Advanced password guessing

Hashcat techniques for the last 20%

Version 1.0 - 2013.05.12

About me

- Name: Jens Steube
- Nick: atom

- Coding Projects:
  - hashcat / oclHashcat

- Work Status:
  - Employed as Coder but not crypto- or security-relevant

Tools Overview

the hashcat universe

Tools overview

Name	Type
hashcat	Multi-purpose cracker on CPU
oclHashcat-plus	Multi-purpose cracker on GPU, Flagship
oclHashcat-lite	Competition cracker on GPU, Performance
hashcat-utils	Set of handy commandline utilities in password guessing
maskprocessor	Standalone word-generator with mask support, very fast
statsprocessor	Standalone word-generator based on markov-chains

Masks

Why to use them - not how

Masks

- Masks are often used in hashcat, and one can greatly benefit from it if they know how to use them
- Masks are usually a simple topic, but too many people still don’t know how to use them, or why
- I’ll show you a reason why hashcat makes use of them

Masks

- Imagine you want to configure a program to generate all words of:
  - aaa – zzz
- There are many ways to do it, for example:
  - It could have the ability to set a charset (lalpha)
  - It could have the ability to set a password-length (3)
- It’s an intuitional approach - And already requires two parameters to be set

Masks

- For some reason, you have additional information about the password
- You know it ends with 1984
- How would you want the program to accept this additional information?
    - Add a parameter that lets you define a salt to append
- That’s very intuitional, again
- But at this point our program already need 3 parameters

Masks

- One more example
- People tend to capitalize the first letter of a password but not the rest
- How could you tell that to captilize only the first letter?
  - Well, add a flag for this ...
- What about if you know the password capitalizes the first two letters?
- Finally, your program will require more and more parameters

Masks

- Masks can solve this!
- Don’t worry, they are by far not as complex as regular expressions
- Two reasons:
  - Need to be calculated fast (see performance table)
  - Need to be easy to understand
- To learn how to use mask attacks with hashcat, read the "Mask Attack" article on the hashcat wiki, it's only 2 pages

maskprocessor

High-performance standalone word-generator

maskprocessor

- Maskprocessor is a standalone program that requires at least one parameter: The mask
- It then prints all words from the selected keyspace to stdout or to a file
- There are many scenarios where you can use this program

maskprocessor

- For example: Aircrack-ng. Aircrack-ng? Yes!
- Aircrack-ng does not have support for masks, but it does have support for reading candidates from stdin
- The command:
  - mp64 ?l?l?l?l?d?d?d?d | aircrack-ng -w –
- Works on Linux and Windows. Yes, windows can do pipes!
- You don’t need to write it into a wordlists and waste gigabytes of hdd space plus that would produce unnecessary I/O while loading it from disc
- In case you ever wished aircrack-ng should have brute-force abilities for WPA/WPA2 you can do that this way (have fun)

maskprocessor

- Another nice example for how to use maskprocessor is when you want to generate rules. Rules? Yes!
- I will explain rules a bit more later, but for now Imagine you want to crack a password and you know it starts with a uppercase letter and ends with a digit
- You could use grep and pick the right words from your dictionary
- But you could also add all uppercase letters and all digits to all of your words in the dictionary
- That sounds crazy but from my experience it’s the better attack

maskprocessor

- A way to do this is to use rules. I'll explain rules later in more detail but for now its enough to know its a little programming language
- With a rule you can only append or prepend 1 specific character. You can not select a range. But you can have as many rules as you want
- That makes 26 * 10 rules in total. You want to write that per hand? Have fun
- You can code a little script to do it or you use maskprocessor to do it:
  - mp64 -o bla.rule '^?l $?d‘

maskprocessor

- If you‘re stuck with a hashlist there is usually no way around identify the pattern of the cracked passwords
- Once you‘ve figured them out you have another problem: How do I to tell hashcat how to generate the candidates without a specific attack-mode?
- The answer is simple. It’s often possible to write your own attack-modes by a combination of maskprocessor and hashcat rules
- Maskprocessor is very fast: A single CPU core is around 50-100 produced MW/s and more. That's typically fast enough to feed hashcat
- If you‘re writing a cracker you can use maskprocessor to do the password-generator work

statsprocessor

The special maskprocessor

statsprocessor

- The statsprocessor is basically the same as the maskprocessor but with one difference:
    - It's using markov-chains to optimize the output in probabilistic order
- As long as you are not modifying the threshold the number of output to maskprocessor is the same, just the ordering differs
- The calculation makes it a bit slower than mask-processor but when you have a slow algorithm like TrueCrypt that doesn’t matter since the blocking part in this case is the algorithm, not the generator

Attack-modes Overview

All roads lead to the password

Attack-modes

- Hashcat supports basic attack-modes (not discussed here):
  - Dictionary
  - Brute-Force
- Hashcat supports advanced attack-modes:
  - Combinator
  - Table-Lookup
  - Toggle-Case
  - Permutation
  - Fingerprint
  - Hybrid
  - Rule-based

Combinator attack

Attack-modes

combinator-attack

- This is one of my favorite attack-modes when reaching a higher percentage level of cracking a hashlist
- The idea is very simple. You have two dictionaries, not one. They are named as left and right dictionary
- Each word of the right dictionary is appended to each word of the left dictionary
- Another way to explain it is: If your left dictionary contains 100 words and the right dictionary contains 50 words, then the number of total candidates generated is 100 * 50 = 5000

combinator-attack

- This is a good way to produce full names and compound words
- Example, if you have a dictionary that contains only first names:
  - Lucy
  - Ann
- You can use the same dictionary on both sides, thus efficiently create full names:
  - LucyAnn
  - AnnLucy

combinator-attack

- Usually they are not written that way. What you can do is to apply an additional single rule per dictionary. That can be done with the -j and the -k parameters with oclHashcat-plus or with the combinator.rule in hashcat-CPU
- The Idea is to append a "-" character to each of the words from the left dictionary:
    - Lucy-Ann
    - Ann-Lucy
- NOTE: The same works for a space char, too

combinator-attack

- It’s also effective against passphrases
- Dictionary contains:
  - is qazwsxedc key the cure am my <space> pass this Love i
- Results in:
  - this is my pass
  - i am the cure
  - Love is the key
- NOTE: This requires two rounds of hashcat, one using –stdout
- As with all good attack-modes they produce stuff you do not think of in the first place, so it cracked:
  - qazwsxedc<space>

Table attack

Attack-modes

table-attack

- This attack mode is also based on dictionaries. You can attack the following targets well:
  - International characters
  - Toggled-case words
  - Leetspeek
  - Fill “holes” in your dictionary
- The targets also can be combined, like:
  - Toggled-case words + Leetspeak
- The table attack takes a configuration file, the "table"
- Inside the table, you do a simple X=Y binding per line
  - Where X is a character that is to replace with Y
- NOTE: You can use X multiple times

table-attack

- Example table
  - a=A
  - a=@
  - a=ä
  - a=/\
- Example dictionary
  - Anita
- Example candidates generated
  - AnitA
  - Anit@
  - Anitä
  - Anit/\

Toggle-case attack

Attack-modes

Toggle-Case attack

- One of the easiest attack-modes
- This attack simply tries all upper- and lower-case of a word from a dictionary
- If your dictionary contains “abc”, It generates:
  - abc
  - Abc
  - aBc
  - ABc
  - abC
  - AbC
  - aBC
  - ABC

Toggle-Case attack

- While this attack is supported, it does not make sense to do it this way
- Here‘s why: When people use capitalized letters they either use it at the first letter or the in the word
- There is another variant in which people use less or equal capitalized letters than lowercase letters. For example, passwords of length 10 do not have more than 5 uppercased letters
- oclHashcat-plus therefore uses rules to do Toggle-Case attack. There are rules for toggling 1-5 letters in the hashcat rules directory
- Since rules are compatibe between oclHashcat-plus and hashcat, you can also use them in hashcat

Toggle-Case attack

- If you really want to do full toggle-case attack you can still feed oclHashcat-plus from hashcat piped candidates:
  - hashcat-cli -a 2 your.dict --stdout | oclHashcat-plus your.hashlist
- NOTE: This will work efficiently only for slow hashes

Toggle-Case attack

- If you combine the toggle.rule with leetspeak.rule you can crack more sophisticated passwords:
  - oclHashcat-plus your.hashlist -r rules/toggles3.rule -r rules/leetspeak.rule
- Produces:
  - Scotl@nd
  - Sh@mr0ck
  - j3sUsFr3aK
  - AlexAndr1a
  - MyPa$$word
  - $ailorM0on
- Admittedly, the table attack is a much better approach to do this, but there is no table-attack for oclHashcat-plus. This is a good emulation

Permutation attack

Attack-modes

Permutation-attack

- This attack mode was an idea that for some reason never really
  worked well
- I want to show what the Idea was, maybe you can use it
- Permutation attack is exactly what it sounds like:
  - ABC
  - ACB
  - BAC
  - BCA
  - CAB
  - CBA

Permutation-attack

- The original Idea was that if the user has the following word in his dictionary:
  - Pass123
- It will produce the following candidates:
  - pass123
  - pass321
  - 1pass23
  - 3pass21
  - 12pass3
  - 32pass1
  - 123pass
  - 321pass

Permutation-attack

- From my experience these are passwords that people actually use
- NOTES:
    - It's supported in hashcat CPU only, you can use --stdout
    - It's also a standalone binary in hashcat-utils in case you find a different use for it

Fingerprint attack

Attack-modes

Fingerprint-attack

- The fingerprint attack is by far to complex to discuss is in here
- The goal is to crack complex passwords like this:
    - 10-D'Ann
- But in an automated way so that it does not require human attention
- It makes extensive use of the expander utility that comes with hashcat-utils
- Read more about the fingerprint attack on the hashcat wiki

Fingerprint-attack

- We used it at Defcon 2010 when team hashcat won the "Crack Me If You Can" competition
- The autocrack-plus.pl cracking helper also makes use of this
- There are also example videos made by the backtrack developers to explain it, you can find it on youtube.

Rule-based attack

Attack-modes

Rule-based attack

- The rule-based attack is the first attack I do against large unsalted hashlists because its the most economic one
- The chosen candidates have a very high probability and the dictionary this attack bases only can be chosen freely
- Everyone who ever used oclHashcat-plus knows that it requires some workload to run with full speed. That is because the GPU must be remain busy
- If I run just a dictionary again a large hashlist it will crack a lot but the GPU will idle
- Add rules too because it costs you nothing in terms of time. The number of additionally produced candidates are for free because of the performance gain you get

Rule-based attack

- Rules are little programming language. Hashcat (among others) has a built-in interpreter for it. It’s specially designed for word manipulations. The user can program it pretty easily.
- The functions you can use are very basic
- There is a rule to append character and to prepend, you can cut around ranges, reverse the words, etc..
- Read all about how to write and use rules on the hashcat wiki
- There is also a few example rules in the rules/ folder for hashcat and oclHashcat-plus you can take a look at

Rule-based attack

- With hashcat you can let it write debugging information about how the rule engine processed a word to crack a password, what the basic password was, what the rule was, etc. that you can build up statistics about their efficiency
- This is a unique feature
- We have already use it to rules/generate.rule file automatically
- You can also use the --stdout option, see debugging section

Rule-based attack

- There is another unique feature in oclHashcat-plus that allows you to stack rules. You can configure to use multiple rules files.
  - NOTE: that does not mean to execute them in a sequence
- The multi-rule feature combines like the combinator-attack each rule of both rule-files with each other
- You can this way create new attack-modes. There is a special subfolder hybrid/ in the rules/ folder that are simple with maskprocessor generator rules that just appends all letters
- There is another one that does the same, but prepends all letters

Rule-based attack

- If you use them together with -r rules/hybrid/prepend_l.rule -r rules/hybrid/append_l.rule it actually does both things at once with your words
- If you have “xpasswordy" to be cracked, and you dictionary contains "password", you will crack it

Hybrid attack

Attack-modes

Hybrid-attack

- Hybrid attacks is my favourite attack against large unsalted hashlists for dictionary building once I've finished rules
- It's common knowledge people append years, birthdays and number to names, locations, etc, right?
- But which ones and how can you be sure you hit the right one? You cant so you have to guess
- But using brute-force to attack against names and locations seems inefficient, no?

Hybrid-attack

- The hybrid attack has two parameters. One is a dictionary and one is a mask. Again, you see why its important to understand masks here
- Simply defined, the hybrid attack brute-forces a range and this range is appended or prepended to each word from your dictionary
- You can choose whatever side you want the dictionary, the left or the right side. I recommend to try both
- But depending on the side were you place the dictionary, you should change the mask

Hybrid-attack

- When you have the dictionary on the right side it’s more common users choose numbers or symbols to make the password “more secure“
- Example:
  - Julia1984
  - Password1!!@
  - NewYork1+2
- You should craft your mask like this: -1 ?d?s ?1?1?1

Hybrid-attack

- But there is more Fun stuff. You can "exploit" this mode to crack passwords which are only partially in your dictionary.
- For example, you want to crack:
  - thecathat
- But you have just the word "thecat" in your dictionary, the mask ?l?l?l appended to will crack it
- It‘s again one of these attack-modes that will result in cracked passwords you did not think of in the first place or you did not target directly but you‘ll get them as a bonus

Hybrid-attack

- The opposide side is also nice, but you should change the type of masks you're attacking
- Typically this is good if you have partial passwords again and the password to be cracked is capitalized
- You have the password "Telephone" but your dictionary only contains "phone", the mask ?u?l?l?l would crack it

Hybrid-attack

- I'll leave this attack-mode and recommend you my absolute favorite attack:
  - -a 6 my.dict -1 ?l?d?s ?1?1?1

Using hashcat’s --stdout

… to feed other crackers

Using hashcat’s --stdout

- Hashcat is still a young project (compared to other crackers) not all hash-algorithms are supported yet
- If you need to use a different cracker like JtR to crack an unsupported hash you can still use hashcat‘s advanced attack-modes to feed them with candidates
- It‘s simple:
  - hashcat-cli -a 2 my.dict --stdout | john --pipe my.hash
- As long as the cracker supported reading plains from stdin this should work. If you‘re coding a special cracker for something this could help you to focus on the cracking part, not on the generating part.

Debugging

Is it doing what you want it to do?

Debugging

- Often you prepare something you think this is what you want but then it runs and runs and nothing happens
- You begin to think did I everything correctly?
- Attack-modes can become very complex, you better take a look at it!

Debugging

- In hashcat (CPU only!) you can use the --stdout parameter
- As discussed in the previous section, this parameter is primary used to pipe candidates outputs into external programs but you can also use it to see what hashcat is doing
- In oclHashcat-plus you can not, but the attack-modes are compatible. If you want to debug stuff for oclHashcat-plus you can use hashcat
- If the output does not match what you think it does you don’t need to worry any longer

Debugging

- It can also help to learn rules. Try it, just create a single rule-file and place into it:
  - $1
- Save it and then execute hashcat-cli -r my.rule --stdout some.dict
- All candidates should have a 1 appended
- This works for all attack-modes

Thank you for listening!

- Feel free to contact me!

  - via Twitter: @hashcat
  - via Hashcat forum: http://hashcat.net/forum/
  - via IRC: freenode #hashcat
  - via Email: atom at hashcat.net