# Cracking Primer

This article provides guidance for performing basic cracking on a workstation.

*Unless you have experience doing so, or are looking for a challenge, it is recommended that you run the cracking software from your host, and **NOT** from within a VM. While your mileage may vary, better performance and compatibility with GPU-based cracking will be realized from the host..*

## Prerequisites:

You will need to have installed one (or both) of the following cracking applications. You may need to compile the applications for your platform. The process to compile the applications is outside of the scope of this document.

- Hashcat: https://hashcat.net/hashcat/ (See the top of the page)
  - https://github.com/hashcat/hashcat contains the latest dev release of the software
- John the Ripper (JtR):
  - MagnumRipper (Official repo for the Jumbo version which has lots of additional hash formats that can be cracked, collections of useful scripts, etc.): https://github.com/magnumripper/JohnTheRipper - This is the recommended version to use when an abundance of features is preferred.
  - Official Developer's Site: http://openwall.com/john/ - This is the recommended version to use when the Jumbo version is instable, or you only want to use a version with minimal modifications

## Other Useful Items:

**Word Lists - There are tons of repositories hosting word lists, a few are listed below. The specific filenames specified below each location are recommended word lists as they have proven to be effective.**

- https://download.g0tmi1k.com/wordlists/large/
  - crackstation.txt.gz
- https://github.com/danielmiessler/SecLists/tree/master/Passwords
  - rockyou.txt.tar.gz
  - phpbbb.txt
- Cracked passwords from the 2016 LinkedIn dump
- http://weakpass.com/lists
  - Lots of foreign language dictionaries
- https://github.com/berzerk0/Probable-Wordlists
  - Word lists sorted in order of most probable to crack a password (according to the research of the author)

## Determining Hash Types

- Many times, JtR can be run, with the name of the file containing the hashes as a parameter, to determine the type of hash.

Using JtR to Determine the Hash Type

```
# /JohnTheRipper/run/john netntlmv2.txt
Using default input encoding: UTF-8
Loaded 5 password hashes with 5 different salts (netntlmv2, NTLMv2 C/R
[MD4 HMAC-MD5 32/64])
<< SNIP >>
```

JtR determined that the hash type was netntlmv2.

- https://pypi.python.org/pypi/hashID is a free tool to identify hash types
- https://hashcat.net/wiki/doku.php?id=example_hashes - This list can be used to discover which hash code to use with the -m switch in hashcat
- Optionally, you can use the following script:

**hashtype.sh**

```
#!/bin/bash

test "$1" == "" && (/opt/temp/home/opencl/hashcat/hashcat64.bin --help 2>
/dev/null | sed '1,/Hash modes/d' | less) ||
(/opt/temp/home/opencl/hashcat/hashcat64.bin --help 2> /dev/null | sed
'1,/Hash modes/d' | grep -i "$1")
```

```
Usage:
```

./hashtype.sh by itself will list all the hash types and their associated hash code

***- or -***

./hashtype.sh with a greppable string (such as md5, sha, or netnt) will list only hash types that match the string input.

```
./hashtype.sh netnt
   5500 | NetNTLMv1                                        | Network
protocols
   5500 | NetNTLMv1 + ESS                                  | Network
protocols
   5600 | NetNTLMv2                                        | Network
protocols
```

You would use hash code 5600 with the -m switch for NetNTLMv2 hashes

## Get Cracking

## *John the Ripper (JtR)*

### Setting the Hash Type

JtR will attempt to auto-detect the hash type. This feature can be particularly useful when trying to identify a hash, but problematic when the detected hash type is wrong. Frequently this occurs with Windows NTLM and LM hash types.

The following output shows JtR incorrectly identifying the hash type as LM, but printing notifications that the hash type could be one of the alternatives listed.

```
# /JohnTheRipper/run/john some_set_of_ntlm_hashes.txt
Warning: detected hash type "LM", but the string is also recognized as
"dynamic=md5($p)"
Use the "--format=dynamic=md5($p)" option to force loading these as that type
instead
Warning: detected hash type "LM", but the string is also recognized as
"HAVAL-128-4"
Use the "--format=HAVAL-128-4" option to force loading these as that type
instead
Warning: detected hash type "LM", but the string is also recognized as "MD2"
Use the "--format=MD2" option to force loading these as that type instead
Warning: detected hash type "LM", but the string is also recognized as "mdc2"
Use the "--format=mdc2" option to force loading these as that type instead
Warning: detected hash type "LM", but the string is also recognized as
"mscash"
Use the "--format=mscash" option to force loading these as that type instead
Warning: detected hash type "LM", but the string is also recognized as
"mscash2"
Use the "--format=mscash2" option to force loading these as that type instead
Warning: detected hash type "LM", but the string is also recognized as "NT"
Use the "--format=NT" option to force loading these as that type instead
<< SNIP >>
Using default input encoding: UTF-8
Using default target encoding: CP850
Loaded 7528 password hashes with no different salts (LM [DES 128/128 AVX-16])
<< SNIP >>
```

Since we know that the hashes are NTLM hashes, we can re-run JtR with the --format switch to specify the hash type as NT:

```
/JohnTheRipper/run/john --format=NT some_set_of_ntlm_hashes.txt
```

### To Brute Force, or not to Brute Force. That is the Question.

By default, JtR will attempt to brute force the hash. This means that JtR will attempt to guess all possible character combinations, from the shortest length (1 character long), to a maximum length of 13 (or up to 24 if you modify the john.conf configuration file). Though JtR attempts the most common characters and combinations first, brute force may not be the quickest way to recover the plain-text.

- **Word Lists** - Using the --wordlist switch, JtR's default behavior can be modified to use a word list as the source for the guesses JtR uses when trying to crack the hash. The word list switch is assigned a word list by providing the path to a file containing a list of words, one word per line.

  ```
  /JohnTheRipper/run/john --format=NT --
  wordlist=/path_to_wordlists/crackstation.txt some-hashes.txt
  ```
  The preceding command instructs JtR to use the NT hashing algorithm with the crackstation.txt word list on the hashes within some-hashes.txt.

- **Word Lists with Rules** - Rules can be used to manipulate and mangle the original word in the word list such that variations of the original word are tested. Variations such as leet speak (such as replacing e's with 3's or a's with @'s, and prepending or appending commonly used characters, such as two- and four-digit representations of the year, are applied. The following two commands are frequently used, but feel free to explore the john.conf file to see what other rules are available.

  ```
  /JohnTheRipper/run/john --format=NT --
  wordlist=/path_to_wordlists/crackstation.txt --rules some-hashes.txt
  ```
  The preceding command runs the default set of rules as configured in the john.conf file.

  ```
  /JohnTheRipper/run/john --format=NT --
  wordlist=/path_to_wordlists/crackstation.txt --rules=KoreLogic some-
  hashes.txt
  ```
  The preceding command runs the KoreLogic set of rules which are very thorough. On the GtW, the entire set of rules may not be able to be executed in a reasonable time frame.

- **Masks** (Requires Jumbo Patch) - The --mask switch can be used to instruct JtR to generate crack guesses based on a pre-defined mask, instead of a full brute-force. This attack method can reduce the character space for the guesses. A mask is created by specifying the type of character JtR should include in its guesses. The character types are defined as follows (note there are many more options/features, but to keep this tutorial simple only basic information is provided below):
  ```
  ?a - All Characters (upper/lower case letters, numbers, and symbols)
  ?d - Only Numbers (0 - 9)
  ?l - Lower Case Letters (a-z)
  ?s - Symbols (`~!@#$%^&*()_+-={}|[]\;':",./<>?)
  ?u - Upper Case Letters (A-Z)
  ```

  Masks can also use custom character types. A custom character types can be assigned to JtR switches -1 through -9 as follows:
  ```
  -1=?u?l - Specifies that the custom character type ?1 is made up of
  upper and lower case letters
  -4=?d?s - Specifies that the custom character type of ?4 is made up of
  numbers and symbols
  ```

  To specify a mask with JtR, the following commands can be used:
  ```
  /JohnTheRipper/run/john --format=Raw-MD5 --mask=?u?l?l?l?d?s
  some_md5s.txt
  ```

The preceding command specifies that JtR will create guesses in which the **first** character is an *upper case letter*, the **second through fourth** characters are *lower case letters*, the **fifth** character is a *number*, and the **last** character is a *symbol*.

```
/JohnTheRipper/run/john --format=Raw-MD5 --mask=?1?l?l?l?d?4 -1=?u?l -4=?d?s some_md5s.txt
```

The preceding command specifies that JtR will create guesses in which the **first** character is an *upper case OR lower case letter*, the **second through fourth** characters are *lower case letters*, the **fifth** character is a *number*, and the **last** character is a *number OR symbol*.

```
/JohnTheRipper/run/john --format=Raw-MD5 --mask=?d?d?d?d?d?d --min-length=5 --max-length=6 md5_weak_session_ids.txt
```

Note that the mask in the previous command specifies 6 digits/numbers (?d). The mask needs to provide character types up to the maximum length specified. The command instructs JtR to iterate through all 5-character numeric combinations, then all 6-character numeric combinations, for the MD5 hash type.

The --min-length and --max-length switches instruct JtR to increment the guesses from the minimum value specified to the max value, using the character types specified in the mask. If the increment value is smaller than the length of the mask, the mask is truncated to the length corresponding with the increment value. In the case of this example, JtR will drop the last ?d at the end of the mask when the increment value is at 5.

```
/JohnTheRipper/run/john --format=Raw-MD5 --mask=list_o_masks.txt some_md5s.txt
```

Alternatively a list of masks, one mask per line, can be stored in a file and referenced with the --mask switch.

- **Hybrid Masks** (Requires Jumbo Patch) - This mode of generating password guesses combines word lists and masks. When specifying the mask, use ?w to represent the entry from the word list. See the following example:

  ```
  /JohnTheRipper/run/john --format=Raw-MD5 --mask=?s?w?d?d --wordlist=some_small_wordlist.txt some_md5s.txt
  ```

  The preceding command specifies that JtR will prepend each entry in the file some_small_wordlist.txt with a symbol, and append two digits to the end.
  ***NOTE:** The word list should be small, and only 2-3 values for the mask, when using this type of an attack on your GtW, as the number of possible guesses will become too many to cycle through in a reasonable amount of time.*

## Hashcat

One nice feature of the latest version of Hashcat is that it will use all available CPU and GPU resources in parallel, by default. On the MacBook Pro's, this means Hashcat will crack using your CPU, Video GPU, and the GPU in your camera.

### Setting the Hash Type

Hashcat does ***NOT*** auto-detect the hash type like JtR.  To specify the hash type use the corresponding hash code with the -m switch.  To determine the correct hash code, see the section of this document Determining Hash Types.

## Any Manual Labor I've Done was Purely by Mistake.

By default, Hashcat will accept input from standard in (stdin) as the source for the guesses Hashcat uses.  You could enter password guesses one at a time from the command line, but this would be silly unless you were testing.  This method is useful when using Hashcat, or another word list / guess generator, to improve performance for slow hash algorithms (https://hashcat.net/wiki/doku.php?id=frequently_asked_questions#how_to_create_more_work_for_full_speed).  While there are use cases for using hashcat in this way on your GTW, this document will focus on the other methods of generating guesses.

- **Word Lists** - Hashcat's default behavior can be modified by adding the path to a word list after the filename that contains the list of hashes to crack.

  ```
  ./hashcat -m 3000 some-hashes.txt /path_to_wordlists/crackstation.txt
  ```
  The preceding command  instructs Hashcat use the NT hashing algorithm with the crackstation.txt word list on the hashes within some-hashes.txt.

- **Word Lists with Rules** - Rules can be used to manipulate and mangle the original word in the word list such that variations of the original word are tested.  Variations such as leet speak (such as replacing e's with 3's or a's with @'s, and prepending or appending commonly used characters, such as two- and four-digit representations of the year, are applied.  The GtW will not be fast enough to handle large word lists (like crackstation.txt) with any of the large rule files.

  ```
  ./hashcat -m 3000 some-hashes.txt /path_to_wordlists/rockyou.txt -r
  ./rules/best64.rule
  ```
  The preceding command runs the set of rules in the best64 rule file on each word within the rockyou.txt word list.

  Below is a suggested word list size and rule combinations guide.  ***The suggestions are based on hash types that don't require much work effort to crack, such as NT, MD5, LM, etc.  Adjustments should be made for stronger hashing algorithms.***
  - Huge Word Lists (crackstation):  best64.rule, leetspeak.rule, hob064.rule*
  - Medium Word Lists (such as rockyou):  Most Word List Rules (see previous line), InsidePro-PasswordsPro.rule, T0XlCv1.rule, d3ad0ne.rule, d3adhob0.rule*
  - Small Word Lists: Probably can use just about any of the rules files.

  ** The Hob0 rules are pretty effective, though not included with hashcat (https://github.com/praetorian-inc/Hob0Rules).*

- **Masks** - Hashcat can be instructed to generate crack guesses based on a pre-defined mask(s), instead of a full brute-force, by changing the attack mode with the -a switch. This attack method can reduce the character space for the guesses. A mask is created by specifying the type of character Hashcat should include in its guesses. The character types are defined as follows (note there are many more options/features, but to keep this tutorial simple only basic information is provided below):

```
?a - All Characters (upper/lower case letters, numbers, and symbols)
?d - Only Numbers (0 - 9)
?l - Lower Case Letters (a-z)
?s - Symbols (`~!@#$%^&*()_+-={}|[]\;':",./<>?)
?u - Upper Case Letters (A-Z)
```

Masks can also use custom character types. A custom character types can be assigned to Hashcat switches -1 through -4 as follows:

```
-1=?u?l - Specifies that the custom character type ?1 is made up of
upper and lower case letters
-4=?d?s - Specifies that the custom character type of ?4 is made up of
numbers and symbols
```

To specify a mask with Hashcat, the following commands can be used:

```
./hashcat -a 3 -m 500 some_md5s.txt ?u?l?l?l?d?s
```

The preceding command specifies that Hashcat will create guesses in which the **first** character is an *upper case letter*, the **second through fourth** characters are *lower case letters*, the **fifth** character is a *number*, and the **last** character is a *symbol*. **NOTE:** *The position of the mask is* ***after*** *the filename containing the hashes.*

```
./hashcat -a 3 -m 500 -1=?u?l -4=?d?s some_md5s.txt ?1?l?l?l?d?4
```

The preceding command specifies that Hashcat will create guesses in which the **first** character is an *upper case OR lower case letter*, the **second through fourth** characters are *lower case letters*, the **fifth** character is a *number*, and the **last** character is a *number OR symbol*.

```
./hashcat -a 3 -m 500 -i --increment-min=5 --increment-max=6
md5_weak_session_ids.txt ?d?d?d?d?d?d
```

Note that the mask in the previous command specifies 6 digits/numbers (?d). The mask needs to provide character types up to the maximum length specified. The command instructs Hashcat to iterate through all 5-character numeric combinations, then all 6-character numeric combinations, for the MD5 hash type.

The -i switch instructs Hashcat to increment the guesses from the value specified in the --increment-min switch up to the value specified in the --increment-max switch, using the character types specified in the mask. If the increment value is smaller than the length of the mask, the mask is truncated to the length corresponding with the increment value. In the case of this example, Hashcat will drop the last ?d at the end of the mask when the increment value is at 5.

```
./hashcat -a 3 -m 500 some_md5s.txt list_o_masks.txt
```

Alternatively a list of masks, one mask per line, can be stored in a file and referenced by specifying a filename in place of a single mask.

- **Hybrid Masks** - This mode of generating password guesses combines word lists and masks. Hashcat offers two formats: Word List + Mask and Mask + Word List

  To specify a hybrid mask mode the -a switch is used as follows:
  -a 6 = Word List + Mask (Hashcat will append each iteration of the mask to the end of every word in the word list)
  -a 7 = Mask + Wordlist (Hashcat will prepend each iteration of the mask to the beginning of every word in the word list)

  ```
  ./hashcat -a 6 -m 500 some_md5s.txt some_small_wordlist.txt ?d?d
  ```
  The preceding command specifies that Hashcat will append each entry in the file some_small_wordlist.txt with a each iteration of two digit numbers.
  ***NOTE:** The word list should be small, and only 2-3 values for the mask, when using this type of an attack on your GtW, as the number of possible guesses will become too many to cycle through in a reasonable amount of time.*

  ```
  ./hashcat -a 7 -m 500 some_md5s.txt ?s some_small_wordlist.txt
  ```
  *The preceding command specifies that Hashcat will prepend each entry in the file some_small_wordlist.txt with all of the symbols (-=!@...).*
  ***NOTE:** The word list should be small, and only 2-3 values for the mask, when using this type of an attack on your GtW, as the number of possible guesses will become too many to cycle through in a reasonable amount of time.*

  *Be aware that the order of the mask and word list in the two preceding command is important. Be sure to use the correct placement associated with the attack mode that is specified.*

## Other Tidbits

### Benchmarking your system

Hashcat allows you to benchmark a system to determine how many cracks per second will be performed.

To benchmark all algorithms, use the following command:

```
./hashcat --benchmark
```

To benchmark a specific algorithm, use the -m switch with the hash code associated with the hash type to benchmark:

```
./hashcat -m 3000 --benchmark
```

### Some Helpful One-liners

Over the last couple of years, I developed some useful one-liners. Most of them convert hashes into a format usable by Hashcat. The complete list is here: https://github.com/timbo05sec/Tools-n-Stuff/blob/master/Cracking/Cheatsheet.txt