Password Cracking 201: Beyond the Basics

Fig. 1. — Un piano de chats. (D'après une gravure du dix-septième siècle.)
Overview

About me
About you (Types of password crackers)
A brief 101 – password cracking and psychology
Cracking constraints
Core attacks and common pitfalls
Bootstrap tips
Questions
Disclaimers

My interpretation of some community consensus
(mostly from hashcat & John the Ripper)

I am not a lawyer and this is not legal advice

Your organization or jurisdiction may be different

Warning: may irritate and/or bore the experts
About me

$DAYJOB in InfoSec in the financial sector

ISP scars

Independent security researcher

Password auditor and enthusiast

Enjoys long keyboard walks on the beach
About you
(types of password crackers)

- Infrequent audit/recovery (means to an end)
- Internal password auditors
- Password auditing consultants
- Pentesters
- Forensic investigators
- Researchers / academics
- Competition participants
- Enthusiasts
- Bounty crackers
- … and black-hat equivalents of some of these

*These can (and do) overlap*
Learn from other disciplines

* Follow *and interact with* the people outside of your discipline.

* Read their work - but more importantly, *study their goals.*

* Most have a common interest in *efficiency within time constraints* … but there are other common interests

* Some disciplines – pentesters, auditors - grok password selection patterns across many organizations.

**This is a competitive advantage** - often not published, so pay attention when it is
## Password activities: a taxonomy (draft)

<table>
<thead>
<tr>
<th>Activity</th>
<th>Work directly for entity that owns the hashes?</th>
<th>Clear hash ownership?</th>
<th>Authorized by user?</th>
<th>Authorized by hash owner?</th>
<th>Recurring against same targets over time?</th>
<th>Restricted time period?</th>
<th>Exposed to a wide variety of plains over time?</th>
<th>Exposed to a wide variety of hash types over time?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Infrequent recovery (means to an end)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>varies</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Internal corp password audit</td>
<td>Y</td>
<td>Y</td>
<td>N/A</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Password auditing (consultants)</td>
<td>N</td>
<td>Y</td>
<td>N/A</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Internal pentest (red team)</td>
<td>Y</td>
<td>Y</td>
<td>N/A</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>External pentest</td>
<td>N</td>
<td>Y</td>
<td>N/A</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Internal forensics (staff InfoSec)</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>?</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Competition: well-sourced, timed</td>
<td>Y</td>
<td>Y</td>
<td>N/A</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Competition: well-sourced, untimed</td>
<td>Y</td>
<td>Y</td>
<td>N/A</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Cracking for bounty (forums, etc.)</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>varies</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Enthusiast activities</td>
<td>N</td>
<td>?</td>
<td>?</td>
<td>N</td>
<td>varies</td>
<td>Y</td>
<td>Y</td>
<td>?</td>
</tr>
<tr>
<td>Bad guy activities</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>?</td>
<td>varies</td>
<td>varies</td>
<td>?</td>
</tr>
</tbody>
</table>
Prerequisites and assumptions

1. You want to crack passwords:
   - more consistently
   - with more results sooner
   - with better understanding
     … and to continually improve over time
## Prerequisites and assumptions

2. You have general password-cracking software ...

<table>
<thead>
<tr>
<th>FOSS</th>
<th>Free (as in beer), closed source</th>
<th>Commercial</th>
</tr>
</thead>
<tbody>
<tr>
<td>FOSS: Dave Grohl</td>
<td><strong>InsidePro</strong></td>
<td>Commercial: John the Ripper Pro <strong>L0phtcrack</strong></td>
</tr>
<tr>
<td></td>
<td><strong>MDXFind</strong></td>
<td><strong>PassCape</strong></td>
</tr>
<tr>
<td></td>
<td><strong>John the Ripper</strong></td>
<td><strong>Elcomsoft</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Passware</strong></td>
<td><strong>Hashstack</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Tableau</strong></td>
<td><strong>AccessData</strong></td>
</tr>
</tbody>
</table>

**FOSS** = Free (as in beer), closed source
Prerequisites and assumptions

3. You have some hashes (an offline attack):

$bba55a0d356d3af89354f6cd7c95b571$
$f3bbbd66a63d4bf1747940578ec3d0103530e21d$

SCRYPT:1024:1:1:MzM0NA==:LdHQwHICiz9N1HZQ9O1eOkxKCCgnW+Sv015WpZ2Su3A=
$1$57784108$grgZw95/LN9eIXxaETHv00 $apr1$08242617$skwy2o88OWLGaSUqdMGK.$

Administrator::24FCC5E8753EA7DCAD0C0724C88E0133:91D95FEFFC9963CC759FC4B338BDE1BB:::

36BYJUMId/4AM bba55a0d356d3af89354f6cd7c95b571 51c4d477273a064c
t5a8soeju4i8khkrcctu3cp4ddilac8i:.bsideslv.org:82205656:1
Prerequisites and assumptions

4. You are familiar with hashing for password “storage”:

- Not stored, but hashed (one way; resulting string is fixed length)
- Some are salted (good); others are unsalted (naive/outdated)
- Salts slow down cracking large lists, not individual hashes
- Some hashes are slow (bcrypt) – bad for the attacker
- Others are fast (unsalted MD5) – good for the attacker
- If you are saying “decrypted” or “dehashed” … No.

Good overview: https://security.stackexchange.com/questions/211#31846 (Pornin)
Prerequisites and assumptions

5. You know some basic password cracking attacks:

- Straight wordlist / dictionary
- Rules (“password” + “c $1” = “Password1”)
- Combinator (list (“za”, “b”, “c”) produces “bza”, “cza”, “zab” …)
- Masks (“?u?!?!?!?d?d?d?d?s” - matches “Fall2017!”)
- Hybrid dict+mask (lucky?d = lucky1, lucky2 …) or mask+dict

Masks and brute force guess in an intelligent order (Markov)
Prerequisites and assumptions

6. You know some basic password management psychology

• Human memory is finite

• We compensate for this with chunking
Because chess experts can “chunk”, they strongly outperform novices in board layout memorization tasks for valid chess games ...

Source: Stangor, Introduction to Psychology, Ch 8 (fair use)
... but when presented with a random board layout, experts perform no better

Source: Stangor, Introduction to Psychology, Ch 8 (fair use)
Experts memorize complex board layouts efficiently, because they can “chunk” the board into sets of familiar patterns.

Source: Gobet, 2011 in re Chase and Simon 1973 (fair use)
Chunking in password schemes

“My son’s name, all lower case, with his birth year at the end”

kevin1963

... is (very roughly) **five** “chunks” of information:

1) A specific person’s name
2) The case of the name
3) Their specific birth year
4) The length of the birth year (YY vs YYYY)
5) The method/order of appending (vs. “1963kevin”)

Cracking guided by known chunking strategies is *much* less expensive than bruteforce
Attacking the chunks

To crack many passwords using the ‘kevin1963’ scheme:

Wordlist: Facebook names (in frequency order):

http://downloads.skullsecurity.org/passwords/facebook-firstnames-withcount.txt.bz2

Attack: Appending digits to a wordlist (hashcat, on GPU):

$ ./hashcat -a 6 -m 0 test.md5 -j l facebook.txt ?d?d?d?d

For all of SkullSecurity’s Facebook name wordlists (2010):
http://www.skullsecurity.org/blog/?p=887
Cracking constraints

Source: https://www.flickr.com/photos/tanj/3838754485/ - CC-BY-2.0
Cracking constraints: Math/physics

The exponential nature of brute force

See also Rob’s blog post: [http://blog.erratasec.com/2012/08/common-misconceptions-of-password.html](http://blog.erratasec.com/2012/08/common-misconceptions-of-password.html)
And his talk at Passwords ‘13: [https://www.youtube.com/watch?v=dM0iZvR786Q](https://www.youtube.com/watch?v=dM0iZvR786Q)
Cracking constraints: Math/physics

The exponential nature of bruteforce – crosscheck

$ python -c 'print 95**2'
9025
$ python -c 'print 95**3'
857375
$ python -c 'print 95**4'
81450625
$ python -c 'print 95**5'
7737809375
$ python -c 'print 95**6'
735091890625
$ python -c 'print 95**7'
69833729609375
$ python -c 'print 95**8'
6634204312890625
$ python -c 'print 95**9'
630249409724609375
$ python -c 'print 95**10'
59873693923837890625

Assume that we have no idea what characters might be used, (other than that they are printable ASCII / 95 characters)

<------------------------ This is 5.9x10^{19}
Cracking constraints: Math/physics

The exponential nature of bruteforce - crosscheck

# Benchmark for MD5 (mode 0), 6x GTX 1080, no overclock

$ hashcat -b -m 0 --quiet

Hashtype: MD5

Speed.Dev.#1.....: 25434.5 MH/s (52.74ms)
Speed.Dev.#2.....: 24610.4 MH/s (54.48ms)
Speed.Dev.#3.....: 24968.4 MH/s (53.72ms)
Speed.Dev.#4.....: 24923.4 MH/s (53.82ms)
Speed.Dev.#5.....: 24015.1 MH/s (53.70ms)
Speed.Dev.#6.....: 25002.8 MH/s (53.64ms)
Speed.Dev.#*.....: 149.0 GH/s
Cracking constraints: Math/physics

The exponential nature of bruteforce - crosscheck

$ hashcat -b -m 0 --quiet --machine-readable
1:0:1746:4513:53.02:25294033130
2:0:1733:4513:54.12:24427650923
3:0:1771:4513:53.67:24987941094
4:0:1746:4513:53.81:24925294904
5:0:1721:4513:53.10:25257382009
6:0:1784:4513:53.47:24999576814 = 150302475476 hashes/sec

# Hours to exhaust ?a at length 8
$ python -c 'print (6634204312890625/150302475476)/60/60'
12
# Days to exhaust ?a at length 9
$ python -c 'print (630249409724609375/150302475476)/60/60/24'
48
# Years to exhaust ?a at length 10
$ python -c 'print (59873693923837890625/150302475476)/60/60/24/365'
12
Cracking constraints: Math/physics

Jeremi M Gosney @jmgosney · 28 Mar 2016
1/ I've encountered several people lately who use password managers & are generating random passwords 20+ chars long (some as long as 200!)

2/ This is WAY OVERKILL for even raw MD5, let alone anything stronger, so no actual threat modeling is being done wrt password security.

3:45 PM - 28 Mar 2016
Cracking constraints: Math/physics

How long should your random password be?

\[
\text{ceil} ( \log_C (H \times Y \times 31556926 \text{ [sec/year]} ) )
\]

\[
\text{C} = \text{charset count} \\
\text{H} = \text{adversary hashrate} \\
\text{Y} = \text{years to crack}
\]

Source rant: https://twitter.com/jmgosney/status/714599158229786625
Cracking constraints: Math/physics

How long should your random password be?

```
ceil( logC (H * Y * 31556926 [sec/year]) )
```

(Assuming: $C = \text{alphanum}$ (62 chars), $H = 100\text{TH/s}$, $Y = 100$ years)

... the non-Moore’s-law-aware answer is 14 characters:

```
IoI80asu93H6oN
XxCp8KekKhR1A6
vnhkx7qMN0pGO
TKheK2Mzkw63IP
S
```

(and per the anrief.net calculator, w/Moore’s Law, it would be 16 years 2 months)
how many bushels in a ton

Assuming US bushels for "bushels" | Use UK bushels instead

Input interpretation:

convert 1 displacement ton to US bushels

Result:

28.12 bu (US bushels)

Additional conversions:

112.5 pk (US pecks)
35 ft³ (cubic feet)
Cracking constraints: Math/physics

How long should your random pass**phrase** be?

\[
\text{ceil}\left( \log C \left( H \times Y \times 31556926 [\text{sec/year}] \right) \right)
\]
(Assuming: \( C = 17K \text{ word dictionary}, H = 0.1\text{TH/s}, Y = 1 \text{ year} \))

\( = 5 \text{ words}, \text{ regardless of other complexity} \)

```
cain mystery ahoy discourse serpent
stares perkiness begs fleshy form
eternal belonged sane allowing disc
```

Adjust the parameters based on your threat model
Add trivial complexity for any sites that require it
Cracking constraints: Math/physics

Just in case you’re also worried about time travelers or inter-dimensional aliens:

*Landauer's principle*: theoretical minimum energy of **one bit flip** of information

The mass-energy of the sun = $2^{225.2}$ operations (Pornin)

225 bits = **35-character** (95 printable ASCII) passwords

*(Hat tip: James Wu / @analogist_net)*
Cracking constraints: The target

- Speed of the hash
  - You can be lazy experimental with faster hashes
  - Slow hashes = harder w/o knowledge of target
- Size of the target
  - 61M SHA1 from a public leak? Try it all!
  - A single WPA2? Target knowledge needed
- Your knowledge of the target
  - Some good OSINT tools - but there is a limit
  - Demographics and sophistication of the target
- Requirements (usually detectable)
Cracking constraints: Capabilities (1)

• Software availability and algorithm support
• Cracking technique awareness
• Your available attention / energy / motivation
Cracking constraints: Capabilities (2)

- Raw \textit{cracking-specific} compute capability
- Your ability to turn target knowledge into inputs
- Quality of inputs (wordlists, rules, masks, etc.)
- Initial attack “spin-up” / prep latency (script this!)
Platform considerations

Source: Flickr user kentwang - CC BY-SA 2.0
OS / hardware / admin hints

- Use the native OS (not a VM) if you can
- Kali’s OpenCL and Intel OpenCL are problematic
- Use the latest stable video drivers, not OS stock
- Use the john “jumbo” & latest hashcat releases …
- … but also keep betas and latest GitHub available
- … and make it easy to quickly run any of them
- For NVIDIA GPUs, use reference/Founders models
  - Very hardy, built to exact NVIDIA specs
- Consider locking fan speed (80%? 100%?) based on your thermal risk models
Use the FOSS, Luke ...

- No arbitrary caps on target size
- Scriptable
- Multi-OS, multi-platform
- Cross-pollination among projects
- Opportunity to directly contribute
- stdin … and stdout
... but embrace the Dark Side ;)

- Ease of use
  - GUI
  - Canned wordlists and rules
  - Often good attack-plan management
  - Built-in distributed processing
- Some suites have free trials
- Some free front-ends (Hashtopussy, Hashview) have similar features
¿Por que no los dos?

- Use both FOSS and commercial – whichever are the right tools for your use cases
- It’s Free to add FOSS. And some non-free are very affordable
- Play with all of them. More exposure = more perspective
- Press your vendors for more interoperability (stdin/stdout support), custom rules, etc.
Input management

WE'RE GONNA NEED A BIGGER HARD DRIVE.

Wordlist management (1)

- Focus on *human-generated strings*
- Quality over quantity
- Single best public source for founds: hashes.org
  - Based wholly on public leaks (AFAICT)
  - Lists both found and remaining hashes, per leak
  - Has an API for submission of new founds
- For slow hashes, bulk harvest from context (CewL, Maltego, etc.)
Wordlist management (2)

- All human-generated strings are fair game
- Wikipedia/Wikia (long tail of fictional words/phrases)
- Domain names and hostnames (Rapid7 Project Sonar DNS ANY)
- Usernames / email addresses / given/family names …
- Street addresses, lyrics, movie subtitles, Project Gutenberg …
- In other words – things that people remember that you can harvest in bulk
- These need to be normalized and deduped
Wordlist management (3)

- Deduplicate *by type* (superset of **all** email addresses, etc.)
  - `rli` and `rli2` from `hashcat-utils` are the go-to tools
- **Build a superset, but retain per-source dictionaries**
- Watch RuraPenthe’s talk on wordlist grooming:
  https://www.youtube.com/watch?v=IGbceBOVYI
- Some people import into databases … mixed consensus
  - Probably a 301-level activity :)
  - I prefer plain text files (the trade-off is more disk space)
Wordlist conversion

• If not in UTF-8, convert to UTF-8 with `iconv`
• … but keep the originals
• HTML escapes? Use RuraPenthe’s `rurasort`
Baseword extraction

- **Stemming** - deriving original base words from plains
  
  “Ground1234!” → “ground”

- More difficult than it looks – aspell, neural networks
- RuraPenthe’s *rurasort* – lots of different options
Mask management

• Grab other people’s masks, like KoreLogic’s
• Make your own from your previous and current cracks - use PACK’s statsgen and maskgen
  • Clients who don’t let you keep plains might let you keep stats
• Keep masks sorted by frequency
• If you get a new list of masks, remove ones that are already exhausted.
  • Hashcat’s .log files note when they are exhausted (and you can harvest / diff the list)
  • Status code definitions are in include/types.h, STATUS_*
Rules management

- Grabbing rules from others
  - Watch: hashcat forums, bartavelle, EvilMog
  - Not all rules are compatible with all suites – see hashcat wiki
  - Some rules are not GPU-or CPU-ready and will be rejected
    - Strip CPU or GPU rules with hashcat-utils’ cleanup-rules
- Making your own rules
  - Generate from a given dict and plains list, using bartavelle’s rulesfinder (GitHub)
- Deduplicating rules is tricky
  - space vs non-space, and rules that cancel each other out
  - 0xbsec’s duprule looks promising – still in development
Attack methods
Cheat

Soldier of FORTRAN
@mainframed767

Them: I have a twelve GPU password cracking rig
Me: <copy/paste hash in to google search>

7:14 AM - 7 Jul 2017

267 Retweets 665 Likes
Cheat ...

- Useful only for unsalted hashes
- Try Google AND Bing, etc (different results sometimes)
- Some specific sites (no specific recommendation - YMMV)
  
  hashcrack.com  hash-killer.com/hashdb  
  md5center.com  md5-database.xyz  
  md5decryption.com  md5decrypt.net  md5hashing.net  
  md5.my-addr.com  www.nitrxgen.net/md5db

  ... etc.

- Use sites’ own search features – may not be fully spidered
- Sites come and go – scout for new ones
  - Use a web search for a semi-common hash
... but cheat responsibly

- Consider sensitivity of hashes before submitting – they are stored, analyzed, and being cracking (that is what the sites are really for)
- Google/Bing searches may be less likely to expose hashes to third parties (but weigh sensitivity anyway!)
- Third-party pentesters/auditors – think hard
Custom Markov

- Stock Markov sets are often derived from RockYou (100%!)
- This is fine for first passes, but your target may be different
- Use your cracks and hcstatgen to build a custom Markov set:
  
  $ .\hcstatgen.bin out.hcstat < infile

  … and then use it for bruteforce/mask-based attacks

Example: custom Markov based on LinkedIn founds automatically hits strings with “link” “LI” “linked”, etc. more often and earlier in the attack
PRINCE (1)

- PRINCE mode is supported by John natively, and by hashcat as a standalone binary (which you can pipe directly to hashcat)

  In essence, it’s a smart combinator attack that will run forever, combining two words, three words, etc. from a given wordlist

- Using **CPU only** and only RockYou as input, early PRINCE testing cracked 72% of LinkedIn in **24 hours** – *completely automated.*

- Introductory post from the hashcat forums

- Technical details are in slides from atom’s 2014 talk
PRINCE (2)

• I personally credit PRINCE for a lot of what I have learned since it was released … because it showed me what I didn’t know yet.

• I run PRINCE in parallel with other runs – on CPU, on a separate box, etc.

• I also run it as an ‘attack of last resort’ to fall back to when other attacks finish and I don’t have something else queued up.

• Use PRINCE. Seriously.
Broad attack guidance

• Don’t be afraid to feed things into other things

• In epixoip’s Circle City Con class, he described “PRINCEPTION” - feeding PRINCE into itself!

• Stdin/stdout and named pipes are key for this exploration

• Use the raw numbers to measure performance – was it worth it?

• Be ready with bash/Python/perl/PowerShell to try things that don’t have a tool yet (that you know of)

• Back up your results(lists, notes, founds, scripts) to reliable media.
Accurate speed estimation

$ echo -n bsideslv17 | md5sum | awk '{print $1}' >bslv.hash

$ cat bslv.hash
A3c67ba47cfb67c42840acc21a77211f

$ hashcat --speed-only -b -m 0 --quiet --machine-readable
1:0:-1:-1:53.31:114469177

$ hashcat --speed-only -m 0 -a 3 bslv.hash \ 
1:20133346

Actual speed for this attack is ~17.6% of benchmark speed
Use actual speed to create your attack plans
Measuring attack efficiency

• “Wall-clock” time efficiency is relative
• Crack position (number of guesses) is absolute
# Measuring attack efficiency

<table>
<thead>
<tr>
<th>#</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>hash[:salt]</td>
</tr>
<tr>
<td>2</td>
<td>plain</td>
</tr>
<tr>
<td>3</td>
<td>hash[:salt]:plain</td>
</tr>
<tr>
<td>4</td>
<td>hex_plain</td>
</tr>
<tr>
<td>5</td>
<td>hash[:salt]:hex_plain</td>
</tr>
<tr>
<td>6</td>
<td>plain:hex_plain</td>
</tr>
<tr>
<td>7</td>
<td>hash[:salt]:plain:hex_plain</td>
</tr>
<tr>
<td>8</td>
<td>crackpos</td>
</tr>
<tr>
<td>9</td>
<td>hash[:salt]:crack_pos</td>
</tr>
<tr>
<td>10</td>
<td>plain:crack_pos</td>
</tr>
<tr>
<td>11</td>
<td>hash[:salt]:plain:crack_pos</td>
</tr>
<tr>
<td>12</td>
<td>hex_plain:crack_pos</td>
</tr>
<tr>
<td>13</td>
<td>hash[:salt]:hex_plain:crack_pos</td>
</tr>
<tr>
<td>14</td>
<td>plain:hex_plain:crack_pos</td>
</tr>
<tr>
<td>15</td>
<td>hash[:salt]:plain:hex_plain:crack_pos</td>
</tr>
</tbody>
</table>
Measuring attack efficiency

--outfile test1.out --outfile-format 11 \
--quiet --potfile-path=/dev/null

--outfile test2.out --outfile-format 11 \
--quiet --potfile-path=/dev/null

$ cat test1.out test2.out
7e7da6a03ac3b80bad3f338ffffa621d5:hash234:12378814668
7e7da6a03ac3b80bad3f338ffffa621d5:hash234:137798556

The second attack is 100 times faster than the first - regardless of the underlying hardware speed
Measuring attack efficiency

Once you have the numbers, plot them (gnuplot, Excel, etc.)

Source: Matt Weir (lakiw) PRINCE analysis
Analyzing cracks and founds

- The PACK toolkit is an essential set of tools
- Build statistics on letter frequency and mix types
- Build lists of masks based on time constraints
- Other fantastic features – definitely a minimum
Working with non-ASCII

• If your plains are in one character encoding and your targets are in another, this won’t work
• Recommendation: convert everything you get to UTF-8
• If your suite doesn’t include encoding conversion tools, use `iconv` to convert to the target encoding
  • Looks like John has good support (not checked)
  • Hashcat recently added encoding-from / encoding-to flags
• Hashcat can do multibyte bruteforce with a workaround (RuraPenthe’s)- works, but tricky; his good writeup [here](https://security.stackexchange.com/a/154958)
  • Full example using PACK and German characters: https://security.stackexchange.com/a/154958
Universal attack-plan principles

• With inputs, quality and frequency order matter more than quantity
• Start simple – crack the easy stuff quickly
• If you can cap attack runtime, use it (diminishing Markov returns)
• While that is running, analyze plains for untapped patterns
• Arrange attacks in order by efficiency for the time allotted
• Record all attacks, results, and performance (science!)
• Script these and repeat the good attacks on all new founds
• Avoid duplicated masks/wordlists - when it makes sense
• Append a fallback attack to each attack plan
Understandable naive approaches

“Where can I get more wordlists? I have 120GB so far.”
Understandable naive approaches

Understandable naive approaches

3ce8b30b8e25ea5d9a83d4a073d6ddf8

“Ah, this must be MD5 - because it’s 32-character hex.”
Understandable naive approaches

“I’m trying to get hashcat to work in Kali, and ...”

The-Distribution-Which-Does-Not-Handle-OpenCL-Well (Kali) Linux ...

https://hashcat.net › hashcat Forum › Misc › User Contributions ▼

Nov 30, 2016 - 4 posts - 2 authors
Installed The-Distribution-Which-Does-Not-Handle-OpenCL-Well (Kali) Linux on your laptop, but hashcat won't run? You probably don't have ...

hashcat v3.00 + The-Distribution-Which-Does-Not ... 5 posts  Jun 29, 2016
upgrade hashcat 0.46 only in The-Distribution-Which ... 6 posts  Dec 2, 2013

More results from hashcat.net
Learn from history

Great presentation on the history of password security by Solar Designer (@solardiz) and Simon Marechal (@bartavelle):

Other bootstrapping resources

Hashcat:
wiki, FAQ, forums, IRC (#hashcat on Freenode), GitHub

John:
john-users mailing list, docs, GitHub for jumbo version

General:
Hashkiller.co.uk and InsidePro forums
Passwords & Ground1234! conference talks on YouTube
PasswordResearch.com (Bruce K. Marshall)
Cultural hints

• Like many geek areas, password crackers can be rough on the noobs sometimes
• Information sharing beyond a certain point is limited
• Like BSidesLV, related projects are usually “do-oocracies”
• Show up, roll up your sleeves, don’t ask the same question twice, and if someone sends you a link in answer to your question, you’d better read it.
• The shared common interest makes for a great community
In other words ...

*Do not trace the footsteps of the wise; seek what they sought.*

- Bashō (poet, *paraphrasing* Kūkai (!))

*Imitate the intent, not the brush strokes.*

- Kūkai (on calligraphy)
Thanks

AlecMuffett             curlyboi                grempe                  Matlink                 sc00bz
Apingis                 d22                     hashcate ;)             mckusick                sediti0n
atom                     d3ad0ne                  iphelix                  minga                   solardiz
atoponce                 DaKahuna                 jfoug                    neheb                   soxor2212
bartavelle               DidierStevens            Jumpforce               NETMUX                   stanev
bill_e_ghote             digininja                 ken                     nitruxgen                stratoma0ro
Bitweasil                doc2n                     kholia                   _NSAKEY                  Szul
blandyuk                 DoZ10                     kmalvoni                 nuartvision              TOXIC
bratemorse               dropdeadfu                Kryczek                  NullMode                 thorsheim
cantcomputer             empty_knapsack            kwzh                    philsm0d                 undeath
Chick3nman                epixoip                   lakiw                    PwdRsCh                  unix-ninja
claudioandré              ErrataRob                 lars-lyosha              r4d1x                    veorg
coolbry95                EvilMog                    m33x                     richrumble               waffle
CormacHerley             Fist0urs                  m3g9tr0n                 Rjmendez                  winxp5421
countuponsec             floyd                     m8urnett                 Rolf                     Xanadrel
cowboym                   frank-ditrich              magnum                   rurapenthe               xmisery
cperciva                  g0tmilk                    mangix                   ryan-c                   ZerBea
CrackTheHash              GiftsUngiven              gm4tr1x                   s3in!c                    zorinaq

Contact: royce@techsolvency.com | @TychoTithonus

Slides, errata, references:  www.techsolvency.com/talks