CYB101 | Intro to Cybersecurity

Intro to Cybersecurity Summer 2025 (@ Section 1 | Wednesdays 3PM - 5PM PDT)
Personal Member ID#: **100895**

MACHINE ACCESS CHECK

To proceed with this unit, please ensure you have access to a ***cyb101 Ubuntu VM*** using Azure Labs or an "Alternative Option" from the **IDE Setup Guide**.

Which alternative options work for this unit?

- **Google IDX** ✓
- **Azure VM** ✓
- **VMware on Windows / Intel Mac** ✓
- **VMware on M-Chip Mac** ✓
- **Manual VM** ✓

Week 3 Lab: 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 (We'll give you instructions for getting these onto your Ubuntu VM)
- John the Ripper's documentation

Lab Instructions

Step 0: Setting up

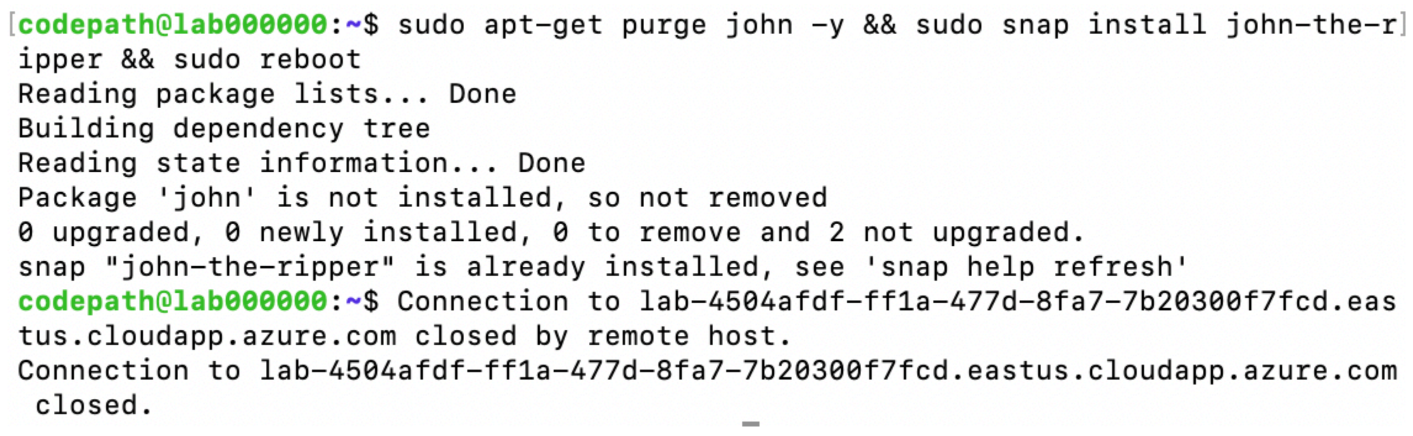
In this step, we'll make sure we have `john` installed and ready to use on our Ubuntu VMs

- [ ] First, let's get connected to our machine. Connect to your Ubuntu VM (using `ssh` or `rdp`).
    - If you used `rdp`, go ahead and open a **terminal** window.
    - If you used `ssh` (*preferred for this lab*), you're already in a terminal!

Now, let's get `john-the-ripper` installed - and also uninstall a previous version of `john` that doesn't work too well anymore:

🎯 **Checkpoint 0**: Run `sudo apt-get purge john -y && sudo snap install john-the-ripper && sudo reboot`. You should see something similar to the following:

NOTE: If you are using Google IDX, John the ripper is already installed for you and you can skip this step.



```
codepath@lab000000:~$ sudo apt-get purge john -y && sudo snap install john-the-r
ipper && sudo reboot
Reading package lists... Done
Building dependency tree
Reading state information... Done
Package 'john' is not installed, so not removed
0 upgraded, 0 newly installed, 0 to remove and 2 not upgraded.
snap "john-the-ripper" is already installed, see 'snap help refresh'
codepath@lab000000:~$ Connection to lab-4504afdf-ff1a-477d-8fa7-7b20300f7fcd.eas
tus.cloudapp.azure.com closed by remote host.
Connection to lab-4504afdf-ff1a-477d-8fa7-7b20300f7fcd.eastus.cloudapp.azure.com
 closed.
```

Note that this reboot will close your ssh/rdp session. Connect to your VM again and the updates should be in place. 👋 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?

A: By making educated guesses! First, John figures out what **hashing algorithm** was used, then starts guessing passwords and comparing its guess hashes to the hash it's trying to crack.

This sounds like it would take a really long time, but computers are pretty fast... Let's see how it goes!

- [ ] Navigate into your provided `unit3` folder in the home directory.
- [ ] **list** the files and find `crackfiles.zip`

🎯 **Checkpoint 1.1**: You should have located the file `crackfiles.zip` on your Ubuntu VM.

Next, go ahead and unzip the folder (you can use the `unzip` command) and take a look at the files (using `ls`). 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 1.2: 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 Ubuntu 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 linux come with wordlists pre-installed, but we aren't so lucky. Let's go ahead and download a new wordlist from the internet:

- [ ] Run the following command:
  `wget https://github.com/brannondorsey/naive-hashcat/releases/download/data/rockyou.txt`

Awesome! We have the `rockyou.txt` wordlist, which is what real-world password crackers use! Use the `ls` command to view the file:

```
codepath@lab000001:~/unit3$ ls
crackfiles.zip  rockyou.txt  cp_leak.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?

ow it's time to explore the `rockyou` wordlist!

- [ ] Run the following command: `less -N ./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.

✨ AI Opportunity

▶ *Use AI to understand technical tools →* `less`

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 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) 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!

```
codepath@lab000001:~$ 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/
Press 'q' or Ctrl-C to abort, almost any other key for status
waterSquirtle    (squirtle)
[redacted]       (charmander)
[redacted]       (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...

'ORDLIST 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, 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 wercase 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*

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
```

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

Well, it's in `~~/snap/john-the-ripper/610/.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 `/snap/john-the-ripper/610/run/john.conf`). 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!

Close Section

▼ 🐶 Pupper (medium)

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

Close Section

▼ 🐱 Kitty (hard)

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

Close Section

▼ 🐺 Doggo (impossible)

It's alphanumeric and 5 digits or less.

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
```

Note: Directly copy the output into a file you wish to use John on!

⚠️ 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 assword files. They can coexist in the same file so go ahead, copy several lines with different hash echanisms, 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?

✨ AI Opportunity

▼ *Use AI to unpack complex challenges* → Trying to crack your own passwords

This can be an arduous task depending on your experience level and how robust you want your program to be, so feel free to ask ChatGPT for help along the way or for assistance with the initial setup!

You can copy the instructions above into ChatGPT, explain what you're trying to do, and tell the AI how you'd like it to respond.