# John the Ripper Password Cracker Workshop

Hi, there fella, today we’ll be learning about how to crack passwords by using John the Ripper, but before we dive into that, first we need to understand how passwords are created.





**Have you heard about cryptography?**

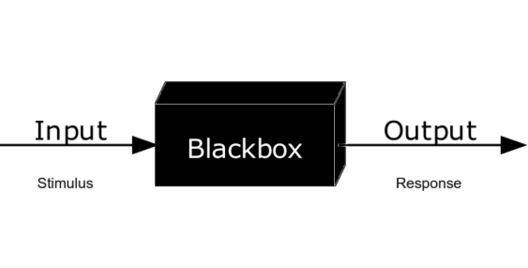
**What is cryptography?**

Cryptography is the study of secure communication techniques that allow only two people(sender and intended) to know the message that lies within. One of the earliest cryptographic techniques is the Ceasar cipher, which is just simply rotating letters N times of the alphabet, for example, “deaf” would be “ghdi ”. Nowadays we use more complex cryptography techniques and algorithms to safely secure messages, a couple of examples are base64 cipher, AES, DES, MD5, Blowfish, SHA-256.

**What is a Hash? How do they work?**

A hash is a random output from a given input that went through a hash algorithm or hash function, this function will create a unique hash that belongs to the initial input. Here are some images to illustrate it better, and we’ll start by using a black box.

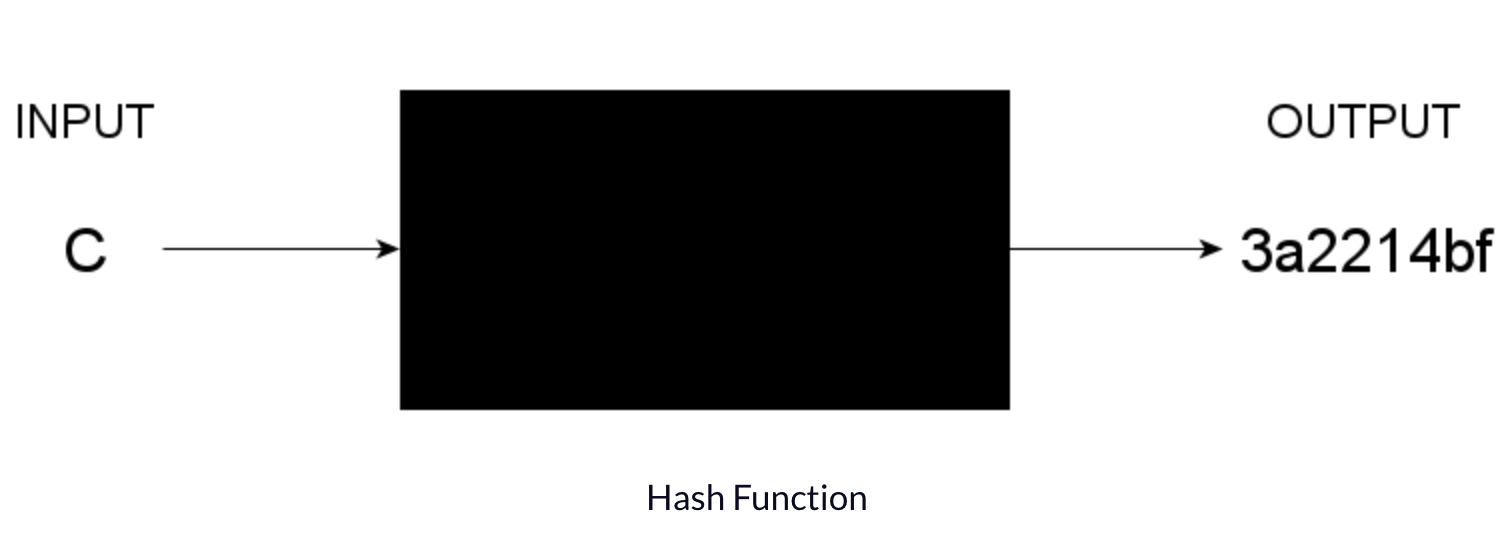
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**What is a black box?**

A black box allows for a system to be viewed as inputs and outputs without knowing how it’s internal components achieve it.

The main purpose behind it is to have a clear understanding of what the inputs are and what the desired output should be.

In this next image we can see how the input goes through the box which is the hash function and the output is different from the input.

****

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

A simple formula we will use to create a hash function is:

**h(x) = x %(mod) 9**

We will use the following example:

h(96) = 96 % 9 = 6

In this case, the input is 96 and the output of our hash function is 6. However this simple function has a flaw, it can create collisions. It is important that only one input is mapped to each output.

## 1. Create Your Own Simple Hash

You are given the following inputs:

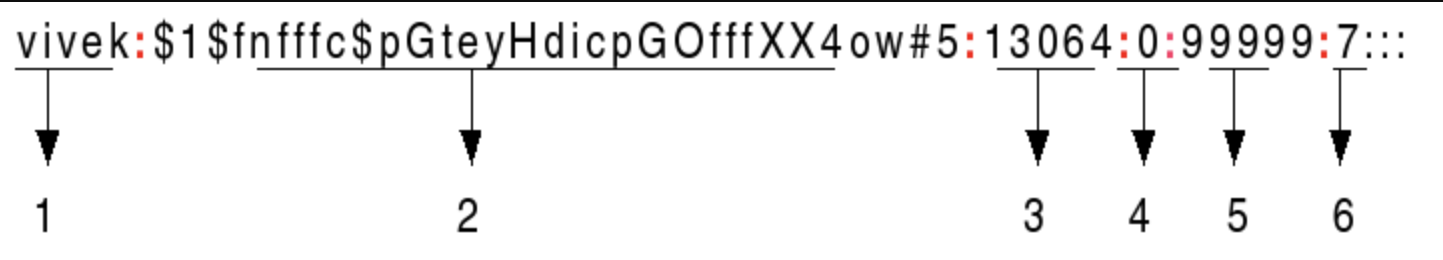
9679, 4199, 1479, 5555

1. **Using the hash function: h(x) = x mod 10”, which inputs will have a collision?**
2. **Why do you think collisions could be bad?**

## 2. Understanding /etc/shadow File

The /etc/shadow file stores user passwords on Linux Systems in an encrypted format. These passwords come from the /etc/passwd file which is a plain text file containing an information profile for each user password.

The following is an example encrypted password stored in the /etc/shadow file.



There is one encrypted password per line, and each field is seperated with a “:” colon.

The data in the fields represent different things:

1. **Username**
2. **Password** (encrypted), in general the password format is: $id$salt$hashed, the $id represents the algorithm used to create the hashed value from the plaintext password.

Here are a few of the $id used on Linux:

$1$ is MD5

$2a$ is Blowfish

$2y$ is Blowfish

$5$ is SHA-256

$6$ is SHA-512

1. **Last Password change:** Days since password has been changed.
2. **Minimum**: The minimum number of days required until a password can be changed again.
3. **Maximum**: The maximum amount of days a user can go without changing a password.
4. **Warn**: The number of days before password expiry when a user will be notified.

You will be using this information on the /etc/shadow file to:

* + Find the /etc/shadow file
  + Identify hash encryptions used on the user passwords
  + Use **John the Ripper** to attempt to crack the user passwords

Using the table below, observe the plaintext passwords next to their corresponding shadow file entries.

1. **In the space provided on the table, determine which algorithm each password is encrypted in, and rank from shortest to longest how long they will take to crack.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Hash Algorithm** | **Rank** | **User Data** | **Shadow File Entries** |
|  |  | User1: kali  Password: kali | kali:$1$QvPAw9OBt..VnSAU$wyeJkkHdxeVtsP7GVe7neOWclI/8PEWHKTHHvbuPUVH21sdOBLC.vcgrS0.WmeNs4sOLX0HeBDQto3YNwHBnV0:18470:0:99999:7::: |
|  |  | User2: lauren  Password: password123 | lauren:$2a$CgpGdeI1r/jDtQfO$ZQF25LJf3sgF0wCrBfAhDW.CUMLo4JnlIVyGNdhmz8NlZqTheRHa9aOqacQVWvycW.OQtfB4B9p2CDP5RmdyX1:18561:0:99999:7::: |
|  |  | User 3: aaron  Password: dogcat$! | aaron:$6$vhDmg172ZxJyTNBP$3OhmVGMD9OIhwPZ6Z4JTx74jYWDRmIBzfrxNM9qGIRMWuKu/kW034zf8ecpf3hEGocVbgtqEjjERwti4lxd0g1:18561:0:99999:7::: |
|  |  | User4: john  Password: cherry!149$ | john:$2y$6YbZJUWthZCsRBfx$3sSMUwmSseE4nOJY0ydJXPpq05HMQoCjPaI/3Y1ElqNre0oGpdHtyUbdrNV1VTYQpw8x65bX7mLzQkpYwiZxa1:18561:0:99999:7::: |
|  |  | User5: mary  Password: Atc459!!! | mary:$6$7Blv4iUwB.oOb/gr$ETz9urciN7TG4bmMpAGGvzuJTlwcurhnPdiTKyHtXSmJC35r7l5CwrHAtt6rtMCuhWbT4HKBLiTAsLoASfw6E0:18561:0:99999:7::: |
|  |  | User6: alice  Password: cDtoLKpy7uy!6k4mn$ | alice:$5$13/XlyAwQoSgZL7A$92QkJNrYyp60B4IkfHwZhHKJm32Rb44pmZcxJAw0.24py9W8S9d.nOF1hQljnd5UMtxkRqXdnyeiTERw1KSDd.:18561:0:99999:7:: |

1. **For the entry you picked to take the shortest amount of time to crack, explain your reasoning:**
2. **For the longest:**

## 3. Get the User Passwords

John the Ripper is a fast password cracking software, we will be using it to detect weak user passwords. It has the following modes:

* Wordlist: John will simply use a file with a wordlist that will be checked against the passwords
* Singlecrack: Uses user account data to guess passwords, and is faster than wordlist.
* Incremental: Most powerful mode, John will try any character combination to resolve the password. (Takes the longest)

1. Login to the Kali VM provided:  
     
    Username: **kali**

Password: **kali**

Open a terminal by right clicking on the desktop and select “Open terminal here”.

In order to utilize the John software, as well as perform the necessary steps to obtain the shadow file, you will need root access.

1. **Elevate to root:**

Enter: ***sudo su***

Enter the VM password.

Begin by using the “unshadow” command provided below, which combines the *passwd* and *shadow* files for John to access both the user information as well as the encrypted passwords.

1. **Use the unshadow function as well as create a temporary file to store the new data in:**  
     
    Enter: ***/usr/sbin/unshadow /etc/passwd /etc/shadow > passwords.txt***

**\*Note:** Carefully check your entry, you may not receive an error message if it is entered incorrectly.

To find instructions and commands for how to use John’s different modes:

Enter: ***john --help***

Your commands will be in the following format:

**john [OPTIONS] [PASSWORD-FILES]**

4. Crack Passwords Using John

1. **Begin by using John’s wordlist mode. A wordlist file has been provided on the Kali machine.**

Enter: ***john --wordlist=”words.txt” passwords.txt***

After each cracking mode use the following command to view the cracked passwords:

Enter: ***john -show passwords.txt***

\***Note**: The passwords will be displayed in the shadow file structure, described previously.

Next, try cracking using the single crack mode.

1. Write the command you used here:

Congratulations! You have cracked all the user passwords in the shadow file on this machine.

1. **List the passwords you found here:**

answer: wordlist mode: kali, dog

single crack mode: ilovecats, jennamarie

## 

## 5. Create your own password hash file:

1. **Begin by creating a .txt file to store the hashes in:**

Enter: ***touch hashes.txt***

Next, we will hash the passwords using a command in the following format:

**echo -n [password] | [hash type]sum | cut -d ‘ ‘ -f 1 >> [output file]**

Let's break down what this command is doing:

|  |  |
| --- | --- |
| echo -n [password] | Uses the ‘echo’ command to display a string on the standard output or file ‘-n’ is used to remove any newline characters from the end of the string [password] is where we place the password we want to hash. |
| **‘ | ’** | This is the symbol used to pipe, a pipe is a form of redirection, or transfer of standard output to some other destination, to send the output of one command/process to another command/process for further processing. |
| [hash type] | The name of hash algorithm |
| cut -d ‘ ‘ -f 1 >> [output file] | The cut command and it’s parameters is used to format the output of the previous pipe  >> is used to pipe the final output into a text file and append it to any existing text in that file  [output file] name of the file where the hash will be written. |

1. **Build the password hash file**

Add the following Passwords to your hash file:

|  |  |
| --- | --- |
| Hash Format: | Password: |
| md5 | 123456 |
| md5 | 7room! |
| md5 | 9/G9t9 |
| sha256 | b0--18 |
| sha256 | AA2vg |
| sha512 | 105346199 |

123456 - 0 seconds to crack

7room! - 20 seconds to crack

9/G9t9 - 41 seconds to crack

b0--18 - 30 seconds to crack

AA2vg - 40 seconds to crack

105346199 - 1:41 seconds to crack

1. **Which passwords do you think will take the longest to crack? Why?**

## 6. Crack Your Password File

To crack your hashfile using John’s incremental mode, use the following format:   
  
**john --incremental=ASCII --format=raw-[hash name] [password file directory/name]**

**\*Note**

* Press any key while running John to see a status and how long it has been running.
* You will need to use multiple commands to crack the entire file due to the different hash formats.
* Include **--incremental=ASCII --format=raw-[hash name]** in your command to view the cracked passwords.

1. **Write the commands you used to crack your file below:**

john --incremental=ASCII --format=raw-md5 /home/kali/Desktop/hashes.txt ~

john --incremental=ASCII --format=raw-sha256 /home/kali/Desktop/hashes.txt ~

john --incremental=ASCII --format=raw-sha516 /home/kali/Desktop/hashes.txt ~

1. **Which two passwords took the shortest amount of time to crack? Explain why you think they cracked faster.**
2. **Which two passwords took the longest amount of time to crack? Explain why you think they took longer.**