****

**Secure Communications 2018/19**

**Lab06  
Salted Hashes**

**James Finglas, B00094138**

Contents

[Introduction: 3](#_Toc2011461)

[Cracking the hashes: 4](#_Toc2011462)

[Conclusion: 8](#_Toc2011463)

# Introduction:

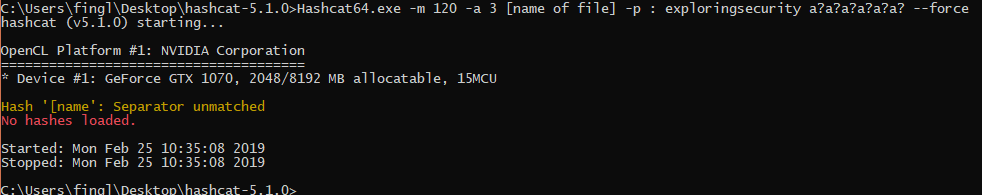
For this Lab we are tasked with assisted a police investigation. We are tasked with cracking 6 salted hashes to aid the police. To facilitate this, based off the polices initial investigation and attempts to crack the passwords we are provided with some simple information and a table as seen below.

* The password policy file was changed in May 2010, passwords created after this date are alphanumeric 5-7 characters in length. Passwords created before these dates are believed to have consisted of only digits and 5-7 characters in length.
* It’s believed that all passwords have used the same salt, and that the value is somewhere in our data.
* The database dump was from MySQL database  The site’s domain name is [www.exploringsecurity.com](http://www.exploringsecurity.com)
* Some of the captured JavaScript code from the site, reveals the salt format as CommonHash($salt,$pass)

  
  
From this information I was able to discern some facts and make some educated guesses to begin my investigation. With some passwords being only numeric it is reasonable to presume that these should be crack-able with a brute force attack. The passwords that are Alpha numeric may require a wordlist, however we will first begin with a brute force attack.  
  
All passwords contain the same salt, so if we examine the information provide we must search for a string that could be used repeatedly. The only piece of information that fits this requirement is the sites domain name. Therefore, we will use this as it is presented as the salt.  
  
The captured java code reveals the salt type. There are two types of encryption that match this type of salt/hash type. MD5 and sha1. MD5 is not very secure and reasonably easy to crack. So, I will assume it is sha1 to start off. If necessary I will return and try MD5.

# Cracking the hashes:

I began my attempts to crack the first hash as I have done before, entering the salt directly in the command and only having the hash in the has file. In the past this proved successful. Where ‘John the ripper has always be able to discern the hash and salt for itself, Hashcat has always required my intervention to discern this in the past. The syntax I used was:

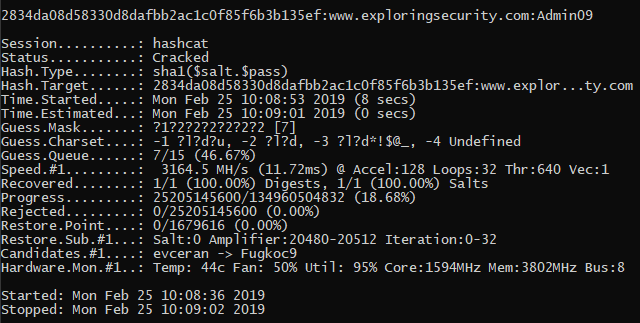


*Here we see the separator error*

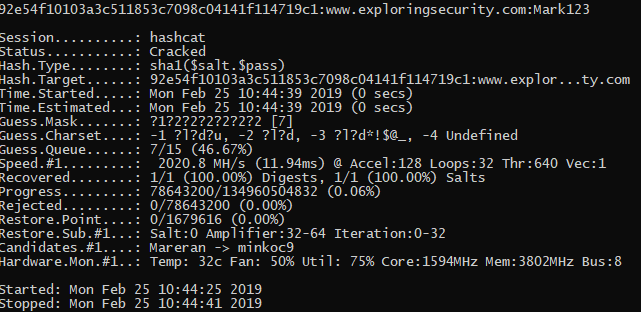
This syntax returned an error regarding a separator mismatch. I could not work out this error. However, during my investigation, I came across forum post where people suggested adding the salt in the file. With no other options I tried this. The format of the file to be cracked was now as follows: 2834da08d58330d8dafbb2ac1c0f85f6b3b135ef:www.exploringsecurity.com.

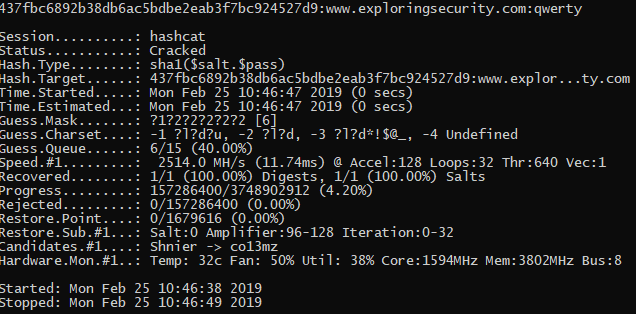
The syntax also needed to be adjusted. The new syntax was: hashcat64.exe -m 120 -a 3 hash1.txt.

This proved successful and I cracked the first hash in under 60 seconds having passed through 6 guess ques and getting a hit on the 7th as seen below with the password being: Admin09.

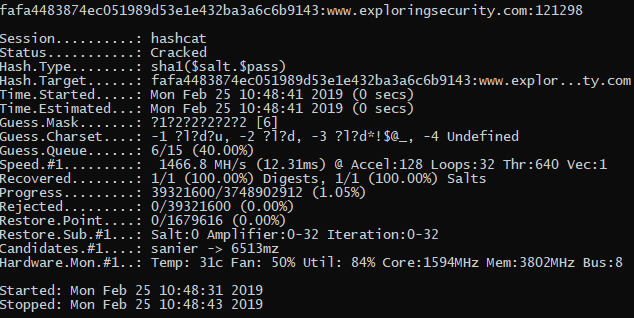


We then proceed to hash number 2 and applied the same method, which again proved successful. The password this time was: Mark123.

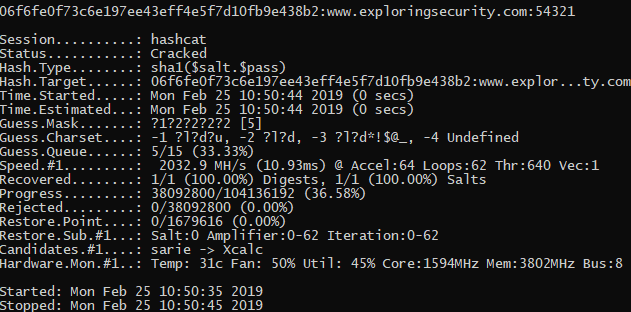
  
  
Next up, hash number 3. And we have another success. The password this time was: qwerty.

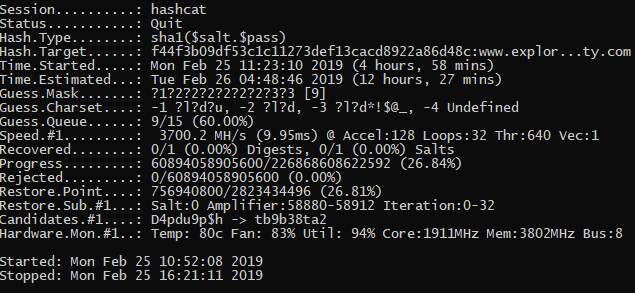


I now moved on to has number 4. This again proved successful. This time on the 6th guess mask with the password being: 121298

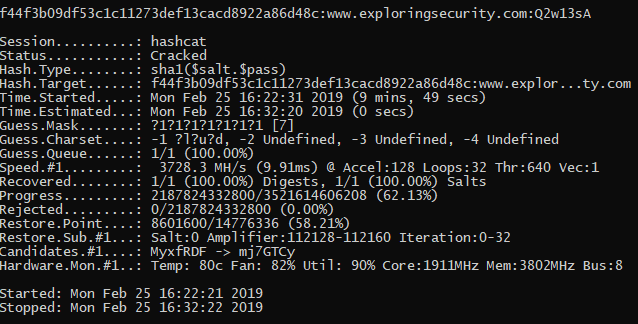


And on to hash number 5. Another success, this time on the 5th guess mask, with the password being: 54321.



Finally, we move on to the last hash. This has proved more difficult. Passing through 7 guess masks to the 8th. This concerned me as I felt like I was getting into areas where passwords could take days to crack. However I left it running for a few hours to see if anything came of it.  
  


Above we can see that the attack has been running for almost 5 hours. I was reasonably sure that something was wrong at this point. So I cancelled the attack and decide to manually force it to re run through the 7th guess mask again before re-evaluating my thought process. This proved to be the correct approach. I cannot say why, but the automatic attack was failing to fully attack the password in the 7th guess mark range. A manual attack with the syntax: C:\Users\fingl\Desktop\hashcat-5.1.0>hashcat64.exe -m 120 -a 3 -1 ?l?u?d hash6.txt ?1?1?1?1?1?1?1, adding a character set and defining the character set as specifically 14 characters long, placing it in the 7th guess mark range forced Hashcat to properly attack the password in this range and sure enough this cracked the password, although it too 10 mins rather than 1 this time. With the password being: Q2w13a



# Conclusion:

This was an enjoyable lab. I learned a few useful things about Hashcat. Where John the Ripper is a useful and powerful tool particularly with its ability to automatically examine formats, Hashcat is more complex. Once the user has correctly determined the format and entered the right information is extremely powerful and more diverse in its ability to attack passwords.  
  
Having made the incorrection assumption based off my previous use of Hashcat that it cannot crack a hash file that contains the salt, I have learned that I must be more careful in the syntax that I use and the way I enter the salt. I must be very careful when examining the format of the password and salt combination.  
  
I am as yet unsure why the given format is $pass.$salt and yet the salt is actually entered after the password in the hash file, however in this case it is not important for me to understand why, only that it is entered in that order.