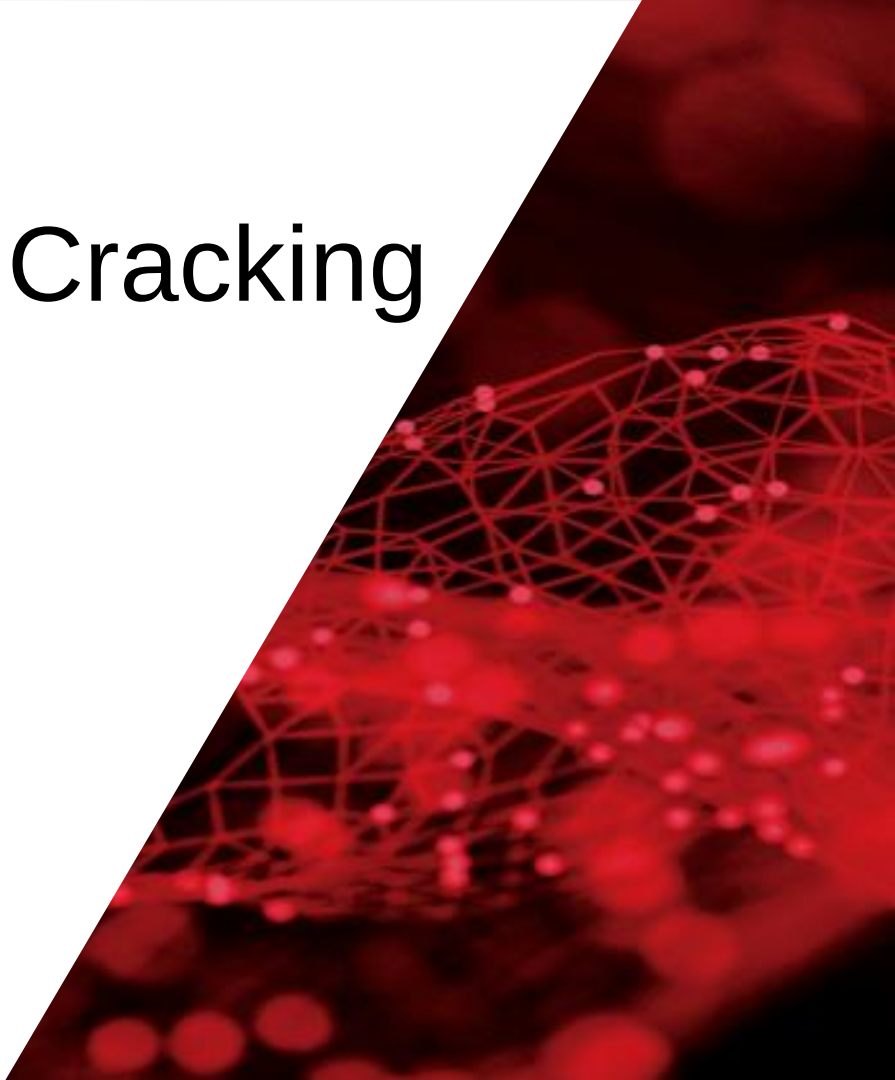
Practical Password Cracking

The First 500 Million

jamie.riden@ioactive.com

IOActive®



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)

Bio

- Built/improved multi-user cracking systems at two UK pentest companies
- Cracking other people's passwords since 2003, red and blue "teams".
- Dev, sysadmin, blue team, now pentester
- From embedded systems up to a Cray
- Erdős number of 4

Why Bother?

- Naïve 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 guidance

Example Hashes

```
6db61eaa9962d0ef0fd6e7676c9f8e68     [NTLM or MD5]
{SHA}AEkejO/OIdcoo8beMltXFsSUoaE=    [Netscape LDAP SHA1]
25f7edf2e81c5c5ae89c86d8d0581ed3f1d0d44657078204a551717f61d52f23
df1bxxx8d32ba0edacf047ca0349d05cf    [SHA512, sometimes SHA2-512]
{SSHA}jmu2X5k8vOJa5XZGeUa9TIrQlpEw   [Netscape LDAP salted SHA1]
1.ke4gjHbX/Xg                       [DEScrypt, 8 char max]
$1$LAr1jfuu$ztACi8wvC1X/cMk7B8Owg/   [md5crypt]
$6$0rqfzlDPSQDWk$gOk9W85/qwUYiDs/hNsgux.Z1GOQVRHnHFNY1dOWPKIzYxk
gLMZcbGoU37xt06xiMzM2.RaPYh/e8niiWtDAQ. [sha512crypt]
$2a$10$XFX4Pyb/KjTgaxH4cFHhbOsP8PsECxxxRw7zwRWTYJi6 [bcrypt]
$9$DDHX/GcKrS.OUZ$OqUtJrbh2eOaNZGWBCxxxD.9r.gpVD2os [scrypt]
```

Password Cracking

- Bad Hashes (fast, unsalted hashes)
- Good Hashes (slow, salted hashes)
- Hashcrack script/tools - http://github.com/blacktraffic/hashcrack
- Offensive uses
- Defensive uses
- Cracking 500 million hashes
- Quick spin against CMIYC 2015 hashes (CTF)

Bad Hashes

- MD5 hash: “secret” -> 5ebe2294ecd0e0f08eab7690d2a6ee69
- MD5 is a quick 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.)

Bad Hashes

- No home-brew crypto please
- Adding “secret” static data to your MD5 doesn’t help as much as you think.
- Improve hash function before you add ‘pepper’. (per app secret)
- Composing bad functions (md5, sha1, etc) doesn’t really help
- Avoid poor Random Number Generators
- **Simpler to use good library function**

Good Hashes

- Blowfish, or argon2, or PBKDF2 etc.

```
echo password_hash("test", PASSWORD_DEFAULT);
$2y$10$.vGA1O9wmRjrwAVXD98HNOgsNpDczlqm3Jq7KnE
d1rVAGv3Fykk1a
```

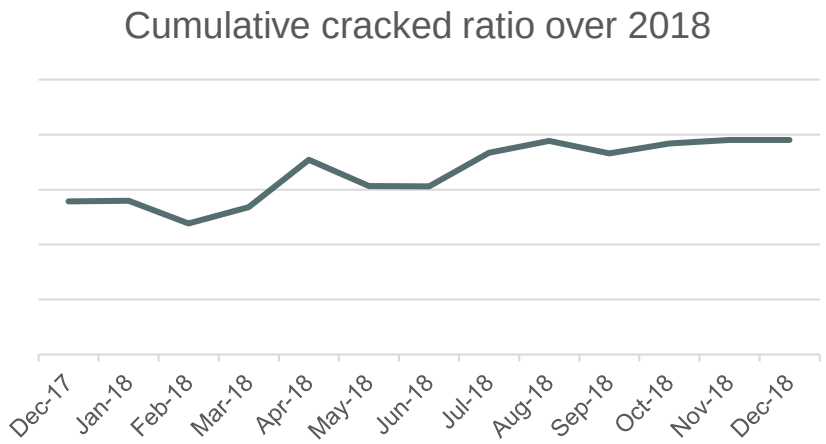
- Salt, cost parameter, very slow – 20 thousand guesses / sec with cost 5.
- Random salt prevents lookup table creation

Hashcrack Script

- Guesses hash type (regular expressions mappings)
- Tries a decode if you need one (mostly via JtR scripts)
- Tries some reasonable dictionaries, rules
- Configurable – mapping files are data
- Can override for specific requirements
- AGPL

Hashcrack Script - Motivation

- “Laziness, hubris, impatience”
- Codify some attack techniques
- Measure, improve
- “front loading”
- Don’t make me think
- Often stuck in
  machine rooms



Hashcrack Script – Constraints & Goals

- 5 main attack types with hashcat:
  - Hybrid (dict+rules), cross-product, masks (3)
- Keep the GPU busy:
  - Fast hashes need more than just dictionary
- Efficiency
  - try not to duplicate guesses (dict + rules)
  - Go for greatest speed/density first
- Re-use pwdump LM cracks for NTLM
  - yes, we still see LM! LM password is uppercase NTLM password

Sketch of an Attack Workflow

Fast hash like pwdump (uid:LM:NTLM)

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

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 last3-4.txt (with common 3-4 char suffixes)
5. Decent sized dict with l33tpasspro rules

Sketch of an Attack Workflow.

Slow Hash like bcrypt:

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

0. (manual) get trunk hashcat from github & compile – 40% quicker
1. Small crib file with company name etc.
2. Previously cracked passwords with no rules
3. Medium dictionary, no rules

Red Team

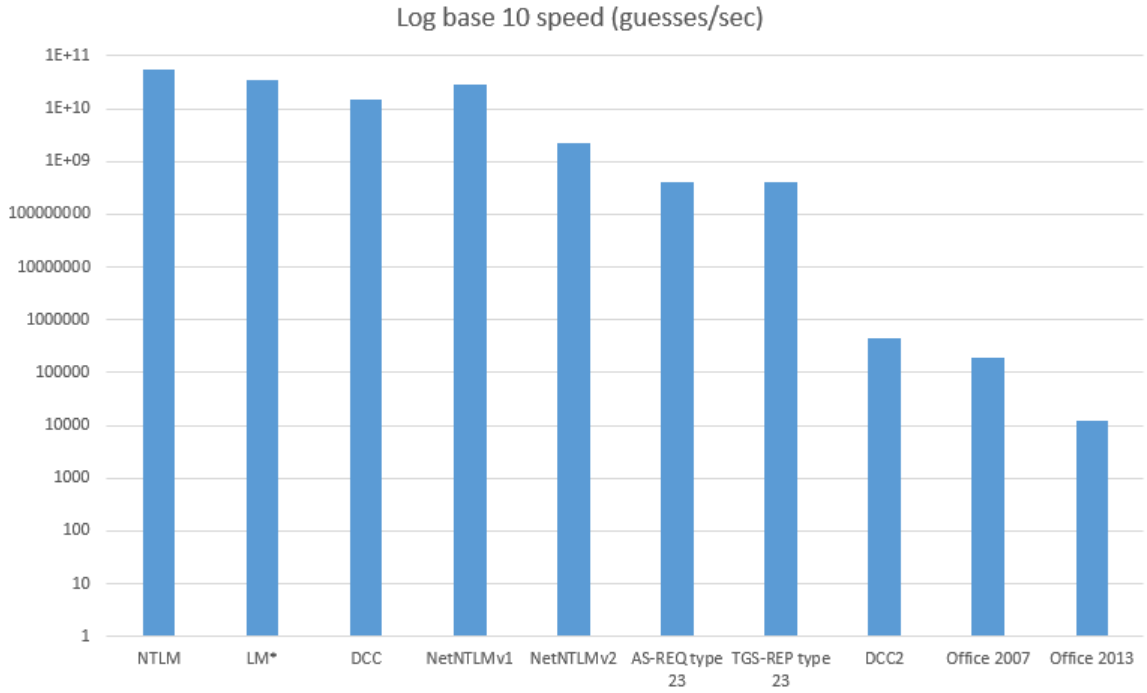
Steal hashes via

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

Red Team

- Once you have the hash, cracking is opaque to the target
- Generally lots of things to try
- Never know what one success will achieve
- Get something started quickly while you have a proper look around
- Once you get some passwords cracked, try them on other things (“crib”)

Guess Speed (Ideal)



* LM is awful for other reasons too

IOActive.

Red Team – Use Cases

```
python3 hashcrack.py -i ifm.zip
```

[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
```

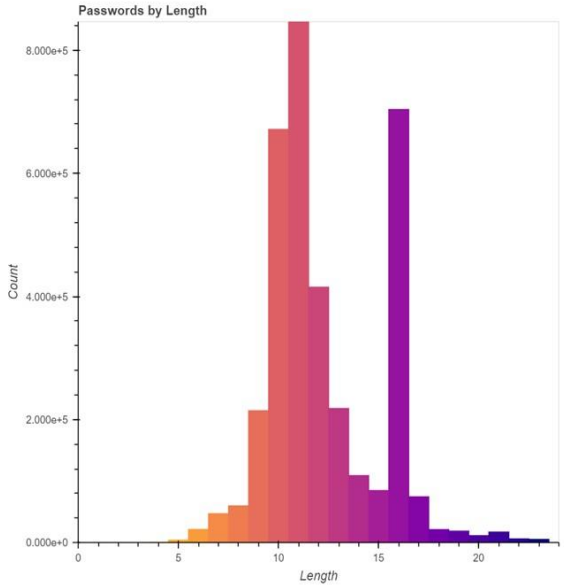
[runs jtr script office2john.py]

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
```

“Boring” Audit Stuff – Pentest or Blue Team

- Export domain passwords
- Crack the hashes
- Uses LM cracks with case permutations, if present
- Draw some graphs if you like
- Update policy, make users change password



Handling painful stuff – e.g. Oracle

```
$ python hashcrack.py -i oracle-hashes.txt -t oracle --
show
CTXSYS:S:54C7Cxxx;T:A10609DFAE71F2xxx1C27D2E4E69CB6BAB96
EDE06F795842CE7
```

Found 112

```
54c7ccf5def04820dxxx7d61a6b:e5ec99bc210308f58747:CTXSYS
```

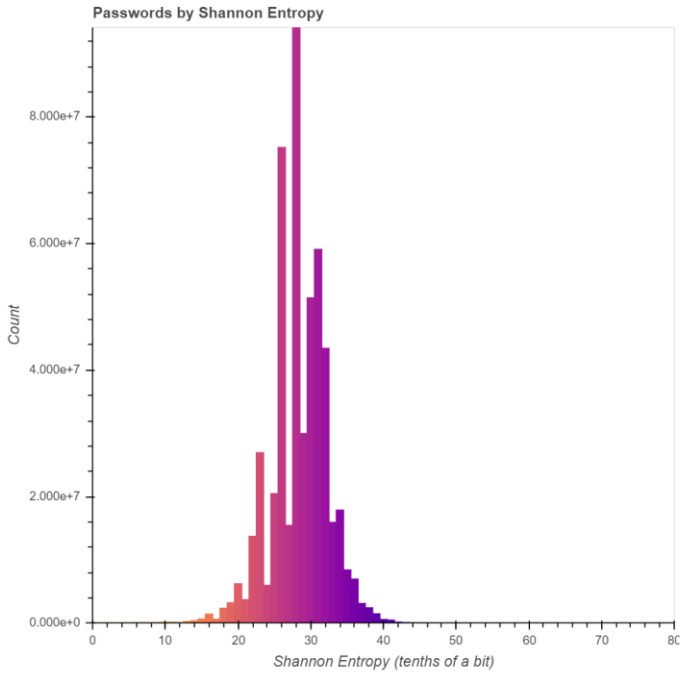
Found 12300

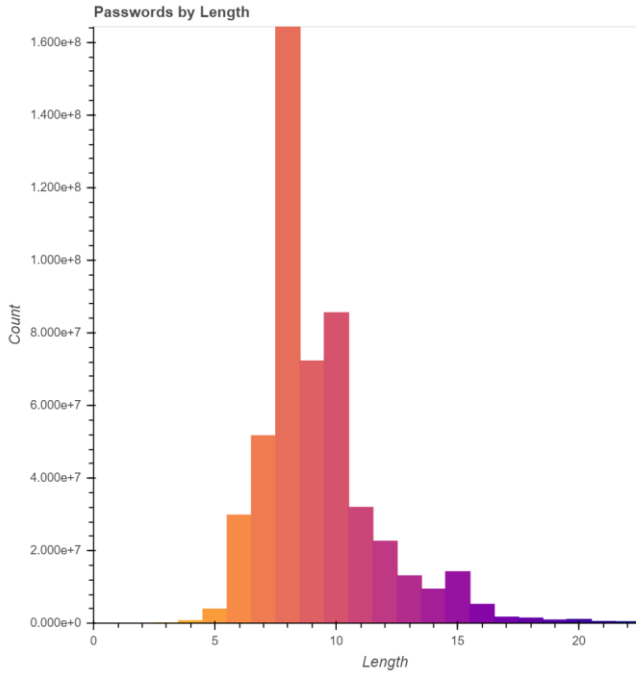
```
A10609DFAE71xxx9CB6BAB96EDE06F795842CE7:CTXSYS
```

Blue Team – Quick Wins

- Stop users from choosing
  - Really weak passwords
  - Widely compromised passwords
  - Passwords based on dictionary words
- Rate-limit password guesses somehow.
- Back off to CAPTCHA for potential brute-force or credential stuffing attacks
- Crack your own password hashes

Stats from the 500 million cracked





IOActive.

Querying Troy Hunt's Compromised List

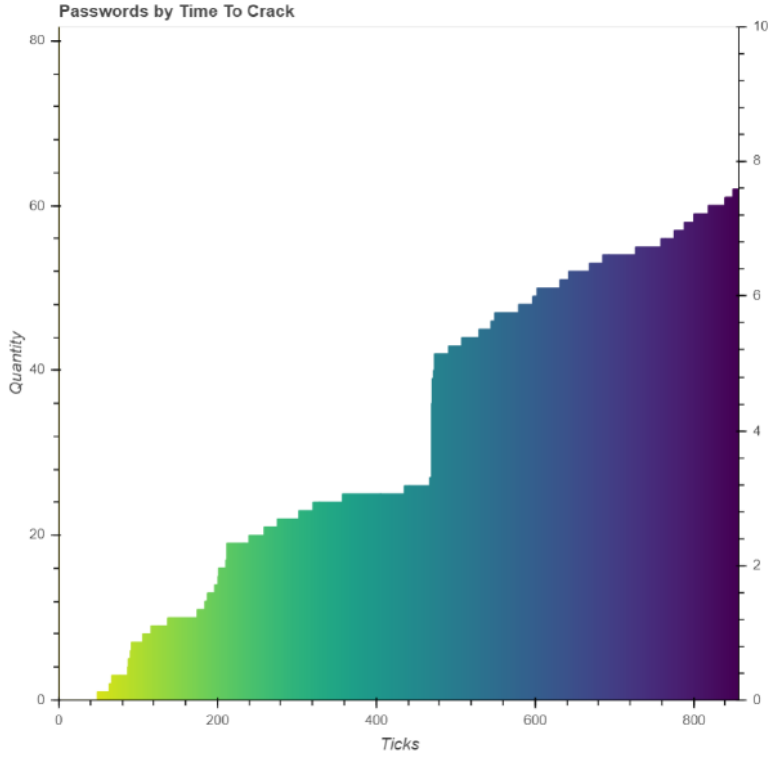
```
$ curl
"https://haveibeenpwned.com/api/v2/pwn
edpassword/5baa61e4c9b93f3f0682250b6cf
8331b7ee68fd8
HTTP/2 200  [found in a breach]
$ curl
"https://haveibeenpwned.com/api/v2/pwn
edpassword/d5c0a7e48a8047f6c59abba1db3
b365ecdaf3663
HTTP/2 404  [not found in breach]
```

What was cracked and when

Can derive from hashcat status output

Minimise the area under this curve:

kill ‘weak’ ones (lhs)



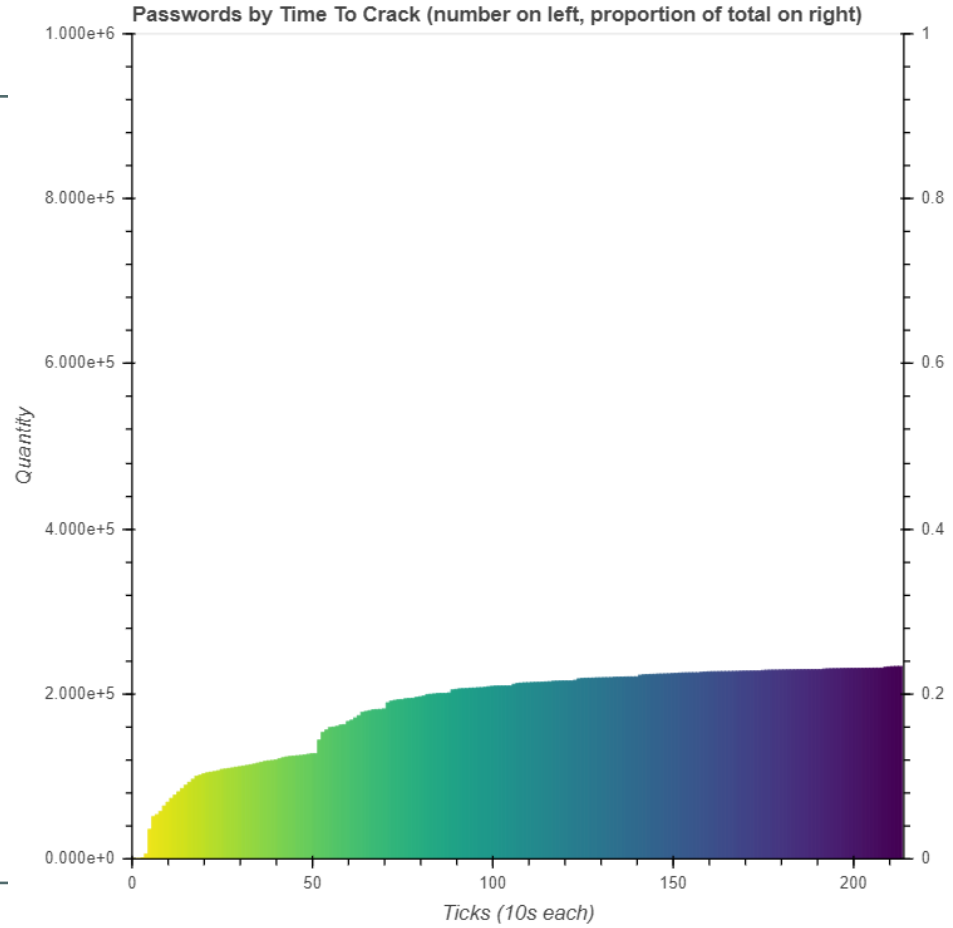
IOActive.

What was cracked and when

First million from HIBP

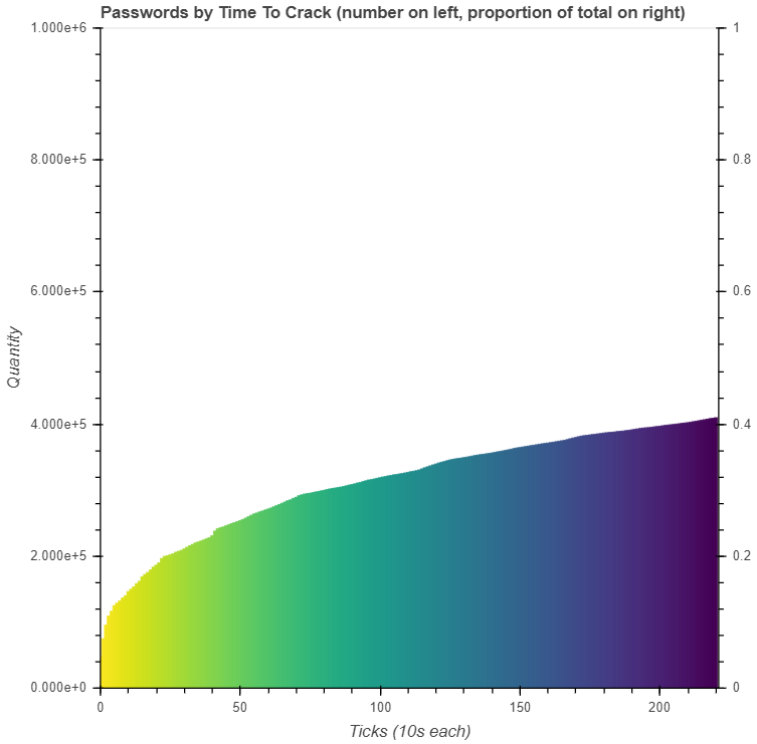
Cracking on laptop

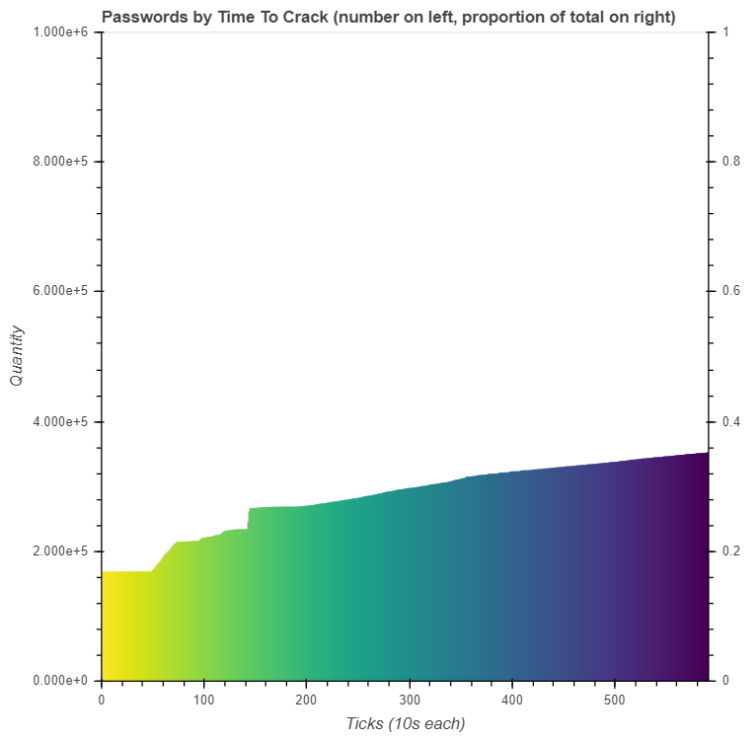
Incremental first then hybrid dict and rules



IOActive.

What was cracked and when – different rules, effect of pot file





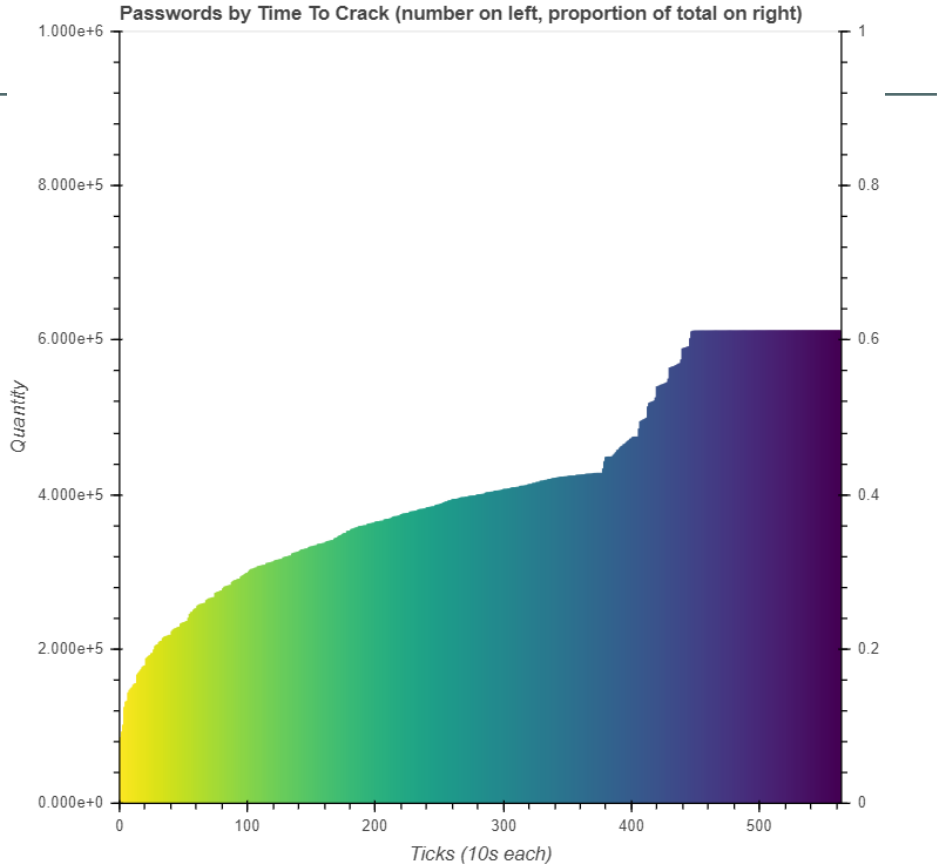
IOActive.

What was cracked and when

Now, on a 1080 Ti

1st: Top 258 Million

2nd: incremental up to 9
- starts about tick 400



IOActive.

What was cracked and when – How to Generate

$ python3 hashcrack.py -i testcase.txt --status greyarea2

(writes the status updates to a file ‘greyarea2’)

$ python graph-by-quality.py greyarea2

(uses bokeh to plot a graph from the status file – probably will be in the hashcat-5.1.0/ subdirectory)

--status is the hashcat flag used to generate this every 10s

Design of a Password Cracking System

power, cooling, contention, latency, throughput

Entry level – laptop. **Do not overheat**.
Modest tower with 1080 Ti (greyarea)
Homebrew multi-card setup
“COTS” hw - Brutalis (https://terahash.com/)
Clustered system, e.g. hashtopolis
Multi-user cluster, custom code

Design of a Password Cracking System

Bulk of this was:

- 1080 Ti (grey area)
- Dell 7510 laptop

Don’t skimp on the card

1080 Ti is 250W, cooling is important

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':' > pwned.txt

Tweak the hashcrack.py code with --bitmap-max=26

Cracking 500 Million Passwords – Split / Crack / Merge

```
$ split -l 50000000 -d pwned.txt       # 50 mil at a go

#!/bin/bash
python3 hashcrack.py -i x00 -t ntlm --pot troy1.pot --remove >/dev/null
… repeat …
cat x?? > pwned-remaining1.txt
rm –f x??
```

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

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?s

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 *aka “information supercollider”*

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

Attack Types and Flags

--loopback –O –w4 # optimise for Linux (-w3 for Windows), feedback

-a1 allows cross-product. combinator and combinator 3 do similar. Or write your own script:

```
$ python leet.py words.txt | ./hashcat64.bin -m 1000
/root/hashcrack/pwned-remaining3.txt  -a0 -O -w4 -r
rules/best64.rule --bitmap-max=26 -r rules/best64.rule
```

OMEN & PRINCE & PCFG as candidate generators as well

Attack Types - Custom

STDIN, use with generator script, e.g. combinator or

$ python scripts/leetify.py dict/Top32Million-probable.txt | ./hashcat-5.1.0/hashcat64.bin -m 1000 -O -w4 pro-hashes.1.txt -r rules/best64.rule

Attack Types - OMEN

#build model

./createNG --iPwdList ../dict/cracked-passwords.txt

#invoke model to generate candidates

./enumNG -p | ../hashcat-5.1.0/hashcat64.bin -m 1000 -O -w4 street-hashes.1.txt -r ../rules/nsav2dive.rule

beb68df17dfbe162212ff8a5d8a254da:Pickey21

31e25e76883ea48224473670cd84aa91:Aller13*

a78be982ee90685cbf8e3b31bd14b4db:Sandand09

Attack Types - Combinator

./hashcat-utils-1.9/bin/combinator3.bin top95k.txt top95k.txt top95k.txt | ./hashcat-5.1.0/hashcat64.bin -m 1000 -O -w4 /root/bits/remaining5.txt -r rules/nsav2dive.rule

1d227xxc91ed5e4fb073:1357iloveyou1357

Attack Types - Combinator

With dict/words.txt three times :

- f1d99b1c353d01xx: blueberrycarrot
- e25d103708a7eaxxc: blueberryfudge
- 99ed45f86ffe309xx: bluebirdpelican123
- 6af87dc65d6cxx6517f: bluefisheggs

Attack Types - Combinator

With dict/top1k.txt three times :

- onrunner12345
- 43211234asdasd
- sunshinelovepark

Beyond 7-bit ASCII

LM seems to depend on codepage.

NTLM is basically widechar MD4. (e.g. –m 900 is raw mode)

See: https://www.blackhillsinfosec.com/cracking-passwords-with-umlauts/

SHA1 and related UNIX hashes like SHA512crypt are generally UTF-8 – web sites may choose their own encoding.

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 – Phases

```
248945621 troy1.pot [defaults -a0]
 96500674 troy2.pot [def incr -a3]
141627978 troy3.pot [breachcompilation]
  2777509 troy4.pot [hcmask incr -a3]
  9121343 troy5.pot [Top 2 Billion]
  1090010 troy6.pot [258Mil/last 4, a1]
  1557763 troy7.pot [other dict / last]
 24902724 troy8.pot [hashes.org]
    68403 troy9.pot [misc ideas]
```

Cracking 500 Million Passwords – Lessons ?

Not a huge amount?

While very much a real world dataset, it's not one you will need to attack in real life.

Do NOT use it as a dictionary for NTLM cracking as is.

First remove anything really weak, saves repetition

```
hashcat64.exe -m 99999 example.txt -O -w3 -a3 --remove -
i --increment-max=7
```

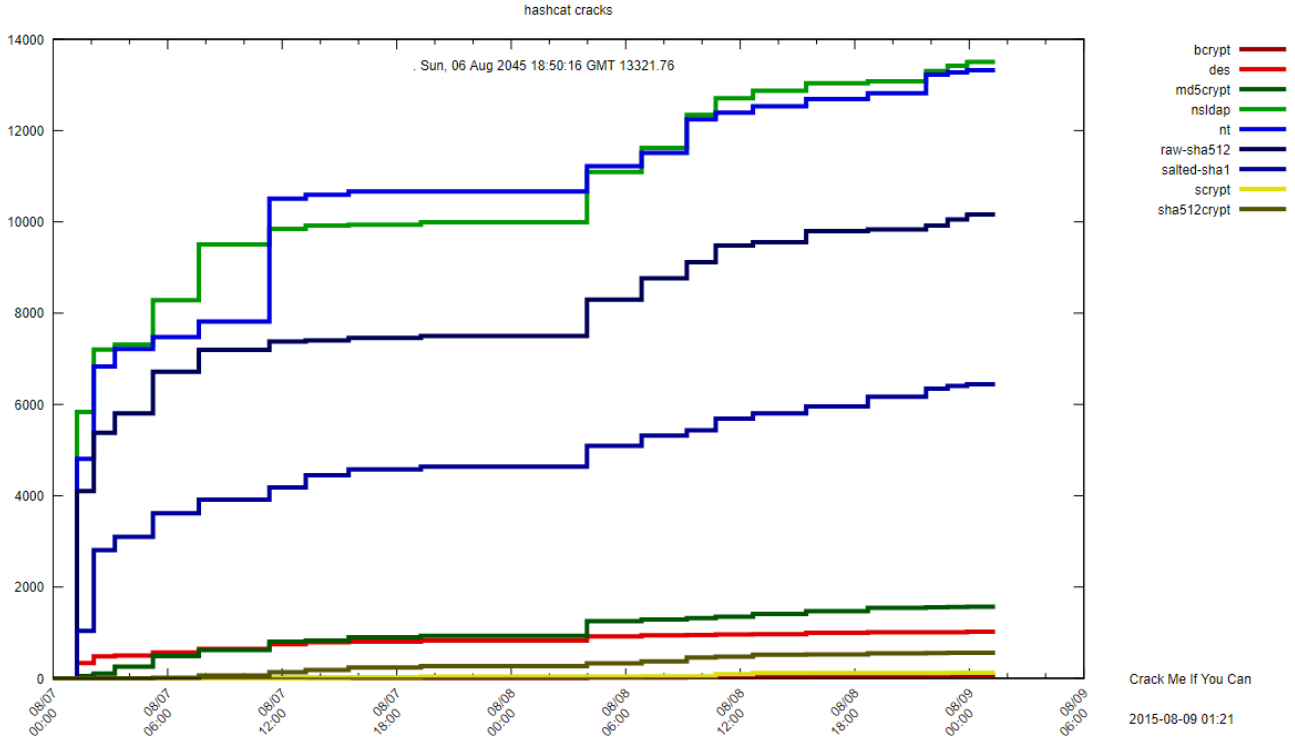
https://contest-2015.korelogic.com/downloads.html

Download, decrypt, unpack

```
$ for i in `ls street` ; do python hashcrack.py -i street/$i ;
done # laptop, Quadro 500
```

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

Crack Me If You Can 2015 – Winners



IOActive.

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

After ~48 hours on a single 1080Ti, cancelling some of the very long runs (minimal user intervention).

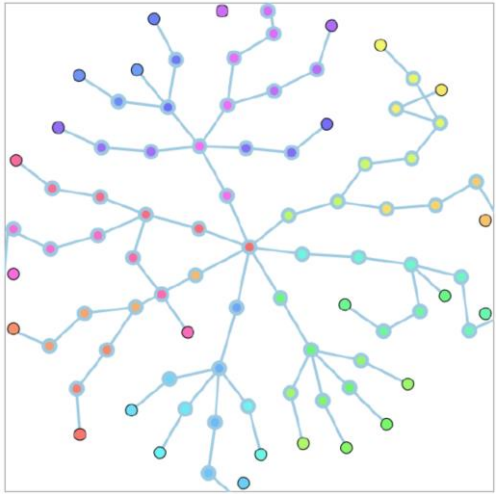
```
$ for i in `ls cmiyc/pro/*` ; do
python3 hashcrack.py -i $i –remove -
-pot cmiyc/pro.pot  ; done

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

Future Work

More graphs and metrics – this is a root word and rules leading to cracks (edge nodes)

~~Multiple back-ends (JtR for e.g. bitlocker)~~ done

PRINCE, OMEN, PCFG etc



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.

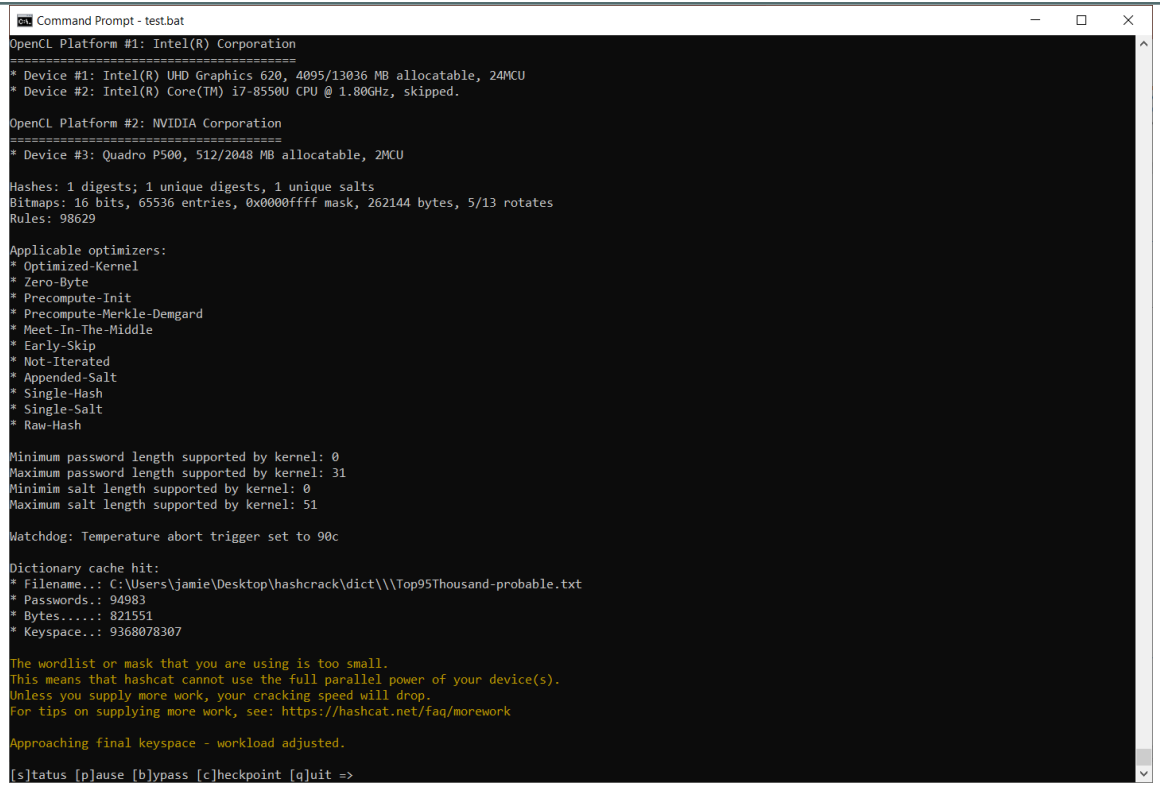
Rules and Dicts

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

Improved? “leetified” versions of nsav2dive, best64 and PasswordsPro rulesets.

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

Live Demo ? ¯_(ツ)_/¯



IOActive.

Questions