CRACKING PASSWORDS 101

A question I have gotten multiple times recently is "*how do I do password cracking?*". People think they need to start with big dictionaries, clever rules, or other stuff. As with most things in life, getting started is not difficult. If anything it is .. repetitive. Seriously. To show you what I mean I figured a practical demostration would be in order. So lets start with the "*rootkit.com*" hashlist (get it here) and get "*hashcat*" and "*john the ripper*". Now follow along and see how you do:

1. Nothing: I am purposely starting with NO dictionary. Nothing. Just the tools and the hashlist. So we start with this:

```
# wc -l ./rootkit.hash
71228 ./rootkit.hash
~tracking = hashes: 71228
~tracking = passwords: 0
```

..and we are going to go for the really low hanging fruit - we are going to try some simple brute-forcing of the passwords:

```
#hashcat-cli32.bin -m 0 -a 3 --remove -o ./rootkit.dic ./rootkit.hash ?d?d?d?d?d?d?d?d?d?d
#hashcat-cli32.bin -m 0 -a 3 --remove -o ./rootkit.dic ./rootkit.hash ?l?l?l?l?l?l?l
#hashcat-cli32.bin -m 0 -a 3 -1 ?l?d --remove -o ./rootkit.dic ./rootkit.hash ?1?1?1?1?1?1?1
```

So how did we do with that? Not a bad start actually:

```
~tracking = hashes: 43446
~tracking = passwords: 27782
```

2. Quick Hashcat Mangling: Hashcat has some mangling rules you can run through that do not take up much time. So we first update our dictionary file:

```
#cat ./rootkit.dic | cut -f 2- -d ":" > ./working.dic
```

..and then we run the mangling rules referencing that dictionary file:

```
#hashcat-cli32.bin -m 0 -r ./rules/best-422RvtEplJ.rule  --remove -o ./rootkit.dic ./rootkit.hash ./
working.dic
#hashcat-cli32.bin -m 0 -r ./rules/best64.rule  --remove -o ./rootkit.dic ./rootkit.hash ./working.dic
#hashcat-cli32.bin -m 0 -r ./rules/best-mpGFdsZoei.rule  --remove -o ./rootkit.dic ./rootkit.hash ./
working.dic
#hashcat-cli32.bin -m 0 -r ./rules/passwordspro.rule  --remove -o ./rootkit.dic ./rootkit.hash ./
working.dic
#hashcat-cli32.bin -m 0 -r ./rules/T0XlC.rule  --remove -o ./rootkit.dic ./rootkit.hash ./working.dic
#hashcat-cli32.bin -m 0 -r ./rules/d3ad0ne.rule  --remove -o ./rootkit.dic ./rootkit.hash ./working.dic
#hashcat-cli32.bin -m 0 -r ./rules/generated.rule  --remove -o ./rootkit.dic ./rootkit.hash ./
working.dic
#hashcat-cli32.bin -m 0 -r ./rules/combinator.rule  --remove -o ./rootkit.dic ./rootkit.hash ./
working.dic
```

So how are we doing now?

```
~tracking = hashes: 36286
~tracking = passwords: 34943
```

3. KoreSecurity Hashcat rules: KoreSecurity released a set of hashcat rules for public use. So update your dictionary again:

```
#cat ./rootkit.dic | cut -f 2- -d ":" > ./working.dic
```

..and then run through the KoreSecurity mangling rules (I use a for loop for simplicity):

```
#for x in `./rules/KoreLogicRules*`
  do
   hashcat-cli32.bin -m 0 -r $x  --remove -o ./rootkit.dic ./rootkit.hash ./working.dic
 done
```

We get some gains:

```
~tracking = hashes: 34807
~tracking = passwords: 36445
```

So with very little work we have found just over 50% of the passwords. Onwards!

4. Hashcat "fingerprinting" attack: This is a interesting one, it is based upon the -generally true- assumption that people use similiar patterns. So what you do is you take the passwords you have found:

```
#cat ./rootkit.dic | cut -f 2- -d ":" > ./working.dic
```

..then use one of the hashcat utility programs to take that dictionary and break each entry into various "patterns":

```
#hashcat-utils-0.9/expander.bin < ./working.dic | sort -u > ./rt-expanded.dic
```

..then hashcat has an attack mode which tries the patterns in the "expanded" dictionary in combination with one another:

```
#hashcat-cli32.bin -m 0 -a 1 --remove -o ./rootkit.dic ./rootkit.hash ./rt-expanded.dic
```

Like I said, it is a generally true assumption, as you can see by our progress:

```
~tracking = hashes: 27457
~tracking = passwords: 43795
```

5. Rinse and Repeat: Nothing fancy, just update your dictionary file and run through steps (2) and (3) until no more major gains:

```
~tracking = hashes: 24159
~tracking = passwords: 47093
```

6. "Dotting the i's": Now that our dictionary is getting bigger we will utilize some of the other hashcat modes. So update your dictionaryand then start with the "permutation" attack:

```
#hashcat-0.42/hashcat-cli32.bin -m 0 -a 4 --remove -o ./rootkit.dic ./rootkit.hash ./working.dic
```

.and then use the new tables method (using the default tables):

```
#for x in digits.table leet.table toggle_case_and_leet.table toggle_case.table
 do
  ./hashcat-cli32.bin -m 0 -a 5 -t ./tables/$x --remove -o ./rootkit.dic ./rootkit.hash ./working.dic
 done
```

We get some new passwords:

```
~tracking = hashes: 23298
~tracking = passwords: 47954
```

7. Hashcat Markov attack: I will oversimplify again and say this is a smarter way of doing brute-forcing. First we create the "training" data:

```
# ./statsprocessor-0.09/hcstatgen.bin ./statsprocessor-0.09/rootkit.hcstat < ./working.dic
Processed input lines: 47955
Writing stats: ./statsprocessor-0.09/rootkit.hcstat
```

Then we use that to generate the guesses. You can play around with the threshold and password length, but generally the higher those values the longer it takes to complete this step. Now since hashcat cannot accept stdin, I pipe the password guesses to a pipe file and then point hashcat to that:

```
#./statsprocessor-0.09/sp32.bin --threshold 200 --pw-max=16 ./statsprocessor-0.09/rootkit.hcstat  > ./
temp.pipe&
#hashcat-cli32.bin -m 0 -a 0 --remove -o ./rootkit.dic ./rootkit.hash ./temp.pipe
```

Very succesful if we look at our progress:

```
~tracking = hashes: 16861
~tracking = passwords: 54391
```

8. Rinse and Repeat again: With the new updated dictionary run through steps (2), (3) and (4) again.

```
~tracking = hashes: 15780
~tracking = passwords: 55472
```

9. JTR specific rulesets used in hashcat: *John the ripper* has it's own mangling rules, but in my experience *hashcat* is a bit

faster. So we will use JTR and it's rules to generate password attempts for hashcat to use. As an example I am going to use a for loop to run throught the KoreSecurity rules for JTR and use that with hashcat:

```
#for x in `cat ../john-1.7.9-jumbo-7/run/john.local.conf | grep -E ".Rules:Kore" | cut -f 2 -d ":" | cut -f 1 -d "]"`m
  do
   ../john-1.7.9-jumbo-7/run/john --rules=$x -w=./working.dic --stdout > ./temp.pipe &
   hashcat-cli32.bin -m 0 -a 0 --remove -o ./rootkit.dic ./rootkit.hash ./temp.pipe
  done
```

Not a lot of progress :(

```
~tracking = hashes: 15478
~tracking = passwords: 55774
```

10. JTR markov mode: JTR also has a *markov* mode, so lets use that:

```
#../john-1.7.9-jumbo-7/run/john -markov:225:0:0:12 --stdout > ./temp.pipe &
#hashcat-cli32.bin -m 0 -a 0 --remove -o ./rootkit.dic ./rootkit.hash ./temp.pipe
```

Not a lot of passwords :(

```
~tracking = hashes: 15466
~tracking = passwords: 55786
```

11. Check dictionary for gaps: Here you use a little guesswork to see if there are any patterns which suggest you could use that to guess passwords. For example I saw special characters and lowercase characters in 1-6 character passwords. In our brute forcing step at the start we did not look for this, but 1-6 characters is not going to take long:

```
#hashcat-cli32.bin -m 0 -a 3 -1 ?a --remove -o ./rootkit.dic ./rootkit.hash ?1?1?1?1?1?1
```

Better progress then the last 2 steps:

```
~tracking = hashes: 15285
~tracking = passwords: 55967
```

12. Hashcat "randomly generated" rules: Hashcat has a mode where you can let it randomly generate mangling rules. Lets update our dictionary and try that:

```
#hashcat-0.42/hashcat-cli32.bin -m 0 -g 90000 --remove -o ./rootkit.dic ./rootkit.hash ./working.dic
```

Not great progress, so we will not repeat this step:

```
~tracking = hashes: 15239
~tracking = passwords: 56013
```

13. Combine markov and mangling rules: Hashcat allows you to combine mangling rules with Markov password guesses. So lets use a for loop to do that:

```
# for x in best-422RvtEplJ.rule best64.rule best-mpGFdsZoei.rule passwordspro.rule T0XlC.rule
  do
    ./statsprocessor-0.09/sp32.bin --threshold 220 --pw-max=13 ./statsprocessor-0.09/rootkit.hcstat > ./temp.pipe &
    hashcat-cli32.bin -m 0 -r ./rules/$x --remove -o ./rootkit.dic ./rootkit.hash ./temp.pipe
  done
```

Not bad progress for this far in the process:

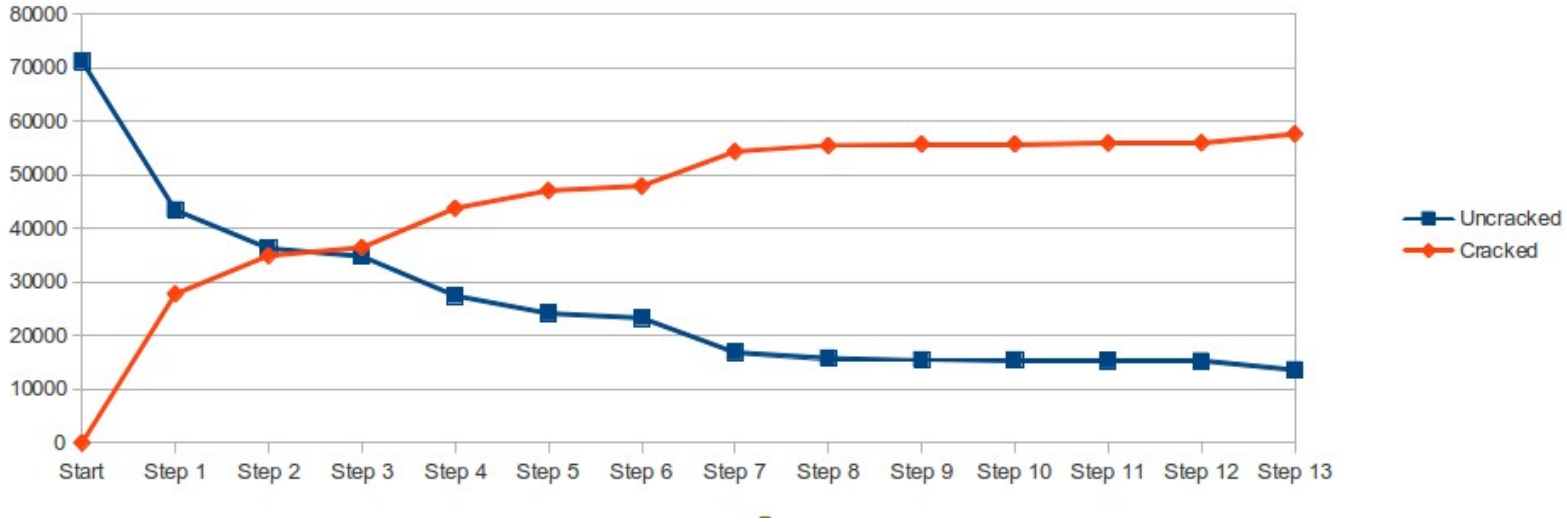
```
~tracking = hashes: 13601
~tracking = passwords: 57651
```

Results:
In a few standard tests we cracked 57627 of 71228 passwords. 80%. Not bad for starting with nothing. And the nice thing is that all of these steps can be scripted very easily to run through them automatically. You will also have noticed this is an iterative process with a large amount of repeated steps. This is the nature of the activity, password cracking is a process that benefits as you learn and expand your resources. In this process we started from a dictionary of nothing, but if we had started with a good dictionary, our gains would have been more. It is also iterative in that the work we have undertaken can be leveraged against others lists, for example using our new dictionary in a plain, simple (no mangling or rules) dictionary attack:

- ~against original gamigo list = *Recovered.: 13024/7004341 hashes*
- ~against original stratfor list = *Recovered.: 4585/860149 hashes*
- ~against original mayhem list = *Recovered.: 1498/130884 hashes*
- ~against original opisrael list = *Recovered.: 155/10802 hashes*
- ~against original blackstar-1 list = *Recovered.: 192/3555 hashes*
- ~against original blackstar-2 list = *Recovered.: 145/2389 hashes*
- ~against original blackstar-3 list = *Recovered.: 389/4262 hashes*
- ~against original unmasked linkedin list = *Recovered.: 2/2935345 hashes*
- ~against original masked linkedin list = *Recovered.: 54106/3487579 hashes*
- ~against original bkav-1 list = *Recovered.: 7540/113224 hashes*
- ~against original bkav-2 list = *Recovered.: 3416/20978 hashes*
- ~against original gawker list = *Recovered.: 109169/743855 hashes*

So you can see straight away that even though our work was focused on the rootkit.com list, the dictionary we have created from that work gives us immediate gains across the other lists. And if we then used all the steps again against a different list (for example gamigo), using our new dictionary as the starting point, we would be able to increase our dictionary. And then that increased dictionary would probably give us some more gains across all the lists and probably against the rootkit.com list as well. This is what I meant by it is an iterative process.

But I must point out, that when we start these processes, same as with our starting steps in the process above, our initial gains will be impressive but successive iterations and steps will yield smaller and smaller gains:



The simple fact is, as you guess the easier passwords you are ending up with the more difficult passwords. At some stage in the process you will cross the point where these types of steps are no longer viable or effective, and it is at that stage when you need to begin trying to come up your own ways to make your cracking more effective (password profiling, new mangling techniques or rules, etc). That is when it becomes not about your CPU/GPU cycles and dictionary, but about you.