My password cracking brings all the hashes to the yard..

Larry Pesce
Hackfest, 2015
You get the theme...

MY MILKSHAKE BRINGS ALL THE BOYS TO THE YARD

AND THEY'RE LIKE, IT'S BETTER THAN YOURS
So let’s start this off right:
About me

- Penetration Tester/Hardware Hacker, @ InGuardians (Sr. Managing Consultant, Director of Research)
- …and the guy who mans the password cracking rig
- SANS Instructor
- Paul’s Security Weekly crew
- Extra class ham radio operator (KB1TNF)
Disclaimer

- This is not the only way - there is no right answer.
- This is what works for me
  - I’m interested in expanding the methodology
  - Feedback welcome!
- We’re talking fairly simple hashes
  - NTLM, MD5, *-HALFCHAL, DES,
  - WPA/2-PSK, bCrypt not so much and compound our problems
- To meet goals, limited to no discussion of
  - Multi-byte input (unicode, language packs, etc)
  - Multiple language sets and conversion
    - Team Hashcat CMIYC 2015
The Problem

• So, you’ve got this huge list of hashed passwords
• Go for the brute force!
  • This becomes overwhelming
• There has to be a better way
  • Throwing money at the problem..
  • …Moore’s law
• There is lots of talk about the technical how, but not a lot about the methodology how
Goals (1)

- To get “just enough”
  - There sill always be that *one* person with the 32 character password
  - Most of our avenues of attack only require 1-2 passwords
  - But they have to be for the *right account* out of many.
• Aiming for > 65% < 85% recovery rate is a great spot
  • Statistically we have a really good chance of finding at least one that will further our aims
  • Only my personal experience
• 10% may be more realistic
  • May get lots of accounts, but might not get lucky
  • Not a bad idea to run with the top 100 most common passwords
  • Maybe even some munging of this top 100…
Start at the Beginning

- Yes, JtR is a great start
- No GPU needed!
- With patches to support for additional hash types
- Requires a wordlist…
- Use rules to modify wordlist
But, where?

- CPU is good in a pinch
- Especially on a pen test without a GPU rig on site
- Maybe get you that *one* account
- GPU can be better
- You'll need to spend time up front
  - Install, drivers, etc.
  - http://www.getpimp.org/ - just install *hashcat*
- Secure remote access
- A modest rig is decent
  - Want to go big? Go see EvilMog today right after this talk in Track 4
  - Crypto-coin folks do great analysis, but needs update from the password side of the house
- Mine crypto currency in the down time
- Maybe even AWS…
Wordlists

- Crafting wordlists is more art than science…
- Modification of those wordlists is art as well
- Get some base wordlists
  - rockyou, ashleymadison (md5…)
  - skullsecurity.org
  - https://xato.net/passwords/ten-million-passwords/ - this may be overkill..
- Add some basics if they aren’t there
  - Seasons, popular baby names.
- This is only the foundation!
- We need to build on this list and how we use it
Pillage...

- Take your base word list and add to it
- Build some structure on the foundation!
- Adding before any munging happens for a broader base
- Think about the victim and location
  - Cities/towns, sports teams, team players, local landmarks, local foods
- The company itself
  - CeWL FTW.
  - Variations on the company name too
- These variations will vary from engagement to engagement
Plunder...

• You’ve got access to systems that you gathered the hashes from...
• ...pillage the village!
  • Some systems will be more fruitful than others
    • DC’s? Not so much.
    • File servers, win
    • User workstations, win
• Grab files!
  • Lots of good *contents* of files
• What about *filenames*?
Burn it down.

• Oh snap, filenames?
  • Sure, specifically, user’s file names in the “My Documents” folders

• Think about this:
  • I have a child, chances are I have a picture, corinn.jpg
  • There is a good chance my password is, Corinn42!
  • (it isnt.)
Filename Acquisition

- Linux
- Strips extension
- Concatenates up to 7 spaces in filenames.
- Additional can remove underscores and dashes if needed.

```
ls -IR | awk -F" " '{print $9$10$11$12$13$14$15$16}F' | awk NF | awk -F. '{print $1}' > filenames.txt
```

- Ugh, Windows.

```
C:\> FOR /R c:\ %i in (*.*) do @echo %i >> filenames.txt
```
Wordlist Munging

- We have a list of words, cool
- Good orgs don’t allow dictionary words….
  - They want numbers, special characters
    - Who knew “?” is not a special character…
- We need to add them in various means
- Let’s talk munging
Wordlist Munging

- Munging can take several forms
- Character replacement (L33t Sp34k)
- Capitalization
- Character addition
  - Numbers, special characters
  - Beginning, end
- Concatenation
- This is where things get interesting…
- …and large.
Wordlist Munging

• We have a couple of ways to do some munging, simple and complex
  • Both are awesome!
• Simple won’t jack up the size of our wordlist too bad.
• Complex can jack our wordlist to unmanageable levels
  • We just need to be smart about how we use it.
• How do we munge?
Simple Munging (1)

- Simple, use a little python
- Create JtR and hashcat rules and can then be used for munging
- Simple character substitution
- Has some interesting dynamic to the wordlist
  - Doesn’t extend word length
  - Does add words
Simple Munging (2)

- https://github.com/inguardians/password_tools
  
- passrulegen_casetoggle.py
  - Modify “d” for more 1337 swaps

- passrulegen_1337toggle.py
  - Modify “l” for max password length

# john --wordlist=base-wordlist.txt --rules:casetoggle.rule --stdout >> casetoggle-wordlist.txt

# oclHashcat -m 1000 -r casetoggle.rule unknown.hash casetoggle-wordlist.txt

* Jarrod Frates (@networkllama)
Complex Munging (1)

- Complex is a multi staged approach
- Wordlists and tool functionality
- More on the tool portion later..
- For strict wordlist munging JtR rules, well, rule
- Especially the Korelogic DEF CON 2010 Crack Me if you Can Rules…
- Prepending, appending, mid-word insertion, l33tsp33k, all the things
Complex Munging (2)

- Korelogic rules
  - `cat rules.txt >> john.conf`
- The result? Without an individual rule, we get MASSIVE word lists.
- We can pass this to our splitting step right away
- We can run more tailored, individual rules based on our investigation
- This will be helpful when we understand password policies
- This can require incremental runs of cracking
  - Each run is "smart" and relatively speedy
  - Each will get us a few percent closer to our sweet spot

```
# john --wordlist=base-wordlist.txt --rules:<rule name> --stdout >> munged-wordlist.txt
```
Wordlist Splitting

• We have a base list that we’ve had our way with…
• It is has potential to be big!
• Also, passwords from 1-20+ characters in no particular order!
• We can take this massive list and split it to be more focused
• We’ll base the split based on what matters..
  • …aside from hash type
  • Length!
Duh!

- Let’s split these lists apart into smaller lists based on character length
- After munging is the time, so we don’t add to the length!
- This was my “duh” moment…
- This may be an iterative process, depending on our
  - Base word list
  - Previous/new iterations of JtR munging
Split it!

• Get split.

• Unix - needs for loop for increment

  # awk -v n=<word length> '{for (i=1; i<=NF; i++) if (length($i) == n) print $i}' <input_list.txt> > output_list_(i).txt

• Windows
  
  • DOS LOL, get bent.

  • Why does text manipulation have to be so hard on Windows?

  • Powershell*

  foreach ($i in $a =
  get-content("text.txt")(cnt = $i | Measure-Object -Character; if
cnt.Characters -eq <X>) {Write-Host $i}) | out-file
  -filepath C:\$iwordlist.txt

* Adam Crompton (@3nc0d3r), Don Weber (@cutaway)
Password Policies

- We can now use our munged and split lists with either JtR or *hashcat.
- We can just run through them all…
- …or we can be smarter about it.
- Ok, so how do we get smart about it?
  - Ask for the password policy!
- Cant ask?
  - We had access to a system to get hashes and to pillage the village
  - Use it!
Unix Password Policy

# cat /etc/login.defs | grep password
password requisite pam_cracklib.so try_first_pass retry=3
minlength=12 lcredit=1 ucredit=1 dcredit=1 ocredit=1 difok=4

try_first_pass = sets the number of times users can attempt setting a good password before the passwd command aborts
minlen = establishes a measure of complexity related to the password length
lcredit = sets the minimum number of required lowercase letters
ucredit = sets the minimum number of required uppercase letters
dcredit = sets the minimum number of required digits
ocredit = sets the minimum number of required other characters
difok = sets the number of characters that must be different from those in the previous password
Windows Password Policy

- **Net command**
  
  ```
  C:\> net accounts
  ...
  ```

- **PowerShell Get-ADDefaultDomainPasswordPolicy RSAT cmdlet**

  ```
  C:\> Get-ADDefaultDomainPasswordPolicy
  ComplexityEnabled : True
  DistinguishedName : DC=nwtraders,DC=msft
  LockoutDuration   : 00:30:00
  LockoutObservationWindow : 00:30:00
  LockoutThreshold  : 5
  MaxPasswordAge    : 42.00:00:00
  MinPasswordAge    : 1.00:00:00
  MinPasswordLength : 7
  objectClass       : {domainDNS}
  objectGuid        : 5765e6a1-cf67-476d-8672-0b8ca3abfac1
  PasswordHistoryCount : 24
  ReversibleEncryptionEnabled : False
  ```

- **Also query the Domain for fine grained password policy for a user**

  ```
  C:\> dsget user testuser -effectivepso
  distinguishedName: CN=testuser,CN=Password Settings Container,CN=System,DC=gs,DC=com;
  dSCorePropagationData: 0x0 = (  );
  instanceType: 0x4 = ( WRITE );
  msDS-LockoutDuration: 0:00:30:00;
  msDS-LockoutObservationWindow: 0:00:30:00;
  msDS-LockoutThreshold: 10;
  msDS-MaximumPasswordAge: 14:00:00:00;
  msDS-MinimumPasswordAge: 1:00:00:00;
  msDS-MinimumPasswordLength: 12;
  msDS-PasswordComplexityEnabled: TRUE;
  msDS-PasswordHistoryLength: 14;
  msDS-PasswordReversibleEncryptionEnabled: FALSE;
  msDS-PasswordSettingsPrecedence: 1;
  msDS-PSOAppliesTo: CN=nor,CN=Users,DC=gs,DC=com;
  name: biztest;
  objectCategory: CN=msDS-Password-Settings,CN=Schema,CN=Configuration,DC=gs,DC=com;
  objectClass (2): top; msDS-PasswordSettings;
  objectGUID: a542fe42-f9d8-44a2-9f2b-905a3dc83f48;
  uSNChanged: 32931;
  uSNCreated: 32927;
  ```
hashcat

- *hashcat is great for brute force
- It is also great for “masked brute force attacks”
  - Where you know some passwords already
  - …say, across many devices one can observe
  - Nothing like password predictability
    - Time to visit the Verizon store…
- Smart pattern matching FTW.
- Have fun finding min, max length and patterns
Static Passwords

- Say this…
- Static portions of passwords
- Portions guessable with public info (WiFi MAC)
- Static separator
- 6 numeric characters unknown…
Smart hashcat

• Choosing the method is helpful

• This one works…

  hashcat -m 1000 -a 3 unknown.hash "MobileEDME-?d?d?d?d?d"

  oclhashcat -m 1000 -w 3 -a 3 unknown.hash "MobileEDME-?d?d?d?d?d"

• Starts at 0 and goes to 999999

• This one works better

  hashcat -m 1000 -a 3 unknown.hash "MobileEDME-?d?d?d?d?d" --pw-min=17

  oclhashcat -m 1000 -w 3 -a 3 unknown.hash "MobileEDME-?d?d?d?d?d" --pw-min=17

• Starts at 000000 and goes to 999999
Conclusions

• I could talk about this for more than my allotted hour! This is a huge subject.

• This is a complex problem.

• This is my solution. I’d love to hear what works for you!

• Wordlists work great with and without GPU, so but get is to your advantage
Thanks!

larry@inguardians.com

@haxorthematrix