Justin Strelka

Password\_cracking

7/7/20

**Cracking Weak Passwords with Hashcat**

**Table of Contents:**

Abstract…………………………………1

Motivation…………………………….1

Introduction…………………………..1

Method/Measurement………….2

Results…………………………………..5

Conclusion……………………………..7

Bibliography…………………………..8

Reflection………………………………8

**Abstract:**

This paper begins with an introduction to what hashes are and how we can use hashcat to crack hashes. There is then a brief outline of how to set up our environment on a Kali Linux VM through VirtualBox. Following will be a demo that will crack a variation of hashes. The paper concludes with a description of why it is important to crack our hashes and how we can make our hashes more secure against cracking.

**Motivation:**

Upon reading this paper you will be able to describe what a hash is and how they are used in our systems to store sensitive information. You will be able to set up your own environment for running hashcat to crack hashes. You will be able to run hashcat on your own system and understand the options provided by the hashcat utility. Lastly, you will be able to understand how passwords can be hardened against cracking and suggest rules to do so.

**Introduction:**

Hashes are used in most modern Operating Systems to store sensitive data such as passwords and they can be cracked using tools such as hashcat. We must understand that easy to crack passwords are a security vulnerability and should be protected against within our systems. We can deploy hashcat on a variety of systems and target the utility at a set file that we know contains hashes we want to crack. We use the term crack because hashes are encrypted pieces of data stored on a system and utilities such as hashcat are designed to run various attempts to try and crack the encryption and display the data in plain text. For the purposes of this paper we will be targeting passwords on a Kali Linux system with hashes stored in the /etc/shadow directory.

**Method/Measurement:**

Before we begin attempting to crack hashes in the /etc/shadow directory on our Kali Linux system we must first ensure that our distribution of Kali Linux has hashcat installed. We can run the following command to ensure hashcat is indeed installed on our system: hashcat –version. The return of the command should display like that in **Figure 1.** You may have a different version number though for the purposes of this paper that is not an issue.

A picture containing meter, clock

Description automatically generated

Figure 1: Proof of hashcat being installed on your system.

Once you have verified that hashcat is installed on your system we can then move forward with understanding how to use the hashcat utility. Hashcat allows us to configure our commands in a variety of ways since different systems and applications use different hashing algorithms. Hashcat can also be ran through your systems GPU if it is supported in the drivers. However, for the purposes of this demonstration we will be running hashcat directly through our GPU as it is easier to configure and VirtualBox does not support use of the GPU through on Windows hosts. [1] If we run the “man hashcat” command as displayed in **Figure 2** we will receive a user manual outlining the various uses and options associated with the hashcat utility.

A screenshot of a cell phone

Description automatically generated

Figure 2: Hashcat manual displayed by “man hashcat” command.

From the hashcat manual page we can see that the general formatting for a hashcat command is as follows: hashcat [options] hashfile [mask|wordfiles|directories]. The hashcat portion of the command obviously tells the systems we wish to execute the hashcat binary. In the options portion of the command we can specify a variety of different options that will alter the way hashcat behaves. Some of the standard options used to define a general hashcat cracking session include the type of device to be used, such as the GPU or CPU by setting the -d flag. We can also choose the attack type we wish to deploy with the -a flag and our options allow us to alter the type of passwords attempted to be cracked. We could link a pre-built word list of common password names here or as we will be doing for the purposes of this demonstration a brute-force attack. For brute-force attacks we will need to specify the characters allotted in each namespace and hashcat will exhaustively search all possible variations of a given namespace. We will also need to specify the hashing algorithm to use for cracking hashes in the targeted file, to do so we can search the man page for the algorithm we would like to use and set the -m flag to the corresponding algorithm in the table. Next the hashfile portion of the command allows us to specify which file that contains hashes to target. Default Debian Linux settings store password hashes in the /etc/shadow directory so this will be specified in our command when deploying hashcat. Lastly we must define the mask or namespace for a brute-force attack or provide wordfiles and dictionaries to various other types of attacks for them to pull words from.

A screenshot of a cell phone

Description automatically generatedNow that we understand the basic settings in our hashcat commands we need to create some users and passwords in our system. We will begin with creating users that have only 4 letter passwords since exhaustively searching each possible combination will require extensive computing time through our VM’s CPU. As shown in **Figure 3** we can add users to our Linux system by obtaining root access and running the following commands: sudo su -; adduser crack1. The system will then prompt you us provide a password for the new user (which will be 4 characters long) and provide any basic information desired about the user. The password we will use for our crack1 user will be “boog” which contains only letters. We will repeat the steps shown in **Figure 3** until we have 4 users created. We will label each user crack1,2,3,4 until all users have been created. Be sure to alter the use of characters numbers and symbols in the four letter passwords created. The following passwords will be used in this demonstration and apply to the user list in ascending order respectively: boog, B@@6, 80@G, bO06. Be sure to pay attention to which characters are o/O’s and which are zeros (0) and sixes(6).

Figure 3:display of crating users in Kali.

Once you have all of the users created in the system we can verify that the users and their hashes exist by viewing the shadow file. To view the shadow file we must still be the root user (sudo su -) and run the command cat /etc/shadow the contents of the shadow file should be displayed to the screen. Scroll to the bottom of the file and search for users crack1,2,3,4 appended to the end of the file. Once you have ensured the file does indeed have all users created we can begin creating our hashcat command to attack the shadow hash file.

The first part of the command requires us to specify the binary we will be executing which is “hashcat”. Next we need to specify the attack type we wish to conduct which is specified by the “-a” flag and we will set our flag to 3. Setting the attack flag to three will set the attack to brute-force mode. Next we must specify the hashing algorithm to use, we are conducting this attack against a Kali Linux shadow file which uses the SHA-512(UNIX) algorithm. The -m flag corresponds to the algorithm ID we wish to attack with, we will be setting our flag to “1800” which corresponds to the Kali hashing algorithm stated previously. We can now set the output file to where our cracked hashes will be written to and stored for our viewing purposes. To set the output file we use the -o flag and then set the value to “cracked.txt”. The last option we will set is the “-O” flag which will optimize the kernel to crack passwords and set a maximum password length.

Once all the flags have been set to our preference we will then specify the hash file that contains the hashes we wish to crack. For us we will be attempting to crack the “/etc/shadow” file which is the default location user passwords are encrypted and stored in Kali. Following the hash file we must specify the mask for the namespace of the passwords we want to crack. We know our passwords are four characters long so we will need a mask that is four characters long to specify the namespace. There are some pre-defined character sets in hashcat which can be observed at the following link: <https://hashcat.net/wiki/doku.php?id=mask_attack>. Since we have letters, numbers, and symbols in our passwords we will set our mask to check the namespace accordingly. For our first round of cracking we will set our mask to “?a?a?a?a”. The final option we will include is the –force flag which bypasses all warnings and will bypass the native devices on your system and will deploy hashcat using the VM’s resources allocated by VirtualBox. Now when we piece this all together our command we will place into the command line is: “hashcat -a 3 -m 1800 -o cracked.txt -O /etc/shadow '?l?l?l?l' --force”.

When running this command the output in the “cracked.txt” file will be placed in the current working directory. I have created a specific directory for this paper that I have navigated to so the output will go directly to a directory of my choosing. When we run our command hashcat will be deployed and will run through optimizing your kernel for cracking. Once hashcat gets your environment established it will begin trying to crack the hashes in the target hash file that match our specified mask. The console will then prompt you with options for related to the current cracking session.

As you can see in **Figure 4** the options we are presented from the hashcat utility:

“[s]tatus [p]ause [b]ypass [c]heckpoint [q]uit =>”. If we choose status we will receive the output shown in **Figure 4** which is an overview of the progress for the current hashcat session. The status feature provides us with the configuration we defined in our command along with other useful information. We can see how long the current session has been running and an estimated time left to completion. We can see how many cases have been tested and how many there are left to completion.

A screenshot of a cell phone screen with text

Description automatically generated

Figure 4: Hashcat running output and options.

The pause option allows us to pause the session in case we have to use our systems resources to do some other computing. Once we have completed we can restart the process where it left off. This feature is very important as the longer and more complex a password gets the more cases will need to be tested taking exponentially more time to crack a password. The bypass option is useful when we plan to test against more than a single wordlist. If we want to skip a particular wordlist to speed up our session we can choose the bypass option and hashcat will begin testing the next wordlist provided. The checkpoint option allows us to update the checkpoint file with where we were in our session so we can stop and restart the session similar to the pause option. Lastly quit is used to exit and cancel a session with no option of restarting where we left off unless we had a previous restore point.

**Results:**

A screenshot of a computer

Description automatically generatedOnce the hashcat session is completed there will be a file created named “cracked.txt” that we specified in our initial command. As shown in **Figure 5** we can see the completion of the hashcat session and the contents of the “cracked.txt” file with our first password cracked, “boog”. To display the contents of the output file (cracked.txt) on your own machine run “cat cracked.txt” which will print the contents of the file to the console. We can also see that to crack a four character password consisting of only lower case letters took only 56 minutes without using the GPU.

Figure 5: Cracked first password “boog” all letters for crack1 user.

We will now begin trying to crack the password hash for the crack2 user. To do so we must alter the mask we are using to match the format of the password. We know the password for the crack2 user is B@@6 which consists of an uppercase letter followed by two symbols and ending with a digit. The hashcat built in charsets will cover all the cases necessary. For an upper case letter we will use the “?u” charset which covers all uppercase letters. Then we will follow with “?s?s” which will represent two symbols and end with a “?d” to represent a digit. The entire mask put together will be as follows, '?u?s?s?d'. We can now deploy hashcat against the same file just with the mask swapped out for our new mask. As you can see in **Figure 6** the hashcat session took a total of 23 minutes only to crack the password for the crack2 user.

A screen shot of a computer

Description automatically generated

Figure 6: Cracked password “B@@6” for crack2 user.

A picture containing sitting, monitor, table

Description automatically generatedNext we will begin cracking the password for the crack3 user. The password for the crack3 user is “80@G” which consists of a digit followed by another digit. After the leading two digits there is a symbol and the password ends with a capital letter. Using the predefined hashcat charsets we can create the following mask to try and crack the password for the crack3 user, '?d?d?s?u'. As you can see in **Figure 7** the hashcat session lasted only 6 minutes.

Figure 7: Output of hashcat for the cracked password “80@G” of the crack3 user.

The last password we are going to crack belongs to the crack4 user. The password for the crack4 user is “bO06” which consists of a letter followed by an uppercase letter and ends with two digits. The mask we will use for this session is as follows, '?l?u?d?d'. As you can see in **Figure 8** the hashcat session took only 2 minutes to crack.

A screenshot of a cell phone

Description automatically generated

Figure 8: Cracked password “bO06” for the crack4 user.

For the purposes of this paper we knew both the length and the format each password was in that we wanted to crack. However, in a real-world scenario this would not be the case. We would not be able to crack these passwords as quickly as we were able to in this scenario because we would have to test formats and lengths that require more cases to test. To show this we will run the same command against the same file however this time we will open our mask up to any combination of letters upper and lower case, digits, and symbols. The mask we will use to do so will again take advantage of the built in hashcat charset “?a” duplicated four times. As you can see in **Figure 9** the estimated time to complete the session has increased significantly to roughly 3.5 days to crack just a for character password. Again here we knew the length of the password we wanted to crack which not knowing the length of the passwords we want to crack would increase our possibilities as well.

A screenshot of a computer

Description automatically generated

Figure 9: Estimated time to crack a four character password using multiple charsets for each character.

**Conclusion:**

It is clear that four letter passwords do not provide enough security and we should consider making passwords long enough that cracking them is just to inefficient that it would deter most malicious users. By have passwords of varying length and the inclusion of symbols, capital letters, and digits increases the security of our passwords exponentially. Moving forward we must understand the dangers having short passwords that do not contain a mixture of charsets. As cybersecurity specialists we must enforce password rules that prevent the use of utilities like hashcat from effectively cracking our hashed passwords and escalating their foothold in a system.

**Bibliography**

[1] <https://forums.virtualbox.org/viewtopic.php?f=6&t=92855>

**Reflection:**

Prior to writing this paper I was unaware of how quickly a malicious user can crack passwords. I did not understand how quantum computing posed a threat towards our hashing algorithms. I still do not fully understand the actual mathematics of why quantum computing poses a threat. However, after being able to crack a four letter password without restrictions in only 3.5 days with a VM that had access to only 2 processors running with a 70% max load cap I realized that theses algorithms are not as secure as I expected. I feel that had I been able to passthrough my GPU with VirtualBox that I would be able to crack a 6-character password in 3.5 days. This is to close for comfort as most password restrictions require the user to have an 8 or more-character password. Moving forward I will be demonstrating this information for my family to make them aware of the implications that not taking their cybersecurity seriously can impose.