CSci 4271W (011 and 012 sections) Lab Instructions

Lab 5 **February 24th, 2025**

Ground Rules. You may choose to complete this lab in a group of up to three students. Before you leave the lab, **make sure you have submitted to Gradescope, you included all group members on the submission, and the autograder found all required files!**

1 John The Ripper

In lecture 10 tomorrow, we will address the topic of authentication, the way that an operating system verifies that a user is who they claim to be. One of the main methods of authentication, that you are familiar with as a user, is by passwords. In principle, the operating system asks a user for their username and password, and then checks that the username corresponds to a valid account, and the password entered by the user matches the stored password for the account. In practice though, it is not a good idea to leave a file of user passwords sitting on disk or even in memory. So instead of storing the password directly, the operating system applies a complicated function to the password, called a *cryptographic hash function* (also sometimes "one-way hash function", or in this application a "password-based key derivation function").

A cryptographic hash function has the property that given a password, it is easy to compute the output of the function (called the "hash"), but given just the output of the function, we don't know any better way to compute the input than to try all possible inputs. So the operating system stores the hash of the password, and when the user logs in, computes the hash of the password entered by the user, and compares these password hashes. If they match, then the operating system can be sure that the user knows the password for the account. But now if the list of password hashes is stolen or leaked, the actual password can't be determined from this list, so the hash is useless for someone else trying to log into the account.

At least this is the idea. In reality, since people tend to choose somewhat predictable passwords, an attacker who gets a copy of the password file can recover a lot of users' passwords just by running a list of the most common passwords through the cryptographic hash function to see if they match the hashes stored in the password file. This is slightly complicated by the fact that most operating systems pick a random "salt" value of a few characters and prepend this to the password before computing the hash. It means that for each user, the attacker has to look at the stored "salt" value, and re-compute the cryptographic hash of each possible password using that user's salt. Another best practice is to make the hash function slower or more resource-intensive to compute such as by interating it many times or using an algorithm that requires a lot of memory.

Since this is a pretty common security-related task, there are several open-source tools – called "password crackers" – that automate this task. The current "champ" is probably hashcat, but that tool requires GPUs (the ability to use the parallel execution units is what makes it fast) and our VMs don't have GPUs. So in this lab, we'll use another older but still popular password cracker, John the Ripper. To match the limited CPU power of the VMs, we'll do experiments using an older Unix hashed password format that uses a pretty fast hash function.

Create a text (markdown) file called `lab5_notes.md`, where you'll keep a running journal of your lab today for submission.

1.1 Installing john

Log in to your 4271 VM and install john with:

```
$ sudo apt-get install john
```

You'll be prompted about installing another package with some password lists; hit enter to accept all of these prompts.

1.2 Download the example file.

Download the file labpasswords that has 4 password hashes:

```
$ git clone https://github.umn.edu/badlycoded/ripperlab.git
$ cp ripperlab/labpasswords .
```

1.3 Start cracking

You can start the cracker running on this file with:

```
$ john labpasswords
```

As john finds matches for the passwords, it will print them out. While you're waiting, you can read a little bit more about what john is doing at https://www.openwall.com/john/doc/EXAMPLES.shtml. TL;DR: john has a list of "most common" passwords (taken from cracking experiments done in the 90s) that it follows in order; if that doesn't succeed then it tries applying some "mangling rules" (like adding single characters in different positions, changing letters to numbers, and a few others) to the words on this list; then if that doesn't succeed it starts trying "randomly generated" passwords according to measurements about which letters are most frequently followed by other letters. So the quality of the word list can make a huge difference in the cracking time.

1.4 Make your own file

By now your john run will have found several of the four passwords, but maybe not all. Record the passwords it has found in your running journal, and stop the session by hitting Ctrl-C. Then check out the default list used by john: http://cs4271.org/password.txt. Pick one word from near the top of the list, one from near the bottom, and add a single digit to the end of one from somewhere in the middle. Record these passwords, then make a file with their hashes by running the following command (on your VM):

```
$ openssl passwd -1 -stdin >mypasswd
```

Then type one of your words per line, starting with the third password. When you've entered all three passwords, hit Ctrl-D. We're going to run john against your password file, but first make a prediction: which password do you think will be cracked first? Which one last? Why? Record your answers in your journal. Also, run cat mypasswd on your VM to see the list of hashes, and paste these into your journal as well.

Now run john on your new password file:

```
$ john mypasswd
```

What order were the passwords found in? Was your prediction correct? Now generate another password file with a single password:

```
$ openssl passwd -1 -stdin > rpw
```

Choose 6 random letters as your password, hit enter, then `Ctrl-D`. Make a prediction about whether `john` will crack this password within 5 minutes, and record your password and prediction in your journal. Then run `john` against your password:

```
$ john rpw
```

While you wait to see if your prediction was correct, read about the rules john can implement at https://www.openwall.com/john/doc/RULES.shtml. If your cracking run gets to 5 minutes without producing a result, hit the space bar in the terminal window to see what password was most recently attempted, record this in your journal, then kill the process with `Ctrl-C`.

1.5 All done!

Once you've recorded the progress of your cracking run and killed the process if it was unsuccessful, you're all set for the lab! As mentioned in the introduction, for this lab you'll submit to Gradescope the markdown file `lab5_notes.md` that you've recorded your progress in. You can submit this to the Lab 5 assignment in Gradescope. Make sure you include all of your group members in the submission!

Once you've submitted the file, the autograder will test to make sure the proper file was submitted, and notify you if it's missing, within a few minutes.

If you want more: Note in particular that is not required that `john` crack all of the passwords in the original `labpasswords` list to complete the lab, and in fact the VMs may not be able to crack the last hash with its default settings during the lab period. But if you want to help it along, you might look to see whether you notice a pattern in the first three passwords. You can construct your own guesses for `john` to try and supply them via the wordlist feature. But this is just for fun.

Congratulations, you've finished Lab 5!