University of Chicago
CMSC 23200/33250: Introduction to Computer Security, Winter 2020
Course Staff: David Cash, Alex Hoover, Rohan Kumar, Blase Ur, Valerie Zhao

Assignment 6: Passwords

Due at 11:59pm Monday, February 24

Introduction

This assignment explores a few different aspects of the password and authentication ecosystems. In particular, you will observe how passwords are stored on Linux systems, gain experience cracking passwords using existing password-cracking software (albeit in configurations you will have to choose), leverage the information you gain in a rate-limited online attack, and take advantage of a side channel to learn more about a system.

In this assignment, the flags you will be aiming to capture are passwords for different users.

Before getting to the problems, we discuss the rules for this assignment and how you can access the assignment infrastructure for your attacks.

Rules

Collaboration policy. Please respect the following collaboration policy: You may discuss the problems with up to 3 other students in the class, *but you must write up your own responses and your own code. You should never see your collaborators' writing, code, flags (passwords in this assignment), or the output of their code.* At the beginning of your submission write-up, you must indicate the names of your (1, 2, or 3) collaborators, if any. You may switch groups between assignments but not within the same assignment.

This should go without saying, but you should not capture anyone else's flags.

Sources. Cite any sources you use. You may Google liberally to learn basic Python, learn how to navigate Linux, and learn how to use Hashcat. Unlike some past assignments, you are welcome to perform additional Google searches on other topics related to this assignment, but be warned that these are unlikely to help you much. You **must**, however, note at the end of your solution for each task any topics you ended up Googling to complete that task.

Campuswire. We encourage you to post questions on Campuswire, but do not include any of your code in the Campuswire posts. If you have a question that you believe will reveal secrets you have discovered while working on the assignment, post privately to just the instructors. If you have a question that you believe will be of general interest or clarifies the assignment, please post publicly. If you are uncertain, post privately; we will make public posts that we believe are of general interest.

Outside attacks. Your attacks for this assignment should be those discussed below. Do not attempt to compromise our server, sniff your classmates' network traffic, or do other nefarious

things. You will not receive credit for breaking into the server. In fact, you will lose credit for doing so.

Grading. Responses will be graded for correctness and clarity.

Assignment Tech Set-Up and Overview

This assignment begins (Problem 1) on your VM from the previous assignment and ends with sending HTTP queries to our server (Problems 2-3).

For Problems 2 and 3, which require that you complete Problem 1 first, you will be making queries to a domain that we surprisingly now own: `https://uchicago.computer/`. We have provided sample code for using both Python 2 and Python 3 to make queries to the server with a `make_query` function. Different from past assignments, our queries to the server are *not* in Base64, but rather reflect basic URL-safe quoting (and unquoting on the server side).

As in past assignments, you may implement your code in whatever programming language you choose, although we recommend Python, for which we can provide the most support. You will be submitting your code (when applicable) alongside descriptions of what you did, as detailed in each task below.

What and How to Submit

You will submit a set of files:

1. A file `<YOUR CNETID>-writeup.pdf/txt` that briefly describes how you solved each problem. This file is shared across all four problems.
2. The individual files (code and/or results) requested by each of the problems, detailed below.

You will upload these files to Canvas. You may zip/tar/gzip/7zip/etc. them or submit them individually.

Final Notes and Hints

Please don't DDOS the server. It will support thousands of queries per student (which you will need), but not millions.

This assignment was built by Blase last year for this class, and it was modified this year based on feedback from last year. Please let us know if you find bugs or need instructions fleshed out by posting on Campuswire.

Problem 1: Password Cracking

200 of your closest friends have somehow created accounts on your VM. Well, now that they're here, you might as well figure out their password.

On your VM, there is a file `/etc/passwd` that anyone can read, but this is not the file you want; you want `/etc/shadow`. Why? See `https://en.wikipedia.org/wiki/Passwd`. The problem is that if you try to access this file as a normal user, access will be denied. However, since we gave you *sudo* access for the last assignment, you can become a superuser and access this file. You now

have the `/etc/shadow` file. You will notice inside the file that there are 200 users with usernames that start with *your* CNetID followed by an underscore. We will refer to these 200 usernames as the users assigned to you. Your first step for this problem is to extract only the lines corresponding to the 200 users assigned to you from the shadow file; there are a handful of accounts made by the system for various purposes.

Unfortunately, this shadow contains hashes, yet you want (plaintext) passwords. Thus, you'll need to crack the passwords! You will probably want to download Hashcat and install the appropriate GPU drivers (if you have an appropriate GPU); it also works on CPUs. It supports Linux, OS X, and Windows.

When you get a set of hashes, the first step is to figure out what hash function was used. Typically, you would use an online version of the Hashtag script (`http://www.onlinehashcrack.com/hash-identification.php`) to identify possibilities. For instance, try 5f4dcc3b5aa765d61d8327deb882cf99, which is "password" hashed with MD5. Once you know which (or which set of) hashes are possible, you will need to know the numerical hash function parameter (the -m parameter) for Hashcat from their list (`https://hashcat.net/wiki/doku.php?id=example_hashes`). In this case, though, you could also learn how the hashes are stored by simply examining the shadow file (see `https://www.cyberciti.biz/faq/understanding-etcshadow-file/`). Before you try to crack any hashes, you will want to extract just the hash portion of each line and feed just these hashes to your cracking software. The examples below assume you saved this file as `hashes.txt`.

You can run Hashcat on your machine or your VM. To run Hashcat, go to a command prompt, navigate to the directory where the files are, and call the appropriate binary. For example, on Ubuntu, I call `./hashcat64.bin`.

This won't do anything, though. You need to point Hashcat towards the file of hashes you want to crack (giving it the full path if it's not in the same directory as your Hashcat executable). You must also specify an attack mode. Your successful cracks will appear in the `hashcat.pot` file in the same directory as Hashcat, though you can change the output file using the -o option.

Here are some sample attack modes:

- `./hashcat64.bin hashes.txt -m 100 -a 3 ?l?l?l?l?l?l?l`
  try to crack hashes in hashes.txt that are hashed with SHA1 (-m 100) using the brute-force/mask mode (-a 3) of Hashcat, trying all 7-character strings of only lowercase letters
- `./hashcat64.bin hashes.txt -m 100 -a 0 pw.txt`
  try to crack hashes in hashes.txt that are hashed with SHA1 (-m 100) using the wordlist mode (-a 0) of Hashcat, drawing its guesses from the file pw.txt (which you would have to provide)
- `./hashcat64.bin hashes.txt -m 100 -a 0 -r ./rules/best64.rule pw.txt`
  try to crack hashes in hashes.txt that are hashed with SHA1 (-m 100) using the wordlist mode (-a 0) of Hashcat, drawing its initial words from pw.txt...and also mangling those entries with the Best64 mangling rules

Note that you'll need to edit the sample commands above to reflect the proper binary for your operating system, specify the mode (-m) that actually applies for the hashes you have, provide valid paths to the rulelist (if applicable) and wordlist (if applicable) you want to run, and so on.

Here are some initial resources that may help, but you will almost certainly want additional wordlists and/or rulelists.

- Lots of password breaches/other word lists: `https://wiki.skullsecurity.org/Passwords`
- Additional wordlists: `http://contest-2010.korelogic.com/wordlists.html`

And this is how you get the password.

That said, you do not have to guess all passwords to receive full credit! **If you successfully guess the passwords for 80 of the 200 users assigned to you who do have accounts on the server, and your code and write-up are sufficiently descriptive, you can receive full credit for this section.** If you find as much joy in cracking passwords as we do, try to crack as many as you can. We'll figure out some reward (snacks, etc.) for the people who crack the most.

What to submit. You do not need to submit any code for this problem.

You must submit the passwords you crack in one of two formats. If you used Hashcat, you can submit the .pot file that Hashcat outputs to store successful cracks as `<YOUR CNETID>-problem1.pot`. If you don't use Hashcat (or if you really want to reformat the output for whatever strange reason), you may instead submit a file `<YOUR CNETID>-problem1.txt` that includes the usernames and corresponding passwords you successfully guessed. Each line of this file should contain a username, followed by a **tab character** (`\t`), followed by the plaintext password that was a successful guess. Do not include the usernames of users whose passwords you did not successfully guess.

In your shared write-up file, `<YOUR CNETID>-writeup.pdf`, describe your approach to solving this problem. In particular, be sure to note what machine you used, what configurations of Hashcat you tried (word lists, rule lists, etc.), and comment briefly on the success or failure of the main cracking strategies you employed.

Problem 2: Using Side Channels to Identify Who Has An Account

In Problem 1, you discovered 200 accounts on the VM that were assigned to you (began with your CNetID). It turns out that some of them, but not all of them, also have accounts on `uchicago.computer` . You can try to log into the server as a given user by making a query to `https://uchicago.computer/<USERNAME>/<PASSWORD>/`, replacing <USERNAME> and <PASSWORD> with the values you want to test. We have once again provided sample code for using both Python 2 and Python 3 to make queries to the server with a `make_query` function. If you choose not to use this function, note that the username and password should both be percent encoded (see `https://en.wikipedia.org/wiki/Percent-encoding`) as performed by the `urllib.parse.quote` function in Python3.

Use information that leaks from the server to determine who (among the 200 usernames assigned to you) does, and who does not, have an account. Note that *time is of the essence.*

What to submit.

Include a code file `<YOUR CNETID>-problem2.py` (or substitute an appropriate extension for the language you used) that includes any code you wrote to solve this problem.

Include a file `<YOUR CNETID>-problem2.txt` that includes just the usernames, one per line, of the users assigned to you who *do* have accounts on the server. **If a user from Problem 1**

appears not to have an account on the server, do not include them in this file.

In your shared write-up file, `<YOUR CNETID>-writeup.pdf`, briefly describe your approach to solving this problem and hypothesize why this information about who has accounts is leaking. Subsequently, discuss how to fix this leak.

Problem 3: Online Attacks (20 points)

In Problem 2, you identified that some of the users who had accounts on the VM also have accounts on `uchicago.computer`, whereas others do not. Problem 3 concerns only the users assigned to you who *do* also have accounts on `uchicago.computer`. You will be attempting to log into the server as them following the same query format as in Problem 3. Note that the server returns either "Success" or "Failure" for each query you make that is not timed out. You can observe this in action for the user "student" whose password for the server is "1234".

Note that this server employs conservative rate-limiting, so your strategy of making lots of guesses in Problem 1 will not work here. In particular, you may only make one guess every five minutes against each account. If under five minutes have elapsed since your last guess, the server will return "Timeout" instead. Thus, you'll need to make every guess count. Hint: what you learned in Problem 1 about each user will help you greatly with this task, though sometimes a little twist is necessary. Note that you **may not** attempt to make guesses against any of your classmates' accounts in this section. We will check the server logs!

You do not have to guess all passwords to receive full credit! If you successfully guess the passwords for **30% of the users assigned to you who have accounts on the server** (i.e., 30% of the people from Problem 2), and your code and write-up are sufficiently descriptive, you can receive full credit for this section.

What to submit. Include a code file `<YOUR CNETID>-problem3.py` (or substitute an appropriate extension for the language you used) that includes any code you wrote to solve this problem.

Include a file `<YOUR CNETID>-problem3.txt` that includes the usernames and corresponding passwords you successfully guessed. Each line of this file should contain a username, followed by a **tab character** (`\t`), followed by the plaintext password that was a successful guess. Do not include the usernames of users who do not have accounts on the system. If you did not guess a particular user's password, don't include them in this file.

In your shared write-up file, `<YOUR CNETID>-writeup.pdf`, briefly describe your approach to solving this problem.