Practical Password Cracking II

The *Last* 50 Million

Acknowledgments

None of this would be possible without

- Hashcat itself
- John the Ripper
- Impacket
- Wordlists (Seclists, probabilistic passwords, Crackstation, hashes.org, etc.)
- D3.js and bokeh (graphing)

https://uncommoncriteria.org/ppc.html pt1

Bio

- I've built/improved multi-user cracking systems at two UK pentest companies
  - PTP; built from scratch
  - NCC; >20 cards, >4kW, >400 users. Improved crack rate (+50% relative), added types
- Cracking other people's passwords on and off since 2003, red and blue team.
- Limited budget.

Why Bother?

- Baseline approach – rockyou.txt & nsav2dive.rule
- Actually, not bad, but we can do better
  - 40+ common hash types to remember
  - Format conversions
  - Pen tests have hard time limits
- “Improving the Security of Your Site by Breaking Into it”
- “31.3. To maximise test coverage the following types of automatic test equipment shall be used where relevant: **password cracking tools**; “ – CHECK
- Non-invasive during red team tests.

Basic Concepts

- Defender uses one-way function to obscure original password
- Recovery is based on repeated guesses
- Hashcat is great, but it’s like trying to find small flecks of gold in a lump of rock
- Except the lump of rock has 8+ dimensions
- The vein (of gold) analogy kind of works
- No right answers, but good library of techniques available

Outline

- Bad Hashes (fast, unsalted hashes)
- Good Hashes (slow, salted hashes)
- Hashcrack script/tools - http://github.com/blacktraffic/hashcrack
- Offensive / defensive uses
- Attacking the last ~53 million out of 553 mil
- “Crack me if you can” example
- Attacking unknown hashes

obDisclaimer

Please do not blow up your laptop.

NEVER USE --hwmon-disable

If you must use a laptop,
ensure it’s adequately cooled

Bad Hashes – For our purposes at least

- MD5 hash: “secret” -> 5ebe2294ecd0e0f08eab7690d2a6ee69
- MD5 is a hash function mapping anything to a 128-bit value.
- Very quick – **35 billion** guesses / sec.
- Can build lookup table (hash, password)
- (Ignore other bad properties of MD5)
- NTLM is basically widechar MD4. LM is worse.
- **Blue team is often stuck with NTLM**

Good Hashes

- Blowfish, or argon2, or PBKDF2 etc.

```
echo password_hash("test",
PASSWORD_DEFAULT);
$2y$10$.vGA1O9wmRjrwAVXD98HNOgsNpD
czlqm3Jq7KnEd1rVAGv3Fykk1a
```

- Salt, cost parameter, slooooooooooooow – 20 thousand guesses / sec with cost 5
- Random salt prevents lookup table creation

Hashes: “how slow is slow enough?”

Ensure you’re using salted passwords properly (x N users)

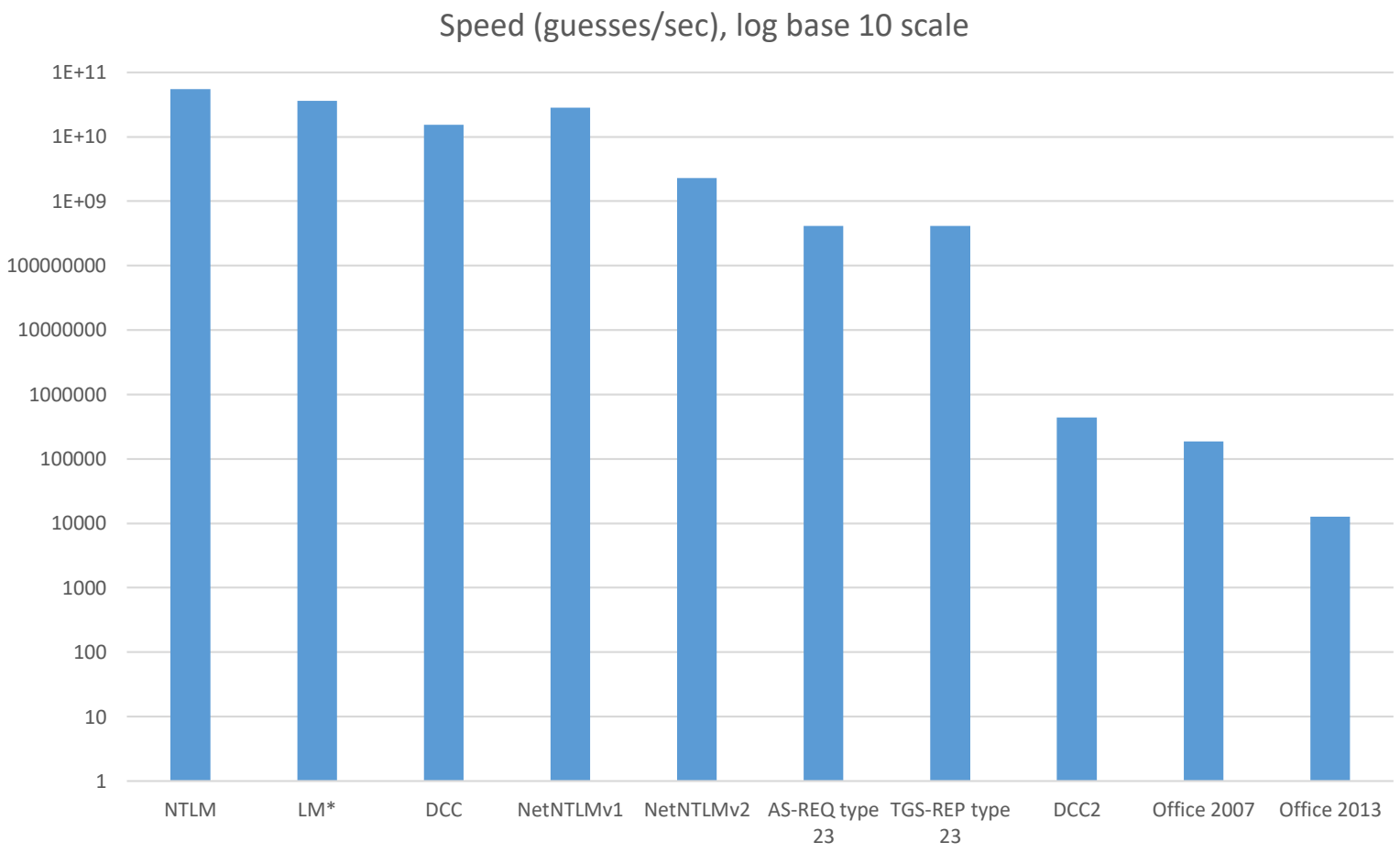
>500ms delay is annoying.

Max out cost factor, so each try is circa 200 ms.

Block weak passwords so median strength improves

One occasion where expiring passwords make sense…

Guess Speed (Ideal)



Hashcrack Script

- Guesses hash type (regexp mappings)
- Tries a decode if you need one (mostly via JtR)
- Tries some reasonable dictionaries, rules
- Configurable – mapping files are data
- Can override for specific requirements
- Ground-up rewrite in python3 of the perl version I built at PTP
- AGPL so no-one can nick it

Hashcrack Script – Constraints & Goals

- Don’t make the user remember ‘m’ number.
  - E.g. 7500 krb5 as-req etype 23
- Keep the GPU busy
  - Fast hashes need more than just dictionary
  - Use appropriate sized dict + rules
- Efficiency
  - Go for greatest **density** first – ie. common passwords
  - Top N Million passwords (probabilistic passwords)
- Automate attacks, e.g.
  - re-use pwdump LM cracks for NTLM
  - Unpack files for the user, e.g. Responder.db, JKS, docx

Main Attack Types

-a0 dict + rules. Rules often needed for throughput

-a0 Top258Million.txt –r rules/l33tnsa.rule

-a3 incremental with given mask, e.g. ?u?l?l?l?l?d ([A-Z][a-z]{4}\d)

or with hcmask file: ?u?l?d,?l?d,?l?d?s,?1?2?2?2?2?3

-a6, -a7 left and right mask –a6 found.txt ?a?a?a -i

-a1 leftdict rightdict “cross-product” *aka “information supercollider”*

Other generator writing to stdin, e.g. perl scripts/leetify.pl | hashcat

Main Attack Types - Rules

swap all chars o to 0 - leetify (word -> w0rd)
so0
append ‘!’
$!
Toggle case of first letter
T0
Enclose in quotes (prepend “, append “)
^”$”
prepend 123
^3^2^1
https://hashcat.net/wiki/doku.php?id=rule_based_attack

Main Attack Types

-a1

Left dict:	your	my	the
Right dict:	cat	dog	moose
	your	**my**	**the**
cat	yourcat	mycat	thecat
dog	yourdog	mydog	thedog
moose	yourmoose	mymoose	themoose

Main Attack Types – Mask Files

```
def talktalkmask(mask,caps,digits):

    if (caps>=1):
        talktalkmask(mask+'?1',caps-1,digits)

    if (digits>=1):
        talktalkmask(mask+'?2',caps,digits-1)

    if caps==0 and digits==0:
        print("ABCDEFGHIJKLMNOPQRSTUVWXYZ,0123456789,"+mask)


talktalkmask('',5,3)
```

Example mappings – looks horrendous, but …

```
# regexp ! m type ! name  - regmap.cfg
(^|:)0x0200!1731!MSSQL2012+
\$gpg\$!gpg-opencl!GPG
\$zip2\$!13600!ZIP
(^|:)md5:1000:!11900!pbkdf2-hmac-md5

#hashtype,dict,rules,inc,name – quickmap.cfg
0:bigdict,bigrules,7,md5
7200:smalldict,smallrules,0,grub
```

Sketch of an Attack Workflow – fast hashes

Fast hash like pwdump (uid:LM:NTLM)

```
python3 hashcrack.py -i pwdump.txt --nuke
```

1. Crack LM first using incremental up to 7 chars
2. Use this as crib for NTLM (all case permutations)
3. Incremental up to 8 for NTLM
4. Run found.txt with various options, if present
   1. -a6 -i ?a?a (found.txt with all 1,2 char suffixes)
   2. -a1 last3.txt (with common 3 char suffixes)
5. Decent sized dict with l33tpasspro rules

Sketch of an Attack Workflow – slow hash

Slow hash like bcrypt:

```
python3 hashcrack.py -i tests/bcrypt.txt
```

0. (manual) download & compile – 40% quicker than 5.1.0
1. Small crib file with company name etc.
    1. Plaintext mode + leet rules -> variants on company name
    2. Run this through more rules
2. Previously cracked passwords with no rules
3. Medium dictionary, no rules

Red Team - Possibilities

Steal hashes via :

- Responder
- Kerberoasting
- Internal Monologue
- Physical theft of device/disk/vmdk etc.
- smb:// URIs and similar leaks
- Dumping NTDS or local registry hives (‘winaudit’)
- Cisco/Juniper/etc configs
- Office docs, zip files, Bitlocker volumes …

Red Team – Contrived Web Example (SQLi)

$ **python sqlmap.py -u "http://192.168.31.130/owaspbricks/content-2/index.php?user=harry" --dump**

[14:15:50] [INFO] the back-end DBMS is MySQL

web server operating system: Linux Ubuntu 10.04 (Lucid Lynx)

web application technology: PHP 5.3.2, Apache 2.2.14

back-end DBMS: MySQL >= 5.0

harry@getmantra.com | 5f4dcc3b5aa765d61d8327deb882cf99

$ hashcat64.exe -m 0 5f4dcc3b5aa765d61d8327deb882cf99 dict\breachcompilation.txt –r InsidePro-PasswordsPro.rule -O

5f4dcc3b5aa765d61d8327deb882cf99:password

Pentest/Blue Teams – Audit Domain Passwords

C:\> **ntdsutil**

ntdsutil: activate instance ntds

ntdsutil: ifm

ifm: create full c:\temp\ifm

[quit <ENTER> quit]

$ **python impacket/examples/secretsdump.py -system SYSTEM -ntds ntds.dit LOCAL**

(or whatever form your main password db is in)

Red Team – Use Cases

python3 hashcrack.py -i ifm.zip -t ifm
[invokes impacket/secretsdump.py to decode]

python3 hashcrack.py -i Responder.db
[uses sqlite3 library to query the db file]

python3 hashcrack.py -i salaries.xlsx
[runs jtr script office2john.py]

Red Team – Use Cases

$ **python3 hashcrack.py -i tests/test-abc.docx -d hashcat.txt**

Reading file: /root/hashcrack/tests/test-abc.docx

RUN: python2 office2john.py tests/test-abc.docx > tests/test-abc.docx.tmp2

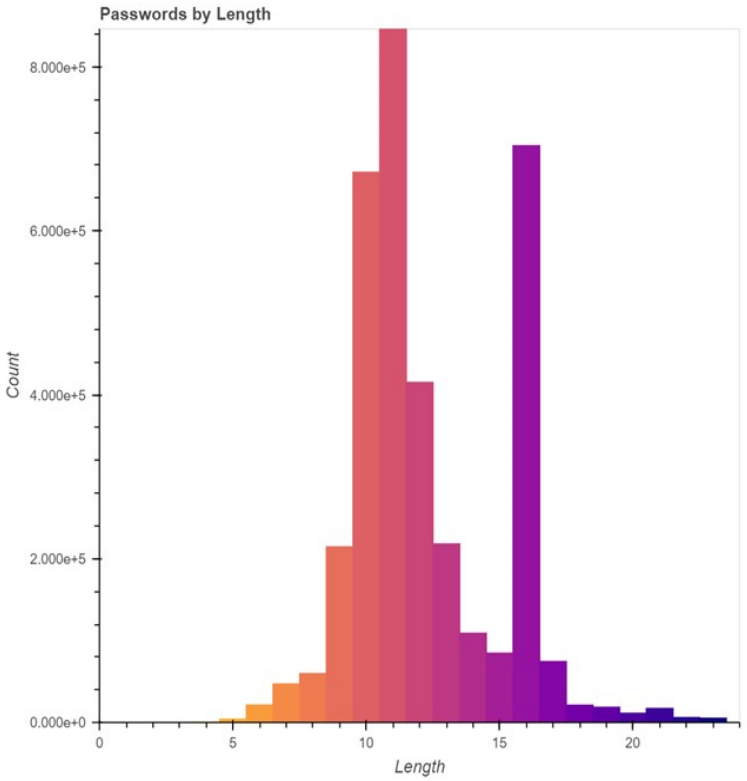
RUN: ./hashcat64.bin -a0 -m 9600 tests/test-abc.docx.tmp2 hashcat.txt -r rules/l33t64.rule --loopback -O -w4

hashcat (v5.1.0) starting...

$office$*2013*100000*256*16*xxxa7f38d98d12:abc

Audit Stuff – Pentest or Blue Team

- Export domain passwords
- Crack the hashes
- Uses LM cracks with case permutations, if present
- Draw some graphs; time to crack, password length
- Update policy, make users change password



Blue Team – approaches

- Stop users from choosing
  - Really weak passwords
  - Widely compromised passwords
  - Passwords based on dictionary words / keyboard
- Fix process; no “Welcome123!” default please
- Rate-limit password guesses.
- Back off to CAPTCHA for potential brute-force or credential stuffing attacks
- Crack your own password hashes

Blue Team – Prevention, Detection

- https://github.com/ryanries/PassFiltEx - password filter module for better policies
- https://haveibeenpwned.com/API/v2 see Searching by range for checking for compromise
- Cheap and cheerful list of sorted NTLM and ‘sgrep’ – homemade breach database.
- Or just go to 2FA already

Blue Team – Audit

- Dump hashes from DC (or whatever)
- Extract with secretsdump
- Crack with hashcat
- Bump policy up if need be.
- Notify users and tick “user needs to change pw”
- Add results to the **front** of your cracking dictionary

Blue Team – breach “database”

$ echo "isthisbad" | python breach.py

$ echo “0fbc4 …4312" | python breach-ntlm.py

This is a 42Gb file **compressed**, so quite big.

Not normal ‘sgrep’, this one, aka “sorted grep”
https://sourceforge.net/projects/sgrep/

Mega.co.nz link :

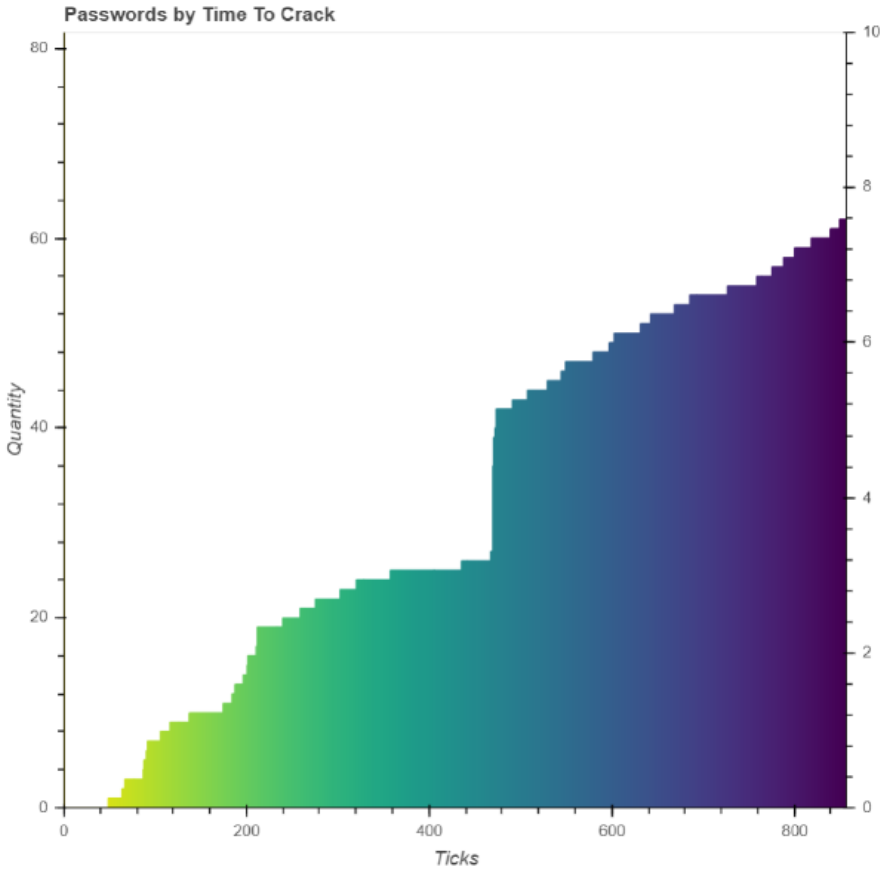
https://mega.nz/#F!e7phQQJJ!E4wAhlA-aB22a5lObJAcmg

Measuring quality (blue) / attack efficiency (red)

Can derive this from hashcat status output

Blue: Minimise the area under this curve.

Red: **maximise** it



hashcat --status >> out

python graph-by-quality.pl hashcat-status-output

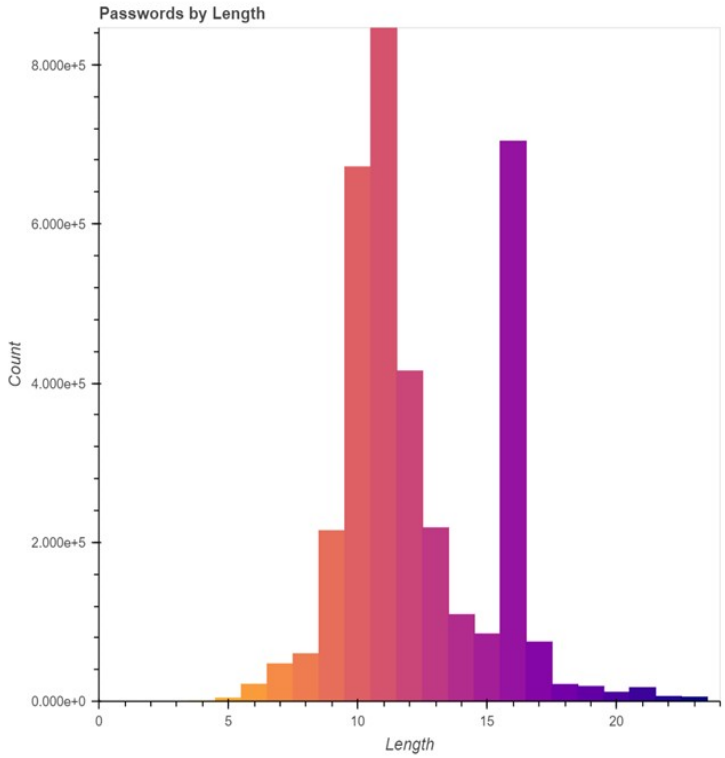
Why not length or entropy as a measure ?

“Password1Password1”

“Qwertyuiop[]123”

Long, decent entropy.

NOT good passwords.

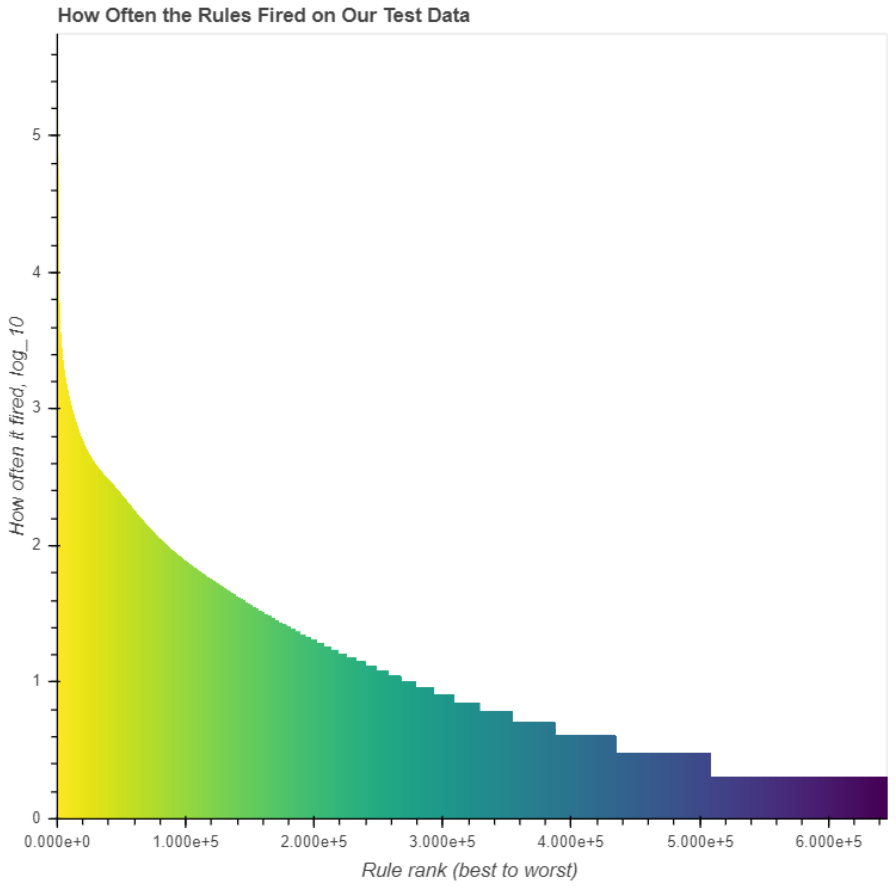


Aside: Rule Evaluation

Debug output shows
rules firing

count them on
“representative” data

This is HIBP / Top258Mil –
See rules/bestN.rule
Generating candidates rules still hard.



Cracking 500 Million Passwords – Getting The Data

https://haveibeenpwned.com/Passwords

wget https://downloads.pwnedpasswords.com/passwords/pwned-passwords-ntlm-ordered-by-hash-v4.7z

7z x pwned-passwords-ntlm-ordered-by-hash-v4.7z

cat pwned-passwords-ntlm-ordered-by-hash-v4.txt | cut -f 1 -d':' > hibp.txt

Tweak the hashcrack.py code with crackopts=" –O **--bitmap-max=28**"

Cracking 500 Million Passwords – Split / Crack / Merge

$ split -l 60000000 -d hibp.txt # max on my card is 60 mil

#!/bin/bash

python3 hashcrack.py -i x00 -t ntlm --pot hibp1.pot --remove >/dev/null

… repeat …

cat x?? > phase-1.txt

rm –f x??

Need >/dev/null for the first couple of phases, too much output

Attack Types and Flags

--loopback –O –w4

-a1 allows cross-product. combinator and combinator3 do similar. Or write your own script:

```
$ python leetify.py words.txt |
./hashcat64.bin -m 1000 hashes.txt -r
rules/best64.rule
```

OMEN & PRINCE & PCFG as candidate generators as well

Main Attack Types

hashcrack.py -i hashfile.txt [-d dict] [-r rules]

hashcrack.py -i hashfile.txt --mask hcmaskfile or literal

hashcrack.py -i hashfile.txt -d dict –lmask <mask>

hashcrack.py -i hashfile.txt -d dict –rmask <mask>

hashcrack.py -i hashfile.txt -d leftdict -e rightdict

Cracking 500 Million Passwords – The easy bit

```
$ wc -l stg*
  227 789 031 stg1-thorough-mask.pot
  164 614 913 stg2-top258m-hybrid.pot
  109 223 798 stg3-breach-hybrid.pot
  501 627 742 total   [ ~48 hours ]

$ wc -l whatremains
53 652 620 whatremains
```

Cracking The Last 50 Million: Tools

PRINCE – combines elements of wordlist to make longer ones.

Then whack it through STDIN to hc, use some rules.

```
$ ./princeprocessor/pp64.bin --pw-min=12 --case-permute dict.txt | ./hashcat64.bin -m 1000 -O -w4 hibp/wr –r /root/hashcrack/rules/best84581.rule
```

2c74f72…007910f9d:Flailability12

Cracking The Last 50 Million

Purple rain attack; lots of random rules (hashcrack.py –R), PRINCE processor

“Supercollider” : -a1 Top304-Thou Top304-Thou etc. ; similar effect to PRINCE / combinator

“Ringing” – feed all the cracks so far back into the process, with whatever rules / transforms we already used.

“Stemming”

```
$ cat hibp-all.txt | perl ../get-passwords-from-pot.pl | sed "s/[^[:alpha:]-]//g" | sort -u > hibp-alpha.txt
```

Cracking The Last 50 Million

Statistical models: pcfg and OMEN

Build statistical model with OMEN

$ ./createNG --iPwdList=../hibp-all.txt

Invoke generator, endless, pipe out mode

$ ./enumNG -e -p | /root/hashcrack/hashcat-5.1.0/hashcat64.bin -m 1000 hashes.txt -r rules/best84581.rule

Cracking The Last 50 Million

Data quality issues. Cull out some email addresses :

-a1 Top2Billion.txt domains.txt (e.g. “@yahoo.com”)

Line endings!

Fudge.rule: $\x0d
Use with –r normal.rule –r fudge.rule

829cc16:$HEX[76666b7..3435**0d**]:vfksirf12345**x0d**
(SINGLE BEST ATTACK ON LAST 50 MILLION!)

Cracking The Last 50 Million

Leetification script (now even leetier) - ./hashcat.py -3

This does one char at a time, not all as hc rules do:

e.g. “S3verusSn@p3”

python3 scripts/leetify.py dict.txt | ./hashcat64.bin -m 1000 hashes.txt –r rules/best84581.rule

:se3 rule -> “SeverusSn@pe” or “S3v3rusSn@p3”, but nothing in between.

Cracking The Last 50 Million

Identify patterns in input data, also password policy

Implement a search using rules, custom generator, masks, or –a1 with custom wordlists

Crib file with company name, location, etc. (--crib)

Feed cracks back in, repeat cycle.

By Wed 22nd Jan, we're down to 29 million

Cracking The Last 50 Million

Friday 24th Jan down to 27 million.

PRINCE run, min 12, using “stemmed” previous answers

Allaboutsavingmoney
Bluntsfordays
IanTheDarkKnight
Yetismokesweed
…

Cracking The Last 50 Million

Monday 27th Jan down to 17 million.

Last run took out ~10 million, using hashes.org found lists.

BUT, this probably contains the answers to a lot of them, as someone fed in the HIBP dataset as input.

However, 1.4m of those were me anyway:

```
# 13. | xxx (that's you!) |  1'405'749
```

See https://blog.cynosureprime.com/2017/08/320-million-hashes-exposed.html

Crack Me If You Can 2015 – Phoning it in

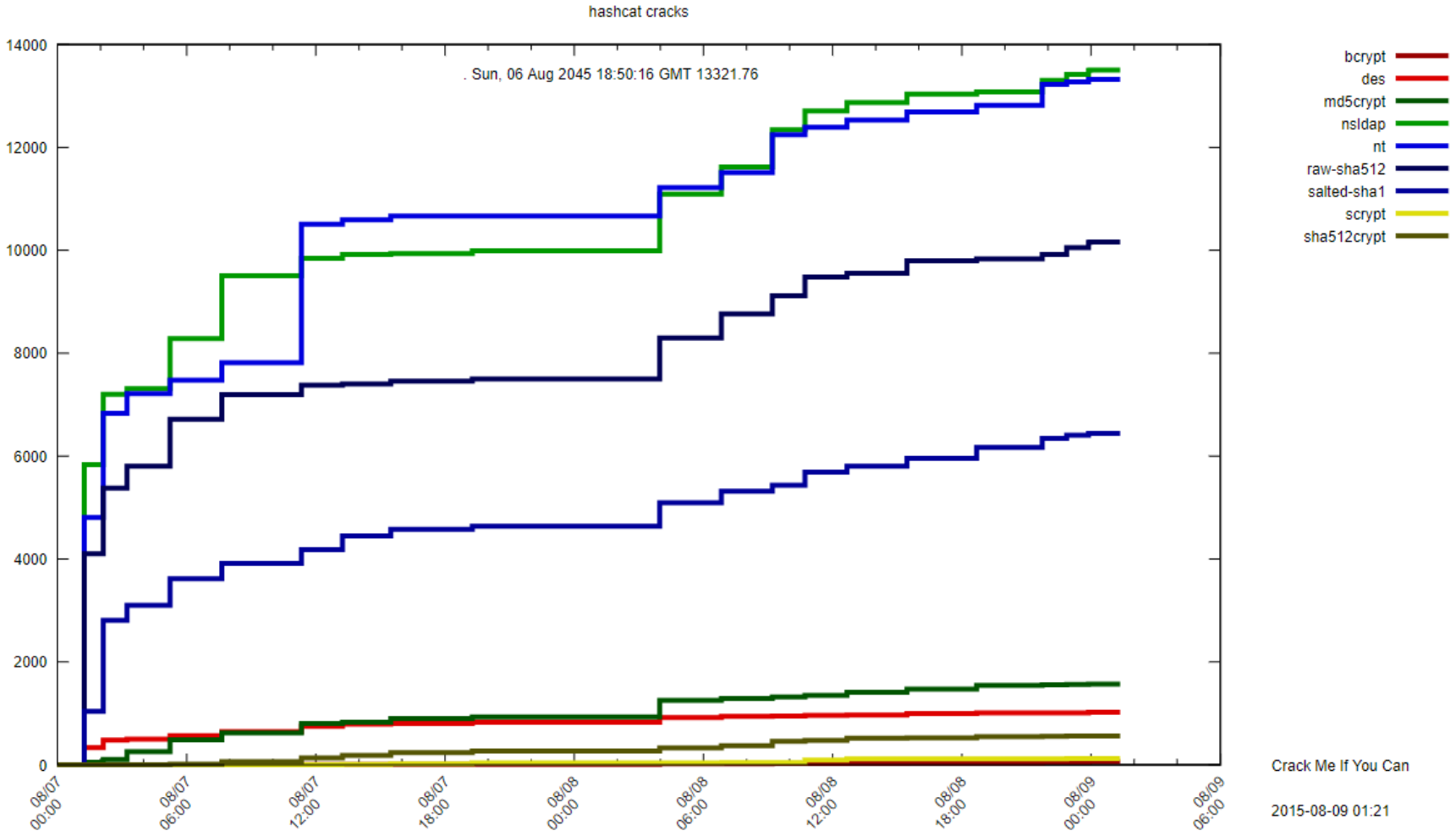
https://contest-2015.korelogic.com/downloads.html

Download, decrypt, unpack. Naïve approach :

```
$ for i in `ls pro` ; do python hashcrack.py -i
pro/$i ; done # 1080Ti
```

(Actually we need to tell it that .1 is NTLM, because both NTLM and MD5 are 32 hex chars.)

Crack Me If You Can 2015 – Winners



Crack Me If You Can 2015 – Our “phone it in” attempt

After ~48 hours on a single 1080Ti, no tuning, cancelling some of the very long runs by hand:

```
$ for i in `ls cmiyc/pro/*` ; do
python3 hashcrack.py -i $i –remove –
-pot cmiyc/pro.pot  ; done

$ wc -l /root/cmiyc/pro.pot
6252 /root/cmiyc/pro.pot
```

Cracking unknown hash types

Look in john/run/ for converters, e.g. pdf2john

Other converters, e.g. https://github.com/floyd-fuh/JKS-private-key-cracker-hashcat

Look at the source code, convert to a known type. (A lot are just different encodings, e.g. base64 not hex)

Write your own hashcat module

Example: Couchbase Admin Password

https://gravitas-shortfall.blogspot.com/2019/12/cracking-couchbase-admin-password.html

Grovel around erlang source and cfg file on disk.

Find HMAC-SHA1 in source code for admin password

Try HMAC-SHA1 (key = $salt) or (key = $pass) ?

```
\0\0
plainm\0\0\00bl/nSj6e7vZS5KQqHmoTER7Z4cgTcDSL5vZ
TeaaFEAqCpxpLh m
```

Couchbase Admin Password

take 0'b...'h - lose the initial '0' and the trailing ‘h’

Base64 decode, then ASCII hex encode:

```
6e5fe74a3e9eeef652e4a42a1e6a13111ed9e1c81370348be6f
65379a685100a82a71a4b
```

salt is first 16 bytes , hmac is next 20

```
salt 6e5fe74a3e9eeef652e4a42a1e6a1311
hmac 1ed9e1c81370348be6f65379a685100a82a71a4b
```

Couchbase Admin Password

For hashcat construct as “hmac:salt”, so :

```
1ed9e1c81370348be6f65379a685100a82a71a4b:6e5fe74
a3e9eeef652e4a42a1e6a1311
```

Then crack with hashcat mode 160 and --hex-salt

```
hashcat64.exe -m 160 salt-n-mac.txt
Top32Million-probable.txt -w3 --hex-salt -O  -r
rules\InsidePro-PasswordsPro.rule
```

Too long; didn't remember

Cracking passwords should be done on pen tests.

Auditing your own passwords isn't so hard (blue team).

Basic attacks are relatively easy.

Some fairly complex attacks out there if you really want the answer.

Hashcrack script can make things easier

References

https://uncommoncriteria.org/ppc.html : Supplementary materials.

https://www.netmux.com/blog/hash-crack-v3 – Like the 'RTFM book' but for password cracking. Highly recommended, name clash is accidental

https://cyberwar.nl/d/1993-FarmerVenema-comp.security.unix-Improving-the-Security-of-Your-Site-by-Breaking-Into-It.pdf Farmer & Venema 1993

https://haveibeenpwned.com/Passwords Troy Hunt's list as SHA1/NTLM

https://blog.cynosureprime.com/2017/08/320-million-hashes-exposed.html
Another (and better) attempt at Troy Hunt's list

References

https://github.com/lakiw/pcfg_cracker – PCFG candidate generator

https://hashcat.net/wiki/doku.php?id=princeprocessor - PRINCE

https://github.com/RUB-SysSec/OMEN - OMEN

https://hashcat.net/wiki/doku.php?id=hashcat_utils – combinator, etc.

https://github.com/rarecoil/pantagrule - lots of rules derived from data

Rules and Dicts

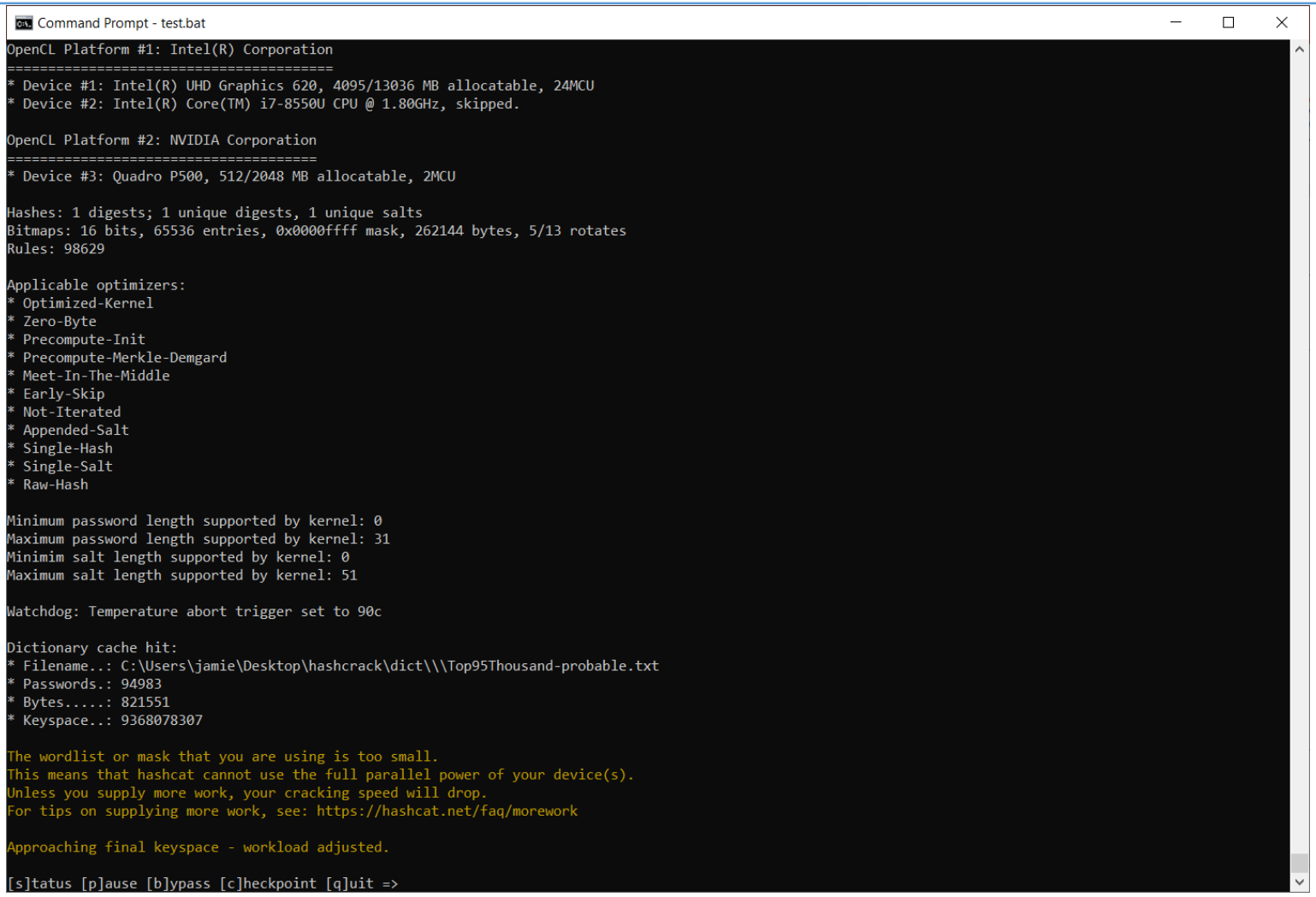
Recommended: nsav2dive.rule
TopNMillion (probabilistic passwords)
Breachcompilation
Hashes.org “left” lists

Improved(?) Rules
“leetified” versions of nsav2dive, best64 and PasswordsPro rulesets.
bestN.rule in rules/ folder of the project on github

New: Insertions.rule : any one or two chars inserted at all possible places in a word.
lastN.txt – all suffixes of N letters taken from breachcompilation or similar.
lastN-M.txt – all suffixes between N and M chars long

Cheap and cheerful “breach DB” https://mega.nz/#F!e7phQQJJ!E4wAhlA-aB22a5lObJAcmg

Live Demo ? ¯_(ツ)_/¯


Command Prompt - test.bat
OpenCL Platform #1: Intel(R) Corporation
* Device #1: Intel(R) UHD Graphics 620, 4095/13036 MB allocatable, 24MCU
* Device #2: Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz, skipped.
OpenCL Platform #2: NVIDIA Corporation
* Device #3: Quadro P500, 512/2048 MB allocatable, 2MCU
Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 98629
Applicable optimizers:
* Optimized-Kernel
* Zero-Byte
* Precompute-Init
* Precompute-Merkle-Demgard
* Meet-In-The-Middle
* Early-Skip
* Not-Iterated
* Appended-Salt
* Single-Hash
* Single-Salt
* Raw-Hash
Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 31
Minimim salt length supported by kernel: 0
Maximum salt length supported by kernel: 51
Watchdog: Temperature abort trigger set to 90c
Dictionary cache hit:
* Filename..: C:\Users\jamie\Desktop\hashcrack\dict\\\Top95Thousand-probable.txt
* Passwords.: 94983
* Bytes.....: 821551
* Keyspace..: 9368078307
The wordlist or mask that you are using is too small.
This means that hashcat cannot use the full parallel power of your device(s).
Unless you supply more work, your cracking speed will drop.
For tips on supplying more work, see: https://hashcat.net/faq/morework
Approaching final keyspace - workload adjusted.
[s]tatus [p]ause [b]ypass [c]heckpoint [q]uit =>

Questions