University of Chicago
CMSC 23200/33250: Introduction to Computer Security, Autumn 2018
David Cash, Blase Ur, and Ben Zhao

Assignment 4: Passwords and Authentication

Due at 11:59pm Tuesday, November 20

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.

Piazza. We encourage you to post questions on Piazza, but do not include any of your code in the Piazza 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 on a Linux virtual machine image (Problems 1-2) and ends with sending HTTP queries to our server (Problems 3-4).

First, if you don't already have virtual machine software, we strongly recommend that you download and install Virtual Box for your operating system: `https://www.virtualbox.org/wiki/Downloads`. Note that VMs sometimes interact poorly with secure boot. Please feel free to post on Piazza if you have any issues getting VirtualBox installed.

If you are registered for 23200 and your last name starts with the letters A – V, follow these instructions: Download the VM image we have prepared for this assignment from `http://securityclass.cs.uchicago.edu/security-2018.ova` (if on campus or connecting through the UChicago VPN) or, if necessary to connect from off campus without the VPN, from `https://super.cs.uchicago.edu/security-2018.ova`.

If you are registered for 23200 and your last name starts with a W, X, Y, or Z, follow these instructions or if you are registered for 33250 and have any last name: Download the VM image we have prepared for this assignment from `http://securityclass.cs.uchicago.edu/security-2018b.ova` (if on campus or connecting through the UChicago VPN) or, if necessary to connect from off campus without the VPN, from `https://super.cs.uchicago.edu/security-2018b.ova`.

Now that you have the VM image, load the image into VirtualBox. The VM is a minimal installation of Ubuntu Linux 18.04. We have made an account for you with the username **student** and the password **uccs**. If you are not seeing the "student" account mixed in among the many others, click the "Not Listed?" button at the bottom, which will bring you to a screen where you can type in the username. At this point, you are ready to complete Problems 1 and 2. Note that on Ubuntu "gedit" is a graphical text editor, "vi" is installed as a text-based text editor, and we have favorited the terminal on your task bar. The terminal is also accessible through the ctrl-alt-t keystroke.

For Problems 3 and 4, which require that you complete Problems 1 and 2 first, you will return to making queries to your favorite server: `http://securityclass.cs.uchicago.edu/`. 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. Different from these past assignments, our queries to the server are *not* in Base64, but rather reflect basic URL-safe quoting (and unquoting on the server side). Recall from the first two assignments that we have deployed this server in private IP space. That means that the server is not accessible from outside the UChicago network. If you are on campus, you should be able to access the server directly. You can test this by loading `http://securityclass.cs.uchicago.edu` in a browser. To access the server off-campus, use the UChicago VPN. See `https://cvpn.uchicago.edu` for instructions.

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. See `https://www.classes.cs.uchicago.edu/archive/2018/fall/23200-1/` for the link to the appropriate Canvas site.

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 fresh for this class. It is the first time it has been deployed in any setting. Please let us know if you find bugs or need instructions fleshed out by posting on Piazza.

Problem 1: Getting Root Access and Stealing the Password File

As previously mentioned, on the VM you have an account with the username **student** and the password **uccs**, and this is a user with intentionally limited access to the system (i.e., not a superuser). However, you are not alone on this machine. There are thousands of other users on the system, 200 of which are assigned to you. You will worry about them in Problem 2. For now, your goal in Problem 1 is just to steal the file containing the password hashes of all other users on the system.

There is a file `/etc/passwd` that you (as the user "student") 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, access will be denied. However, if you carefully peruse /etc/passwd, you might notice another user on the system – `mavenben` – who it turns out has sudo access to the system. This knowledge about mavenben is useful because, if you could log in as mavenben, you could access (and exfiltrate) the shadow file.

So do this. Log in as mavenben and exfiltrate the shadow file. This might seem hard – what could mavenben's password be? However, it turns out that mavenben is not as wise, and definitely not as creative, as their name implies. Good luck!

Note that you don't need to use the GUI to try and log in as mavenben, but can log in as someone else from the command line. See the `su` command.

What to submit.

You are not required to write any code to solve this problem, but it would probably make your life easier to do so. If you wrote code, include a code file `<YOUR CNETID>-problem1.py` (or substitute an appropriate extension for the language you used) that includes any code you wrote to solve this problem.

Don't submit the shadow file. If you complete the other problems, it tells us implicitly that you got the shadow file.

In your shared write-up file, `<YOUR CNETID>-writeup.pdf`, briefly describe your approach to solving this problem and include mavenben's password in bold text.

Problem 2: Password Cracking

You now have the `/etc/shadow` file. You will notice inside the file that there are thousands of users, but only 200 of them have usernames that start with *your* CNetID. 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.

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 can 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. To be consistent with our examples below, save this file as `hashes.txt`.

You can run Hashcat on whatever machine you want; you do not need to use the VM for this problem. To run Hashcat, go to a command prompt, navigate to the directory where the files are, and call the appropriate binary. For example, for the 64-bit CPU-based Hashcat on Ubuntu, I call `./hashcat-cli64.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:

- `./hashcat-cli64.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

- `./hashcat-cli64.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)

- `./hashcat-cli64.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

Here are some resources that may help:

- Lots of sets of password breaches and other word lists: `https://wiki.skullsecurity.org/Passwords`
- An English dictionary: `https://super.cs.uchicago.edu/usable18/dictionarysmall.txt`
- A list of Common first names: `https://super.cs.uchicago.edu/usable18/names.txt`
- Additional wordlists from KoreLogic: `http://contest-2010.korelogic.com/wordlists.html`

You do not have to guess all passwords to receive full credit! If you successfully guess the passwords for 75% of the 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.

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>-problem2.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>-problem2.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 of the main cracking strategies you employed.

Problem 3: Using Side Channels to Identify Who Has An Account

In Problem 2, 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 securityclass.cs.uchicago.edu. You can try to log into the server as a given user by making a query to `http://securityclass.cs.uchicago.edu/assignment4/<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* in solving this problem.

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 just the usernames, one per line, of the users assigned to you in Problem 2 who *do also* have accounts on the server. If a user from Problem 2 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.

Problem 4: Online Attacks (20 points)

In Problem 3, you identified that some of the users who had accounts on the VM also have accounts on securityclass.cs, whereas others do not. Problem 4 concerns only the users assigned to you who *do* also have accounts on securityclass.cs. 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. You can observe this in action for the user "student" whose password for the server is "12345".

Note that this server employs conservative rate-limiting, so your strategy of making lots of guesses in Problem 2 will not work here. In particular, you may only make one guess every five minutes against each account. Thus, you'll need to make every guess count. Hint: what you learned in Problem 2 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 67% of the 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.

What to submit. Include a code file `<YOUR CNETID>-problem4.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>-problem4.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.