# CIT 285 - Lab #6: Passwords

## Lab Preparations

In this lab perform the steps in your Kali VM. **Copy and pasting commands from this lab document to the terminal will result in errors for some commands, due to character set differences.**  Type the commands yourself, using bash features like command history and pathname completion to reduce the amount of repetitive typing.

## 1: MD5 Hash Cracking

Password leaks have shown us that many web applications use simple MD5 passwords without salts. These passwords are easily cracked by anyone with a web browser to access the wide variety of reverse hash lookup database web sites.

Using reverse hash databases such as the ones listed on the Cybersecurity Resources page of the class web site, crack the MD5 hashes in the left column of the table below. Write the corresponding password and the database you found it in in the right two columns of the table. *There are numerous free, web-based hash cracking tools on available. All hashes in the table can be cracked, which means you may need to try multiple hash cracking databases*.

|  |  |  |
| --- | --- | --- |
| **MD5 hashes** | **Passwords** | **Database** |
| 84f928034c38d9a079d8bd411d820a1f | ILUVCAMERON | **hashkiller** |
| 844427886b4adb91cff8e12d59cfff1a | mylinkedin | Hashkiller |
| 509081b7f8d9d0d03d6f44fcf938dc5c | mylinkedin1 | Hashkiller |
| 0e13ca0098665801d3e304351adb16d3 | slartibartfast | Hashkiller |
| e541ca7ecf72b8d1286474fc613e5e45 | Ncc-1701 | Hashkiller |
| a2e42a76643b77bfccf66229ae6deac1 | gonewiththewind | Hashkiller |
| 8e698ed576c307a4faaa42cec81abc6d |  | Hashkiller |
|  | egadbaddad |  |

## 2: UNIX Password Database

SSH from your Kali VM to your Metasploitable 2 VM and examine the contents of the passwd file. Read the man page to learn the meaning of each colon-separated fields in the file.

**On Kali**

**# ssh msfadmin@META\_IP**

**When connection is open to Meta VM issue the commands:**

**$ cat /etc/passwd**

**$ man 5 passwd**

2.1: What is the string used as a placeholder in place of the password in /etc/passwd?

Text

Description automatically generated

**X is used as a place holder**

2.2: Attempt to examine the contents of the shadow file.

* What error message do you see?
* How is that error message related to protecting user passwords?

**# cat /etc/shadow**

**Our permission is denied. Only the root and user can access this file, it is meant to keep information secure to certain people. This applies a limited access.**

2.3: The mkpasswd command can be used to generate password hashes suitable for use in a UNIX password database. In the example below, a salted password hash for the word “test” was created using the sha-512-crypt format, which is often used in modern UNIX systems.

$ mkpasswd -m sha-512 test | tee passtemp

**$**6**$**BAJIXVc3cajY**$**Bmyb7qy0jhZpWcYMSvn0aNDSJJniPyZTR9C8rjgA.gtSOTlcnpFV8xbQefJegVoiYe1OYK9qULst3rvuUwOB60

The output from above has been divided into three fields below, each separated by a **$** using the ‘sed’ command to print each component on its own line.

$ sed 's/\$/\n/g' passtemp

6

BAJIXVc3cajY

Bmyb7qy0jhZpWcYMSvn0aNDSJJniPyZTR9C8rjgA.gtSOTlcnpFV8xbQefJegVoiYe1OYK9qULst3rvuUwOB60

* The first field is a number that indicates the type of password hash, which we know is sha-512 since we used that argument in the mkpasswd command
  + *See “man crypt” for a full list of password hash types.*
* The second field is a random salt generated by mkpasswd.
* The third field is a sha-512-crypt hash of the password “test” and the random salt in the second field. Remember that sha-512-crypt is not just a SHA-512 hash, it's a SHA-512 hash iterated 5000 times to increase the amount of time required to crack it.

Run the same mkpasswd command line below in a new Kali terminal. Write or screenshot the outputs from the command then answer the following questions.

* Which fields of the output are different?
* Why are they different?

**$ mkpasswd -m sha-512 test**

**The second and third field outputs are different. Since the salt is random the outputs will be random.**

## 3: Password Cracking with Word Lists

In this part of the lab, we will use John the Ripper, a powerful open source password cracking tool. It can crack passwords using word lists and a set of permutation rules, as well as brute force guessing. John supports dozens of password hash formats. While John is capable of guessing the hash format, it is best to specify the hash format because some hash formats look identical. For example, raw MD5, LM, and NTLM hash formats are all represented by 32 hexadecimal characters. The best way to distinguish the hashes is to know what type of system the hashes were retrieved from.

* John's documentation can be found at <http://www.openwall.com/john/doc/>. You can also get help with the command john --help | less. In this lab, we use John version 1.9.0-jumbo-1.

The command below will create a new passwd2 file, add a user account and use the mkpasswd command to create a hash of a given password.



* Notice in the command in the image, there are backticks and not single quotes around the mkpasswd command.

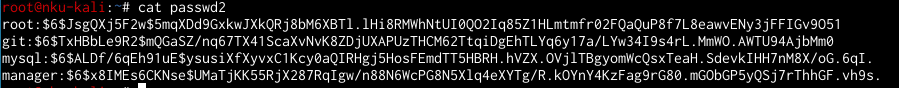
Use to the cat command to verify your passwd2 file is similar to the image below



Next, create hashed passwords for the remaining user accounts listed in the table below. **Remember to use redirection arrows in your command to prevent overwriting the file**. If not, you’ll overwrite everything each time the command is issues.

|  |  |
| --- | --- |
| **Useraccount** | **Password to Hash** |
| root | password |
| git | password1 |
| mysql | password2019 |
| manager | p@ssword |

After all accounts and password hashes have been created, use the cat command to verify your passwd file is similar to the image below.



3.1: We will first use John the Ripper in word list mode without any permutation rules. In this mode, John will try the passwords in the word list one by one, storing matches in the **john.pot** file located in your **~/.john** directory by default. If a word list is not specified on the command, John will use its default word list.

* How many passwords does John's default word list contain?

**$ wc -l /usr/share/john/password.lst**

3557 passwords

3.2: Let us attempt to crack the **passwd2** file we created at the beginning of this section using wordlist mode. When you run John the first line will display how many password hashes were found, along with how many salts, if any were used. In our example, the first line of output should be:

Loaded 4 password hashes with 4 different salts (sha512crypt, crypt(3)

If this line displays 0 (zero) password hashes found, that means that John could not find any hashes in the specified file. In this case, it is likely an incorrect file or hash format was given

Issues the following command

**$ john –-crack-status --wordlist –-format=sha512crypt passwd2**

When using the --crack-status option in the command, John will report each password found once cracked. The report lines look like the following, which should appear soon after you type the command above.

Loaded 4 password hashes with 4 different salts (sha512crypt [64/64])

Loaded 4 password hashes with 4 different salts (sha512crypt [64/64])

password (root)

guesses: 1 time: 0:00:00:00 0.04% c/s: 66.66 trying: password1

password1 (git)

guesses: 2 time: 0:00:00:00 0.08% (ETA: Fri Jan 6 13:19:51 2017) c/s: 78.94 trying: 123456789

guesses: 2 time: 0:00:00:24 DONE (Fri Jan 6 13:20:16 2017) c/s: 286 trying: sss

Issue the following command

**$ john --show passwd2**

* Based on the output from the two commands issued, which passwords were you able to crack in this mode? Include the outputs from both commands below.

We are able to crack into 1 hash password.

3.3: One technique to improve our ability to crack passwords is to use a better wordlist. While increasing the size of wordlists helps, it is important to have the right words, that is, words that people are likely to use as passwords. Due to the high frequency of large password leaks, there are freely available lists of millions of commonly used passwords. We will use one of these lists already installed on Kali in our attempt to crack the remaining passwords in our file.

**# cd /usr/share/wordlists**

**# ls -l**

**# cp rockyou.txt ~/rockyou1.txt**

**# cd ~**

**# wc -l rockyou1.txt**

* How many lines (i.e. passwords) are in this list?

14344392

3.4: Try to crack the remaining passwords using the large wordlist above.

**$ john –crack-status –-max-run-time=180 --wordlist=./rockyou1.txt   
--format=sha512crypt passwd2**

Your command should be similar to the image.



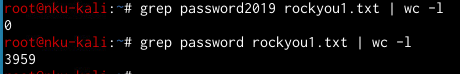
* How many additional passwords were you able to find in the 3 minutes allocated by the max-run-time option?
* Which new password were you able to crack in this mode? Include the output of the following command in the box below

**# john --show passwd2**

I struggled with this question but I think I was able to find 2 more, and able to crack 1 new password in this mode.

## 4: Password Cracking with Permutation Rules

We had to use a large word list and a considerable amount of processor time to crack one additional password. Even if we let the command above run for over 15 minutes, John would not find the remaining password in you passwd2 file because it is not in the rockyou1.txt word list as you can determine with a quick grep command.



It would be faster to find such passwords by using John's permutation rules, which can append or prepend numbers or symbols to words in the list or replace characters in words. The permutation rules also have the advantage of being able to find all such appends and substitutions, not just the changes that exist in the original word list.

The word list we are using has a wide variety of combinations of passwords followed by four digits, but it is missing the one that we need.



In this section, we will use the KoreLogic John the Ripper ruleset from the “Crack me if you can” password cracking contest held at DefCon 2010. A description of the rules can be found at <http://contest-2010.korelogic.com/rules.html>.

Download the ruleset with the following command, then create a very short wordlist containing only one word, password.

**# wget http://contest-2010.korelogic.com/rules.txt**

**# echo “password” > word.lst**

We will use this new file as John's configuration file via the **--config** option. To use a specific ruleset from the configuration, use the **--rules** option with the name of the ruleset.

4.1: To see how John produces new candidate passwords from its permutation rules, we will use the **--stdout** option which prints the words produced by the rules to STDOUT. We use I/O redirection to save the words to a file.

Use this ruleset to append 4 numbers to each word.

**# john --wordlist=word.lst --config=./rules.txt   
--rules=KoreLogicRulesAppend4Num --stdout > passwords4num.lst**



Next, issue the following commands and answer the questions based on the output returned.

**$ wc -l passwords4num.lst**

**$ head passwords4num.lst**

**$ tail passwords4num.lst**

* How many words are produced?
* What is John doing in addition to appending 4 numbers to increase the number of words from the expected 10,000?
  + Use the output of the head and tail commands to determine the answer to this question.

**20,000 words are produced. John is adding the appended 10000 twice, making 20000 words.**

4.2: While the permutation ruleset above produces every variation of password with a four-digit number appended, users do not pick their passwords randomly. We know that people are more likely to choose numbers that have meaning for them. In particular, people are more likely to choose years. The Kore Logic rules include a ruleset designed to do just that, which we will use in this question.

Issue the following commands:

**# john --wordlist=word.lst --config=./rules.txt   
--rules=KoreLogicRulesAppendYears --stdout > passwordsyears.lst**



**# wc -l passwordsyears.lst**

**# head -1 passwordsyears.lst**

**# tail -1 passwordsyears.lst**

* How many passwords are produced with the year ruleset?
* What is the earliest year produced?
* What is the latest year?

240 passwords are produced. 1900 is the earliest year produced. 2019 was the latest year

4.3: Crack the last password in the passwd2 file using the word list created from the KoreLogicRulesAppendYears ruleset.

* How long did it take to crack this password? Use the wall clock (real) time output from the **time** command.

**$ time john --crack-status --format=sha512crypt --wordlist=./passwordsyears.lst --config=./rules.txt passwd2**

* Verify John found all the passwords with the command below and provide the output.

**# john –-show passwd2**

400 seconds

## Submission

Upload a completed copy of this document to Canvas by the due date