# Week 6 Lab: Password Cracking

Include section screenshots in the Week 6 LPR. This lab will use the Kali Linux Virtual Machine.

## Part 1: Test Password Security

1. Visit the following URL:

   https://lowe.github.io/tryzxcvbn/

   This website estimates the strength of passwords you enter. Passwords never leave your browser – if you are so inclined, you can confirm as much by perusing the source code and/or by opening your browser developer tools and observing the (java)script that runs when you enter passwords.

2. Try out different passwords to see how strong they are.

   **Optional:** If you want to learn more about password strength estimation, see this video and paper.

## Part 2: Check an Account for a Prior Data Breach

1. Check to see if one of your online accounts has already been breached.

   Visit: https://haveibeenpwned.com. Type in one of your email accounts or usernames to see if it has already been compromised in a data breach.

2. Next visit: https://haveibeenpwned.com/Passwords

   Try out some passwords to see if they have already been compromised in a data breach.

3. Finally, visit: https://haveibeenpwned.com/NotifyMe

   Sign up to be notified when one of your accounts is breached in the future.

**Question:** Was one of your accounts breached? If so, which one(s)? and what will you do to make it more secure?

## Part 3. Sign-up for Two Factor Authentication

Visit twofactorauth.org and browse through the categories to find an online service that you regularly use (e.g., Gmail, Snapchat, Instagram, Facebook, etc.). Click the box-arrow icon in the "Docs" column to learn how to set two factor authentication for that service. Sign up for 2FA for at least one account.

**Alternatively:** Visit dongleauth.info to find an online service that allows you to use WebAuthn or Universal Second Factor (U2F).

**Question:** Which service did you enable 2FA for?

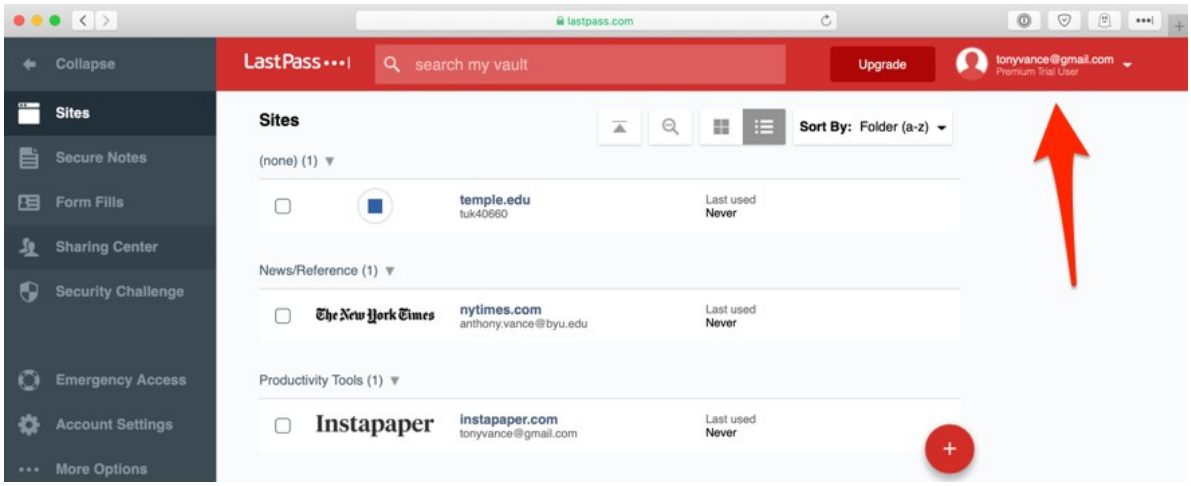
## Part 4. Install and Set up a Password Manager

If you're not already using one, set up a password manager. I recommend creating an account with LastPass (free, or premium version $24 per year), or my favorite, 1Password (first six months free for students using this link  $36 per year). See here for a comparison of leading password managers.

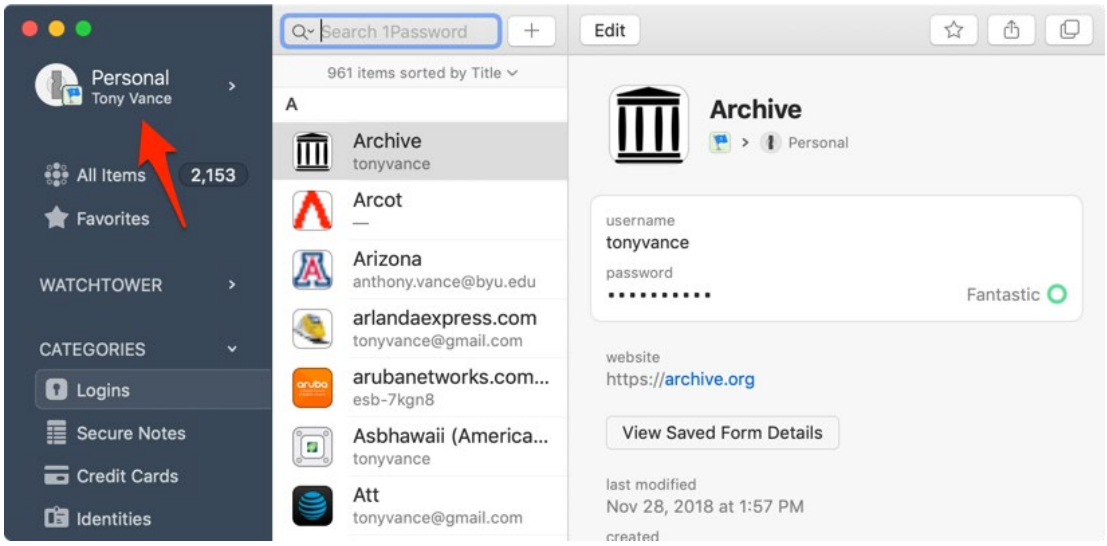
Next, install the browser extension for your password manager (see here for LastPass; see here for 1Password). With the browser extension installed, log into a website for which you have an account. Your password manager will ask to save the password after each login. Do this for three sites.

Take a screenshot of your password manager showing **saved entries for at least three sites** you visited. Also, make sure the screenshot **shows your username** in the top left- or right-hand corner. Submit your screenshot on Canvas.

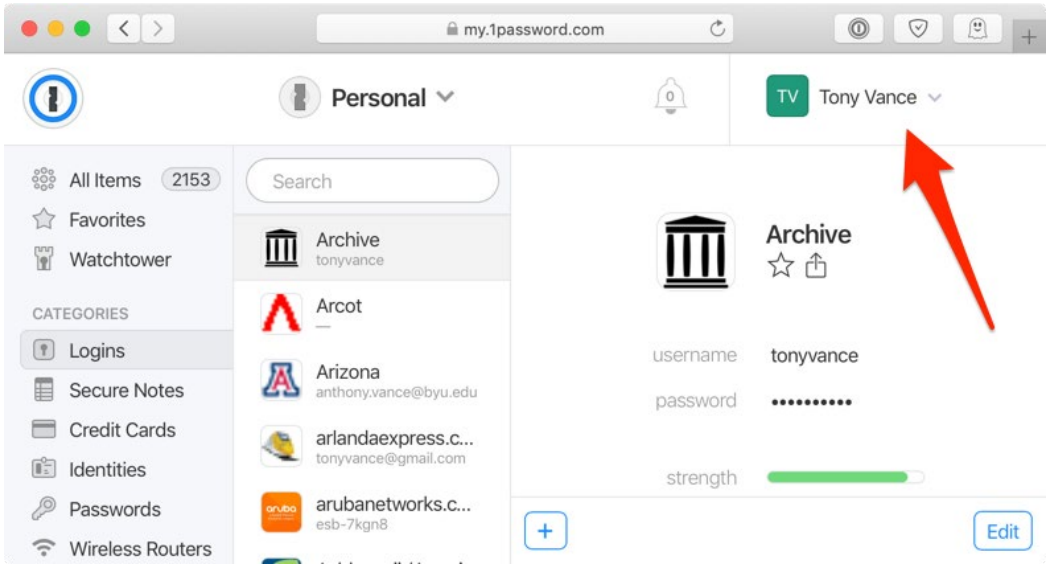
**LastPass.com example screenshot:**




**1Password app example screenshot:**




**1Password.com example screenshot:**



## Part 5: Online Password Attack

In this section, you will launch attacks against login services, in an attempt to brute-force a username/password.

**Note:** This section uses the Kali VM

**Heads up!** Pay careful attention to each step in this section.

This attack uses `/usr/share/wordlists/rockyou.txt.gz`, which comprises all unique passwords from the 32 million RockYou password breach.

1. Open a terminal and elevate to a root shell (or, alternatively, run all

   commands below with `sudo`): `sudo -s`

   `su root` – Note: If you used the VM provided for this class, root is already escalated and you won't need to do this.

2. First, if the file `/usr/share/wordlists/rockyou.txt` is not present, unzip the password list.

   ```
   gunzip /usr/share/wordlists/rockyou.txt.gz
   ```

3. We will use the 'rockyou' password list to launch an online password attack using `THC-Hydra`. The attack will be launched against a website that is managed by Dr. David Eargle at the University of Colorado Boulder, who authorizes students to launch the attack **only** as specified in the instructions below.

   Visit https://is.theorizeit.org in a browser, and note that it is an installation of mediawiki. Now browse to subdirectory https://is.theorizeit.org/auth/. It requires a login. Let's say that you wanted to crack the password for this route. Imagine that you knew, or guessed, that one of the usernames was `istheory`.

   Open a terminal in your Kali VM. Read the explanation below for each command argument. Then, type the following, all on one line (remember that you can use tab-completion for the rockyou path):

   ```
   hydra -V -l istheory -P /usr/share/wordlists/rockyou.txt https-get://is.theorizeit.org/auth/
   ```

   **Note:** The trailing slash (`/`) in the final

   argument is needed. Where:

   - `hydra` is the password cracking tool to execute.
   - `-V` means verbose, and will show you the username and password combination being attempted.
   - `-l istheory` sets "istheory" as the login name. **Note:** that's a lowercase 'L.'
   - `-P /usr/share/wordlists/rockyou.txt` is the password
     - dictionary file to use. the password is case-sensitive!
   - `https-get` means a GET request over HTTPS. Note that Hydra supports many protocols (e.g., ftp, ssh).
   - `is.theorizeit.org/auth/` is the password-protected URL to be accessed.

   **Tip** This is a good opportunity to look at a command's built-in help documentation. Run `hydra -h` and peruse the output to find each of the above arguments, and read what they do. What would a lowercase `-p` have signified, instead of the uppercase `-P` used? What would an uppercase `-L` have signified? Argument casing matters for this program!

   **Note:** you can also use THC-Hydra with web forms: http://insidetrust.blogspot.com/2011/08/using-hydra-to-dictionary-attack-web.html

   In Greek and Roman mythology, Hydra is a mythical sea monster with many heads. When a head is cut off, it is replaced by another. `THC Hydra`, the tool you are using, likewise launches multiple "heads" (tasks), each of which makes a battery of attacks (password guesses) before dying off and being replaced by another head (task). By default, Hyrda runs with 16 concurrent heads.

   The Hyrda output will tell you at what time it started, how many passwords it has tried so far, and at what time it stopped.

   **Question:** What was the password (Scan the results to find the line beginning with `[443][http-get]`)?

   **Question:** Approximately how many passwords a second were you able to try? **Hint:** You may need to calculate this from the start and end time along with number of guesses made.

   **Question:** For the LPR submission, look at sample output from a `hydra` run, and determine how many passwords were tried per second in the sample output.

## Part 6: Offline Attack Using Hashcat

While in the previous section attacks were launched over the network, in this section, you will start with a *password hash* stored on your computer, and you will launch attacks against it locally. No authentication service such as a webserver sits in front of your attempts, potentially throttling the speed at which you can make guesses. Instead, you can make guesses as fast as your computer and the particular hash algorithm will allow.

If you're feeling adventurous, you can install Hashcat on your host computer, where you'll get massive speed improvements compared to running it in your Kali VM. Hashcat needs to be able to directly interface with the CPU/GPU in order to crack fast—it can't so as well in a virtualized environment.

For Windows or Linux, download Hashcat here. For Mac, I recommend that you first install Homebrew (follow instructions at https://brew.sh), then run the command,
`brew install hashcat`. In Kali

1. On your Kali machine, open up a terminal and navigate to your home directory (`/root/`).
2. Hashcat uses a command-line interface. To see available options and syntax, type:

   ```
   hashcat --help
   ```
3. First, a dictionary attack will be performed against a password-protected Word document. The following command uses a Python script to obtain the hash of the Word document password called `hashcat.doc`. This document has been prepared specifically for this lab. The word document has been password-protected. The hash of the password is stored within the metadata of the document file itself. You can use the python script called `office2john.py` to extract the hash. `john` in that script name refers to JtR, John the Ripper, and the python script prints the hash in a format that `john` can accept. We will edit the output to prepare for its use with `hashcat`.

   First, obtain the necessary files:

   - office2john.py can be obtained here. To obtain it in Kali, run:

     ```
     wget https://raw.githubusercontent.com/magnumripper/JohnTheRipper/bleeding-jumbo/run/office2john.py
     ```
   - hashcat.doc is available here. To obtain it, run:

     ```
     wget https://raw.githubusercontent.com/deargle/security-assignments/master/labs/files/hashcat.doc
     ```

   Then, run the following command from the directory to where you downloaded `office2john.py`

   and `hashcat.doc`: `python office2john.py hashcat.doc`
4. In the output, you'll see the name of the file, followed by the type. The type is shown with a `$` at the beginning and end of it. You need to convert the output to a format that `hashcat` can recognize. To do so, copy the type and everything up until but not including `:::`.

   For example, if the script output the following:

   ```
   hashcat.doc:$oldoffice$1*04477077758555626246182730342136*b1b72ff351e41a7c68f6b45c4e938bd6*0d95331895e99f73ef8b
   6fbc4a78ac1a:::::hashcat.d
   ```
   You would extract just the hash, similar to the following:

   ```
   $oldoffice$1*04477077758555626246182730342136*b1b72ff351e41a7c68f6b45c4e938bd6*0d95331895e99f73ef8b6fbc4a78ac1a
   ```

   **Save the extracted hash into a file in your home directory.** Name the file whatever you like.

   **Note:** make sure the entire hash is on one line within the text file. **Don't add extra spaces at the end.** If you get a "line-length exception" in the next step, make sure there's not a typo in the beginning of the hash.

   If you want, you can use text redirection (e.g., `>`) to put the output of office2john.py into a file for you. Then you would just have to open up the file and remove the extra stuff.
   Also, if you want, you might attempt to `echo -n 'thehash' > afile.txt` the hash into a file. But beware! The hash contains `$` signs, which in bash indicate a variable when couched in double-quotes. If you want to echo the hash into a file, use single quotes, and you won't be bitten by bash variable expansion. Be sure to `cat` the contents of your hashfile after you made it to make sure it looks right.
5. While still in your home directory, run the following command (all on one line). Reference the hash file you just created, and choose an arbitrary name for an output file. Once the password is cracked, you will read your output file to see the cracked password. It will be appended to the end of the hash following a colon (`:`) symbol.

**Note:** In the commands below, the < > notation indicates placeholders, because I do not know what filenames you chose. Replace the placeholders, **including replacing the < > symbols!** -- with the names of the actual files you are using. Remember that you can use tab-completion to help you spell out the filenames.

```
hashcat --force -a 0 -m 9700 -o <outputFileName.txt> <HashInputFileName.txt> /usr/share/wordlists/rockyou.txt
```

Or alternatively, if you prefer to do it without making an input file, put the hash string right in the terminal, surrounded by single quotes (**not double-quotes!**). Type the quotes in yourself – do not copy-paste them.

```
hashcat --force -a 0 -m 9700 -o <outputFileName.txt> '<hash string>' /usr/share/wordlists/rockyou.txt
```

Peruse `man hashcat` to get an idea of what the arguments do. In our particular instance, the switches correspond to the following:

| | |
|---|---|
| `--force` | This is necssary to get hashcat to run in a VM environment (it doesn't normally like to do so). |
| `-a 0` | Straight dictionary attack against the hash |
| `-m <Office_Flag>` | The corresponding flag for the version of Office in use (see table in `hashcat --help`) |
| `--status` | Provides an update of the status of the process without giving a prompt The location where the cracked hashes will be saved. |
| `-o <Output_File>` | The results will also be saved on the .pot file, unles otherwise specified. In our case, we disabled it. |
| `<Hash>` | The saved password hash. |
| `<Dictionary>` | The list of words that will be used to try and crack the password. |

**Question:** What is the password for `hashcat.doc`?

6. Follow the same process as above, but this time for word doc file `john.doc` (available here, use `wget` as above to obtain it from url `https://raw.githubusercontent.com/deargle/security-assignments/master/labs/files/john.doc`).

   **Question:** What is the password for `john.doc`?

7. Examine the hashcat cracking benchmarks for a Brutalis. The Brutals has 8 graphics cards, each of which can simultaneously work on cracking hashes. The measured speed for each card is shown, along with a cumulative speed at the bottom (`Speed.Dev.#*`). Use the cumulative speed for all Brutalis-related calculations in this lab.

   Cracking speeds are in the following format:

   ```
   H/s    Hashes per second
   KH/s   Kilohashes per second (Thousands of hashes per
   second) MH/s     Megahashes per second (Millions of
   hashes per second) GH/s  Gigahashes per second
   (Billions of hashes per second) TH/s    Terahashes
   per second (Trillions of hashes per second) PH/s
         Petahashes per second (Quadrillions of hashes
   per second)
   ```

   **Question:** How many passwords per second can Hashcat running on a Brutalis try on a `.doc` file (i.e., hashtype "MS Office <= 2003 MD5 + RC4, oldoffice$0, oldoffice$1")?

   **Question:** How much faster is Hashcat in cracking .doc MS Office documents (option 9700, "Hashtype: MS Office <= 2003 MD5 + RC4, oldoffice$0, oldoffice$1") compared to Office 2013 documents (option 9600, "Hashtype: Office 2013")?

   **Optional:** Install `hashcat` on your own machine (not the VM). See how your benchmarks compare against a Brutalis. Note that running benchmarks on the VM will break once it reaches `scrypt` before complete results are reported.

   ```
   hashcat -b --force
   ```

   **Question:** How does an offline password attack compare with the online hydra attack you attempted earlier?

## Part 7. Cracking LinkedIn Hashes Using Hashcat

In this section, you'll see how many hashes you can recover from the 2016 LinkedIn password breach. The LinkedIn hacker, a Russian, was sentenced in US court to seven years in jail on September 29, 2020. This breach of 177,500,189 unsalted SHA1 password hashes represents the data of all LinkedIn users as of 2012. Among these passwords, only 61,829,207 are unique.

However, in interest of your time, this section will require you to crack only 500,000 of these passwords. After you complete this lab, you're welcome to crack all of the LinkedIn hashes. Ask me for a copy.

1. Download a copy of the file

`LinkedIn_HalfMillionHashes.txt` from here. Right-click this link, select "copy link", then paste it into Kali after `wget`.

2. To get your feet wet, perform a "straight" dictionary attack using the `rockyou.txt` wordlist again. This attack will try each entry in the rockyou dataset with no permutations.

   Before you run the following command, read the `man` page to make sure you understand what each argument/switch does. Once you have done so, run the command.

   ```
   hashcat --force -m 100 --remove --outfile=LinkedIn_cracked.txt LinkedIn_HalfMillionHashes.txt
   /usr/share/wordlists/rockyou.txt
   ```

   **Note:** The above command may take 5–10 minutes to run. To see the status of a running job in Hashcat, press the `s` key (it might take up to 15 seconds for Hashcat to report its status).

   `hashcat` will report how many passwords it "recovered" when it finishes.

   These commands use the `--remove` flag. This will remove cracked hashes from the input file. So, if you run these commands more than once without changing anything, it won't crack anything after the first time.

   If you accidentally delete your cracked outfile, you will need to delete your hashcat "potfile" too before you try to recreate the outfile. You have to do this because otherwise, hashcat won't write any already-cracked hashes found in the potfile to the outfile. The `hashcat.potfile` is stored in a hidden direction in the home directory of whomever you run the command as.

   To do this, `rm ~/.hashcat/hashcat.potfile`.

   Don't forget to also start with a fresh 500k hashlist, because the `--remove` flag would have removed rows from that file as the hashes were cracked and inserted into the potfile.

   You can always **open another terminal session** and count the number of lines in your outfile (`LinkedIn_cracked.txt`) to see how many you've cracked so far, in total:

   ```
   wc -l LinkedIn_cracked.txt
   ```

   Or count the number of passwords left (it started with half a million):

   ```
   wc -l LinkedIn_HalfMillionHashes.txt
   ```

   To see hashes cracked in real time, in another terminal shell, type the command: `tail -f LinkedIn_cracked.txt`. Type `control+c` to exit the `tail` command.

   **Question:** How many passwords were you able to recover using the Hashcat command above?

3. Run another attack that uses a rules-based method.

   Rules apply common patterns to password dictionaries to crack even more hashes. You can read about rules in Hashcat here: https://hashcat.net/wiki/doku.php? id=rule_based_attack.

   The "best64.rule" is one of the most effective sets of Hashcat rules. It is continually refined using input and testing from the password cracking community. You can view the contents of the best64.rule here:

   https://github.com/hashcat/hashcat/blob/master/rule

   s/best64.rule You can read an explanation of these

   set of rules here:

   http://kaoticcreations.blogspot.com/2011/09/explanation-of-

   hashcat-rules.html Run the following command once you

   understand what the `-r` flag does.

   ```
   hashcat --force -m 100 --remove --outfile=LinkedIn_cracked.txt LinkedIn_HalfMillionHashes.txt -r
   /usr/share/hashcat/rules/best64.rule /u
   ```

   **Question:** How many total passwords were you able to recover after using this rules based attack in combination with the earlier straight attack?

4. Run another attack that uses a hybrid method that uses a dictionary attack combined with a “mask,” which is a pattern that is appended to each password in the password dictionary:

   ```
   hashcat --force -m 100 --remove --outfile=LinkedIn_cracked.txt LinkedIn_HalfMillionHashes.txt -i -a 6
   /usr/share/wordlists/rockyou.txt ?
   ```

   The `?d?d` at the end means to append two digits between 0–9 each at the end of each password in the rockyou.txt password dictionary.

   **Question:** How many total passwords were you able to recover after using this hybrid attack combination with the earlier straight and rules-based attacks?

5. If you would like to try using a different character set for your mask, you can use the following masks below. Note that each mask below is for one character. If you wanted to test four digits at the end of each password, the mask would be: `?d?d?d?d`.

   ```
   ?l = abcdefghijklmnopqrstuvwxyz
   ?u = ABCDEFGHIJKLMNOPQRSTUVWXYZ
   ?d = 0123456789
   ?s =  !"#$%&amp;'()*+,-./:;&lt;=&gt;?@[\]^_`{|}~
   ?a = ?l?u?d?s
   ?b = 0x00 - 0xff
   ```

   **Optional:** Experiment with other rules found in `/usr/share/hashcast/rules`.

   **Optional:** Another common password pattern is to prepend digits at the beginning of passwords. If you would like try this mask, run the following command:

   ```
   hashcat --force -m 100 --remove --outfile=LinkedIn_cracked.txt LinkedIn_HalfMillionHashes.txt -i -a 7 ?d?d
   /usr/share/wordlists/rockyo
   ```

Want even more practice? You can download the massive Troy Hunt haveibeenpwned SHA1 password hash list on the bottom of this page. Also, see Daniel Miessler's wordlist collection for more wordlists besides RockYou to try.

## Part 8. Secure Password Hashing

Refer again to the benchmark output for a Brutalis.

**Question:** How much slower is Hashcat in cracking Bcrypt hashes

compared to SHA1 hashes? Read about the Bcrypt algorithm here, and also

here

**Question:** Imagine that Bcrypt is set to a work factor of 12. How many hashing rounds will Bcrypt go through to compute the final hash?

**Question:** An attacker knows that a user generated their password using 8 random lowercase letters exclusively (so character space of 26, length of 8). On average, an attacker needs to try only half of all possible passwords in order to brute force the password. The attacker has access to a Brutalis. How long would it take to crack the password hash if SHA1 had been used? Bcrypt with the benchmarks shown for a Brutalis?

## Part 9. Create a Targeted Wordlist Using CeWL

CeWL (Custom Word List Generator) is a command-line tool that creates custom wordlists from a target website. This can be useful for cracking a password of an organization or individual that also has a website or social media profile. Because people often use information about themselves or their organization when creating passwords, custom wordlists can be very effective.

Imagine that you exfiltrated the following MD5 hash from a database on :

```
cf4aff530715824c055892438a1ab6b2
```

You want to create a custom dictionary using the words on `neurosecurity.byu.edu` to see if you can crack the hash.

1. Create a custom dictionary using CeWL for the website

   neurosecurity.byu.edu: `cewl -v -d 2 -m 5 -w`

   `custom_dict.txt https://neurosecurity.byu.edu` Where:

   - `-v` runs CeWL in verbose mode.
   - `-d` is the depth to “spider” or crawl the website

- `-m` is the minimum word length
- `-w custom_dict.txt` is the name of your new custom wordlist or dictionary.

Give the command a minute or two to complete.

2. Check how many entries are in the `custom_dict.txt` file:

```
wc -l custom_dict.txt
```

3. Look at the words in

```
custom_dict.txt: less

custom_dict.txt
```

4. Permute the words in the `custom_dict.txt` wordlist using the "best64" rule, and append the output to `custom_dict.txt` (all one line):

```
hashcat custom_dict.txt -r /usr/share/hashcat/rules/best64.rule --stdout >> custom_dict.txt
```

5. Check how many entries are in the custom_dict.txt file now:

```
wc -l custom_dict.txt
```

6. Run Hashcat using custom_dict against the MD5 hash (all one line):

```
hashcat --force -a 0 -m 0 cf4aff530715824c055892438a1ab6b2 custom_dict.txt
```

Where `-m 0` signifies `md5` mode, and `-a` specifies "straight attack mode" (do not permutate the wordlist, because we already did)
The password will be reported towards the top of the output in the format: `hash:password`. If you miss the output, you can view it in your potfile once you have cracked it by running

```
hashcat --show cf4aff530715824c055892438a1ab6b2
```

7. Confirm that you found the correct password:

```
echo -n "<the plaintext password>" | md5sum
```

We include the `-n` flag because otherwise, the `echo` command will append a newline character, which will throw off the hash.

**Question:** What is the plaintext of the hash?

## Learn more:

- Podcast: Story behind the game-changing RockYou password breach:
- https://darknetdiaries.com/episode/33/ "Hash Crack: Password Cracking Manual (v3)"
- by @Netmux, one of the foremost password crackers.
  https://arstechnica.com/security/2013/03/how-i-became-a-password-cracker/
- Official Hashcat documentation: https://hashcat.net/wiki/

### *References:*