TryHackMe – John the Ripper - The Basics

Learn how to use John the Ripper, a powerful and adaptable hash-cracking tool.

Note: This room was completed in different days, so you will see different IPs.

Task 1: Introduction

John the Ripper is a free and open-source password-cracking tool. It can crack passwords stored in various formats, including hashes, passwords, and encrypted private keys. It can be used to test passwords' security and recover lost passwords. In other words, John the Ripper is a well-known, well-loved, and versatile hash-cracking tool. It combines a fast-cracking speed with an extraordinary range of compatible hash types.

Learning Objectives

Upon the completion of this room, you learn about using John for:

- Cracking Windows authentication hashes
- Crack **/etc/shadow** hashes
- Cracking password-protected Zip files
- Cracking password-protected RAR files
- Cracking SSH keys

Starting with Task 4, you need to apply either on the attached VM, the AttackBox, or your own system. For your convenience, you can download the required task files as a single Zip file from this task.

Task 2: Basic Terms

What are Hashes?

A hash is a way of taking a piece of data of any length and representing it in another fixed-length form. This process masks the original value of the data. The hash value is obtained by running the original data through a hashing algorithm. Many popular hashing algorithms exist, such as MD4, MD5, SHA1 and NTLM. Let's try and show this with an example:

If we take "polo", a string of four characters, and run it through an MD5 hashing algorithm, we end up with an output of **b53759f3ce692de7aff1b5779d3964da**, a standard 32-character MD5 hash.

Likewise, if we take "polomints", a string of 9 characters, and run it through the same MD5 hashing algorithm, we end up with an output of **584b6e4f4586e136bc280f27f9c64f3b**, another standard 32-character MD5 hash.

What Makes Hashes Secure?

Hashing functions are designed as one-way functions. In other words, it is easy to calculate the hash value of a given input; however, it is a hard problem to find the original input given the hash value. In simple terms, a hard problem quickly becomes computationally infeasible in computer science. This computational problem has its roots in mathematics as P vs NP.

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.

While this is a fascinating mathematical concept that proves fundamental to computing and cryptography, it is entirely outside the scope of this room. But abstractly, the algorithm to hash the value will be "P" and can, therefore, be calculated reasonably. However, an "un-hashing" algorithm would be "NP" and intractable to solve, meaning that it cannot be computed in a reasonable time using standard computers.

Where John Comes in

Even though the algorithm is not feasibly reversible, that doesn't mean cracking the hashes is impossible. 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, called a dictionary. You can then compare these hashes to the one you're trying to crack to see if they match. If they do, you know what word corresponds to that hash- you've cracked it!

This process is called a **dictionary attack**, and John the Ripper, or John as it's commonly shortened, is a tool for conducting fast brute force attacks on various hash types. This room will focus on the most popular extended version of John the Ripper, **Jumbo John**.

Question 1: What is the most popular extended version of John the Ripper?

Answer: *Jumbo John*

Task 3: Setting Up Your System

Throughout the tasks of this room, we will be using the following:

- The "Jumbo John" version of John the Ripper
- The RockYou password list

If you use the attached virtual machine or the AttackBox, you don't need to install John the Ripper on your system. Consequently, feel free to skip through the installation section. If you prefer to use your system to follow along, please read along to learn how to proceed with the installation. We should note that if you use a version of John the Ripper other than Jumbo John, you might not have some of the required tools, such as **zip2john** and **rar2john**.

Installation

John the Ripper is supported on many Operating Systems, not just Linux Distributions. Before we go through this, there are multiple versions of John, the standard "core" distribution, and multiple community editions, which extend the feature set of the original John distribution. The most popular of these distributions is the "**Jumbo John**," which we will use specific features of later.

AttackBox and Kali

Jumbo John is already installed on the attached virtual machine and on the AttackBox, so if you plan to use either one, you need not take any further action. Furthermore,
offensive Linux distributions like Kali are shipped with Jumbo John installed.

You can double-check this by typing `john` into the terminal. You should be met with a usage guide for John, with the first line reading “John the Ripper 1.9.0-jumbo-1” or something similar with a different version number.

Other Linux Distributions

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`. Unfortunately, at the time of writing, these versions provided core functionality and missed some of the tools available through Jumbo John.

Consequently, you need to consider building from the source to access all the tools available via Jumbo John. The official installation guide provides detailed installation and build configuration instructions.

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

Now that we have `john` ready, we must consider another indispensable component: wordlists.

As we mentioned earlier, to use a dictionary attack against hashes, you need a list of words to hash and compare; unsurprisingly, this is called a wordlist. There are many different wordlists out there, and a good collection can be found in the SecLists repository. There are a few places you can look for wordlists for attacking the system of choice; we will quickly run through where you can find them.

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

RockYou

For all of the tasks in this room, we will use the infamous `rockyou.txt` wordlist, a very large common password wordlist obtained from a data breach on a website called rockyou.com in 2009. If you are not using any of the above distributions, you can get the `rockyou.txt` wordlist from the SecLists repository under the `/Passwords/Leaked-Databases` subsection. You may need to extract it from the `.tar.gz` format using `tar xvzf rockyou.txt.tar.gz`.

Now that we have our hash cracker and wordlists all set up, let’s move on to some hash cracking!

To follow along, first, let's start the **Virtual Machine** by pressing the Start **Machine button** below.

The machine will start in **Split-Screen** view. In case the VM is not visible, use the blue **Show Split View** button at the top of the page.

You can also access the virtual machine using SSH at the IP address `MACHINE_IP` using the following credentials:

- Username: `user`
- Password: `Tryhackme123!`

Question 2: Which website's breach was the rockyou.txt wordlist created from?

Answer: *rockyou.com*

Task 4: Cracking Basic Hashes

There are multiple ways to use John the Ripper to crack simple hashes. We'll walk through a few before moving on to cracking some ourselves.

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

John has built-in features to detect what type of hash it's being given and to select appropriate rules and formats to crack it for you; this isn't always the best idea as it can be unreliable, but 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

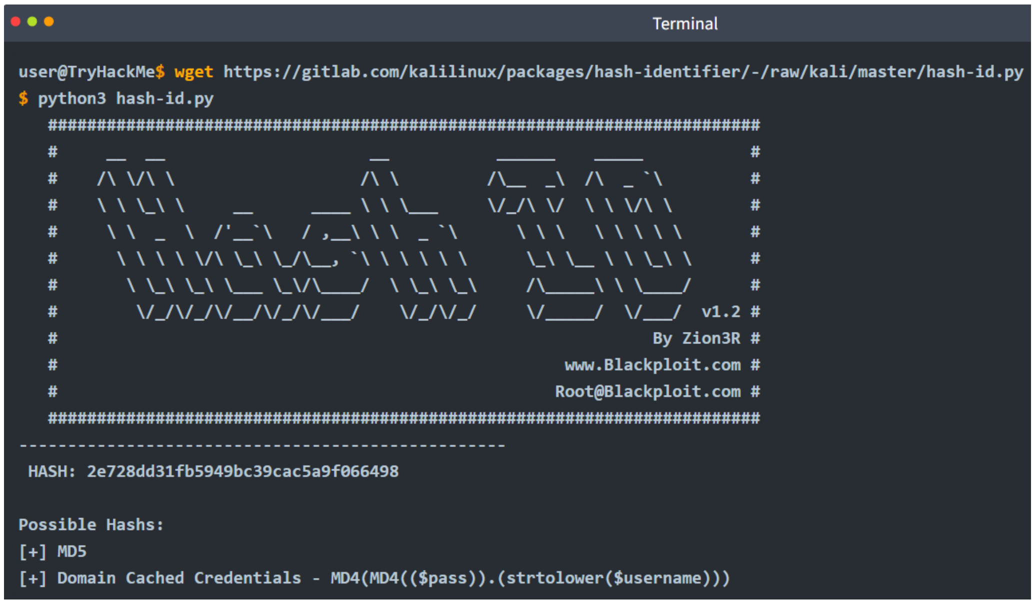
Example Usage:

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

Identifying Hashes

Sometimes, John won't play nicely with automatically recognizing and loading hashes, but that's okay! 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, giving you more options if the first one fails.

To use hash-identifier, you can use **wget** or **curl** to download the Python file **hash-id.py** from its GitLab page. Then, launch it with **python3 hash-id.py** and enter the hash you're trying to identify. It will give you a list of the most probable formats. These two steps are shown in the terminal below.



```
user@TryHackMe$ wget https://gitlab.com/kalilinux/packages/hash-identifier/-/raw/kali/master/hash-id.py
$ 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)))
```

wget https://gitlab.com/kalilinux/packages/hash-identifier/-/raw/kali/master/hash-id.py

python3 hash-id.py

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
```

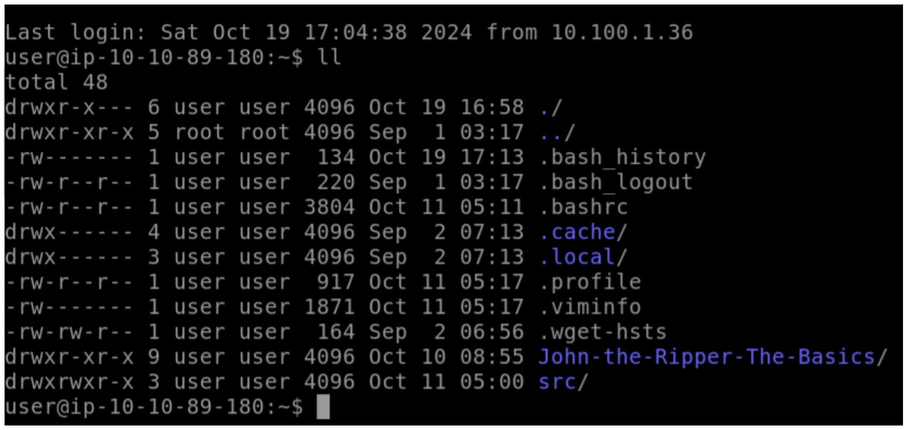
A Note on Formats:

When you tell John to use 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, though this doesn't always apply. 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

Now that you know the syntax, modifiers, and methods for cracking basic hashes, try it yourself! The files are located in `~/John-the-Ripper-The-Basics/Task04/` on the attached virtual machine.

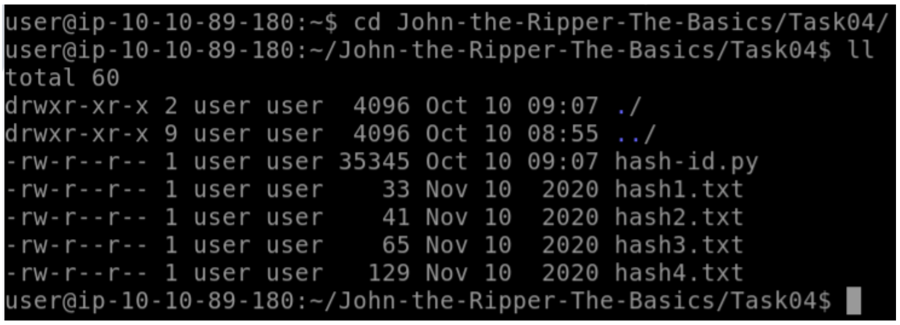
ll



```
Last login: Sat Oct 19 17:04:38 2024 from 10.100.1.36
user@ip-10-10-89-180:~$ ll
total 48
drwxr-x--- 6 user user 4096 Oct 19 16:58 ./
drwxr-xr-x 5 root root 4096 Sep  1 03:17 ../
-rw------- 1 user user  134 Oct 19 17:13 .bash_history
-rw-r--r-- 1 user user  220 Sep  1 03:17 .bash_logout
-rw-r--r-- 1 user user 3804 Oct 11 05:11 .bashrc
drwx------ 4 user user 4096 Sep  2 07:13 .cache/
drwx------ 3 user user 4096 Sep  2 07:13 .local/
-rw-r--r-- 1 user user  917 Oct 11 05:17 .profile
-rw------- 1 user user 1871 Oct 11 05:17 .viminfo
-rw-rw-r-- 1 user user  164 Sep  2 06:56 .wget-hsts
drwxr-xr-x 9 user user 4096 Oct 10 08:55 John-the-Ripper-The-Basics/
drwxrwxr-x 3 user user 4096 Oct 11 05:00 src/
user@ip-10-10-89-180:~$
```

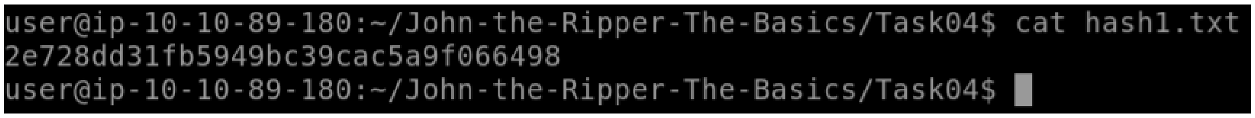
cd John-the-Ripper-The-Basics/Task04/

ll



```
user@ip-10-10-89-180:~$ cd John-the-Ripper-The-Basics/Task04/
user@ip-10-10-89-180:~/John-the-Ripper-The-Basics/Task04$ ll
total 60
drwxr-xr-x 2 user user  4096 Oct 10 09:07 ./
drwxr-xr-x 9 user user  4096 Oct 10 08:55 ../
-rw-r--r-- 1 user user 35345 Oct 10 09:07 hash-id.py
-rw-r--r-- 1 user user    33 Nov 10  2020 hash1.txt
-rw-r--r-- 1 user user    41 Nov 10  2020 hash2.txt
-rw-r--r-- 1 user user    65 Nov 10  2020 hash3.txt
-rw-r--r-- 1 user user   129 Nov 10  2020 hash4.txt
user@ip-10-10-89-180:~/John-the-Ripper-The-Basics/Task04$
```

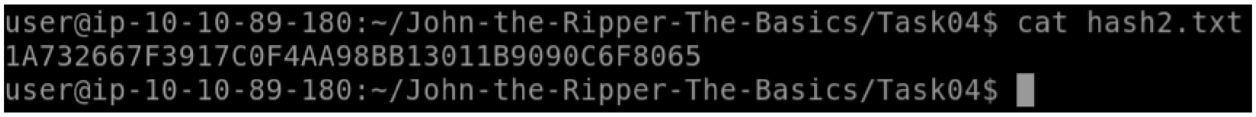
cat hash1.txt



```
user@ip-10-10-89-180:~/John-the-Ripper-The-Basics/Task04$ cat hash1.txt
2e728dd31fb5949bc39cac5a9f066498
user@ip-10-10-89-180:~/John-the-Ripper-The-Basics/Task04$
```

2e728dd31fb5949bc39cac5a9f066498

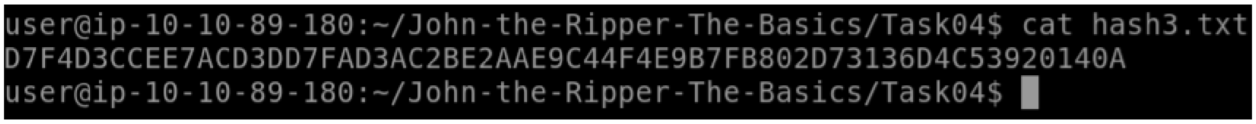
cat hash2.txt



```
user@ip-10-10-89-180:~/John-the-Ripper-The-Basics/Task04$ cat hash2.txt
1A732667F3917C0F4AA98BB13011B9090C6F8065
user@ip-10-10-89-180:~/John-the-Ripper-The-Basics/Task04$
```

1A732667F3917C0F4AA98BB13011B9090C6F8065

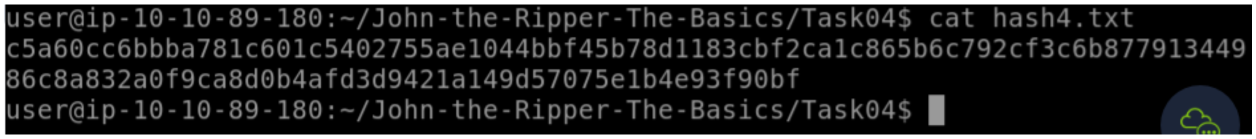
cat hash3.txt



```
user@ip-10-10-89-180:~/John-the-Ripper-The-Basics/Task04$ cat hash3.txt
D7F4D3CCEE7ACD3DD7FAD3AC2BE2AAE9C44F4E9B7FB802D73136D4C53920140A
user@ip-10-10-89-180:~/John-the-Ripper-The-Basics/Task04$
```

D7F4D3CCEE7ACD3DD7FAD3AC2BE2AAE9C44F4E9B7FB802D73136D4C53920140A

cat hash4.txt

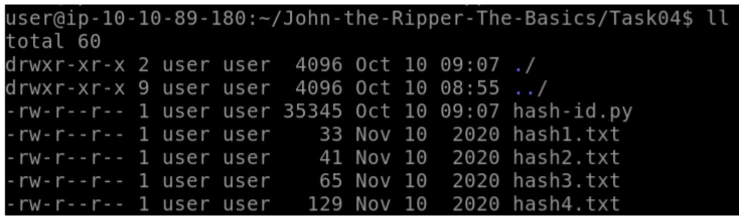


```
user@ip-10-10-89-180:~/John-the-Ripper-The-Basics/Task04$ cat hash4.txt
c5a60cc6bbba781c601c5402755ae1044bbf45b78d1183cbf2ca1c865b6c792cf3c6b877913449
86c8a832a0f9ca8d0b4afd3d9421a149d57075e1b4e93f90bf
user@ip-10-10-89-180:~/John-the-Ripper-The-Basics/Task04$
```

c5a60cc6bbba781c601c5402755ae1044bbf45b78d1183cbf2ca1c865b6c792cf3c6b877913449

86c8a832a0f9ca8d0b4afd3d9421a149d57075e1b4e93f90bf

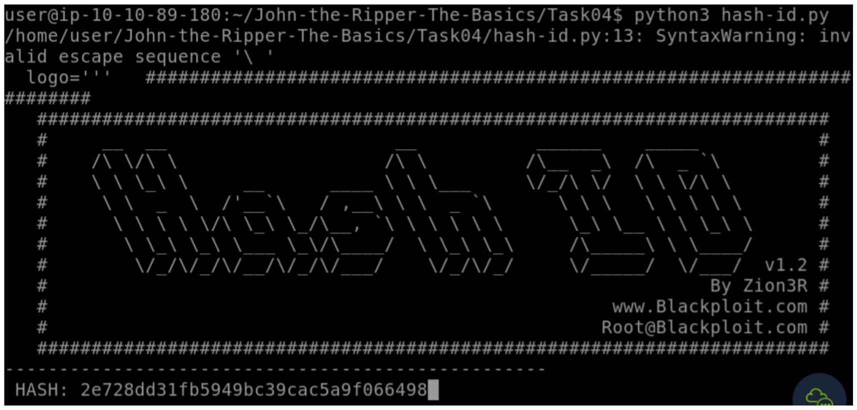
We can notice that we have ***hash-id.py***.



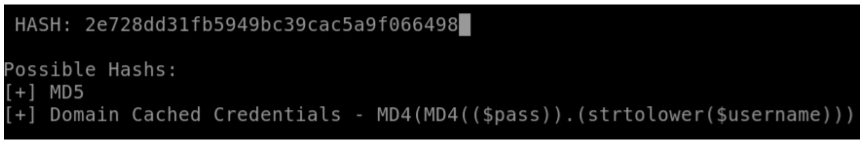
```
user@ip-10-10-89-180:~/John-the-Ripper-The-Basics/Task04$ ll
total 60
drwxr-xr-x 2 user user  4096 Oct 10 09:07 ./
drwxr-xr-x 9 user user  4096 Oct 10 08:55 ../
-rw-r--r-- 1 user user 35345 Oct 10 09:07 hash-id.py
-rw-r--r-- 1 user user    33 Nov 10  2020 hash1.txt
-rw-r--r-- 1 user user    41 Nov 10  2020 hash2.txt
-rw-r--r-- 1 user user    65 Nov 10  2020 hash3.txt
-rw-r--r-- 1 user user   129 Nov 10  2020 hash4.txt
```

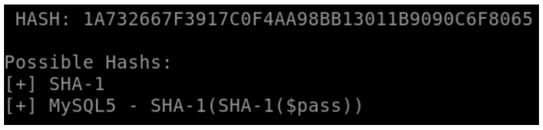
python3 hash-id.py

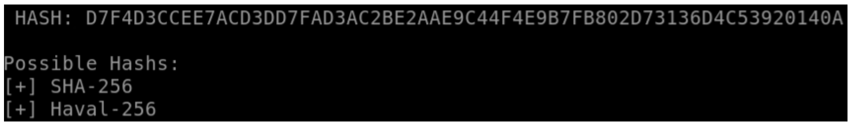
Let's introduce the hashes now.

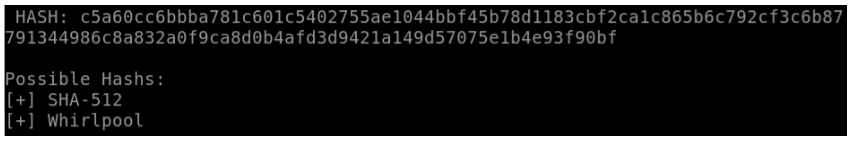


```
user@ip-10-10-89-180:~/John-the-Ripper-The-Basics/Task04$ python3 hash-id.py
/home/user/John-the-Ripper-The-Basics/Task04/hash-id.py:13: SyntaxWarning: inv
alid escape sequence '\ '
  logo='''   ###################################################################
########
   #########################################################################
   #     __  __                     __           ______    _____           #
   #    /\ \/\ \                   /\ \         /\__  _\  /\  _ `\         #
   #    \ \ \_\ \     __      ____ \ \ \___     \/_/\ \/  \ \ \/\ \        #
   #     \ \  _  \  /'__`\   / ,__\ \ \  _ `\      \ \ \   \ \ \ \ \       #
   #      \ \ \ \ \/\ \_\ \_/\__, `\ \ \ \ \ \      \_\ \__ \ \ \_\ \      #
   #       \ \_\ \_\ \___ \_\/\____/  \ \_\ \_\     /\_____\ \ \____/      #
   #        \/_/\/_/\/__/\/_/\/___/    \/_/\/_/     \/_____/  \/___/  v1.2 #
   #                                                             By Zion3R #
   #                                                    www.Blackploit.com #
   #                                                   Root@Blackploit.com #
   #########################################################################
--------------------------------------------------
 HASH: 2e728dd31fb5949bc39cac5a9f066498
```


```
HASH: 2e728dd31fb5949bc39cac5a9f066498

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

```
HASH: 1A732667F3917C0F4AA98BB13011B9090C6F8065

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

```
HASH: D7F4D3CCEE7ACD3DD7FAD3AC2BE2AAE9C44F4E9B7FB802D73136D4C53920140A

Possible Hashs:
[+] SHA-256
[+] Haval-256
```

```
HASH: c5a60cc6bbba781c601c5402755ae1044bbf45b78d1183cbf2ca1c865b6c792cf3c6b87
791344986c8a832a0f9ca8d0b4afd3d9421a149d57075e1b4e93f90bf

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

hash1 → MD5

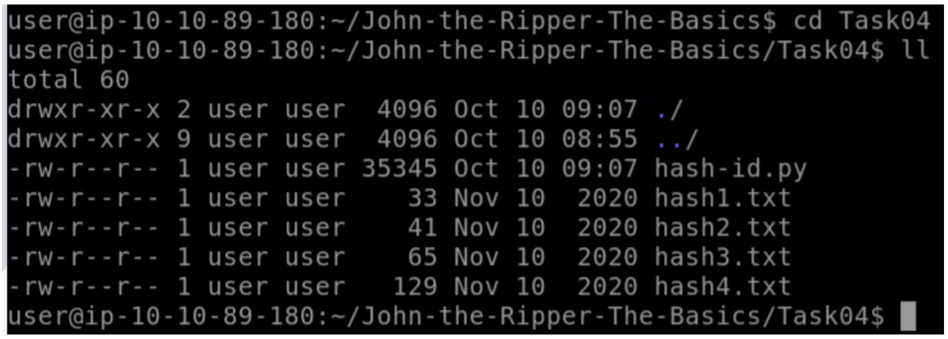
hash2 →SHA-1

hash3 →SHA-256

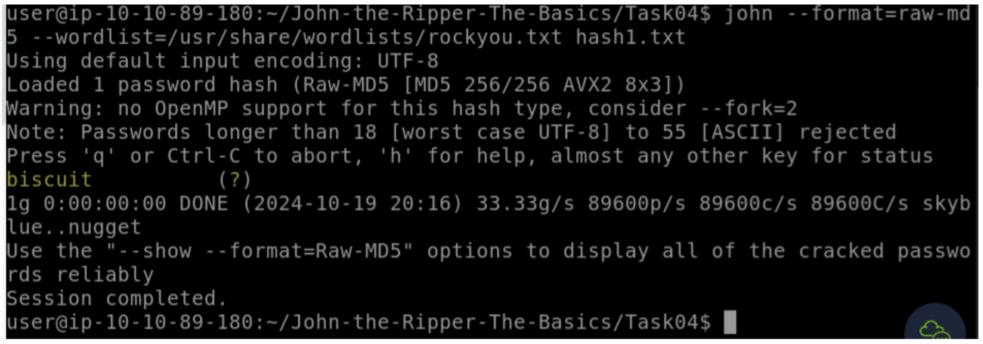
hash4 → Whirlpool

cd Task04

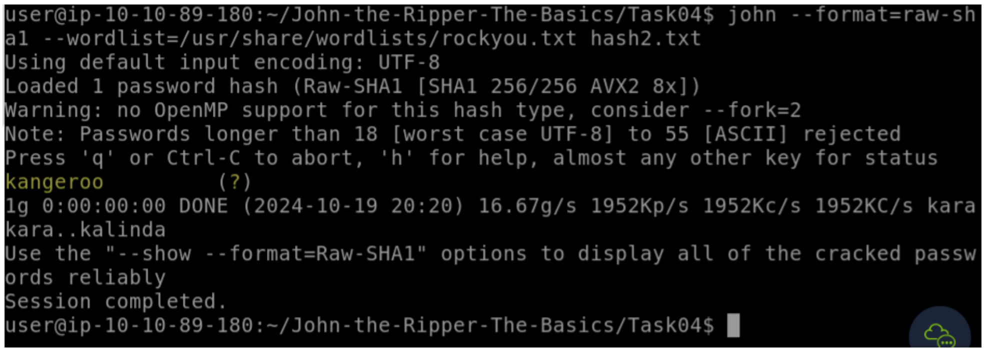
ll


```
user@ip-10-10-89-180:~/John-the-Ripper-The-Basics$ cd Task04
user@ip-10-10-89-180:~/John-the-Ripper-The-Basics/Task04$ ll
total 60
drwxr-xr-x 2 user user  4096 Oct 10 09:07 ./
drwxr-xr-x 9 user user  4096 Oct 10 08:55 ../
-rw-r--r-- 1 user user 35345 Oct 10 09:07 hash-id.py
-rw-r--r-- 1 user user    33 Nov 10  2020 hash1.txt
-rw-r--r-- 1 user user    41 Nov 10  2020 hash2.txt
-rw-r--r-- 1 user user    65 Nov 10  2020 hash3.txt
-rw-r--r-- 1 user user   129 Nov 10  2020 hash4.txt
user@ip-10-10-89-180:~/John-the-Ripper-The-Basics/Task04$
```

john --format=raw-md5 --wordlist=/usr/share/wordlists/rockyou.txt hash1.txt

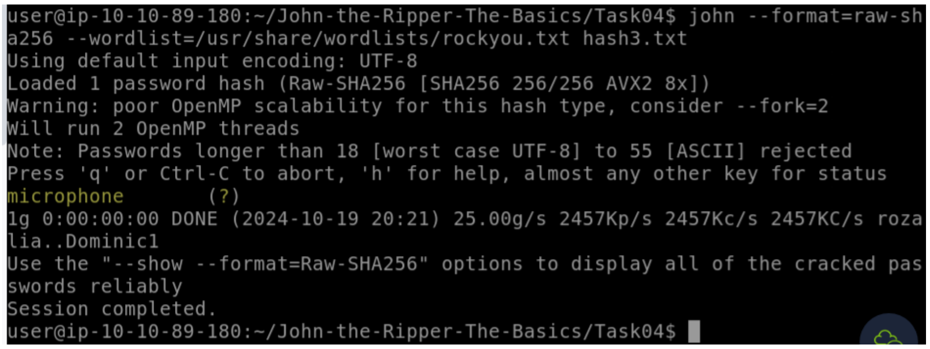

```
user@ip-10-10-89-180:~/John-the-Ripper-The-Basics/Task04$ john --format=raw-md
5 --wordlist=/usr/share/wordlists/rockyou.txt hash1.txt
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-MD5 [MD5 256/256 AVX2 8x3])
Warning: no OpenMP support for this hash type, consider --fork=2
Note: 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 (2024-10-19 20:16) 33.33g/s 89600p/s 89600c/s 89600C/s skyb
lue..nugget
Use the "--show --format=Raw-MD5" options to display all of the cracked passwo
rds reliably
Session completed.
user@ip-10-10-89-180:~/John-the-Ripper-The-Basics/Task04$
```

john --format=raw-sha1 --wordlist=/usr/share/wordlists/rockyou.txt hash2.txt



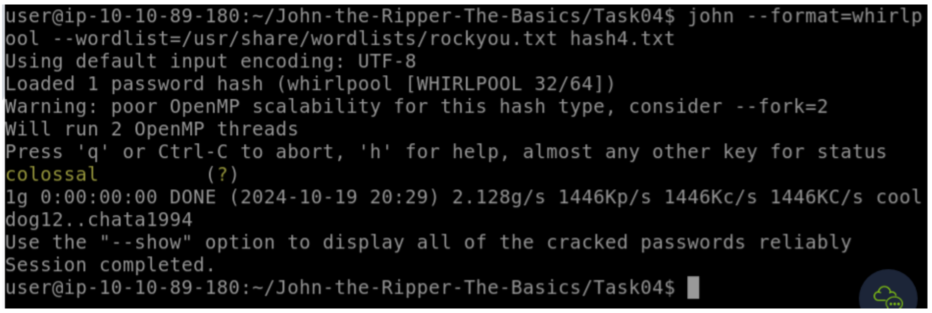
```
user@ip-10-10-89-180:~/John-the-Ripper-The-Basics/Task04$ john --format=raw-sh
a1 --wordlist=/usr/share/wordlists/rockyou.txt hash2.txt
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-SHA1 [SHA1 256/256 AVX2 8x])
Warning: no OpenMP support for this hash type, consider --fork=2
Note: 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 (2024-10-19 20:20) 16.67g/s 1952Kp/s 1952Kc/s 1952KC/s kara
kara..kalinda
Use the "--show --format=Raw-SHA1" options to display all of the cracked passw
ords reliably
Session completed.
user@ip-10-10-89-180:~/John-the-Ripper-The-Basics/Task04$
```

john --format=raw-sha256 --wordlist=/usr/share/wordlists/rockyou.txt hash3.txt



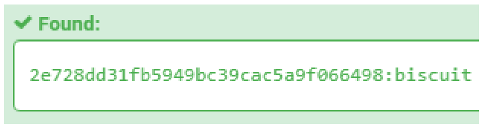
```
user@ip-10-10-89-180:~/John-the-Ripper-The-Basics/Task04$ john --format=raw-sh
a256 --wordlist=/usr/share/wordlists/rockyou.txt hash3.txt
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-SHA256 [SHA256 256/256 AVX2 8x])
Warning: poor OpenMP scalability for this hash type, consider --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 (2024-10-19 20:21) 25.00g/s 2457Kp/s 2457Kc/s 2457KC/s roza
lia..Dominic1
Use the "--show --format=Raw-SHA256" options to display all of the cracked pas
swords reliably
Session completed.
user@ip-10-10-89-180:~/John-the-Ripper-The-Basics/Task04$
```

john --format=whirlpool --wordlist=/usr/share/wordlists/rockyou.txt hash4.txt



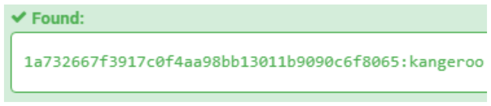
```
user@ip-10-10-89-180:~/John-the-Ripper-The-Basics/Task04$ john --format=whirlp
ool --wordlist=/usr/share/wordlists/rockyou.txt hash4.txt
Using default input encoding: UTF-8
Loaded 1 password hash (whirlpool [WHIRLPOOL 32/64])
Warning: poor OpenMP scalability for this hash type, consider --fork=2
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 (2024-10-19 20:29) 2.128g/s 1446Kp/s 1446Kc/s 1446KC/s cool
dog12..chata1994
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
user@ip-10-10-89-180:~/John-the-Ripper-The-Basics/Task04$
```

Another option is to decrypt the hashes with this tool: [Decrypt MD5, SHA1, MySQL, NTLM, SHA256, MD5 Email, SHA256 Email, SHA512, Wordpress, Bcrypt hashes for free online]



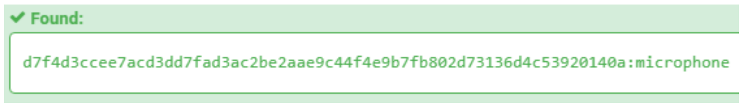
✔ Found:

```
2e728dd31fb5949bc39cac5a9f066498:biscuit
```



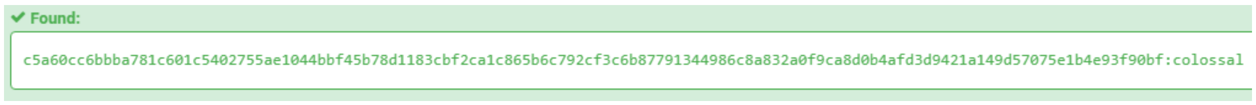
✔ Found:

```
1a732667f3917c0f4aa98bb13011b9090c6f8065:kangeroo
```



✔ Found:

```
d7f4d3ccee7acd3dd7fad3ac2be2aae9c44f4e9b7fb802d73136d4c53920140a:microphone
```


Found:

```
c5a60cc6bbba781c601c5402755ae1044bbf45b78d1183cbf2ca1c865b6c792cf3c6b87791344986c8a832a0f9ca8d0b4afd3d9421a149d57075e1b4e93f90bf:colossal
```

Question 3: What type of hash is hash1.txt?

Answer: *md5*

Question 4: What is the cracked value of hash1.txt?

Answer: *biscuit*

Question 5: What type of hash is hash2.txt?

Answer: *sha1*

Hint: Write the name without the dash "-".

Question 6: What is the cracked value of hash2.txt?

Answer: *kangeroo*

Question 7: What type of hash is hash3.txt?

Answer: *sha256*

Hint: Write the name without the dash "-".

Question 8: What is the cracked value of hash3.txt?

Answer: *microphone*

Question 9: What type of hash is hash4.txt?

Answer: *whirlpool*

Hint: It is not SHA-512.

Question 10: What is the cracked value of hash4.txt?

Answer: *colossal*

Hint: You won't need the "raw" prefix for this one.

Task 5: Cracking Windows Authentication Hashes

Now that we understand the basic syntax and usage of John the Ripper, let's move on to cracking something a little bit more complicated, something that you may even want to attempt if you're on an actual Penetration Test or Red Team engagement. Authentication hashes are the hashed versions of passwords stored by operating systems; it is sometimes possible to crack them using our brute-force methods. To get your hands on these hashes, you must often already be a privileged user, so we will explain some of the hashes we plan on cracking as we attempt them.

NTHash / NTLM

NThash is the hash format modern Windows operating system machines use to store user and service passwords. It’s also commonly referred to as NTLM, which references the previous version of Windows format for hashing passwords known as LM, thus NT/LM.

A bit of history: the NT designation for Windows products originally meant New Technology. It was used starting with Windows NT to denote products not built from the MS-DOS Operating System. Eventually, the “NT” line became the standard Operating System type to be released by Microsoft, and the name was dropped, but it still lives on in the names of some Microsoft technologies.

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**. You may not have to crack the hash to continue privilege escalation, as you can often conduct a “pass the hash” attack instead, but sometimes, hash cracking is a viable option if there is a weak password policy.

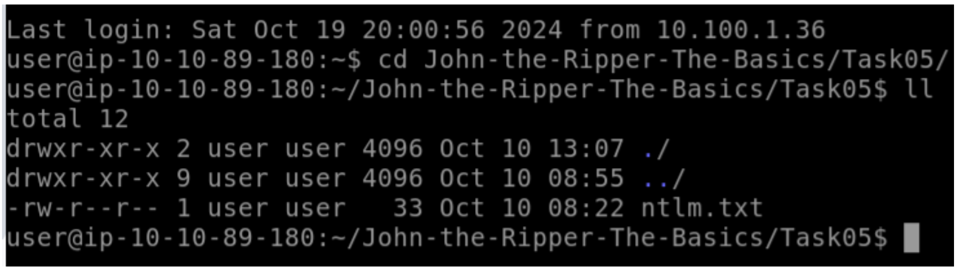
Practical

Now that you know the theory behind it, see if you can use the techniques we practised in the last task and the knowledge of what type of hash this is to crack the **ntlm.txt** file! The file is located in **~/John-the-Ripper-The-Basics/Task05/**.

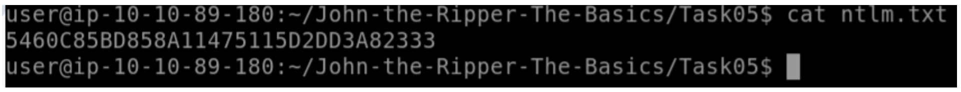
***cd* John-the-Ripper-The-Basics/Task05/**

ll

cat ntlm.txt



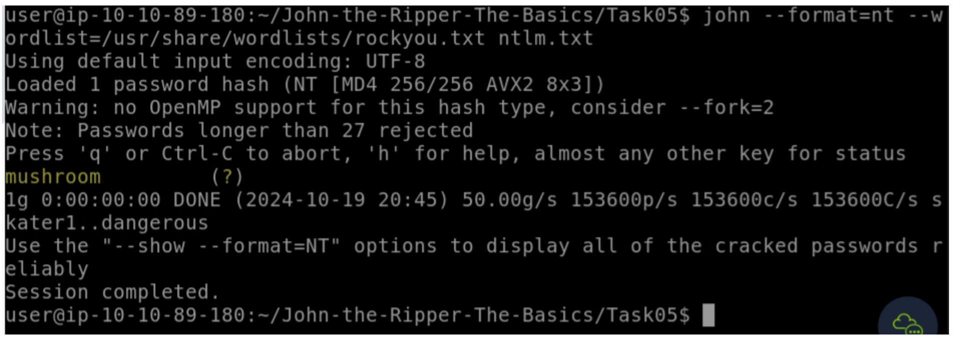
```
Last login: Sat Oct 19 20:00:56 2024 from 10.100.1.36
user@ip-10-10-89-180:~$ cd John-the-Ripper-The-Basics/Task05/
user@ip-10-10-89-180:~/John-the-Ripper-The-Basics/Task05$ ll
total 12
drwxr-xr-x 2 user user 4096 Oct 10 13:07 ./
drwxr-xr-x 9 user user 4096 Oct 10 08:55 ../
-rw-r--r-- 1 user user   33 Oct 10 08:22 ntlm.txt
user@ip-10-10-89-180:~/John-the-Ripper-The-Basics/Task05$ 
```



```
user@ip-10-10-89-180:~/John-the-Ripper-The-Basics/Task05$ cat ntlm.txt
5460C85BD858A11475115D2DD3A82333
user@ip-10-10-89-180:~/John-the-Ripper-The-Basics/Task05$ 
```

5460C85BD858A11475115D2DD3A82333

`john --format=nt --wordlist=/usr/share/wordlists/rockyou.txt ntlm.txt`



```
user@ip-10-10-89-180:~/John-the-Ripper-The-Basics/Task05$ john --format=nt --w
ordlist=/usr/share/wordlists/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 (2024-10-19 20:45) 50.00g/s 153600p/s 153600c/s 153600C/s s
kater1..dangerous
Use the "--show --format=NT" options to display all of the cracked passwords r
eliably
Session completed.
user@ip-10-10-89-180:~/John-the-Ripper-The-Basics/Task05$
```

Question 11: What do we need to set the --*format* flag to in order to crack this hash?

Answer: *nt*

Question 12: What is the cracked value of this password?

Answer: *mushroom*

Task 6: Cracking /etc/shadow Hashes

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. It contains one entry per line for each user or user account of the system. This file is usually only accessible by the root user, so you must have sufficient privileges to access the hashes. However, if you do, there is a chance that you will be able to crack some of 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

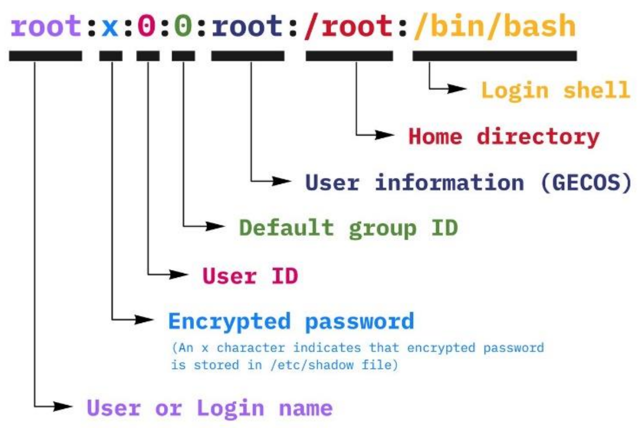
Contains the **/etc/passwd** line for the root user:

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

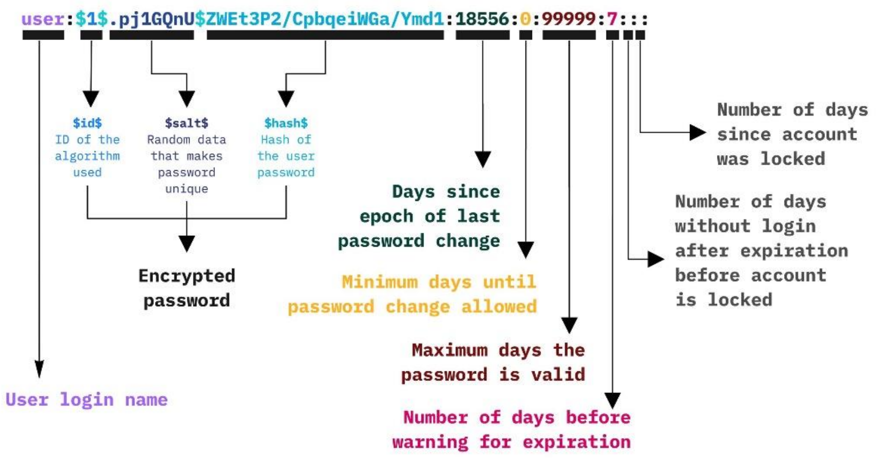
FILE 2 - local_shadow

Contains the **/etc/shadow** line for the root user: **root:$6$2nwjN454g.dv4HN/$m9Z/r2xVfweYVkrr.v5Ft8Ws3/YYksfNwq96UL1FX0OJjY1L6l.DS3KEVsZ9rOVLB/ldTeEL/OIhJZ4GMFMGA0:18576::::::**

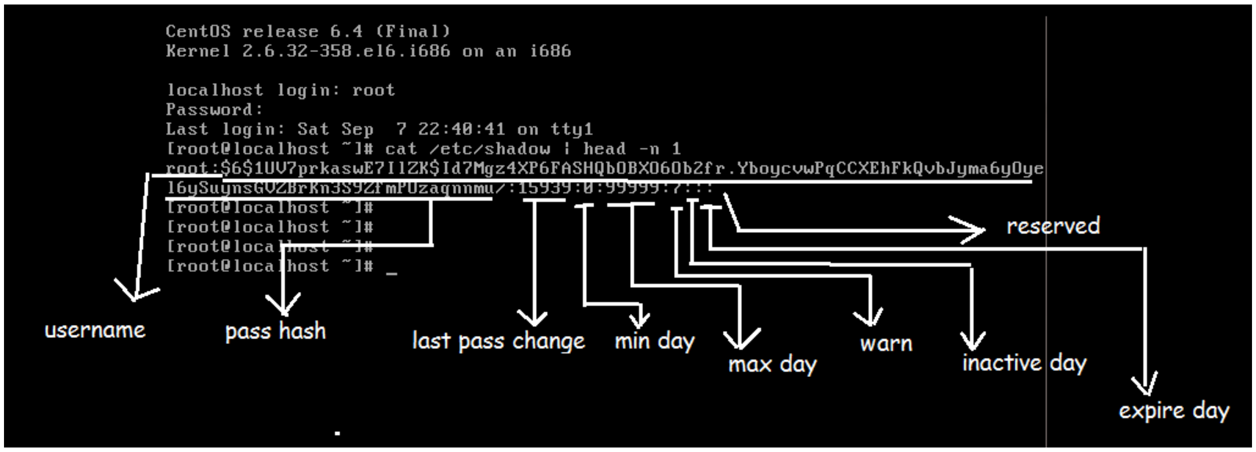
An entry from '/etc/passwd' explained: The ***passwd*** file contains information about the users, their login name, user and group IDs, home directory and other information.


root:x:0:0:root:/root:/bin/bash
Login shell
Home directory
User information (GECOS)
Default group ID
User ID
Encrypted password
(An x character indicates that encrypted password is stored in /etc/shadow file)
User or Login name

An entry from '/etc/shadow' explained: The ***shadow*** file contains the actual user encrypted password along with other information.


user:$1$.pj1GQnU$ZWEt3P2/CpbqeiWGa/Ymd1:18556:0:99999:7:::
id
ID of the algorithm used
$salt$
Random data that makes password unique
$hash$
Hash of the user password
Encrypted password
User login name
Days since epoch of last password change
Minimum days until password change allowed
Maximum days the password is valid
Number of days before warning for expiration
Number of days since account was locked
Number of days without login after expiration before account is locked

Another example:



```
CentOS release 6.4 (Final)
Kernel 2.6.32-358.el6.i686 on an i686

localhost login: root
Password:
Last login: Sat Sep  7 22:40:41 on tty1
[root@localhost ~]# cat /etc/shadow | head -n 1
root:$6$1UV7prkaswE7IlZK$Id7Mgz4XP6FASHQbOBXO6Ob2fr.YboycvwPqCCXEhFkQvbJyma6yOye
l6ySuynsGVZBrKn3S9ZfmPUzaqnnmu/:15939:0:99999:7:::
[root@localhost ~]#
[root@localhost ~]#
[root@localhost ~]#
[root@localhost ~]# _
```

Cracking

We can then feed the output from **unshadow**, in our example use case called **unshadowed.txt**, directly into John. We should not need to specify a mode here as we have made the input specifically for John; however, in some cases, you will need to specify the format as we have done previously using: **--format=sha512crypt**

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

Practical

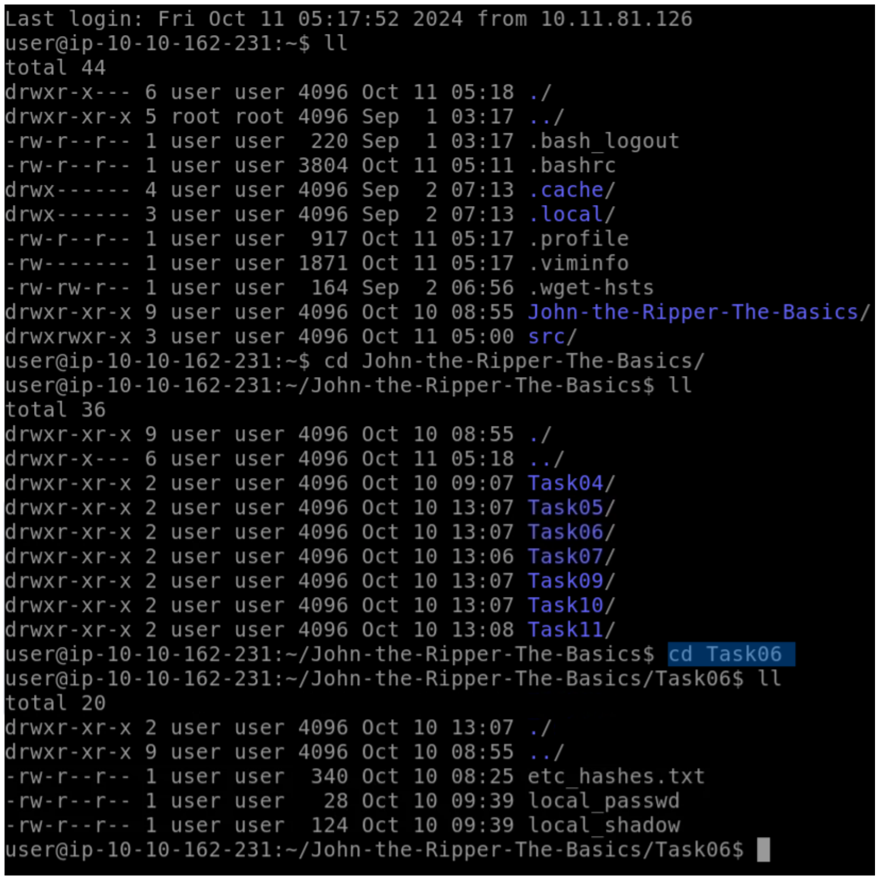
Now, see if you can follow the process to crack the password hash of the root user provided in the **etchashes.txt** file. Good luck! The files are located in **~/John-the-Ripper-The-Basics/Task06/**.

ll

cd John-the-Ripper-The-Basics/

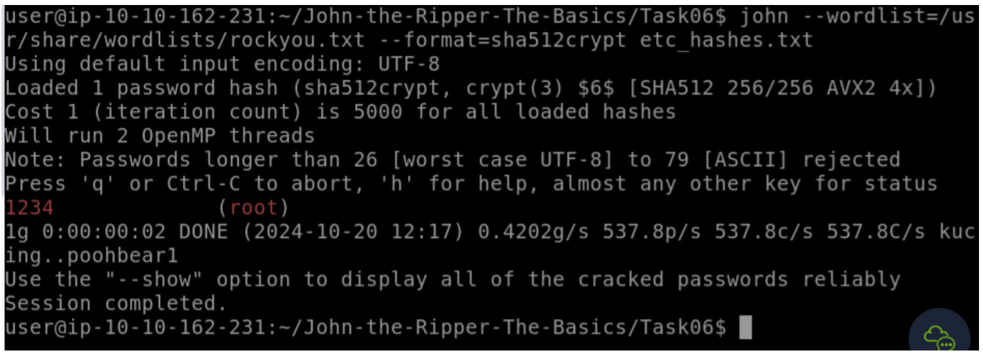
cd Task06

ll



```
Last login: Fri Oct 11 05:17:52 2024 from 10.11.81.126
user@ip-10-10-162-231:~$ ll
total 44
drwxr-x--- 6 user user 4096 Oct 11 05:18 ./
drwxr-xr-x 5 root root 4096 Sep  1 03:17 ../
-rw-r--r-- 1 user user  220 Sep  1 03:17 .bash_logout
-rw-r--r-- 1 user user 3804 Oct 11 05:11 .bashrc
drwx------ 4 user user 4096 Sep  2 07:13 .cache/
drwx------ 3 user user 4096 Sep  2 07:13 .local/
-rw-r--r-- 1 user user  917 Oct 11 05:17 .profile
-rw------- 1 user user 1871 Oct 11 05:17 .viminfo
-rw-rw-r-- 1 user user  164 Sep  2 06:56 .wget-hsts
drwxr-xr-x 9 user user 4096 Oct 10 08:55 John-the-Ripper-The-Basics/
drwxrwxr-x 3 user user 4096 Oct 11 05:00 src/
user@ip-10-10-162-231:~$ cd John-the-Ripper-The-Basics/
user@ip-10-10-162-231:~/John-the-Ripper-The-Basics$ ll
total 36
drwxr-xr-x 9 user user 4096 Oct 10 08:55 ./
drwxr-x--- 6 user user 4096 Oct 11 05:18 ../
drwxr-xr-x 2 user user 4096 Oct 10 09:07 Task04/
drwxr-xr-x 2 user user 4096 Oct 10 13:07 Task05/
drwxr-xr-x 2 user user 4096 Oct 10 13:07 Task06/
drwxr-xr-x 2 user user 4096 Oct 10 13:06 Task07/
drwxr-xr-x 2 user user 4096 Oct 10 13:07 Task09/
drwxr-xr-x 2 user user 4096 Oct 10 13:07 Task10/
drwxr-xr-x 2 user user 4096 Oct 10 13:08 Task11/
user@ip-10-10-162-231:~/John-the-Ripper-The-Basics$ cd Task06
user@ip-10-10-162-231:~/John-the-Ripper-The-Basics/Task06$ ll
total 20
drwxr-xr-x 2 user user 4096 Oct 10 13:07 ./
drwxr-xr-x 9 user user 4096 Oct 10 08:55 ../
-rw-r--r-- 1 user user  340 Oct 10 08:25 etc_hashes.txt
-rw-r--r-- 1 user user   28 Oct 10 09:39 local_passwd
-rw-r--r-- 1 user user  124 Oct 10 09:39 local_shadow
user@ip-10-10-162-231:~/John-the-Ripper-The-Basics/Task06$
```

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



```
user@ip-10-10-162-231:~/John-the-Ripper-The-Basics/Task06$ john --wordlist=/us
r/share/wordlists/rockyou.txt --format=sha512crypt etc_hashes.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 (2024-10-20 12:17) 0.4202g/s 537.8p/s 537.8c/s 537.8C/s kuc
ing..poohbear1
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
user@ip-10-10-162-231:~/John-the-Ripper-The-Basics/Task06$
```

Question 13: What is the root password?

Answer: ***1234***

Task 7: Single Crack Mode

So far, we've been using John's wordlist mode to brute-force simple and not-so-simple hashes. But 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.)

This technique is called word mangling. John is building its dictionary based on the information 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 on relevant factors for the target you're trying to crack. This exploits how poor passwords can be based on information about the username or the service they're logging into.

GECOS

John's implementation of word mangling also features compatibility with the GECOS field of the UNIX operating system, as well as other UNIX-like operating systems such as Linux. GECOS stands for General Electric Comprehensive Operating System. In the last task, 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. It stores general information about the user, such as the user's full name, office number, and telephone number, among other things. John can take information stored in those records, such as full name and home directory name, to add to the wordlist it generates when cracking `/etc/shadow` hashes with single crack mode.

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`

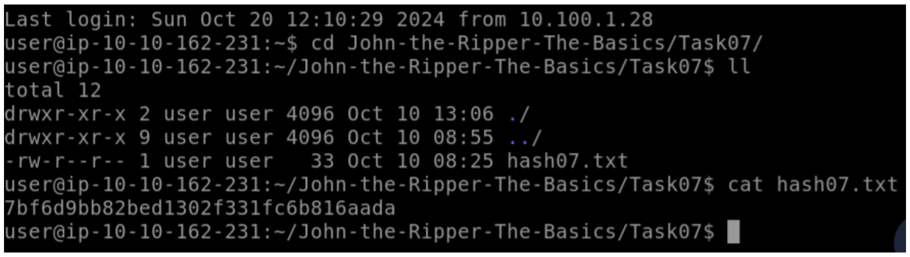
Practical

Now that you're familiar with the Syntax for John's single crack mode, access the hash and crack it, assuming that the user it belongs to is called "Joker". The file is located in **~/John-the-Ripper-The-Basics/Task07/**.

cd John-the-Ripper-The-Basics/Task07/

ll

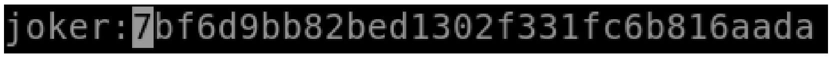
cat hash07.txt



```
Last login: Sun Oct 20 12:10:29 2024 from 10.100.1.28
user@ip-10-10-162-231:~$ cd John-the-Ripper-The-Basics/Task07/
user@ip-10-10-162-231:~/John-the-Ripper-The-Basics/Task07$ ll
total 12
drwxr-xr-x 2 user user 4096 Oct 10 13:06 ./
drwxr-xr-x 9 user user 4096 Oct 10 08:55 ../
-rw-r--r-- 1 user user   33 Oct 10 08:25 hash07.txt
user@ip-10-10-162-231:~/John-the-Ripper-The-Basics/Task07$ cat hash07.txt
7bf6d9bb82bed1302f331fc6b816aada
user@ip-10-10-162-231:~/John-the-Ripper-The-Basics/Task07$ 
```

vi hash07.txt

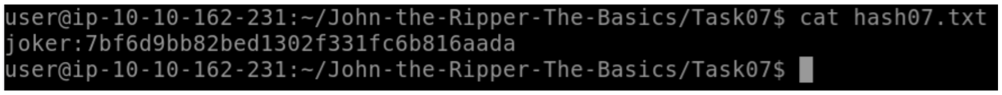
Add the user in front of the hash, save & close.



```
joker:7bf6d9bb82bed1302f331fc6b816aada
```

joker:7bf6d9bb82bed1302f331fc6b816aada

Now, verify the changes with ***cat hash07.txt***



```
user@ip-10-10-162-231:~/John-the-Ripper-The-Basics/Task07$ cat hash07.txt
joker:7bf6d9bb82bed1302f331fc6b816aada
user@ip-10-10-162-231:~/John-the-Ripper-The-Basics/Task07$ 
```

In Task 4 we have the python3 hash-identifier. Let's see the format of joker's hash.

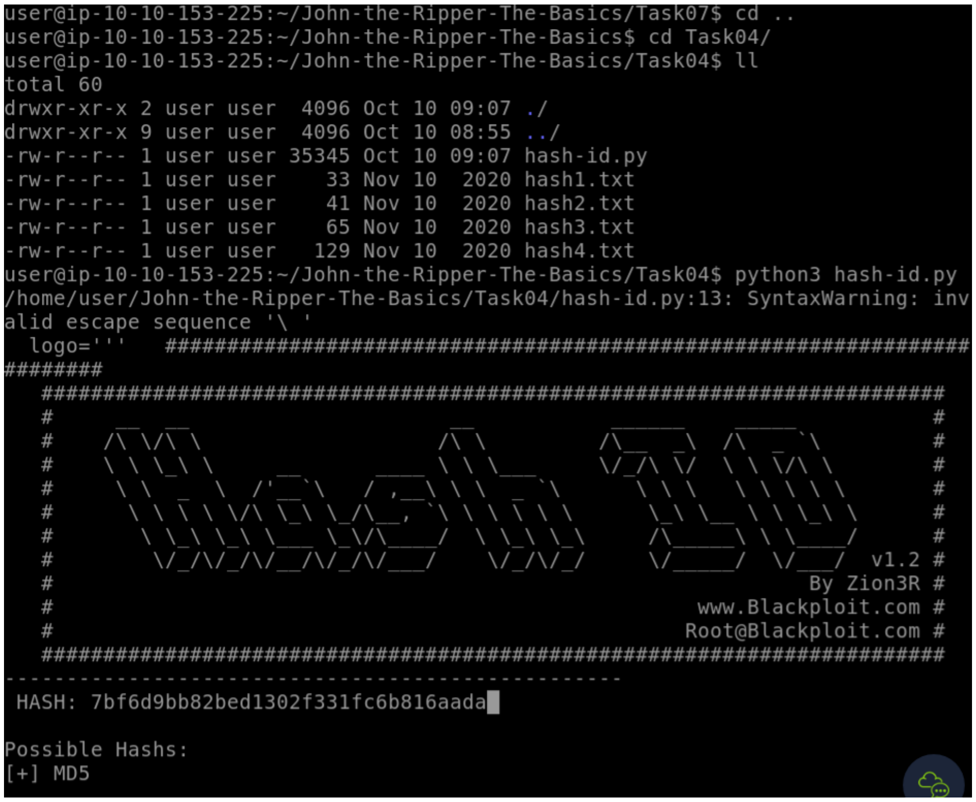
cd ..

cd Task04/

ll

python3 hash-id.py

7bf6d9bb82bed1302f331fc6b816aada

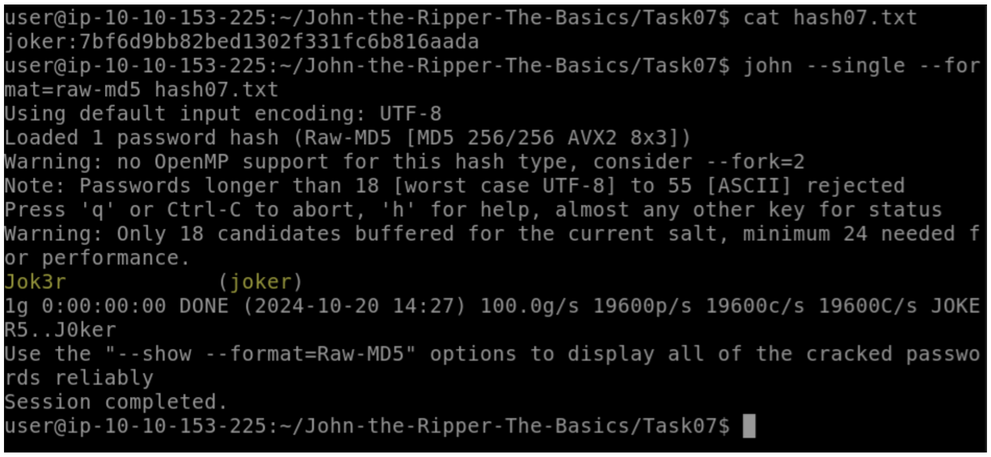


```
user@ip-10-10-153-225:~/John-the-Ripper-The-Basics/Task07$ cd ..
user@ip-10-10-153-225:~/John-the-Ripper-The-Basics$ cd Task04/
user@ip-10-10-153-225:~/John-the-Ripper-The-Basics/Task04$ ll
total 60
drwxr-xr-x 2 user user  4096 Oct 10 09:07 ./
drwxr-xr-x 9 user user  4096 Oct 10 08:55 ../
-rw-r--r-- 1 user user 35345 Oct 10 09:07 hash-id.py
-rw-r--r-- 1 user user    33 Nov 10  2020 hash1.txt
-rw-r--r-- 1 user user    41 Nov 10  2020 hash2.txt
-rw-r--r-- 1 user user    65 Nov 10  2020 hash3.txt
-rw-r--r-- 1 user user   129 Nov 10  2020 hash4.txt
user@ip-10-10-153-225:~/John-the-Ripper-The-Basics/Task04$ python3 hash-id.py
/home/user/John-the-Ripper-The-Basics/Task04/hash-id.py:13: SyntaxWarning: inv
alid escape sequence '\ '
  logo='''   ##############################################################
########
   #########################################################################
   #     __  __                     __           ______    _____           #
   #    /\ \/\ \                   /\ \         /\__  _\  /\  _ `\         #
   #    \ \ \_\ \     __      ____ \ \ \___     \/_/\ \/  \ \ \/\ \        #
   #     \ \  _  \  /'__`\   / ,__\ \ \  _ `\      \ \ \   \ \ \ \ \       #
   #      \ \ \ \ \/\ \_\ \_/\__, `\ \ \ \ \ \      \_\ \__ \ \ \_\ \      #
   #       \ \_\ \_\ \___ \_\/\____/  \ \_\ \_\     /\_____\ \ \____/      #
   #        \/_/\/_/\/__/\/_/\/___/    \/_/\/_/     \/_____/  \/___/  v1.2 #
   #                                                             By Zion3R #
   #                                                    www.Blackploit.com #
   #                                                   Root@Blackploit.com #
   #########################################################################
--------------------------------------------------
 HASH: 7bf6d9bb82bed1302f331fc6b816aada

Possible Hashs:
[+] MD5
```

So, MD5 is the format. Let's return to Task07 and use the command:

john --single --format=raw-md5 hash07.txt



```
user@ip-10-10-153-225:~/John-the-Ripper-The-Basics/Task07$ cat hash07.txt
joker:7bf6d9bb82bed1302f331fc6b816aada
user@ip-10-10-153-225:~/John-the-Ripper-The-Basics/Task07$ john --single --for
mat=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 f
or performance.
Jok3r            (joker)
1g 0:00:00:00 DONE (2024-10-20 14:27) 100.0g/s 19600p/s 19600c/s 19600C/s JOKE
R5..J0ker
Use the "--show --format=Raw-MD5" options to display all of the cracked passwo
rds reliably
Session completed.
user@ip-10-10-153-225:~/John-the-Ripper-The-Basics/Task07$
```

Question 14: What is Joker's password?

Answer: *Jok3r*

Task 8: Custom Rules

What are Custom Rules?

As we explored what John can do in Single Crack Mode, you may have some ideas about some good mangling patterns or what patterns your passwords often use that could be replicated with a particular mangling pattern. The good news is that you can define your rules, which John will use to create passwords dynamically. 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 organizations will require a certain level of password complexity to try and combat dictionary attacks. In other words, when creating a new account or changing your password, if you attempt a password like `polopassword`, it will most likely not work. The reason would be the enforced 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!`

Consider the password with a capital letter first and a number followed by a symbol at the end. This familiar pattern of the password, appended and prepended by modifiers (such as capital letters or symbols), is a memorable pattern that people use and reuse when creating passwords. This pattern can let us exploit password complexity predictability.

Now, this does meet the password complexity requirements; however, 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`.

Let's go over the syntax of these custom rules, using the example above as our target pattern. Note that you can define a massive level of granular control in these rules. I suggest looking at the wiki here to get a full view of the modifiers you can use and more examples of rule implementation.

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 a John argument.

We then use a regex style pattern match to define where the word will be modified; again, we will only cover the primary 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

These can be used in combination to define where and what in the word you want to modify.

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 **" "**. 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

Please note that:

- **[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] [!£$%@]"
```

Utilizes the following:

- **c**: Capitalizes the first letter
- **Az**: Appends to the end of the word
- **[0-9]**: A number in the range 0-9
- **[!£$%@]**: The password is followed by one of these symbols

Using Custom Rules

We could 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]`

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.

Jumbo John already has an extensive list of custom rules containing modifiers for use in almost all cases. If you get stuck, try looking at those rules [around line 678] if your syntax isn't working correctly.

Question 15: What do custom rules allow us to exploit?

Answer: ***Password complexity predictability***

Question 16: What rule would we use to add all capital letters to the end of the word?

Answer: ***Az"[A-Z]"***

Question 17: What flag would we use to call a custom rule called THMRules?

Answer: ***--rule=THMRules***

Task 9: Cracking Password Protected Zip Files

Yes! You read that right. We can use John to crack the password on password-protected Zip files. Again, we'll use a separate part of the John suite of tools to convert the Zip file into a format that John will understand, but we'll use the syntax you're already familiar with for all intents and purposes.

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.

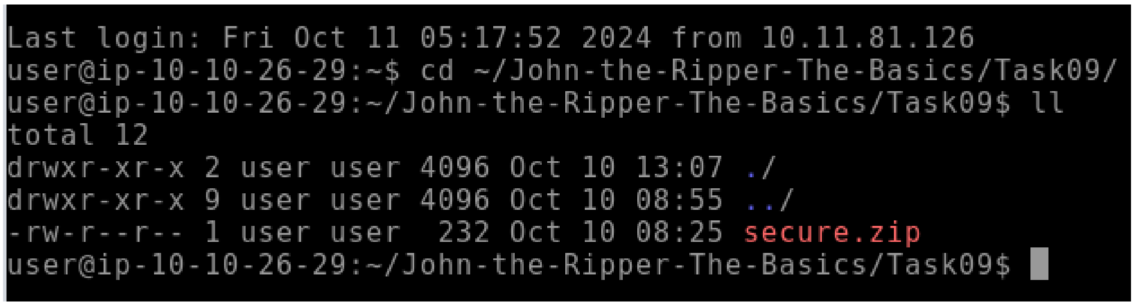
```
john --wordlist=/usr/share/wordlists/rockyou.txt zip_hash.txt
```

Practical

Now, have a go at cracking a “secure” Zip file! The file is located in **~/John-the-Ripper-The-Basics/Task09/**.

cd ~/John-the-Ripper-The-Basics/Task09/

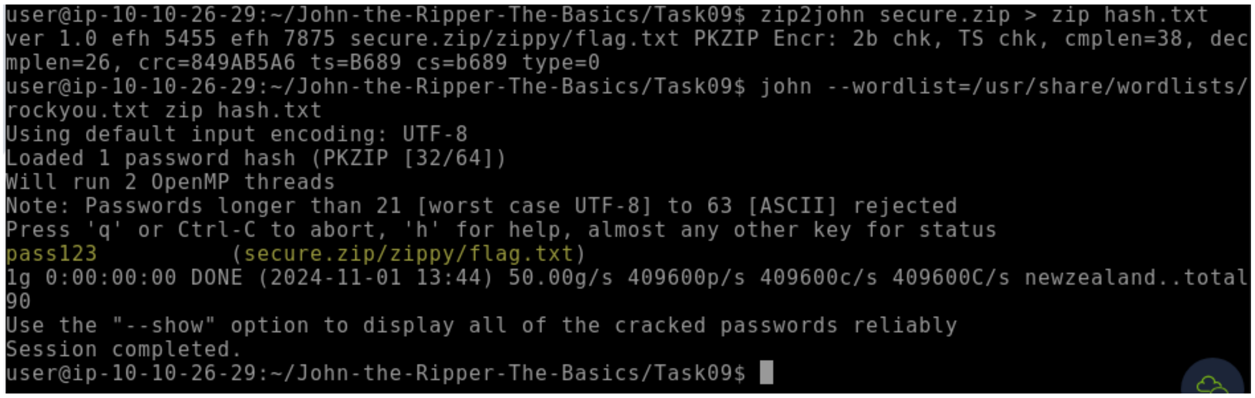
ll



```
Last login: Fri Oct 11 05:17:52 2024 from 10.11.81.126
user@ip-10-10-26-29:~$ cd ~/John-the-Ripper-The-Basics/Task09/
user@ip-10-10-26-29:~/John-the-Ripper-The-Basics/Task09$ ll
total 12
drwxr-xr-x 2 user user 4096 Oct 10 13:07 ./
drwxr-xr-x 9 user user 4096 Oct 10 08:55 ../
-rw-r--r-- 1 user user  232 Oct 10 08:25 secure.zip
user@ip-10-10-26-29:~/John-the-Ripper-The-Basics/Task09$
```

zip2john secure.zip > zip_hash.txt

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

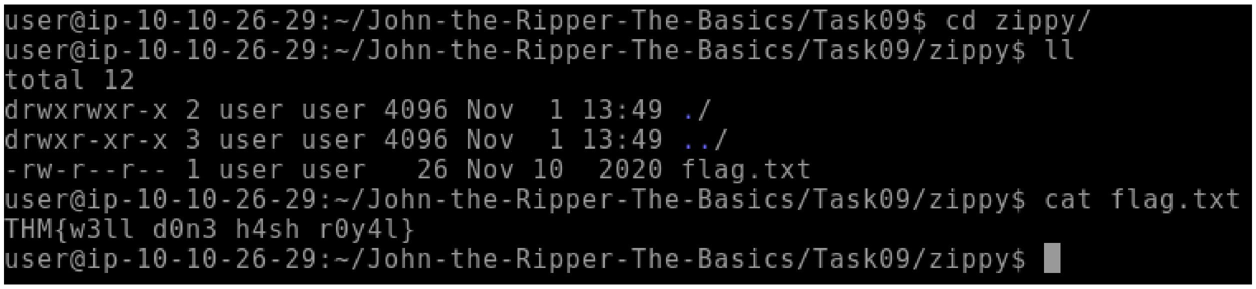


```
user@ip-10-10-26-29:~/John-the-Ripper-The-Basics/Task09$ zip2john secure.zip > zip hash.txt
ver 1.0 efh 5455 efh 7875 secure.zip/zippy/flag.txt PKZIP Encr: 2b chk, TS chk, cmplen=38, dec
mplen=26, crc=849AB5A6 ts=B689 cs=b689 type=0
user@ip-10-10-26-29:~/John-the-Ripper-The-Basics/Task09$ john --wordlist=/usr/share/wordlists/
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 (2024-11-01 13:44) 50.00g/s 409600p/s 409600c/s 409600C/s newzealand..total
90
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
user@ip-10-10-26-29:~/John-the-Ripper-The-Basics/Task09$
```

unzip secure.zip

cd zippy

cat flag.txt



```
user@ip-10-10-26-29:~/John-the-Ripper-The-Basics/Task09$ cd zippy/
user@ip-10-10-26-29:~/John-the-Ripper-The-Basics/Task09/zippy$ ll
total 12
drwxrwxr-x 2 user user 4096 Nov  1 13:49 ./
drwxr-xr-x 3 user user 4096 Nov  1 13:49 ../
-rw-r--r-- 1 user user   26 Nov 10  2020 flag.txt
user@ip-10-10-26-29:~/John-the-Ripper-The-Basics/Task09/zippy$ cat flag.txt
THM{w3ll d0n3 h4sh r0y4l}
user@ip-10-10-26-29:~/John-the-Ripper-The-Basics/Task09/zippy$
```

Question 18: What is the password for the secure.zip file?

Answer: ***pass123***

Question 19: What are the contents of the flag inside the zip file?

Answer: *THM{w3ll_d0n3_h4sh_r0y4l}*

Task 10: Cracking Password-Protected RAR Archives

Cracking a Password-Protected RAR Archive

We can use a similar process to the one we used in the last task to obtain the password for RAR archives. If you aren't familiar, RAR archives are compressed files created by the WinRAR archive manager. Like Zip files, they compress folders and files.

Rar2John

Almost identical to the `zip2john` tool, we 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`.

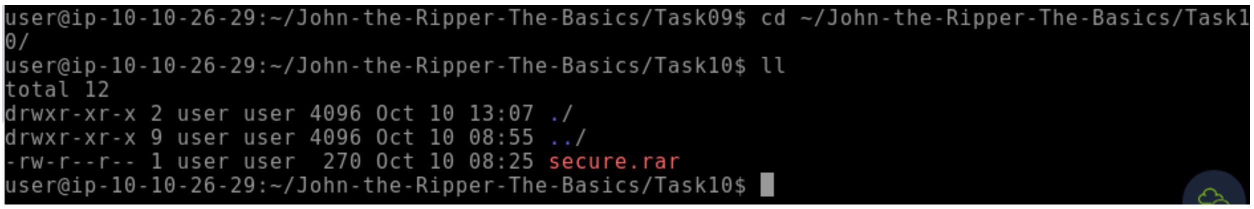
```
john --wordlist=/usr/share/wordlists/rockyou.txt rar_hash.txt
```

Practical

Now, have a go at cracking a "secure" RAR file! The file is located in `~/John-the-Ripper-The-Basics/Task10/`.

cd ~/John-the-Ripper-The-Basics/Task10/

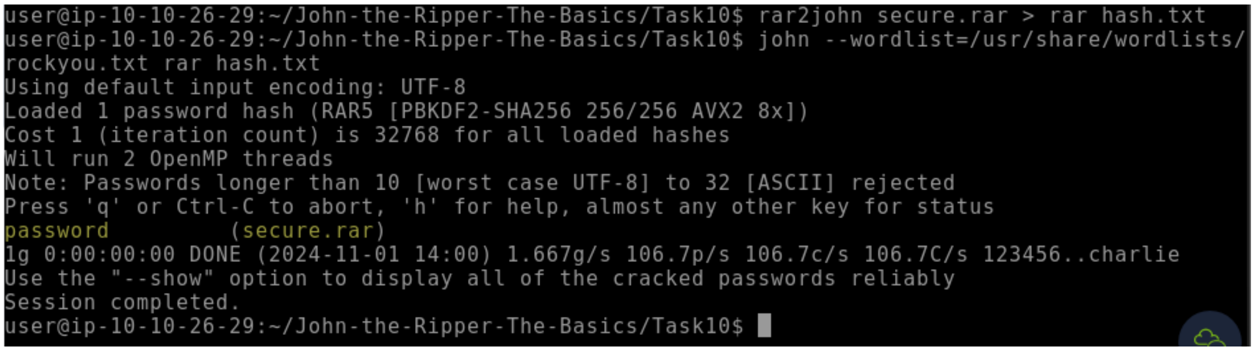
ll



```
user@ip-10-10-26-29:~/John-the-Ripper-The-Basics/Task09$ cd ~/John-the-Ripper-The-Basics/Task1
0/
user@ip-10-10-26-29:~/John-the-Ripper-The-Basics/Task10$ ll
total 12
drwxr-xr-x 2 user user 4096 Oct 10 13:07 ./
drwxr-xr-x 9 user user 4096 Oct 10 08:55 ../
-rw-r--r-- 1 user user  270 Oct 10 08:25 secure.rar
user@ip-10-10-26-29:~/John-the-Ripper-The-Basics/Task10$ █
```

rar2john secure.rar > rar_hash.txt

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

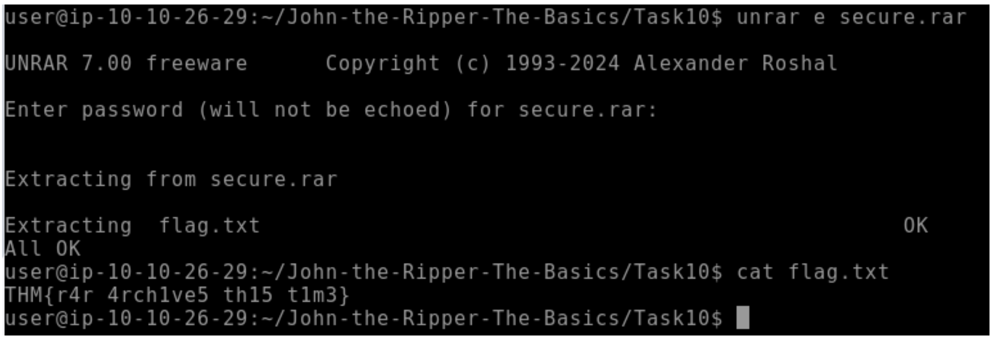


```
user@ip-10-10-26-29:~/John-the-Ripper-The-Basics/Task10$ rar2john secure.rar > rar_hash.txt
user@ip-10-10-26-29:~/John-the-Ripper-The-Basics/Task10$ john --wordlist=/usr/share/wordlists/
rockyou.txt rar_hash.txt
Using default input encoding: UTF-8
Loaded 1 password hash (RAR5 [PBKDF2-SHA256 256/256 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 (2024-11-01 14:00) 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.
user@ip-10-10-26-29:~/John-the-Ripper-The-Basics/Task10$ █
```

If you don't have unrar (sudo apt-get install unrar)

unrar e secure.rar

cat flag.txt



```
user@ip-10-10-26-29:~/John-the-Ripper-The-Basics/Task10$ unrar e 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-10-10-26-29:~/John-the-Ripper-The-Basics/Task10$ cat flag.txt
THM{r4r 4rch1ve5 th15 t1m3}
user@ip-10-10-26-29:~/John-the-Ripper-The-Basics/Task10$ █
```

Question 20: What is the password for the secure.rar file?

Answer: *password*

Question 21: What are the contents of the flag inside the zip file?

Answer: *THM{w3ll_d0n3_h4sh_r0y4l}*

Task 11: Cracking SSH Keys with John

Cracking SSH Key Passwords

Okay, okay, I hear you. There are no more file archives! Fine! Let's explore one more use of John that comes up semi-frequently in CTF challenges—using John to crack the SSH private key password of **id_rsa** files. Unless configured otherwise, you authenticate your SSH login using a password. However, 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; here, we will be using John to crack this password to allow authentication over SSH using the key.

SSH2John

Who could have guessed it, another conversion tool? Well, that's what working with John is all about. As the name suggests, **ssh2john** converts the **id_rsa** private key, which is used to log in to the SSH session, into a hash format that John can work with. Jokes aside, it's another beautiful example of John's versatility. The syntax is about what you'd expect. Note that 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

For the final time, 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:

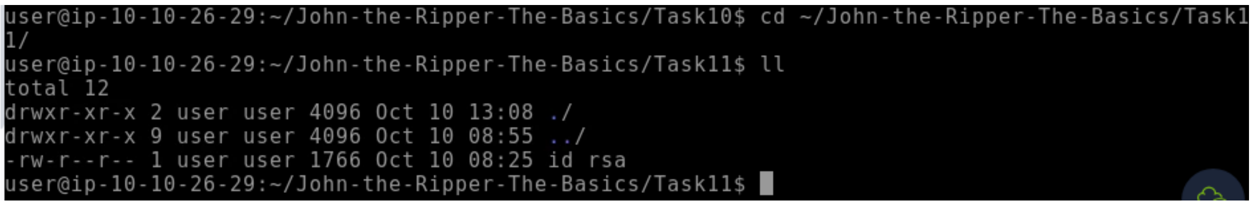
john --wordlist=/usr/share/wordlists/rockyou.txt id_rsa_hash.txt

Practical

Now, I'd like you to crack the hash of the **id_rsa** file relevant to this task! The file is located in **~/John-the-Ripper-The-Basics/Task11/**.

cd ~/John-the-Ripper-The-Basics/Task11/

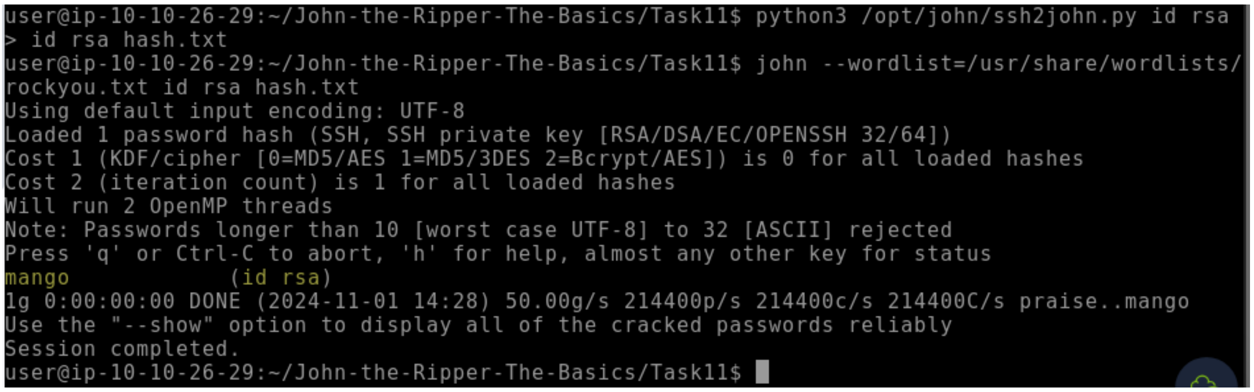
ll



```
user@ip-10-10-26-29:~/John-the-Ripper-The-Basics/Task10$ cd ~/John-the-Ripper-The-Basics/Task1
1/
user@ip-10-10-26-29:~/John-the-Ripper-The-Basics/Task11$ ll
total 12
drwxr-xr-x 2 user user 4096 Oct 10 13:08 ./
drwxr-xr-x 9 user user 4096 Oct 10 08:55 ../
-rw-r--r-- 1 user user 1766 Oct 10 08:25 id rsa
user@ip-10-10-26-29:~/John-the-Ripper-The-Basics/Task11$
```

python3 /opt/john/ssh2john.py id_rsa > id_rsa_hash.txt

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



```
user@ip-10-10-26-29:~/John-the-Ripper-The-Basics/Task11$ python3 /opt/john/ssh2john.py id rsa
> id rsa hash.txt
user@ip-10-10-26-29:~/John-the-Ripper-The-Basics/Task11$ john --wordlist=/usr/share/wordlists/
rockyou.txt id rsa hash.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 (2024-11-01 14:28) 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.
user@ip-10-10-26-29:~/John-the-Ripper-The-Basics/Task11$
```

Question 22: What is the SSH private key password?

Answer: *mango*

Happy Hacking! 😸

> *Thanks and Regards,*
>
> *ShadowGirl* 🦹‍♀️😉