Lab 4: Passwords

A542 - Technical Foundations of Cybersecurity
Spring 2024

John the Ripper(JtR) (`https://www.openwall.com/john/`) is a useful tool for cracking password hashes. A version of john-the-ripper is already installed for you to use on the burrow server, you can execute it using the john command. In this lab, you'll be using it to crack hashed passwords.

1. Downloading Hashes Files

1. SSH to burrow, and in your home directory, create a new directory called `hashes` using the `mkdir` command.

2. Change directory (`cd`) to `hashes` and use the `nano` command to create 4 files with the given hashes. To do that, follow the steps below:

(a) Type `nano alice1` on the terminal, press enter. You will create a file called `alice1` and will be taken inside it. Copy and paste the following into it (in a single line):

```
$apr1$pPmOnZYu$cgLPRQy2OYR8fACwIW8Ls0
```

Press `ctrl+X`, type `y`, and then press enter twice to save and exit the file.

(b) Write `nano alice2` on the terminal, press enter. Copy and paste the following into it (in a single line):

```
$apr1$d.WxKyLH$zZFjXdR2b9HB2kHK2QZZN0
```

Press `ctrl+X`, type `y`, and then press enter twice to save and exit the file.

(c) Write `nano alice3` on the terminal, press enter. Copy and paste the following into it (in a single line):

```
$apr1$hrZTEJOr$0SEv2CjeQmKq4saZnCtd8.
```

Press `ctrl+X`, type `y`, and then press enter twice to save and exit the file.

(d) Write `nano alice4` on the terminal, press enter. Copy and paste the following into it (in a single line):

```
$apr1$BK.34kf3$IFA5TplksWGXKZr0QSkLs.
```

Press `ctrl+X`, type `y`, and then press enter twice to save and exit the file.

3. Run the command `ls` to check and see if all four files have been created properly.

2. Basic Usage

1. Each of the created files contains the hash of the password that you need to crack.

2. You can use John to crack passwords by running the following command: $ `john ~/hashes/<name of the file>`

3. You can see the cracked password by using the command $ `john --show ~/hashes/<name of the file>`

4. If you face problems, go to the link `http://www.openwall.com/john/doc/EXAMPLES.shtml` and read about features and usage.

5. Go to the link `https://www.openwall.com/john/doc/RULES.shtml` and read about word mangling rules.

3. Password Cracking [15 points]

You need to crack all four files you downloaded using John.

1. The first hash (`alice1`) is very easy to crack because the password used to generate this hash is a word taken from the dictionary. You should be able to crack the password using the command in section 2.2.

Password: gocougs

2. `[5 points]`The second hash (`alice2`) is a bit more complicated than the first. The password used to generate this hash is made up of a word from the dictionary followed by a two-digit number. So if `apple` is a word from the dictionary then a possible password that fits the above description is `apple45`. You will not be able to crack this password by using the default configuration of John-the-Ripper. You can verify this by running the command in section 2.2 for the second hash file (alice2). However, this will run a long time before giving up, so please kill the process by pressing q after a minute or so to avoid overconsumption of burrow's CPU.
In order to crack this password we have to create a rule, which mangles the words in the dictionary to meet the above description and add it to a config file. Copy the default config file from /etc/john/john.conf to the ".john" directory in your home folder.

Password: Pentium97

```
$ cp /etc/john/john.conf ~/.john
```

The rule $`[0-9]`$`[0-9]` basically asks John-the-Ripper to append each word in the dictionary with two numbers (each between 0 and 9) before checking to see if it was used to generate the hash. We can now add the rule under `[List.Rules:Wordlist]` in the `john.conf` file. Use a command-line editor, like `nano`, or `vim`, to modify the file.
Note: We suggest you add the rule right below `[List.Rules:Wordlist]` so that John-the-Ripper runs this rule first (before checking for other rules). Now you should be able to crack the password using the config file:

```
$ john ~/hashes/<name_of_the_file>
```

3. `[5 points]` The password used to generate the third hash (`alice3`) is very similar to the second one except for a minor difference. This password has two numbers (between 1 and 2) after the first two characters and two numbers (between 1 and 2) at the end. Again, if `apple` is the word in the John-the-Ripper dictionary, the password would be something like `ap11ple12`. Generate a rule and add it to the `john.conf` file. Now, try cracking the password using the previous command.

Rule: i2[1-2]i3[1-2]$[1-2]$[1-2]

Password: va12l12

4. `[5 points]` The fourth hash (`alice4`), has been generated using a word

from the Dutch dictionary. So you will have to download the Dutch dictionary from this URL to crack this password.

Password: bijeengekomen

- Download the file using the `wget` command.

```
$ wget ftp://ftp.openwall.com/pub/wordlists/languages/Dutch/1-clean/lower.gz
```

- Unzip the file using the command:

```
$ gunzip lower.gz
```

- Rename the file from `lower` to `dutch` by using the command:

```
$ mv lower dutch
```

Now crack the password by giving the Dutch word list as input to John-the-Ripper.

```
$ john --wordlist=dutch ~/hashes/<name_of_the_file>
```

Note: Dictionaries for other languages can be downloaded from:

`ftp://ftp.openwall.com/pub/wordlists/languages.`

If there is more than one dictionary file for any of the languages, you will need to try some or all of them. Also, if John-the-Ripper does not crack the password in 3-5 minutes, you should quit the process using `q`, or `Control+C` and try other files. John-the-Ripper uses a lot of CPU time (and students are trying to share the CPU!), so it is important to not keep instances of John-the-Ripper running too long.

4. Questions [30 points]

1. [5 points] Is John-the-Ripper only useful for malicious hackers? If not, who else could benefit from using this tool? Justify your answer.
   - John the Ripper is useful for more than malicious hackers. It can be used for organizations or people who wish to use it for good purposes. A business or person could have forgotten their password but knows the format and could utilize this application to crack their own passwords.

2. [5 points] Explain briefly the process used by John-the-Ripper to crack passwords?

- First, there must be an existing hashed file of the password. Next, you command John to crack the password. John will crack it based on the conditions you give it with its algorithm. The conditions can include wordlists, numbers, or rule-based.

3. [5 points] Why does John-the-Ripper not run all possible permutations and combinations (as directed by the various rules) for each word in the dictionary by default?

- It is impractical to run all possible permutations and combinations. The resources required would be substantial. The costs would greatly outweigh the benefits of running all possibilities to crack a password.

4. [5 points] Based on your experience cracking the passwords, what kind of passwords do you think are secure? Justify your answer.

- Based on this lab I have realized a password that is longer is much better. Also, passwords that are arbitrary numbers, special characters, and random lower/uppercase letters are secure. This is because John only has so many capabilities so having a long arbitrary password will make it nearly impossible for John to crack it.

5. [5 points] In many systems, passwords are stored as salted hashes. Explain what a salt of size `n` bits provides in terms of security against precomputed dictionary offline attacks.

- Salt is added to a password before it's hashed to protect from a dictionary and rainbow table attack. N bits increase complexity by 2^n possibilities. The more salt bits the more secure a password is from a dictionary attack. The pre-computed dictionary attack is no longer of use to an adversary once salt has been added.

6. [5 points] Suppose you were designing a website with password-based user authentication. What are the security implications of using the same salt for all passwords vs. using different salts for each password, with regard to precomputed dictionary offline attacks?

- Using the same salt for every password would be a bad security practice. An adversary could find patterns in the hashed passwords and take the time to crack the salt. Once the salt is cracked, they could apply this to all the precomputed dictionary attacks making simple passwords vulnerable.
- Using a different salt for every password would eliminate patterns and randomize every password making it nearly impossible for an adversary to crack. The adversary would have to crack every salt individually which is not realistic.