Hashcat Guide

Hashcat is a password cracking tool used to crack hashes. It is great for brute forcing! And generating hashes with patterns (masks).

It supports many hashing algorithms such as
- Microsoft LM hashes
- Microsoft NTLM hashes
- MD4
- MD5
-SHA-family
-Unix Crypt formats
-MySQL
-Cisco PIX
-more

and works by taking in user input such as wordlists or masks, generating the hash and looking for a match with the provided hash. It is the self-proclaimed world's fastest CPU-based password recovery tool and is free software!

Hashcat versions are available for Linux, OS X, and Windows and can come in CPU-based or GPU-based variants.

You can download hashcat here.

https://hashcat.net/hashcat/

How to use hashcat

Typing hashcat -h we can see all the options available in hashcat.

The basic use of hashcat is as follows:

> hashcat -m 0 -a 0 hashfile.txt wordlist.txt

This will do a straight md5 hash of every word in your wordlist.txt and look for a match with the hash in hashfile.txt

Types of modes

```
  # | Mode
 ===+======
  0 | Straight
  1 | Combination
  3 | Brute-force
  6 | Hybrid Wordlist + Mask
  7 | Hybrid Mask + Wordlist
```

Straight

The dictionary attack, or "straight mode," is a very simple attack mode. It is also known as a "Wordlist attack".

All that is needed is to read line by line from a textfile (aka "dictionary" or "wordlist") and try each line as a hash candidate.

The command for the Combinator Attack in hashcat is -a 0

Combination

Words from your wordlist are combined with each other to create a more comprehensive wordlist. Eg. Each word of a dictionary is appended to each word in a dictionary.

Example:
If our wordlist contains the words:
pass
12345
omg
Test

Hashcat creates the following password candidates.
passpass
pass12345
passomg
passTest
12345pass
1234512345
12345omg
12345Test
omgpass
omg12345
omgomg
omgTest
Testpass
Test12345
Testomg
TestTest

The command for the Combinator Attack in hashcat is -a 1
You need to specify exactly 2 dictionaries in you command line: e.g.

./hashcat64.bin -m 0 -a 1 hash.txt dict1.txt dict2.txt

If you wish to add rules to either the left or right dictionary or both at once then you can use the -j or -k commands.

-j, --rule-left=RULE Single rule applied to each word on the left dictionary

-k, --rule-right=RULE Single rule applied to each word on the right dictionary

Example.
Dictionary 1
yellow
green

black
Blue

Dictionary 2
car
Bike

Commands
-j '$-'
-k '$!'

The output would be…
Yellow-car!
Green-car!
Black-car!
Blue-car!
Yellow-bike!
Green-bike!
black-bike!
Blue-bike!

Brute-Force Attack

Tries all combinations from a given Keyspace. It is the easiest of all the attacks. In Brute-Force we specify a Charset and a password length range. The total number of passwords to try is Number of Chars in Charset ^ Length.

The command for the Combinator Attack in hashcat is -a 3

Built-in charsets

- ?l = abcdefghijklmnopqrstuvwxyz
- ?u = ABCDEFGHIJKLMNOPQRSTUVWXYZ
- ?d = 0123456789
- ?s = «space»!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
- ?a = ?l?u?d?s
- ?b = 0x00 - 0xff

Mask Attack

Mask attacks try all combinations from a given keyspace just like in Brute-Force attack, but more specific.

For each position of the generated password candidates we need to configure a placeholder. If a password we want to crack has the length 8, our mask must consist of 8 placeholders.

- A mask is a simple **string** that configures the keyspace of the password candidate engine using placeholders.
- A placeholder can be either a custom charset variable, a built-in charset variable or a static letter.
- A variable is indicated by the ? letter followed by one of the built-in charset (l, u, d, s, a) or one of the custom charset variable names (1, 2, 3, 4).
- A static letter is not indicated by a letter. An exception is if we want the static letter ? itself, which must be written as ??.

Built-in charsets

- ?l = abcdefghijklmnopqrstuvwxyz
- ?u = ABCDEFGHIJKLMNOPQRSTUVWXYZ
- ?d = 0123456789
- ?s = «space»!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
- ?a = ?l?u?d?s
- ?b = 0x00 - 0xff

Cracking Windows Hashes

So you've got Domain Admin and you've dumped the domain hashes. Well done! Now we need to crack the hashes.

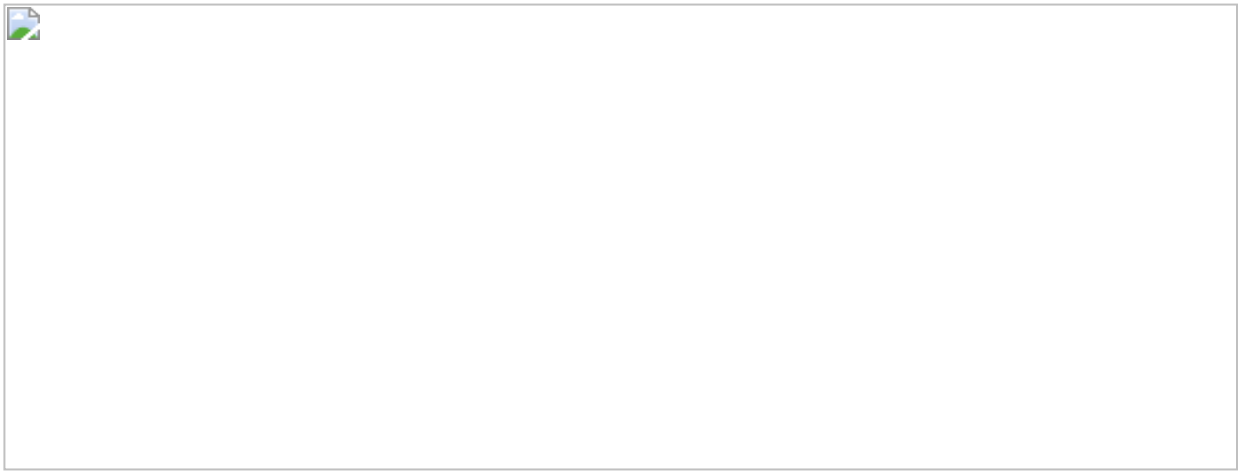
Background
Before we go ahead and start cracking, let's quickly go over how Windows stores it's password hashes. There are two types of hashes used by

Windows:

LM - Stored as a 32bit hexadecimal string. LM hashes contain multiple vulnerabilities and are very easy to crack as a result. If you're unsure why they are vulnerable, go google it. (smile)

NTLM - Stored as a 32bit hexadecimal string. Computers have gotten to the stage where even NTLM hashes are relatively easy to crack, especially if the password is less than 8 characters.

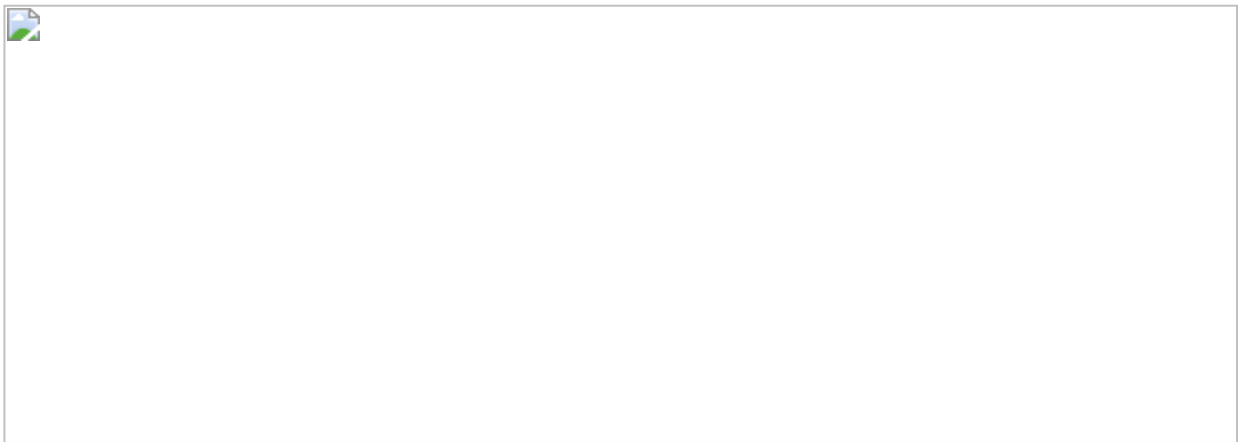
When you dump the hashes from a domain controller, you will probably see them in the format below (hashes changed from their original form):



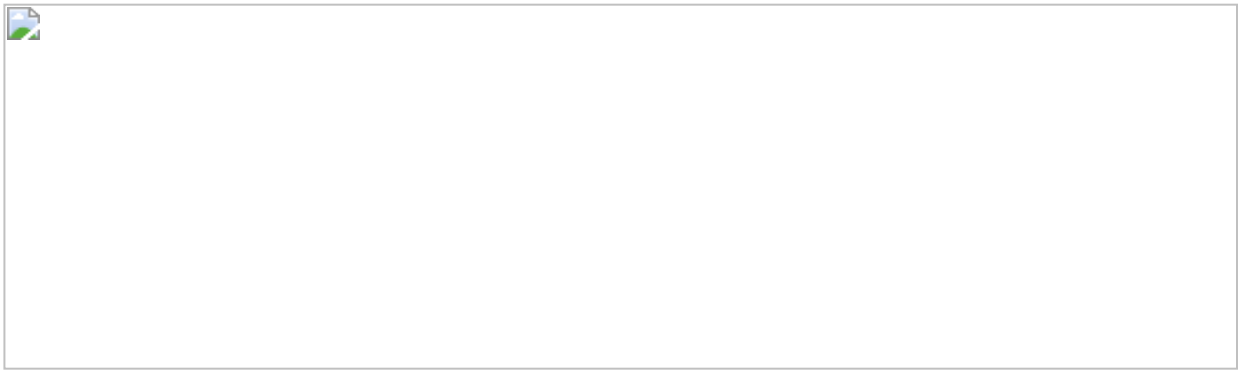
Some of the LM hashes are "aad3b435b51404eeaad3b435b51404ee" which denotes an empty password. Note that this probably does not mean that the password is actually empty. Rather, it means that an LM hash is not stored for that user.

Also, you might see some usernames that end in '$'. These are computer accounts. Computer accounts use randomly generated passwords (32 characters) that are changed automatically every 30 days - cracking them is highly unlikely. It's safe to remove them, which we will do in the next section.

Preparation
Unfortunately, hashcat does not recognise the format shown above, so we'll have to use some commandline-fu to get it into the correct format. I'm going to assume you output your dump into a file called 'hashdump.txt'.



Remove all accounts with a '$'.
grep -v "\\$" hashdump.txt > hashdump-tmp.txt



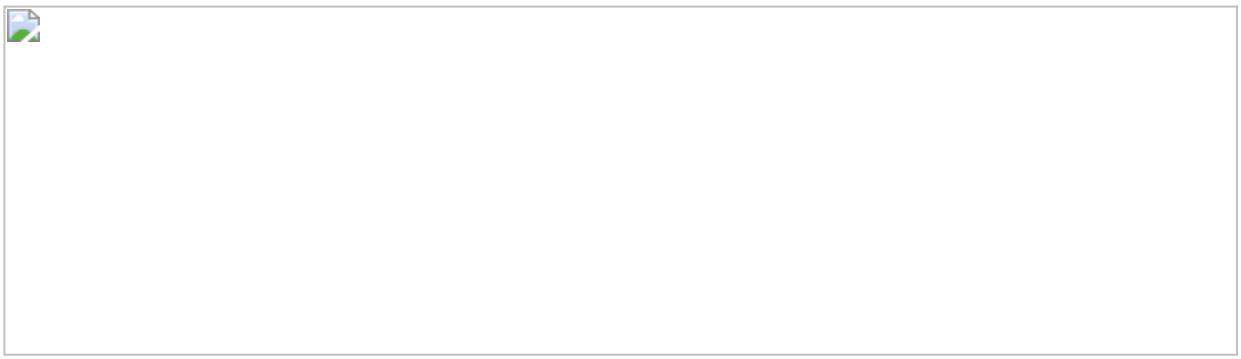
You can see that the service accounts have been removed

Now we're going to split the file up into two files. One for LM and one for NTLM. We won't be needing LM hashes that look like 'aad3b435b51404eeaad3b435b51404ee' so we'll get rid of those also.
cat hashdump-tmp.txt | cut -d: -f1,3 | grep -v "aad3b435b51404eeaad3b435b51404ee" > lm.txt
cat hashdump-tmp.txt | cut -d: -f1,4 > ntlm.txt

We now have our two file that we will work from called "lm.txt" and "ntlm.txt". It is safe to remove our temporary hashdump file.
rm hashdump-tmp.txt

NOTE: It is good practice to keep the original hashdump.txt file just in case we need it later.

Let's now create a custom wordlist that is specific for this client. We are all professionals so this should be done every time!
cewl -m 8 -w custom-wordlist.txt http://www.example.org

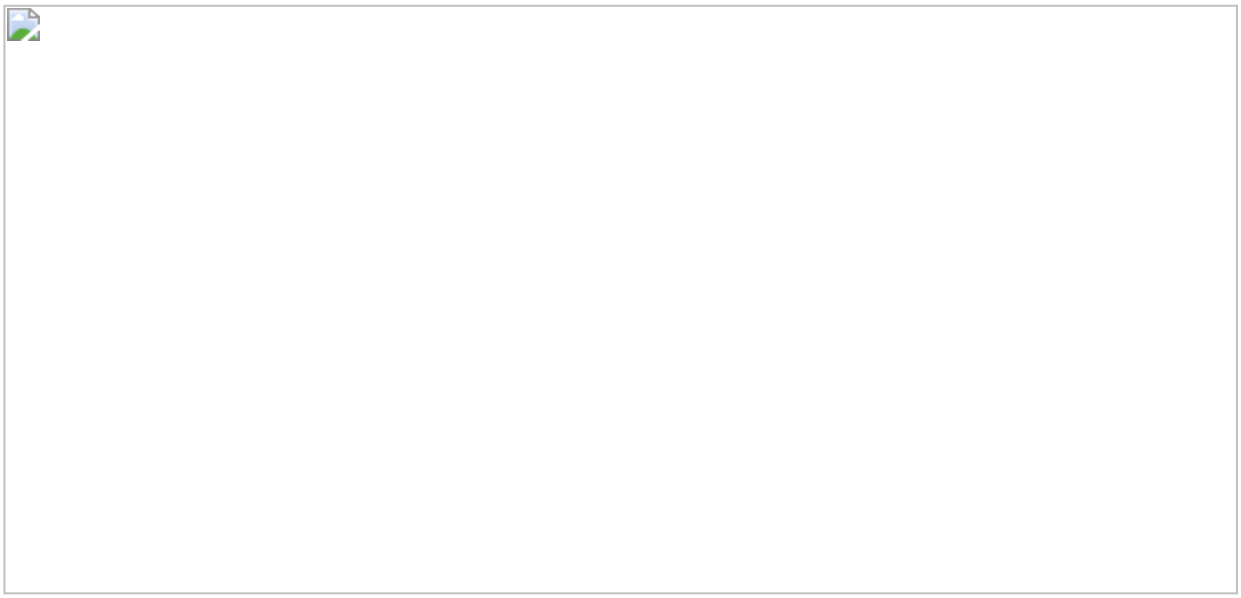


We will use this when we get to the NTLM cracking part.

Good start, but the list is quite short. Let's use john's rule engine to mangle the words and give us a better wordlist.
john --wordlist=custom-wordlist.txt --rules --stdout > tmp.txt; mv tmp.txt custom-wordlist.txt

That's better! Now we we are ready to start cracking some hashes!

LM Hashes
This part of the process is very easy since we are simply doing a brute force attack against all 7 character (or less) passwords. If you have no LM hashes to crack, you can skip this section and more on to NTLM cracking.
We are going to use a mask based brute force attack using the following command.
sudo hashcat -m 3000 -a 3 --username -i -o lm-out.txt lm.txt ?a?a?a?a?a?a?a
The -m 3000 says we are targeting LM hashes.
The -a 3 means we are performing a mask brute force attack attack.
The --username tell hashcat we have formatted our input file to contain usernames at the start.
The -i means we will increment through the character lengths, starting at length 1 and adding 1 extra character until we reach what we have specified (7 characters in this case).
The -o lm-out.txt is where we will output our results.
The lm.txt is our input file.
The ?a?a?a?a?a?a?a is the mask we are using. ?a means all upper, lower, numeric and special characters. There are other mask groups, so check out the help menu for those.



This will run for about 15 minutes. Which is pretty damn FAST!

Now that we have our output file, let's use stitch.py and capstar.py to create a wordlist with all of our currently cracked passwords.
stitch -lm lm.txt lm-out.txt | cut -d: -f3 | grep -v "*******" | capstar - > lm-wordlist.txt

We will use this in the next section to derive password casing.
That's all for LM hashes! You can see why storing LM hashes is such a bad idea. We effectively brute forced the entire LM password space in less than 15 minutes!

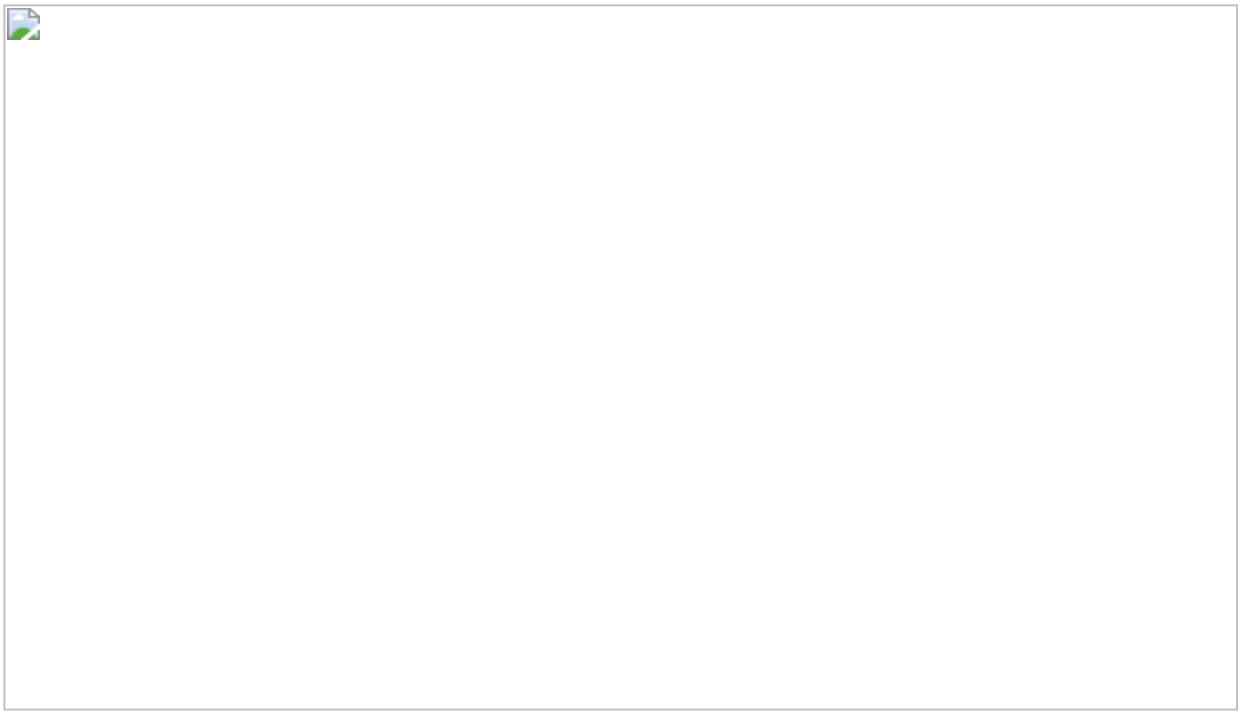
NTLM Cracking

Now we get to the good stuff! All modern Windows Domain Controllers store NTLM hashes for domain accounts. As such, if we get a hold of the hashdump, we will inevitably end up cracking NTLM.

Something to keep in mind is password policy. There is no point trying a mask with only lowercase characters if the password policy enforces complexity. Similarly, if the minimum password length is 8 characters, don't even bother with 7 character passwords.

Okay! Let's get started!

We will start by getting out the very obvious passwords. Let's run hashcat against some of our wordlists.
hashcat -m 1000 -a 0 --username -o ntlm-out.txt ntlm.txt /usr/share/wordlists/crackstation.txt /usr/share/wordlists/rockyou.txt custom-wordlist.txt lm-wordlist.txt



Already you have probably found a good number of passwords! But we are far from done.

To cover short passwords, we will also perform a bruteforce attack on all passwords 7 character or less.
hashcat -m 1000 -a 3 --username -i -o ntlm-out.txt ntlm.txt ?a?a?a?a?a?a?a

This will take approximately 45 minutes. So go grab a coffee and relax...

And..... We're back! As a baseline, we could end here. We have most likely gotten a fair number of passwords (about 50%). But as l33t haxxorz that's not really good enough. We we'll get into some more advanced attacks.

Let's combing our two previous attacks to perform a hybrid attack! A hybrid attack involves using wordlists in combination with a mask. We will try a few different masks with the crackstation-human-only.txt wordlist.
hashcat -m 1000 -a 6 --username -i -o ntlm-out.txt ntlm.txt /usr/share/wordlists/crackstation-human-only.txt ?d?d?d?d

Here are some other masks to try:

?l?l?l?l
?s?s?s?s
?d?d?s?s

You can try other masks if you like.

You may have noticed that a lot of passwords follow the mask "?u?a?a?a?a?d?d?s". We will abuse this fact and start running a some common masks.
hashcat -m 1000 -a 3 --username -o ntlm-out.txt ntlm.txt ?u?a?a?a?a?d?d?s

Here are some other masks to try:
?l?l?l?l?l?l?l?l
?l?l?l?l?l?l?l?l?l
?u?l?l?l?l?l?d?d
?u?l?l?l?l?l?d?d?s
?u?l?l?l?l?l?d?d?d
?u?l?l?l?l?d?d?d?d
?l?l?l?l?l?l?d?d
?l?l?l?l?l?l?d?d?s
?l?l?l?l?l?l?d?d?d
?l?l?l?l?l?d?d?d?d

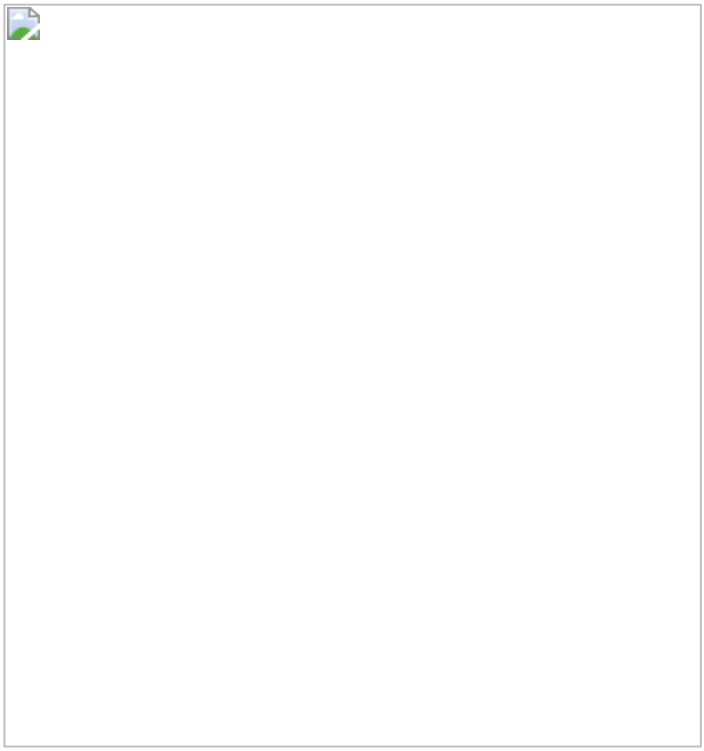
There are many many more, but I will leave the rest to your imagination. (smile) Remember we have already brute forced all passwords 7 characters or less, so there is no need to try them again.

We'll take a step back and go back to using wordlists, but now let's add some rules on top! In hashcat, we can choose a rules file that will transform each word in the wordlist. For example, there is a rule to "l33tify" each word; Transforms password -> p@55w0rd.
hashcat -m 1000 -a 0 --username -r /usr/share/hashcat/cudaHashcat/rules/rockyou-30000.rule -o ntlm-out.txt ntlm.txt /usr/share/wordlists/crackstation-human-only.txt

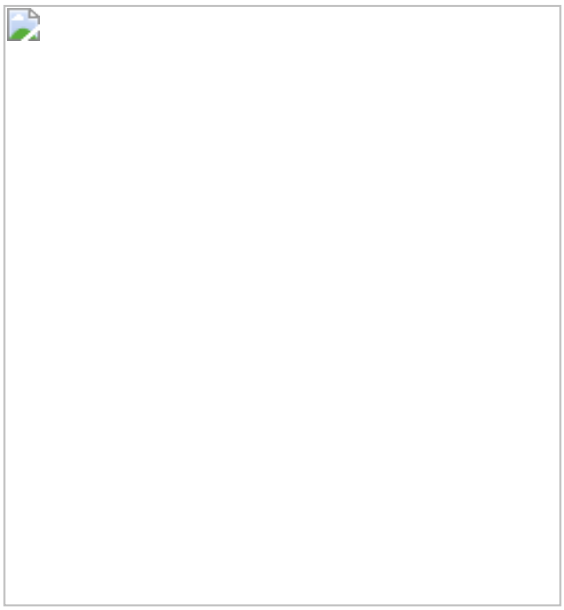
Here are some more rules you can try:
Ninja-leetspeak.rule
d3ad0ne.rule
Hob0Rules - (Reference: https://github.com/praetorian-inc/Hob0Rules)

There are many more rules you can try if you like.

This next part I call the password feedback loop. We are going to use existing results to make new a new custom wordlist and a new list of masks. The reason this can be successful is that organisations often have a password "culture". What I mean by that is employees may have been trained to select passwords of a certain mask or use specific words common to the organisation.
Let's use hcmasks.pl to create a list of most common masks that we have cracked so far.
cut -d: -f2 ntlm-out.txt| hcmasks



This format is no good for us, so we'll use our commandline-fu again to fix that. In the process we will also remove any masks that are less than 7 characters.
cut -d: -f2 ntlm-out.txt| hcmasks | tr -s ' ' | cut -d' ' -f3 | grep -e '[^\ ]\{16,\}' | head -n 20 > feedback-masks.txt



This gives us the top 20 most used masks for that organisation. We will use these masks shortly.

The second part of the password feedback loop is to simply create a new wordlist from the already cracked passwords.
cut -d: -f2 ntlm-out.txt > feedback-wordlist.txt

Time to use our feedback masks! The important part here is to keep an eye on the estimated time. It is pointless to run a mask that will take weeks to complete. I personally cancel any mask that takes longer than 5 minutes, but it's up to you.
for i in `cat feedback-masks.txt`;do hashcat -m 1000 -a 3 --username -o ntlm.out.txt ntlm.txt $i; done

For those that take too long, you can just press 'q' and it will move on to the next mask.

From here we can go back to Step 2 and start the process all over again using our new feedback-wordlist.txt.

Congratulations! You have cracked a bunch of hashes. What I have described above is a very structured approach. But remember, you can mix and match any or these steps. For example, you can do a combination of rules, masks and wordlists. You can choose different masks, different wordlists. Be imaginative and refine the process to make it work for you.