Introduction to Cybersecurity Elisabeth Oswald

System Security: Password Cracking

2023

Overview

This lab sheet complements the other lab sheet pertaining to the "System Security" topic of Introduction to Cybersecurity. As I keep mentioning, system security is a vast topic and therefore we can only dip into some aspects. This lab is devoted to the topic of password security—passwords are still omnipresent and they represent one the most challenging aspects of real world cybersecurity systems.

We authenticate users via passwords: they often implement the "something you know" version of authentication. Even if more and more systems are now utilising more than a single factor for authentication, typically one of the factors is a password.

Password security is thus important, and I would like you to get a better understanding of how password "crackers" work. Thereby I don't mean a theoretical understanding of password hashing and the like, but I want you to get your hands dirty and actually get familiar with one of the most widely used password cracking tools today: `hashcat`.

Requirements

To do this lab you need to install `hashcat` on your system. You can find download options at `https://hashcat.net/hashcat/`.

Linux: there are binaries available of hashcat that should work out of the box.

MacOs: you may be able to install it via brew (`brew install hashcat`), or, then clone their git and run `make` in the respective directory (in my experience, this works a treat).

Windows: there are binaries available of hashcat that should work out of the box. (You must be able to use the CommandLine though).

On Moodle you are given a dictionary file `rockyou.txt`, which you should download and place in the folder in which you execute hashcat. There is also a file called `hashfile.txt`, which you need section six.

What you should get out of this.

Authentication and access control are vital components of any security infrastructure. The use of passwords is still among the most widely deployed ways of authenticating users. In this lab we look at password security, and do some hands on password cracking using the `hashcat` tool. Using this tool, you should get a first hand experience of how easy it is to crack passwords that follow a simple pattern, or that are "popular".

We do this activity largely based on MD5 hashes. You should know that MD5 is an insecure hash function, in the sense that it is nowadeays easy to find two inputs that hash to the same value. This is clearly an issue, but the reason why we do this activity based on MD5 is that it is also very quick to compute. In a practical setting, clearly all your alarm bells should go off if you were to discover that a system was still based on MD5 hashes.

Assessment

There is a dedicated Moodle quiz for this activity: in contrast to the other labs/work sheets, you need to progress linearly through this one as activities built upon each other, and each activity is mapped to a Moodle quiz question.

Generate some hashes

In order to compute MD5 hashes you can use `md5` (MacOs) or `md5sum` (linux), or you can use Python `import hashlib`. For instance under Linux you can generate the md5 hash of the string "password" like this:

```
echo -n "password" | md5sum | tr -d "-" >> pw_hashes.txt
```

You can also do this via the Python hashlib library, which should come with Python 3 per default.

In your Python environment, import the hashlib library, and the compute the MD5 hash of your name via
`hashlib.ALGO_NAME(b"password").hexdigest()`
whereby you replace "ALGO_NAME" with "md5" and "STRING" with your chosen string. You can then save the resulting MD5 hashes into a text file.

When you are able to do this, you should be able to solve the first question of the corresponding quiz.

Crack your first hash

Now we'll use hashcat to invert a hash function. We will do this with MD5 so it doesn't take too long. Hashcat is actually a really cool tool, that supports GPU usage and parallelism, so if you run it on something beefier than just your standard machine then it can do a lot!

But we start small: let's run hashcat on the string "6a67dfc8c2584354e4c8ce9302da7dfb": `./hashcat -m 0 -a 3 6a67dfc8c2584354e4c8ce9302da7dfb`[1]
(the `m` flag 0 selects MD5 and the `a` flag selects brute force cracking mode). Hashcat should terminate within a minute and claim success (the "Status" should show "Cracked"). Hashcat puts the recovered input into a file with the extension "potfile". In Linux or MacOs you can use `cat` to show the content of the file:
`cat hashcat.potfile` In Windows you may have to open the file in an editor.

Hashcat appends all further successful cracking results in this file per default. Now you should be able to solve the second quiz question.

Crack a longer password

So far we have dealt with short passwords (you might have noticed). Let's move up to something slightly longer.

Beware: ensure that you know how to terminate a process in case hashcat becomes unresponsive!

Try and crack the MD5 hash `c226ad1a3da0beec54add9f75c8b6467`.

It is unlikely that you will succeed (unless you run hashcat on something that is performance enhanced) in a reasonable period of time.

Hashcat enables to "weaken" brute force search by allowing you to specify a "search mask" for a password. This is a powerful tool, and makes sense if you have reason to believe that a password has a particular structure. For instance, if you suspect that a password consists of 5 characters, whereby the first character is likely to be an upper case, the next three characters are lower case and the last character is a symbol you can supply `?u?l?l?l?s` as an argument to instruct hashcat to narrow its brute force search to passwords that have this structure.

Try and crack the MD5 hash from before assuming that the first letter is upper case, the second character is a digit, the third and fourth characters are upper case, the following character is lower case, and the final two characters are digits (you may have to consult the hashcat help).

[1] You don't need the "./" if you are working under Windows,

Now you are ready to do the third quiz question. The password for the third question is structured as well. You can assume that the first character is upper case, the second character is a digit, the third and fourth characters are lower case, and the final characters are upper case (this is deliberately vague).

Crack many passwords

In real life scenarios, adversaries are typically not after the password of a specific user. Instead, they will leverage any weak password, given an exfiltrated file containing hashed passwords.

Make a ".txt" file containing a few hashed passwords like "i2c, 1234, abc, 7890, $password, 4doon3?1, yan123, pwd%".

Linux: if doing this on the command line, then
`echo -n "itc" | md5sum | sed 's/[-[:blank:]]//g' >> hashfile.txt` (repeated for different passwords) should work.

MacOs: if doing this on the command line, then `echo -n 'itc' | md5 >> hashfile.txt` (repeated for different passwords) should work.

(You can view the file via `cat hashfile.txt`).

Python: if you are generiting md5 hashes in Python then you can create a textfile via

```
txtfile = open("hashfile.txt", "w");
txtfile.write(hashlib.md5(b"itc").hexdigest() + "\n");
txtfile.write(hashlib.md5(b"1234").hexdigest() + "\n")
.
.
txtfile.close()
```

Now run hashcat over this file via `./hashcat -m 0 -a 3 hashfile.txt`, and stop if after a minute or so (either via [q]uit, or by pressing ctrl-c). The potfile should reveal that hashcat broke the short passwords very quickly.

Now you are ready for question four: use the file `hashfile.txt` that is supplied on Moodle and find the weak password in there.

Crack passwords using a dictionary

In the real world scenario that we have begun to investigate, we are often given a password file that contains the hashes of many passwords, and we hope that at least one of the

passwords is weak enough that we can crack it.

Create a list of 10 passwords of different strengths (according to your understanding of "strength") and then use the supplied dictionary "rockyou.txt" to help hashcat crack at least one password: i.e. use something like
`./hashcat -m 0 -a 0 hashes.txt rockyou.txt`. Include one of your own recent passwords. Report briefly on your results in the essay type quiz question (i.e. question number five).

The file "rockyou.txt" comes from an attack that took place in 2009 on the website `rockyou.com`. They had their users passwords stored in plain on their server, which was vulnerable to an SQL injection. They had about 32M users, which all had their passwords compromised as a result: `https://www.theguardian.com/technology/blog/2009/dec/15/rockyou-hacked-passwords`. The supplied file contains these compromised passwords and is a pretty good example of a useful dictionary.

Work with salted passwords

Salting passwords refer to the idea that if we add a random string to a user password before hashing, then we increase the search space for a brute force attack (even if a dictionary is used) proportional to the salt. This makes time-memory trace offs, like computing hashes for the most popular passwords, ineffective.

Salting works by creating a random string for each user password and either appending or prepending that string prior to computing the hash. The salt is then stored alongside the hash in the password file.

Create a hashcat compatible salted password file for the following passwords and salts:

itc	salt0
345	salt1
hello	salt2

Now use hashcat to break the passwords in this file, using the provided dictionary "rockyou.txt". Now you are ready for the final question in the quiz.