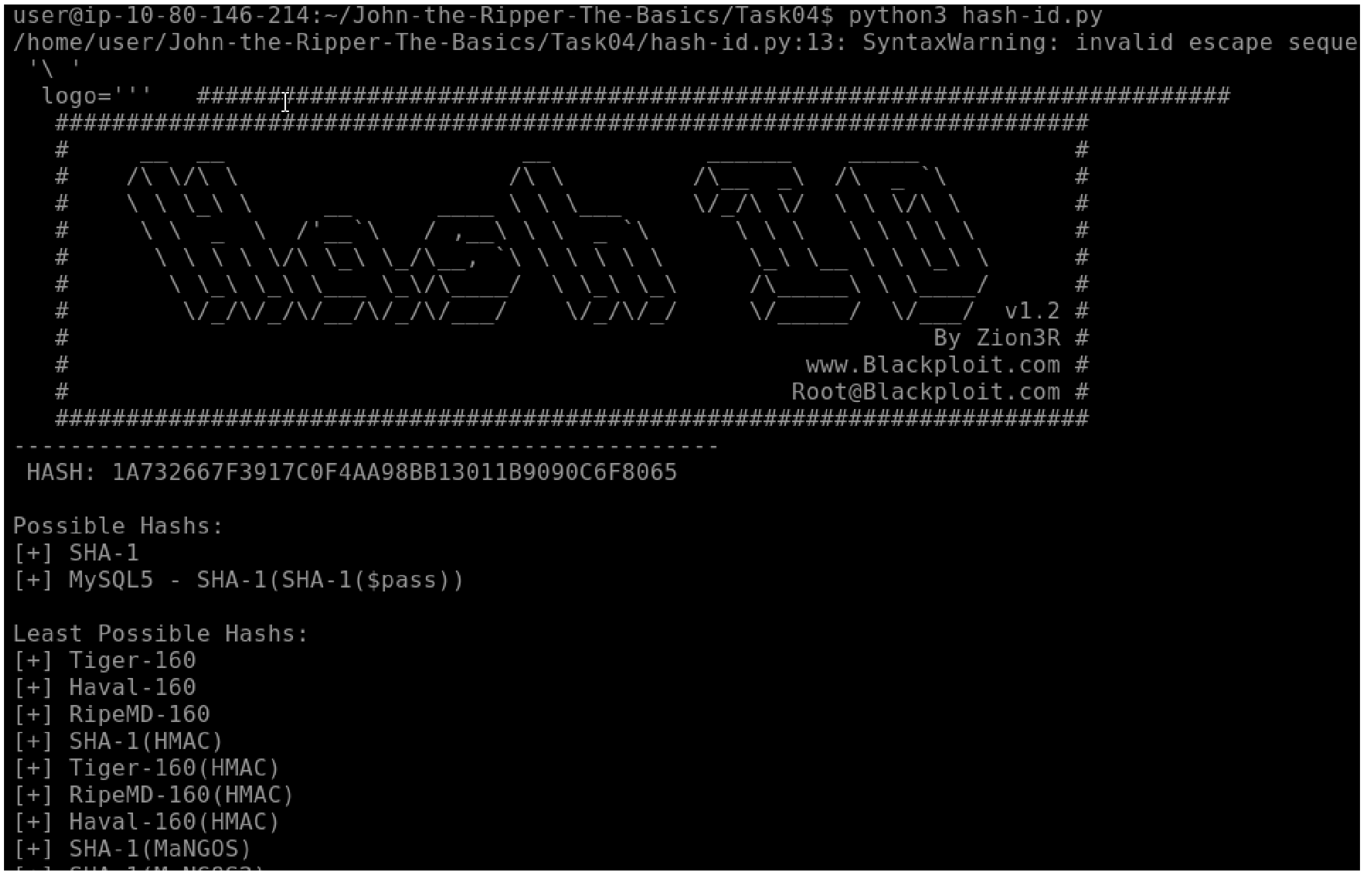
#1



```
user@ip-10-80-146-214:~/John-the-Ripper-The-Basics/Task04$ john --format=raw-md5 --wordlist=/usr/
hare/wordlists/rockyou.txt hash1.txt
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-MD5 [MD5 256/256 AVX2 8x3])
Warning: no OpenMP support for this hash type, consider --fork=2
Note: Passwords longer than 18 [worst case UTF-8] to 55 [ASCII] rejected
Press 'q' or Ctrl-C to abort, 'h' for help, almost any other key for status
biscuit          (?)
1g 0:00:00:00 DONE (2026-01-20 19:35) 16.67g/s 44800p/s 44800c/s 44800C/s skyblue..nugget
Use the "--show --format=Raw-MD5" options to display all of the cracked passwords reliably
Session completed.
```

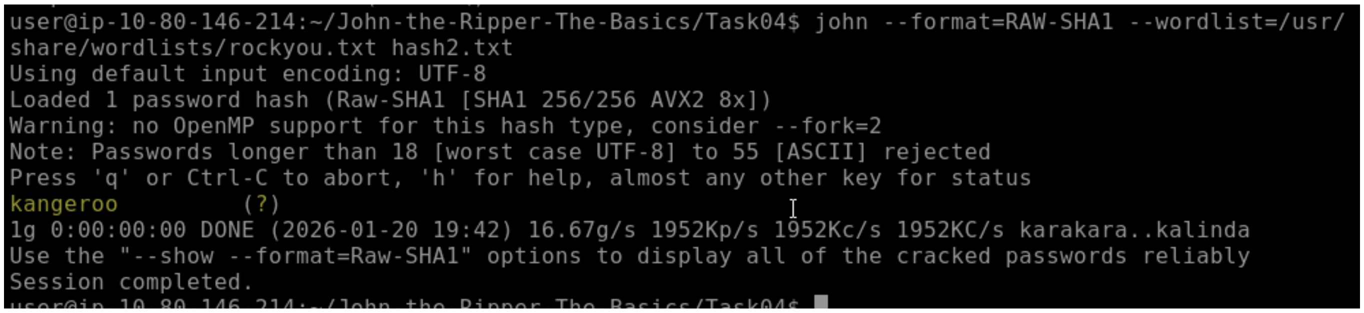
#2

in this I used cat on the hash file to get the hash then i used the python3 command to run the hash-id.py script which helps find the type of hash it is i put the hash in and it said it was sha1



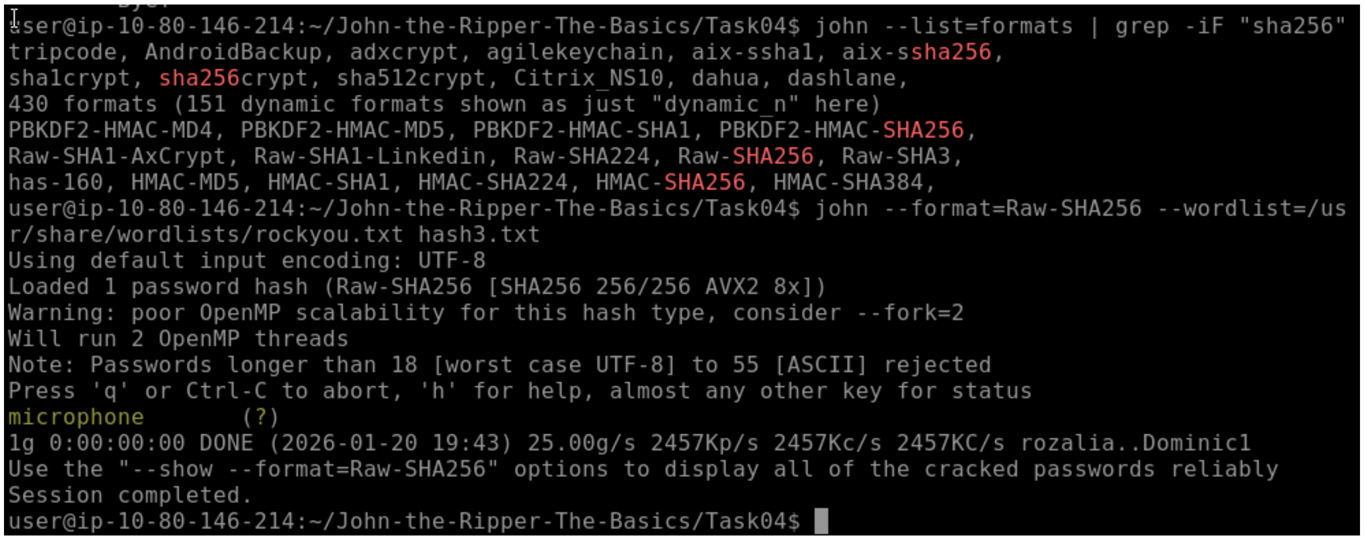
```
user@ip-10-80-146-214:~/John-the-Ripper-The-Basics/Task04$ python3 hash-id.py
/home/user/John-the-Ripper-The-Basics/Task04/hash-id.py:13: SyntaxWarning: invalid escape seque
 '\ '
  logo='''   #########################################################################
   #########################################################################
   #     __  __                     __           ______    _____           #
   #    /\ \/\ \                   /\ \         /\__  _\  /\  _ `\         #
   #    \ \ \_\ \     __      ____ \ \ \___     \/_/\ \/  \ \ \/\ \        #
   #     \ \  _  \  /'__`\   / ,__\ \ \  _ `\      \ \ \   \ \ \ \ \       #
   #      \ \ \ \ \/\ \_\ \_/\__, `\ \ \ \ \ \      \_\ \__ \ \ \_\ \      #
   #       \ \_\ \_\ \___ \_\/\____/  \ \_\ \_\     /\_____\ \ \____/      #
   #        \/_/\/_/\/__/\/_/\/___/    \/_/\/_/     \/_____/  \/___/  v1.2 #
   #                                                             By Zion3R #
   #                                                    www.Blackploit.com #
   #                                                   Root@Blackploit.com #
   #########################################################################
--------------------------------------------------
 HASH: 1A732667F3917C0F4AA98BB13011B9090C6F8065

Possible Hashs:
[+] SHA-1
[+] MySQL5 - SHA-1(SHA-1($pass))

Least Possible Hashs:
[+] Tiger-160
[+] Haval-160
[+] RipeMD-160
[+] SHA-1(HMAC)
[+] Tiger-160(HMAC)
[+] RipeMD-160(HMAC)
[+] Haval-160(HMAC)
[+] SHA-1(MaNGOS)
```

I then used john to crack the hash



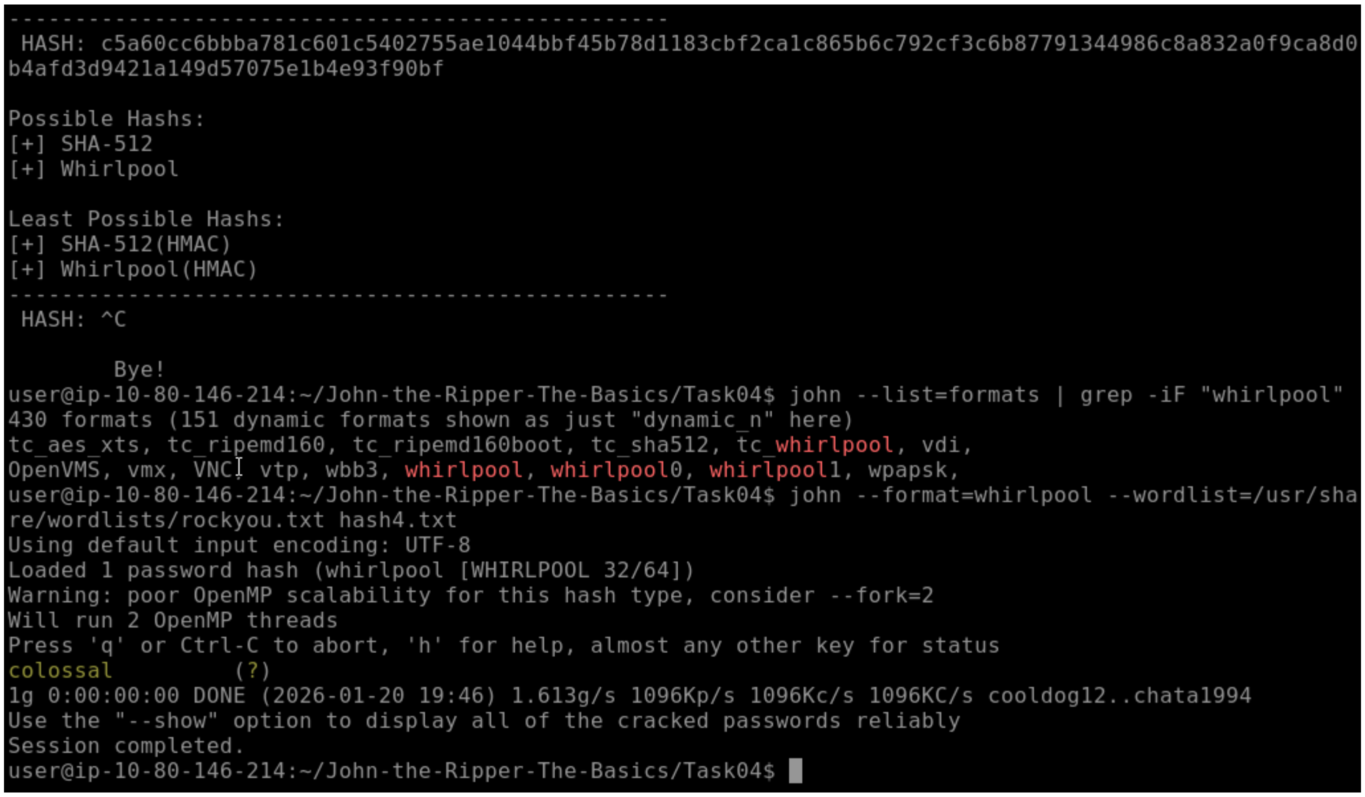
```
user@ip-10-80-146-214:~/John-the-Ripper-The-Basics/Task04$ john --format=RAW-SHA1 --wordlist=/usr/
share/wordlists/rockyou.txt hash2.txt
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-SHA1 [SHA1 256/256 AVX2 8x])
Warning: no OpenMP support for this hash type, consider --fork=2
Note: Passwords longer than 18 [worst case UTF-8] to 55 [ASCII] rejected
Press 'q' or Ctrl-C to abort, 'h' for help, almost any other key for status
kangeroo         (?)
1g 0:00:00:00 DONE (2026-01-20 19:42) 16.67g/s 1952Kp/s 1952Kc/s 1952KC/s karakara..kalinda
Use the "--show --format=Raw-SHA1" options to display all of the cracked passwords reliably
Session completed.
```

#3



```
user@ip-10-80-146-214:~/John-the-Ripper-The-Basics/Task04$ john --list=formats | grep -iF "sha256"
tripcode, AndroidBackup, adxcrypt, agilekeychain, aix-ssha1, aix-ssha256,
sha1crypt, sha256crypt, sha512crypt, Citrix_NS10, dahua, dashlane,
430 formats (151 dynamic formats shown as just "dynamic_n" here)
PBKDF2-HMAC-MD4, PBKDF2-HMAC-MD5, PBKDF2-HMAC-SHA1, PBKDF2-HMAC-SHA256,
Raw-SHA1-AxCrypt, Raw-SHA1-Linkedin, Raw-SHA224, Raw-SHA256, Raw-SHA3,
has-160, HMAC-MD5, HMAC-SHA1, HMAC-SHA224, HMAC-SHA256, HMAC-SHA384,
user@ip-10-80-146-214:~/John-the-Ripper-The-Basics/Task04$ john --format=Raw-SHA256 --wordlist=/us
r/share/wordlists/rockyou.txt hash3.txt
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=2
Will run 2 OpenMP threads
Note: Passwords longer than 18 [worst case UTF-8] to 55 [ASCII] rejected
Press 'q' or Ctrl-C to abort, 'h' for help, almost any other key for status
microphone       (?)
1g 0:00:00:00 DONE (2026-01-20 19:43) 25.00g/s 2457Kp/s 2457Kc/s 2457KC/s rozalia..Dominic1
Use the "--show --format=Raw-SHA256" options to display all of the cracked passwords reliably
Session completed.
user@ip-10-80-146-214:~/John-the-Ripper-The-Basics/Task04$ 
```

#4



```
--------------------------------------------------
 HASH: c5a60cc6bbba781c601c5402755ae1044bbf45b78d1183cbf2ca1c865b6c792cf3c6b87791344986c8a832a0f9ca8d0
b4afd3d9421a149d57075e1b4e93f90bf

Possible Hashs:
[+] SHA-512
[+] Whirlpool

Least Possible Hashs:
[+] SHA-512(HMAC)
[+] Whirlpool(HMAC)
--------------------------------------------------
 HASH: ^C

        Bye!
user@ip-10-80-146-214:~/John-the-Ripper-The-Basics/Task04$ john --list=formats | grep -iF "whirlpool"
430 formats (151 dynamic formats shown as just "dynamic_n" here)
tc_aes_xts, tc_ripemd160, tc_ripemd160boot, tc_sha512, tc_whirlpool, vdi,
OpenVMS, vmx, VNC, vtp, wbb3, whirlpool, whirlpool0, whirlpool1, wpapsk,
user@ip-10-80-146-214:~/John-the-Ripper-The-Basics/Task04$ john --format=whirlpool --wordlist=/usr/sha
re/wordlists/rockyou.txt hash4.txt
Using default input encoding: UTF-8
Loaded 1 password hash (whirlpool [WHIRLPOOL 32/64])
Warning: poor OpenMP scalability for this hash type, consider --fork=2
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, 'h' for help, almost any other key for status
colossal         (?)
1g 0:00:00:00 DONE (2026-01-20 19:46) 1.613g/s 1096Kp/s 1096Kc/s 1096KC/s cooldog12..chata1994
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
user@ip-10-80-146-214:~/John-the-Ripper-The-Basics/Task04$ 
```

Cracking Windows Authentication Hashes

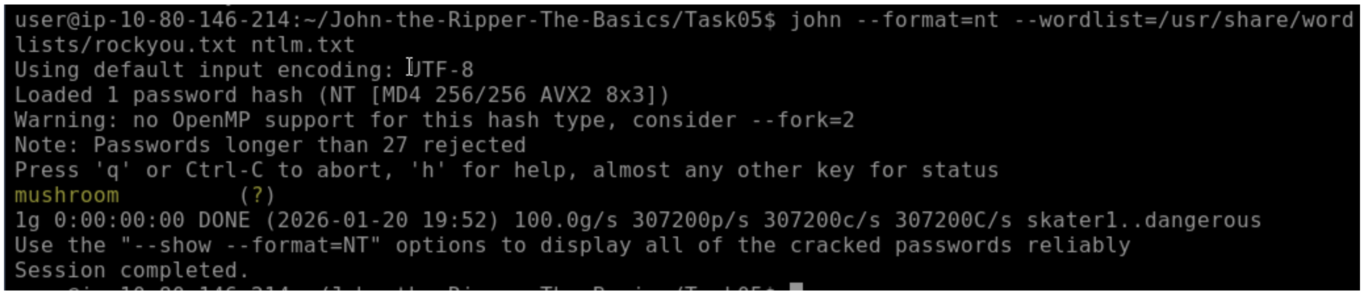
Authentication hashes are the hashed versions of passwords stored by operating systems

To get your hands on these hashes, you must often already be a privileged user

NTHash / NTLM

NThash is the hash format modern Windows operating system machines use to store user and service passwords

In Windows, SAM (Security Account Manager) is used to store user account information, including usernames and hashed passwords. You can acquire NTHash/NTLM hashes by dumping the SAM database on a Windows machine, using a tool like Mimikatz, or using the Active Directory database: `NTDS.dit`


```
user@ip-10-80-146-214:~/John-the-Ripper-The-Basics/Task05$ john --format=nt --wordlist=/usr/share/word
lists/rockyou.txt ntlm.txt
Using default input encoding: UTF-8
Loaded 1 password hash (NT [MD4 256/256 AVX2 8x3])
Warning: no OpenMP support for this hash type, consider --fork=2
Note: Passwords longer than 27 rejected
Press 'q' or Ctrl-C to abort, 'h' for help, almost any other key for status
mushroom         (?)
1g 0:00:00:00 DONE (2026-01-20 19:52) 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 Hashes from /etc/shadow

The `/etc/shadow` file is the file on Linux machines where password hashes are stored. It also stores other information, such as the date of last password change and password expiration information

This file is usually only accessible by the root user, so you must have sufficient privileges to access the hashes

Unshadowing

John can be very particular about the formats it needs data in to be able to work with it; for this reason, to crack `/etc/shadow` passwords, you must combine it with the `/etc/passwd` file for John to understand the data it’s being given. To do this, we use a tool built into the John suite of tools called `unshadow`. The basic syntax of `unshadow` is as follows:

`unshadow [path to passwd] [path to shadow]`

- `unshadow`: Invokes the unshadow tool
- `[path to passwd]`: The file that contains the copy of the `/etc/passwd` file you’ve taken from the target machine
- `[path to shadow]`: The file that contains the copy of the `/etc/shadow` file you’ve taken from the target machine

Example Usage:

`unshadow local_passwd local_shadow > unshadowed.txt`

Note on the files

When using `unshadow`, you can either use the entire `/etc/passwd` and `/etc/shadow` files, assuming you have them available, or you can use the relevant line from each, for example:

FILE 1 - local_passwd

Contains the `/etc/passwd` line for the root user:

`root:x:0:0::/root:/bin/bash`

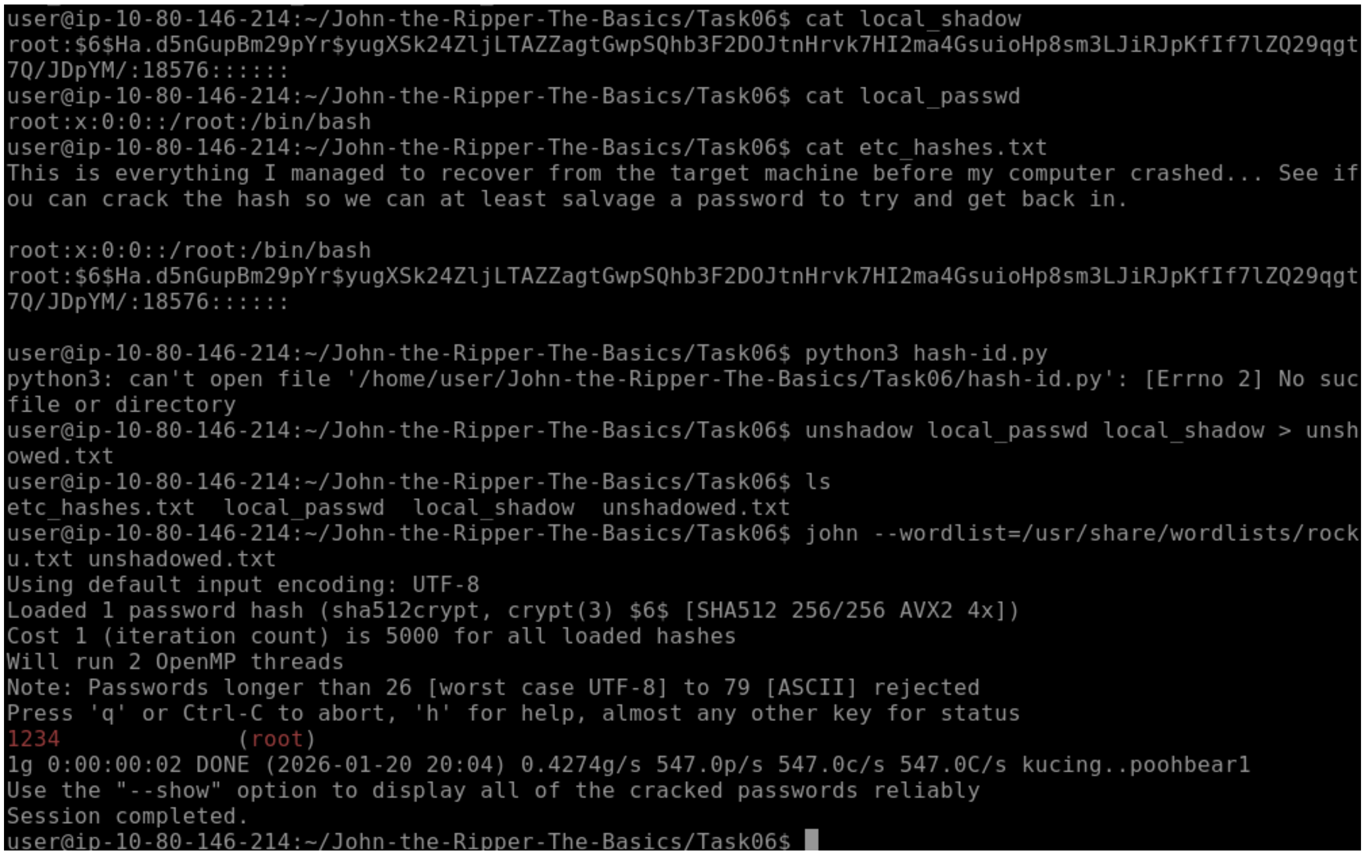
FILE 2 - local_shadow

Contains the `/etc/shadow` line for the root user: `root:$6$2nwjN454g.dv4HN/$m9Z/r2xVfweYVkrr.v5Ft8Ws3/YYksfNwq96UL1FX0OJjY1L6l.DS3KEVsZ9rOVLB/ldTeEL/OIhJZ`

Cracking

We can then feed the output from `unshadow`, in our example use case called `unshadowed.txt`, directly into John

john --wordlist=/usr/share/wordlists/rockyou.txt --format=sha512crypt unshadowed.txt



```
user@ip-10-80-146-214:~/John-the-Ripper-The-Basics/Task06$ cat local_shadow
root:$6$Ha.d5nGupBm29pYr$yugXSk24ZljLTAZZagtGwpSQhb3F2DOJtnHrvk7HI2ma4GsuioHp8sm3LJiRJpKfIf7lZQ29qgt
7Q/JDpYM/:18576::::::
user@ip-10-80-146-214:~/John-the-Ripper-The-Basics/Task06$ cat local_passwd
root:x:0:0::/root:/bin/bash
user@ip-10-80-146-214:~/John-the-Ripper-The-Basics/Task06$ cat etc_hashes.txt
This is everything I managed to recover from the target machine before my computer crashed... See if
ou can crack the hash so we can at least salvage a password to try and get back in.

root:x:0:0::/root:/bin/bash
root:$6$Ha.d5nGupBm29pYr$yugXSk24ZljLTAZZagtGwpSQhb3F2DOJtnHrvk7HI2ma4GsuioHp8sm3LJiRJpKfIf7lZQ29qgt
7Q/JDpYM/:18576::::::

user@ip-10-80-146-214:~/John-the-Ripper-The-Basics/Task06$ python3 hash-id.py
python3: can't open file '/home/user/John-the-Ripper-The-Basics/Task06/hash-id.py': [Errno 2] No suc
file or directory
user@ip-10-80-146-214:~/John-the-Ripper-The-Basics/Task06$ unshadow local_passwd local_shadow > unsh
owed.txt
user@ip-10-80-146-214:~/John-the-Ripper-The-Basics/Task06$ ls
etc_hashes.txt  local_passwd  local_shadow  unshadowed.txt
user@ip-10-80-146-214:~/John-the-Ripper-The-Basics/Task06$ john --wordlist=/usr/share/wordlists/rock
u.txt unshadowed.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 2 OpenMP threads
Note: Passwords longer than 26 [worst case UTF-8] to 79 [ASCII] rejected
Press 'q' or Ctrl-C to abort, 'h' for help, almost any other key for status
1234             (root)
1g 0:00:00:02 DONE (2026-01-20 20:04) 0.4274g/s 547.0p/s 547.0c/s 547.0C/s kucing..poohbear1
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
user@ip-10-80-146-214:~/John-the-Ripper-The-Basics/Task06$ 
```

Single Crack Mode

John also has another mode, called the **Single Crack** mode. In this mode, John uses only the information provided in the username to try and work out possible passwords heuristically by slightly changing the letters and numbers contained within the username.

Word Mangling

The best way to explain Single Crack mode and word mangling is to go through an example:

Consider the username “Markus”.

Some possible passwords could be:

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

GECOS

GECOS stands for General Electric Comprehensive Operating System

we looked at the entries for both `/etc/shadow` and `/etc/passwd`. Looking closely, you will notice that the fields are separated by a colon `:`. The fifth field in the user account record is the GECOS field.
stores general information about the user, such as the user’s full name, office number, and telephone number, among other things.

Using Single Crack Mode

To use single crack mode, we use roughly the same syntax that we’ve used so far; for example, if we wanted to crack the password of the user named “Mike”, using the single mode, we’d use:

`john --single --format=[format] [path to file]`

- `--single`: This flag lets John know you want to use the single hash-cracking mode
- `--format=[format]`: As always, it is vital to identify the proper format.

Example Usage:

`john --single --format=raw-sha256 hashes.txt`

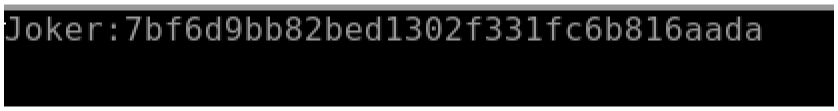
A Note on File Formats in Single Crack Mode:

If you’re cracking hashes in single crack mode, you need to change the file format that you’re feeding John for it to understand what data to create a wordlist from. You do this by prepending the hash with the username that the hash belongs to, so according to the above example, we would change the file `hashes.txt`

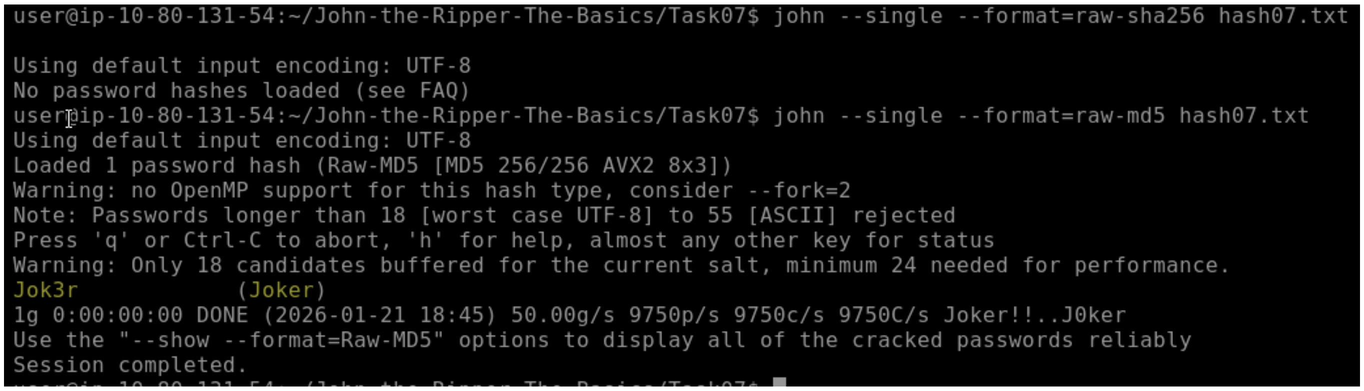
From `1efee03cdcb96d90ad48ccc7b8666033`

To `mike:1efee03cdcb96d90ad48ccc7b8666033`

Task:
I went into the hashfile and prepended the name Joker onto the hash


```
Joker:7bf6d9bb82bed1302f331fc6b816aada
```

I then searched up the hash type in a hash type identifier and it was md5 then i used the john single crack command to crack the hash


```
user@ip-10-80-131-54:~/John-the-Ripper-The-Basics/Task07$ john --single --format=raw-sha256 hash07.txt

Using default input encoding: UTF-8
No password hashes loaded (see FAQ)
user@ip-10-80-131-54:~/John-the-Ripper-The-Basics/Task07$ john --single --format=raw-md5 hash07.txt
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-MD5 [MD5 256/256 AVX2 8x3])
Warning: no OpenMP support for this hash type, consider --fork=2
Note: Passwords longer than 18 [worst case UTF-8] to 55 [ASCII] rejected
Press 'q' or Ctrl-C to abort, 'h' for help, almost any other key for status
Warning: Only 18 candidates buffered for the current salt, minimum 24 needed for performance.
Jok3r            (Joker)
1g 0:00:00:00 DONE (2026-01-21 18:45) 50.00g/s 9750p/s 9750c/s 9750C/s Joker!!..J0ker
Use the "--show --format=Raw-MD5" options to display all of the cracked passwords reliably
Session completed.
```

Custom Rules

The ability to define such rules is beneficial when you know more information about the password structure of whatever your target is.

Common Custom Rules

Many organisations will require a certain level of password complexity

As a result, you may receive a prompt telling you that passwords have to contain at least one character from each of the following:

- Lowercase letter
- Uppercase letter
- Number
- Symbol

Password complexity 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!`

is a memorable pattern that people use and reuse when creating passwords
as attackers, we can exploit the fact that we know the likely position of these added elements to create dynamic passwords from our wordlists.

How to create Custom Rules

Custom rules are defined in the `john.conf` file. This file can be found in `/opt/john/john.conf` on the TryHackMe Attackbox. It is usually located in `/etc/john/john.conf` if you have installed John using a package manager or built from source with `make`.

the wiki here to get a full view of the modifiers you can use and more examples of rule implementation.

`[List.Rules:THMRules]` is used to define the name of your rule; this is what you will use to call your custom rule a John argument.

- `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 must define what characters should be appended, prepended or otherwise included

We do this by adding character sets in square brackets `[ ]` where they should be used. These follow the modifier patterns inside double quotes `" "`.

- `[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`
- `[!£$%@]`: Will include the symbols `!`, `£`, `$`, `%`, and `@`

Putting this all together, 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:

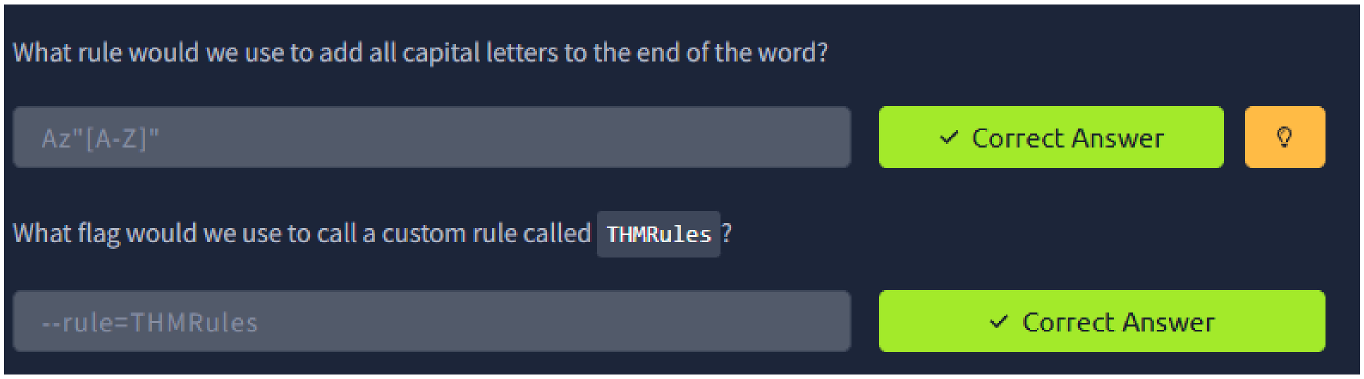
`[List.Rules:PoloPassword]`

`cAz"[0-9] [!£$%@]"`

Using Custom Rules

then call this custom rule a John argument using the `--rule=PoloPassword` flag.

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


What rule would we use to add all capital letters to the end of the word?

Az"[A-Z]"

What flag would we use to call a custom rule called `THMRules`?

--rule=THMRules

Cracking Password Protected Zip Files

Zip2John

Similarly to the `unshadow` tool we used previously, we will use the `zip2john` tool to convert the Zip file into a hash format that John can understand and hopefully crack

The primary usage is like this:

```
zip2john [options] [zip file] > [output file]
```

- `[options]`: Allows you to pass specific checksum options to `zip2john`; this shouldn't often be necessary
- `[zip file]`: The path to the Zip file you wish to get the hash of
- `>`: This redirects the output from this command to another file
- `[output file]`: This is the file that will store the output

Example Usage

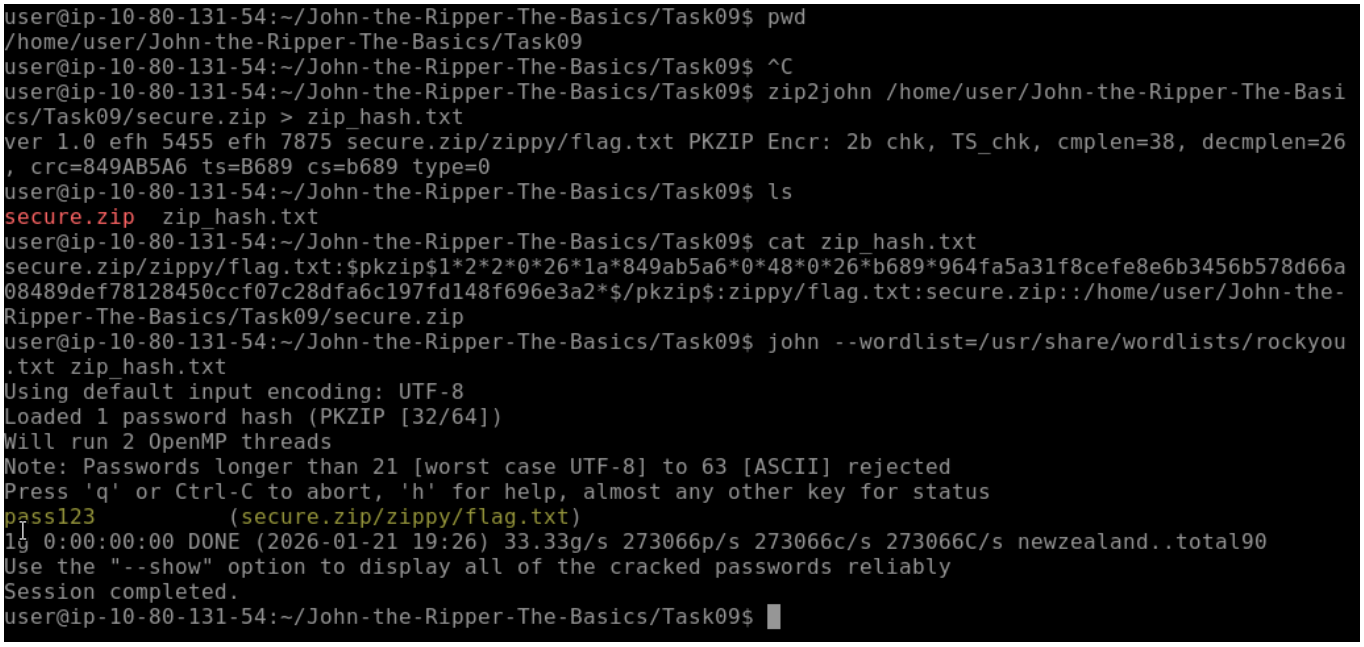
```
zip2john zipfile.zip > zip_hash.txt
```

Cracking

We're then able to take the file we output from `zip2john` in our example use case, `zip_hash.txt`, and, as we did with `unshadow`, feed it directly into John as we have made the input specifically for it.

```
john --wordlist=/usr/share/wordlists/rockyou.txt zip_hash.txt
```

Practical

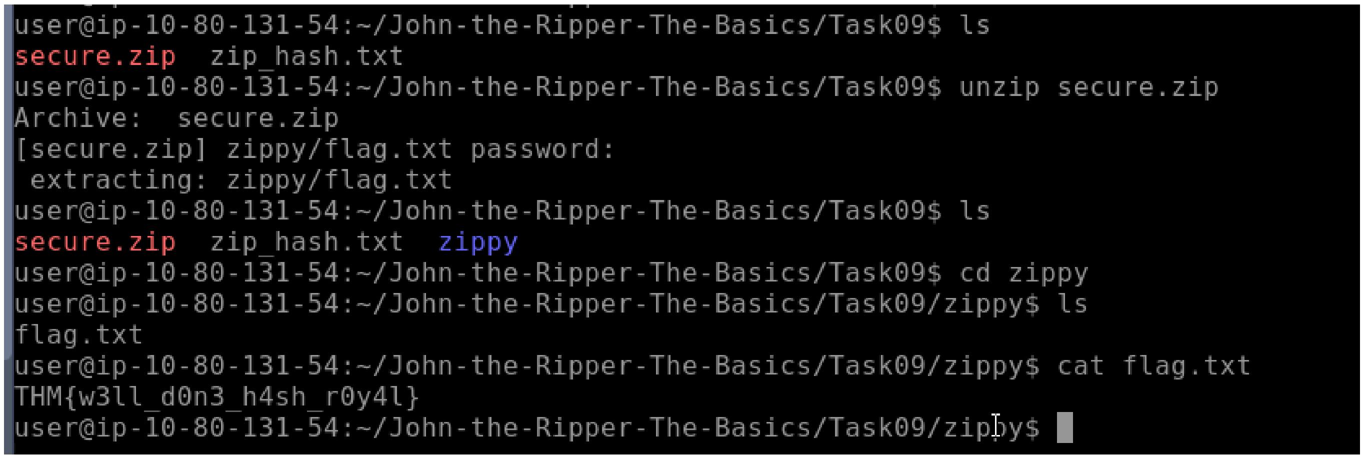


```
user@ip-10-80-131-54:~/John-the-Ripper-The-Basics/Task09$ pwd
/home/user/John-the-Ripper-The-Basics/Task09
user@ip-10-80-131-54:~/John-the-Ripper-The-Basics/Task09$ ^C
user@ip-10-80-131-54:~/John-the-Ripper-The-Basics/Task09$ zip2john /home/user/John-the-Ripper-The-Basi
cs/Task09/secure.zip > zip_hash.txt
ver 1.0 efh 5455 efh 7875 secure.zip/zippy/flag.txt PKZIP Encr: 2b chk, TS_chk, cmplen=38, decmplen=26
, crc=849AB5A6 ts=B689 cs=b689 type=0
user@ip-10-80-131-54:~/John-the-Ripper-The-Basics/Task09$ ls
secure.zip  zip_hash.txt
user@ip-10-80-131-54:~/John-the-Ripper-The-Basics/Task09$ cat zip_hash.txt
secure.zip/zippy/flag.txt:$pkzip$1*2*2*0*26*1a*849ab5a6*0*48*0*26*b689*964fa5a31f8cefe8e6b3456b578d66a
08489def78128450ccf07c28dfa6c197fd148f696e3a2*$/pkzip$:zippy/flag.txt:secure.zip::/home/user/John-the-
Ripper-The-Basics/Task09/secure.zip
user@ip-10-80-131-54:~/John-the-Ripper-The-Basics/Task09$ john --wordlist=/usr/share/wordlists/rockyou
.txt zip_hash.txt
Using default input encoding: UTF-8
Loaded 1 password hash (PKZIP [32/64])
Will run 2 OpenMP threads
Note: Passwords longer than 21 [worst case UTF-8] to 63 [ASCII] rejected
Press 'q' or Ctrl-C to abort, 'h' for help, almost any other key for status
pass123          (secure.zip/zippy/flag.txt)
1g 0:00:00:00 DONE (2026-01-21 19:26) 33.33g/s 273066p/s 273066c/s 273066C/s newzealand..total90
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
user@ip-10-80-131-54:~/John-the-Ripper-The-Basics/Task09$
```

I used pwd to print the working directory of the hash file then i used zip2john [directory of file] > zipped hash text file

then I used the standard john command with the rockyou wordlist on the zipped hash text file i made to get the password for the zip file in yellow

i then could use the unzip [zip file] command to unzip then it required a password which is the password i cracked then i got another directory i went into which contained a flag



```
user@ip-10-80-131-54:~/John-the-Ripper-The-Basics/Task09$ ls
secure.zip  zip_hash.txt
user@ip-10-80-131-54:~/John-the-Ripper-The-Basics/Task09$ unzip secure.zip
Archive:  secure.zip
[secure.zip] zippy/flag.txt password:
 extracting: zippy/flag.txt
user@ip-10-80-131-54:~/John-the-Ripper-The-Basics/Task09$ ls
secure.zip  zip_hash.txt  zippy
user@ip-10-80-131-54:~/John-the-Ripper-The-Basics/Task09$ cd zippy
user@ip-10-80-131-54:~/John-the-Ripper-The-Basics/Task09/zippy$ ls
flag.txt
user@ip-10-80-131-54:~/John-the-Ripper-The-Basics/Task09/zippy$ cat flag.txt
THM{w3ll_d0n3_h4sh_r0y4l}
user@ip-10-80-131-54:~/John-the-Ripper-The-Basics/Task09/zippy$
```

Cracking Password-Protected RAR Archives

RAR archives are compressed files created by the WinRAR archive manager. Like Zip files, they compress folders and files.

will use the `rar2john` tool to convert the RAR file into a hash format that John can understand. The basic syntax is as follows:

`rar2john [rar file] > [output file]`

- `rar2john`: Invokes the `rar2john` tool

- `[rar file]` : The path to the RAR file you wish to get the hash of
- `>` : This redirects the output of this command to another file
- `[output file]` : This is the file that will store the output from the command

Example Usage

`/opt/john/rar2john rarfile.rar > rar_hash.txt`

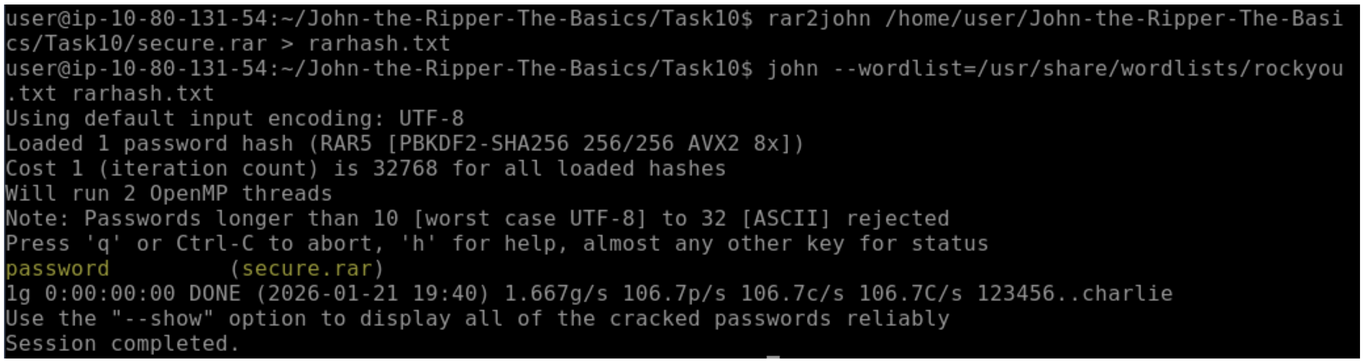
Cracking

Once again, we can take the file we output from `rar2john` in our example use case, `rar_hash.txt`, and feed it directly into John as we did with `zip2john`.

`john --wordlist=/usr/share/wordlists/rockyou.txt rar_hash.txt`

Practical

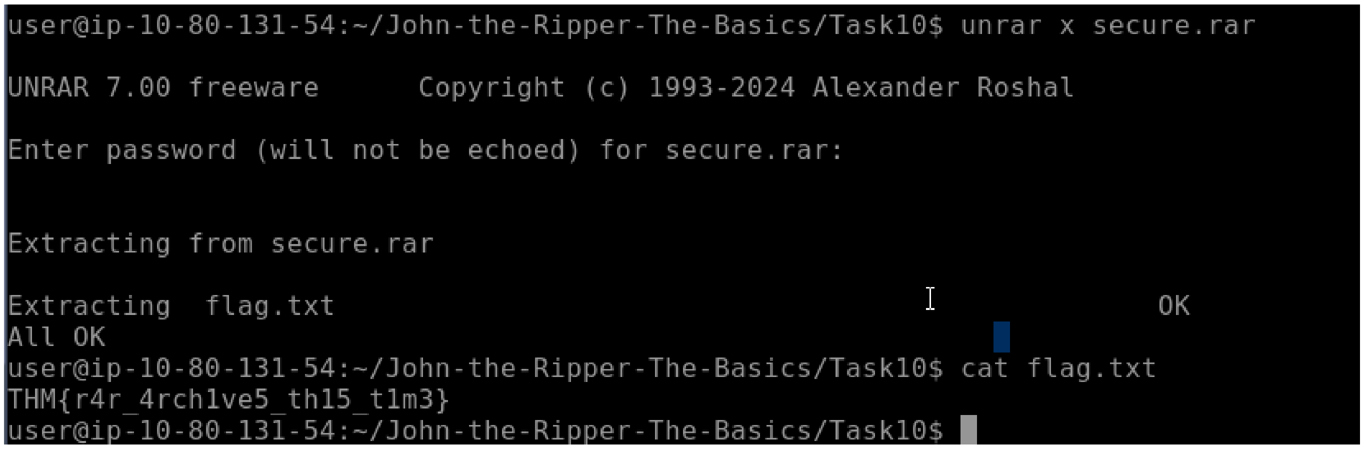
like the zip file i used the rar2john on the directory with the rar file to convert contents into a text file



```
user@ip-10-80-131-54:~/John-the-Ripper-The-Basics/Task10$ rar2john /home/user/John-the-Ripper-The-Basi
cs/Task10/secure.rar > rarhash.txt
user@ip-10-80-131-54:~/John-the-Ripper-The-Basics/Task10$ john --wordlist=/usr/share/wordlists/rockyou
.txt rarhash.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 2 OpenMP threads
Note: Passwords longer than 10 [worst case UTF-8] to 32 [ASCII] rejected
Press 'q' or Ctrl-C to abort, 'h' for help, almost any other key for status
password         (secure.rar)
1g 0:00:00:00 DONE (2026-01-21 19:40) 1.667g/s 106.7p/s 106.7c/s 106.7C/s 123456..charlie
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

I then used the normal john command with the rockyou directory on the rar text file to get the password of the rar file

i unrar'd the rar file with unrar x [rarfilename] command which gave me the flag.txt which i opened to get the flag



```
user@ip-10-80-131-54:~/John-the-Ripper-The-Basics/Task10$ unrar x secure.rar

UNRAR 7.00 freeware      Copyright (c) 1993-2024 Alexander Roshal

Enter password (will not be echoed) for secure.rar:


Extracting from secure.rar

Extracting  flag.txt                                                  OK
All OK
user@ip-10-80-131-54:~/John-the-Ripper-The-Basics/Task10$ cat flag.txt
THM{r4r_4rch1ve5_th15_t1m3}
user@ip-10-80-131-54:~/John-the-Ripper-The-Basics/Task10$
```

Cracking SSH Keys with John

using John to crack the SSH private key password of `id_rsa` files

you can configure key-based authentication, which lets you use your private key, `id_rsa`, as an authentication key to log in to a remote machine over SSH. However, doing so will often require a password to access the private key

will be using John to crack this password to allow authentication over SSH using the key

SSH2John

`ssh2john` converts the `id_rsa` private key

if you don't have `ssh2john` installed, you can use `ssh2john.py`, located in the `/opt/john/ssh2john.py`. If you're doing this on the AttackBox, replace the `ssh2john` command with `python3 /opt/john/ssh2john.py` or on Kali, `python /usr/share/john/ssh2john.py`

`ssh2john [id_rsa private key file] > [output file]`

- `ssh2john`: Invokes the `ssh2john` tool
- `[id_rsa private key file]`: The path to the id_rsa file you wish to get the hash of
- `>`: This is the output director. We're using it to redirect the output from this command to another file.
- `[output file]`: This is the file that will store the output from

Example Usage

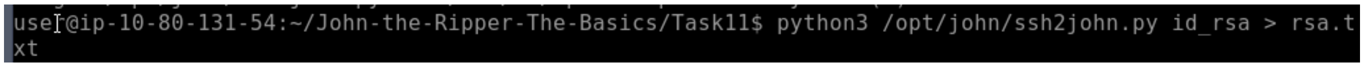
`/opt/john/ssh2john.py id_rsa > id_rsa_hash.txt`

Cracking

we're feeding the file we output from ssh2john, which in our example use case is called `id_rsa_hash.txt` and, as we did with `rar2john`, we can use this seamlessly with John:

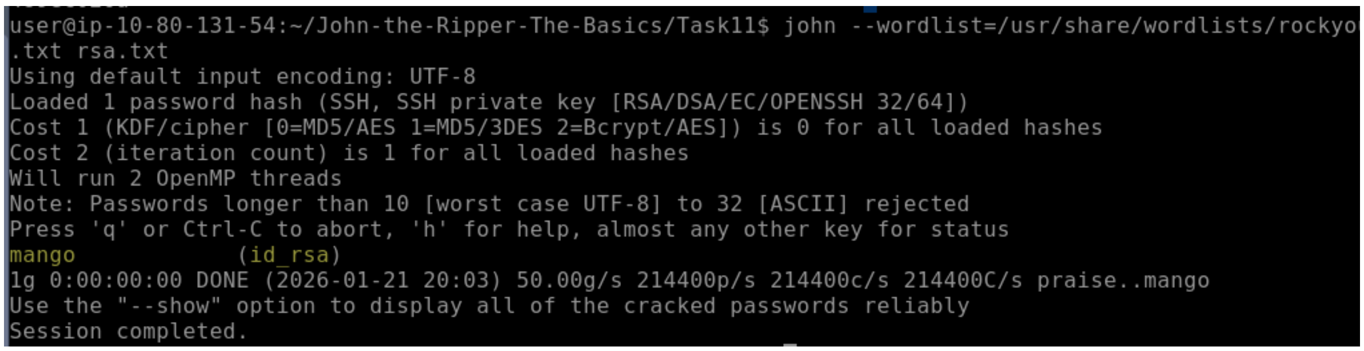
`john --wordlist=/usr/share/wordlists/rockyou.txt id_rsa_hash.txt`

have to use this command to convert the rsa key to a text file



```
user@ip-10-80-131-54:~/John-the-Ripper-The-Basics/Task11$ python3 /opt/john/ssh2john.py id_rsa > rsa.t
xt
```

i then used normal john with wordlist against the textfile to get the password for the key



```
user@ip-10-80-131-54:~/John-the-Ripper-The-Basics/Task11$ john --wordlist=/usr/share/wordlists/rockyo
.txt rsa.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 2 OpenMP threads
Note: Passwords longer than 10 [worst case UTF-8] to 32 [ASCII] rejected
Press 'q' or Ctrl-C to abort, 'h' for help, almost any other key for status
mango            (id_rsa)
1g 0:00:00:00 DONE (2026-01-21 20:03) 50.00g/s 214400p/s 214400c/s 214400C/s praise..mango
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```