## **Lab #3: Password Cracking with John**

### **Overview**

This lab will introduce you to one of the most commonly used password crackers, John the Ripper or just John once you become friends. John is a free and open-source password cracker, which can identify many types of password storage schemes and apply guessing/cracking algorithms against those hashes.

While John is generally run locally on your testing computer it has options for distributing the guessing actions across multiple machines to coordinate running the algorithm across many more processors without duplicating effort (as in if one computer has already guessed or is tasked to guess 'Password1234' none of the others are tasked to try it).

### **🎯 Goals**

By the end of this lab you will be able to...

* Know what John can do and identify some limitations
* Run John against a hashed Linux password file
* See the difference in speed between brute-force and educated guessing

### **Resources**

* [crackfiles.zip](https://courses.codepath.org/course_files/cyb101/lab_3/crackfiles.zip) (We'll give you instructions for getting these onto your Kali VM)
* [John the Ripper's documentation](https://www.openwall.com/john/doc/)

### **Lab Instructions**

#### **Step 0: Setting up**

In this step, we'll make sure we have john installed and ready to use on our Kali VMs

* First, let's get connected to our machine. Connect to your Kali VM (using ssh or rdp).
  + If you used rdp, go ahead and open a **terminal** window.
  + If you used ssh, you're already in a terminal!
* Next we'll check to see if john is already installed. Try executing the john command and see what happens.

Most likely, you'll see something like this:

┌──**(**kali㉿kali**)**-[~]

└─$ john

Command 'john' not found, but can be installed with:

sudo apt install john

Do you want to install it? **(**N/y**)**

Hmmm... Well then, let's install it!

* Type y if prompted, or just run the following command:

┌──**(**kali㉿kali**)**-[~]

└─$ sudo apt install john

* Type y when prompted, and john should install on your Kali system!

🎯 **Checkpoint 0**: Run the john command again. You should see something similar to the following:

┌──**(**kali㉿kali**)**-[~]

└─$ john

John the Ripper 1.9.0-jumbo-1+bleeding-aec1328d6c 2021-11-02 10:45:52 +0100 **[**linux-gnu 64-bit x86\_64 AVX512BW AC]

Copyright **(**c**)** 1996-2021 by Solar Designer and others

Homepage: https://www.openwall.com/john/

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

Use --help to list all available options.

👋 Hello, John!

#### **Step 1: Find some password hashes to crack!**

As we briefly mentioned in the Unit 1 lab, unless the developer *really messed up* 👀, passwords will be stored and passed as **hashes**, never in **plain text**.

As a reminder, **hashes** are one-way calculations done against a file that *always have the same result for the same file*. This means something neat if you're a secure resource:

* You don't need to store my password, just its hash!

For example, let's say I make an account on your server:

username: admin

password: chinchillaFriend42

* Rather than risk storing my password directly, you just make a note that my user's password, chinchillaFriend42, hashes to 6ccf.
* Later, I try to log in, and submit a password: chinchillaEnemy42, which hashes to 29da
* By comparing the hashes, you can figure out if I entered the correct password!
  + (In this case, I did not, since 6ccf does not equal 29da. Alas!)

So, if we want to crack passwords on a machine... we won't be looking for plain text. We'll be looking for hashes.

🤔 Q: Um, if hashes can't be reversed, how can John possibly crack them?

* Download [crackfiles.zip](https://courses.codepath.org/course_files/cyb101/lab_3/crackfiles.zip) onto your Kali box, by either...
  1. Using RDP to open a Kali web browser and download the file directly.
  2. Downloading the file to your local machine and using [scp](https://linuxhandbook.com/scp-command/) (**s**ecure **c**o**p**y) to move it to Kali.
* Follow one of these sets of instructions:

**Option 1:** Falkon

1. RDP into your Kali machine
2. Navigate to [https://courses.codepath.org](https://courses.codepath.org/) on your Kali web browser (Falkon)
3. Login to the course portal (on Kali)
4. Download <https://courses.codepath.org/course_files/cyb101/lab_3/crackfiles.zip>

Close Section

**Option 2:** scp (**s**ecure **c**o**p**y)

1. Download <https://courses.codepath.org/course_files/cyb101/lab_3/crackfiles.zip>
2. From a local terminal, cd into the folder you downloaded the file into
   * Example: cd ~/Downloads
3. Run the following command: scp -i [your pem file] crackfiles.zip [user]@[kali-ip]:[destination]
   * Example: scp -i azure\_kali\_key.pem crackfiles.zip kali@20.222.45.198:~/Downloads/crackfiles.zip

Close Section

🎯 **Checkpoint 0**: You should have the file crackfiles.zip on your Kali machine.

Once you've got the file on Kali, go ahead and unzip the folder (you can use the unzip command) and take a look at the files. You should have:

* crackA.txt
* crackB.txt
* crackC.txt
* crackChallenge.txt
* lower.lst

As you can probably guess, all of the crackX files are password files! Feel free to look at them with a text editor, or terminal command like cat or less. The other file, lower.lst, is a wordlist file... we'll get to that in the next step.

🎯 **Checkpoint 1**: We can view our individual password files and are ready to start cracking!

🤔 Q: Okay, but in the real world no one is providing me a file. Where are password hashes actually stored?

A: In a Linux environment, you'd want to look in /etc/passwd and /etc/shadow. However, there a couple reasons you might not want to crack these:

* If you do it on *someone else's machine*, that's unethical (and usually illegal!)
* If you do it on *your own machine*, you're creating plain text files with your passwords and exposing them to risk!

#### **Step 2: Explore the wordlists**

In this step, we'll add some popular **wordlists** to our Kali boxes, and explore a bit.

💡 Remember how we said john (and other tools) crack password hashes by making educated guesses?

To make that easier, people create text files full of the most commonly used passwords -- Literally, a pass***word-list***.

Some versions of Kali come with wordlists pre-installed. If so, you'd find them in /usr/share/wordlists. However, you'll most likely need to install them:

* Run the following command: sudo apt install wordlists
* Now let's **l**i**s**t the files in that folder: ls /usr/share/wordlists

┌──**(**kali㉿kali**)**-[~]

└─$ ls /usr/share/wordlists

john.lst rockyou.txt.gz

Awesome! We have the rockyou.txt wordlist, which is what real-world password crackers use! The .gz suffix means it's compressed, though, so we'll need to unzip it:

* Use the gunzip command to unzip the wordlist:

┌──**(**kali㉿kali**)**-[~]

└─$ sudo gunzip -d /usr/share/wordlists/rockyou.txt.gz

Repeating thels command shows the file is now uncompressed!

┌──**(**kali㉿kali**)**-[~]

└─$ ls /usr/share/wordlists

john.lst rockyou.txt

🤔 Q: What about the other wordlists?

Other wordlists are useful! They have a lot of foreign language lists, some typical default passwords for a lot of different devices that might have been left as default (such as routers and switches) among others.

What reasons and targets can you think of that might create a need for different wordlists?

Now it's time to explore the rockyou wordlist!

* Run the following command: less -N /usr/share/wordlists/rockyou.txt
  + (The -N flag makes line numbers show up. If you don't want line numbers, you can run less without it.)

💡 Tip: less has its own keyboard controls! Use Space to go to the next "page" of entries, and Q to exit.

Now that you're in there you can actually do some searching with the & key.

* Type &puppy then hit Enter to see all entries that contain the text "puppy".

1556 puppy

1647 puppylove

2769 puppy1

3412 puppys

3664 puppydog

8464 puppyluv

11011 puppy123

13688 puppylove1

15080 puppy2

... //etc

Now it's your turn!

* Try out some other options and explore what sorts of passwords people thought were secure at one time or another.

⚠️ Warning: If you choose to search for your own password, make sure no one else can see your screen!

💡 Note: For this lab, we're not going to use rockyou.txt. Why? Because it's **big**, and takes a **long time** to run. Instead, we'll use a much smaller wordlist, the provided lower.lst. Feel free to check it out too with less -N lower.lst!

🎯 **Checkpoint 2**: We know where our wordlists are, and have taken a peek inside them!

#### **Step 3: Cracking passwords with John**

This is the step you've been waiting for! It's time to crack the passwords in the provided file. We're going to do this in a few different ways:

* Run John against our files in single crack mode
* Run John against our files in wordlist crack mode
* Stop and resume John in the middle of a crack.
* Run John against our files in incremental mode (brute-force)
* Stop John WITHOUT saving our place in the algorithm

**SINGLE CRACK MODE (crackA.txt)**

Single crack is a mode that uses information about the user (stored in [the GECOS fields](https://en.wikipedia.org/wiki/Gecos_field)) to make educated guesses about the password.

* For example, if the username is admin, single-crack mode will guess passwords like admin, admin1, ADMIN, admin=, etc...

GECOS fields aren't commonly used today, but they could also contain information like the user's full name, email address, and phone numbers.

* If this data exists, John will use elements from all these fields to make guesses.

✏️ Your turn!

* First, take a look at crackA.txt. What additional data do you see for users squirtle, charmander, and bulbasaur?
* Now let's try running John in single crack mode: john --single crackA.txt
  + Did the passwords crack?

Successful cracking will look something like this, except... we've censored the last two passwords!

┌──**(**kali㉿kali**)**-[~]

└─$ john -single crackA.txt

Using default input encoding: UTF-8

Loaded 3 password hashes with 3 different salts **(**md5crypt, crypt**(**3**)** $1$ **(**and variants**)** **[**MD5 512/512 AVX512BW 16x3]**)**

Press 'q' or Ctrl-C to abort, almost any other key **for** status

waterSquirtle **(**squirtle**)**

███████████ **(**charmander**)**

██████████ **(**bulbasaur**)**

🎯 **Checkpoint 3**: You should have successfully all three pokemon's passwords! Run john --show crackA.txt to view them.

Okay, but what about one of our other files? 🤔

* Try running John in single crack mode against crackB.txt
  + (If you open this file, you'll notice there's no GECOS field data!)

Uh oh! We may need something fancier for this one...

**WORDLIST MODE (crackB.txt)**

Let's bring back the wordlists from step 2! John's **wordlist mode** will take any wordlist as a dictionary and try every password in there. (It will also do basic *mangling*, trying different mixes of upper/lowercase letters, etc.)

**🔑 Jim**'s password is crackable with the wordlist directly. Let's start with that:

* john --wordlist=lower.lst crackB.txt

Ugh.... this is taking FOREVER. What if my laptop dies, I have something else to do, etcetera?

* Stop john any time by pressing the q key. It may take a moment while John saves your place!
* Resume any time by running: john --restore
* Wait for John to finish. It may take 2-3 minutes. Press any key (except q) to see your progress!

If all goes well, you should crack one of the passwords! (Oh, Jim...) But there are three passwords in the file. To get the other passwords, we'll need to add some *mangling rules*:

**🔑 Dwight** used some [l33tspeak](https://en.wikipedia.org/wiki/Leet), which we can check for with --rules=l33t

* john --wordlist=lower.lst crackB.txt --rules=l33t
  + Once it cracks, go ahead and stop john.

**🔑 Pam** tried mixing up her lower and upper case letters.

* john --wordlist=lower.lst crackB.txt --rules=shifttoggle
  + Sneaky, Pam, but we still got it!

🎯 **Checkpoint 4**: You should have successfully cracked Jim, Dwight, and Pam's passwords! Run john --show crackB.txt to view them.

**INCREMENTAL MODE (crackC.txt)**

Finally, there's incremental. This mode is the most powerful... but also the most slow.

Have you ever tried to guess someone's PIN number by just trying things? *1111, 1112, 1113, etc..* Well, John's incremental mode does this at a huge scale.

* By default, it will try every legal permutation of all 97 ASCII characters up to 13 characters long. That's over **67 septillion** possibilities, and will take a **really long time**.
* To speed things up, we can make some educated guesses about how people *usually* construct their passwords.

**🔑 pinball**: This password is strictly **numeric** and **4-6 digits long**.

* john --incremental=digits --min-length=4 --max-length=6 crackC.txt
* Once pinball cracks, stop John with q

**🔑 pacman**: This password follows a common pattern: A number, an uppercase letter, and some lowercase letters. To do this, we'll use a mask.

* john --mask=?d?u?l?l crackC.txt
  + Where '?u' represents an uppercase letter, '?l' represents a lowercase letter, and '?d' is a digit.
* Once pacman cracks, you can stop John if it's not done yet.

**🔑 frogger**: This password follows an even more common pattern: A 4-letter word, a number, and an exclamation mark! (Ever make a password that resembles that?)

* Take a guess at what the --mask should be. (Hint: You can put the ! directly into the mask)

💡 Stuck?

Try this: john --mask=?l?l?l?l?d! crackC.txt

🎯 **Checkpoint 5**: You should have successfully cracked all the games' passwords! Run john --show crackC.txt to view them.

For more information on masking, check out: <https://www.openwall.com/john/doc/RULES.shtml>

*Extra Info: Some notes about how John handles long sessions*

* John auto-saves where it is in it's guessing scheme every 10 minutes by default. If you stop it before it has exhausted everything it will try in the algorithm you've chosen it will attempt to save its place so it doesn't waste computing cycles guessing the others again. It will not be immediate while it's writing to the file it needs in order to bookmark its place. You can prevent it from doing this by hitting Ctrl-C TWICE.
* NOTE: If you do this but then get back to cracking the passwords in that file you may lose up to 10 minutes of time getting back to where you were in the process. Ctrl-C is so common for anyone who works in the command line to stop an action it's going to be your default soon if something isn't going your way in the terminal it isn't already. Seeing that nothing seemed to happen immediately might cause you to try it again directly after, and if so you would prevent John from saving a bookmark. This part of the exercise is mostly to show you how easy it is to screw up, but if you are truly done and moving on it can prevent you from having to wait for John to write the file.

Close Section

**AUTO MODE**

Finally, if you just need it cracked and don't care about optimizing it... John does have an "auto-run" mode.

It will try the following, in order, until it is successful:

* single-crack mode
* wordlist mode (using John's built-in password.lst wordlist)
* incremental mode (brute-force)

This is the simplest mode to invoke. Simply run:

$ john crackA.txt

This mode can be useful, but it may not be optimal if you already have some information about the password(s) in question!

#### **Step 4: Where'd it all go? ...**

We've already seen that John stores cracked passwords. To check what we've already cracked for a file, we can use the --show option when we run John:

$ john --show crackA.txt

This is great... but where does john keep it?

Well, it's in ~/.john/john.pot. (Toilet humor, anyone?)

Go ahead and less this file to see all the passwords you've cracked so far.

Another important file is the configuration file john.conf (also located in ~/.john). Take a look at it, we don't need to make any changes to it, but you can see for your own use in the future how you could do things like change the default wordlist, use only Idle time on your system so it's not bogging it down when you need to do something else, make it save its bookmark more or less frequently, or have it beep if you find a password.

Finally, we can see all the different sessions you have, both the ones you aborted early and any that are still running:

$ john --status

🎉 Congratulations, you've cracked passwords with John! 🎉

If you have time left over, continue on to the stretch tasks to grow your knowledge further!

### **Stretch Features**

#### **Step 4: A Challenge**

A challenger approaches!

For the stretch features of this lab, see if you can crack the hashes in crackChallenge.txt. There are four users to crack:

* 🦜 Birb (easy)
* 🐶 Pupper (medium)
* 🐱 Kitty (hard)
* 🐺 Doggo (impossible)

Try your best! All the tools you need are at your fingertips!

But, if you get stuck...

**HINTS**

🦜 Birb (easy)

You're looking for a word included in lower.lst -- no mangling needed!

ANS: john --wordlist=lower.lst crackChallenge.txt

Close Section

🐶 Pupper (medium)

Have you tried looking at the crackChallenge.txt file? What's different about this entry?

ANS: john -single crackChallenge.txt

Close Section

🐱 Kitty (hard)

You still want to use lower.list, but you'll need a --rules flag…

ANS: john --wordlist=lower.lst crackChallenge.txt --rules=All

Close Section

🐺 Doggo (impossible)

It's alphanumeric and 5 digits or less.

john --incremental=Alnum --max-length=5 crackChallenge.txt

Close Section

🎉 Congratulations 🎉

You've completed the password lab AND the stretch goals! 🚀

## **Extra Practice**

🤔 *"Wait, can people crack MY passwords?!"*

* Play around with John and see if various passwords are crackable with the wordlist here's how to create some password hashes to mess around with, try this - it will spit out a username and password line in the format john is expecting:

*# You might need to install mkpasswd first, that's okay!*

$ echo -n theusername: ; mkpasswd -m md5 thepassword

$ echo -n theusername: ; mkpasswd -m sha-256 thepassword

$ echo -n theusername: ; mkpasswd -m sha-512 thepassword

⚠️ Warning: Using your own passwords here could expose them!

Instead, try using a similar password to see how crackable yours is.

* For example, if your password is fish29, try another 4-letter 2-number password, like lamp36.

MD5, SHA-256, and SHA-512 are different hash algorithms that you might find in these types of password files. They can coexist in the same file so go ahead, copy several lines with different hash mechanisms, usernames and passwords and try a few and see if John is able to crack them.

* If you have some experience with coding you may want to try writing your own password cracking program!
  + Think of both the technical elements and human elements! If a capital letter and a number are required for complexity reasons, what will a human likely do?
  + If the password requirements are at least 8 characters and less than 12 what can you do?