Practical Password Cracking

“wannabes worry about clock speed – real computer companies worry about cooling”

Jamie Riden
Email: jamie@blacktraffic.co.uk
Twitter: @pedantic_hacker
Password Cracking

Bad hashes and why they’re bad

Good hashes and why they’re good

Protecting your users from themselves

Cracking tools and techniques
Problem

We want to store the user password in a reasonably safe way.

That means, we can check it’s correct but if an attacker breaches the system, they can’t just recover the password.

The solution is a one-way function, of which a hash is one example.

Obviously we want a one-way function with low number of collisions.
Supposedly a “one way” function should be hard to reverse.

We can make lots of guesses and see whether the answer is the same. Quick function => quick guesses.

Example MD5 hash: “secret” -> 5ebe2294ecd0e0f08eab7690d2a6ee69

Collisions are so unlikely they’re not worth worrying about.

This is nothing to do with hash tables.
Properties of Hash Functions

Maps arbitrary data to fixed length – eg any input produces 256-bit output.

Don’t want predictable collisions.

In many branches of Computer Science, faster is better (not here).

Small change in input produces large change in output.

Should be difficult to reverse.
Examples – MD5

MD5 is a quick hash function mapping anything to a 128-bit value.

Unsalted hash, so feasible to build a lookup table.

```bash
$ for w in `cat /usr/share/dict/words` ; echo -n $j ; echo -n $j | md5sum ; done > lookup.txt
```

MD5 is very quick – so guessing is quick.
Examples – Salted MD5

We don’t want people to build a lookup table, so we chuck a large random number (salt) into each hash.

\[
\text{Stored hash} := \text{salt} ++ \text{md5(salt} ++ \text{password)}
\]

Makes a lookup table unfeasibly large. Slows hashcat to O(n).

Example is md5/shaXcrypt off UNIX – also does many “rounds”

root: $6$1Zd0wvLZ$zb4l0ouYxxx:::
Examples – Salted MD5

$ for i in `seq 1 9` ; do openssl passwd -1 password ; done

TYPE / SALT / HASHED PASSWORD

$1$ybI6m63ao4AqUQ5AqRzX4n2b6BvcR0
$1$e97Z4W.Vsx9sbekkkDpZWluzw4FrFJ.
$1$7dyr5uqa$1Rz6NnnZD1Uszcsv5OQuu0
$1$QhqY.UV4$1uSEu.3mIx5ZaqehnNkNv.
$1$9l.Ernhe$BZmHo1AaTf1MEVin.kcTO/
$1$KMfvTerj$REG7pa24ZndxoMHmHbYMOn1
$1$em5.2cPE$Pb6ud.Uxjikg4w4n9KeLZ1
$1$q824kpm2$MTyqG5Q7Si6o5T7uVs69Z/
$1$tXVYUqFs$RoyzeFdFcfkgEXsf.6hb.
Non-solutions #1

Random junk appended/prepended.
Web app using plain MD5 “7JjUe83k” + password + “He03Kje2UekEkmPa3MeRbKntw8T9Ons5”.

^k^3^8

Attack:
[List.Rules:SillyAppJTR]
Az"He03Kje2UekEkmPa3MeRbKntw8T9Ons5"A0"7JjUe83k"

(for each candidate, append "He03Kje2UekEkmPa3MeRbKntw8T9Ons5", prepend "7JjUe83k", then test)
Non-solutions #1

$./john jamietest --wordlist=testdict.txt --format=raw-shal --rules=SillyAppJTR
7JjUe83kFOOBARHe03Kje2UekEkmPa3MeRbKntw8T9Ons5 (jamie)

Recovers the original. A bit slower but not really very useful in terms of protection.

DO NOT ROLL YOUR OWN CRYPTO

HMAC-MD5 is there if you need to introduce a secret.
Non-solutions #2

Any of:

md5(sha1(password))
md5(md5(salt) + md5(password))
sha1(sha1(password))
sha1(str_rot13(password + salt))

(No computationally harder)
Non-solutions #3

NTLM is based off MD4, unsalted -> so hashcat doesn’t slow down as number of hashes increase. Same for all unsalted.

LM is even worse – upper case, chop into 2 x 7 char bits

There’s no reason to prevent long passwords / special characters in web sites.
NTLM Dump – DC

C:\>ntdsutil
ntdsutil: activate instance ntds
ntdsutil: ifm
ifm: create full c:\temp\ifm
ifm: quit
ntdsutil: quit

python impacket/examples/secretsdump.py -system SYSTEM -ntds ntds.dit LOCAL
NTLM Dump – non-DC

reg save HKLM\system system.reg
reg save HKLM\security security.reg
reg save HKLM\sam sam.reg

python impacket/examples/secretsdump.py -system
system.reg  -security security.reg -sam sam.reg LOCAL
Other “Webby” Examples

https://hashcat.net/wiki/doku.php?id=example_hashes

mysql4.1/5: fcfc1b8749cf99d88e5f34271d636178fb5d130

phpass, WordPress (MD5), Joomla (MD5):
  $P$984478476IagS59wHZvyQMArzfx58u.

phpass, phpBB3 (MD5):
  $H$984478476IagS59wHZvyQMArzfx58u.

JWT: eyJhbGciOiJIUzI1NiJ9.e..HRnrkiKmio2t3JqwL32guY

Django SHA-1: sha1$fe76b$02d5916...f044887f4b1abf9b013

Apache MD5: $apr1$71850310$gh9m4xcAn3MGxogwX/ztb.
Getting hold of hashes – contrived ex.

```bash

[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 ./rules/InsidePro-PasswordsPro.rule -O

5f4dcc3b5aa765d61d8327deb882cf99:password
```
Actual Solutions – Better Hashing

PBKDF2 (RFC2898) – takes a number of “rounds” or iterations to make it costly. e.g. System.Cryptography.Rfc2898DeriveBytes

WPA2: PBKDF2(HMAC–SHA1, passphrase, ssid, 4096, 256)

Bcrypt, Scrypt, Argon (not many implementations yet)
Actual Solutions – Better Hashing


```php
echo password_hash("rasmuslerdorf", PASSWORD_DEFAULT);
```

$2y$10$.vGA1O9wmRjrwAVXD98HNOgsNpDczlqm3Jq7KnEd1rVAGv3Fykk1a
Requirements

Salt should be from CSPRNG (java SecureRandom, etc)

Salt should be long enough to make lookup table infeasible.

Hashing should be done server-side.

Hash should be computationally expensive (not one round of MD5)
“Pepper”

Using an application secret as well as the salt – this is not stored with the hash. Might help, might not.

Doesn’t make it computationally harder, but you have to find the “pepper” first.

Most apps don’t do this; salted hashes should be OK without extra bits.
Actual Solutions – Better Hashing

bcrypt and SHA512crypt take a similar approach so that making guesses is costly.

$ ./hashcat64.bin -m 1800 sha512.txt -a 3
Hash.Type........: sha512crypt, SHA512 (Unix)
Speed.GPU.#1....: 38749 H/s

$6$9sirPrQg$keedQFI0yFrljxxxxxiA2l7eksg1:toor
bcrypt and other algorithms use number of rounds or cost factor so you can make a hash computation take longer.

```php
$options = [
    'cost' => 12,
];

echo password_hash("rasmuslerdorf", PASSWORD_BCRYPT, $options);

$2y$12$QjSH496pcT5CEbzjD/vtVeH03tfHKFy36d4J0Ltp31Rtee9HDxY3Ks
```
Helping Your Users

Number of users vs. Complexity of password
Helping Your Users

Try to enforce length/complexity.
But be aware, “Password123!” meets most length/complexity guidelines.
Check for dictionary words?
Check for password stuffing (someone replaying passwords found in another breach) – e.g. rate limit, CAPTCHAs, account lockout etc.
Crack your own passwords and expire the compromised ones.
Check for breached passwords at set time – see below:
Checking for Breached Passwords

$ curl
"https://haveibeenpwned.com/api/v2/pwnedpassword/" `echo -n password | sha1sum | cut -f 1 -d' '` -D
HTTP/2 200  [ found in a breach ]

$ curl
"https://haveibeenpwned.com/api/v2/pwnedpassword/" `echo -n psdfasdfsdfgasdfgasdgasdgassword | sha1sum | cut -f 1 -d' '` -D
HTTP/2 404  [ not found in breach ]
Tools - Hashcat

Very good GPU cracker, but also does CPU / FPGA.

Get the binaries from the net, install the latest NVIDIA drivers and it should be ready.

On Linux, needs an X server running to overlock.

On Windows, use MSI Afterburner.
Tools - Hashcat

Basic usage – mode –a0 is assumed if not specified – dict+rules
hashcat64.exe –m <hash type> hashlist.txt dictionary.txt rules.rule

Incremental:
Hashcat64.exe –m <hash type> -a3 hashlist.txt [ <mask> ]
Where ?l lower case ?u upper case ?d digit ?s special

Tools - Hashcat

xorg.conf:
Section "Device"
 Identifier     "Device0"
 Driver         "nvidia"
 VendorName     "NVIDIA Corporation"
 BoardName      "GeForce GTX 1080 Ti"
 Option         "Coolbits" "13"
 Option         "RegistryDwords" "PowereMizerEnable=0x1; PowerMizerLevelAC=0x3; PerfLevelSrc=0x2222"
 Option         "AllowEmptyInitialConfiguration" "true"
EndSection
Tools – Hashcat overclocking

#!/bin/bash

export MEMCLOCK=200 # don’t blame me if this breaks your card
export GFXCLOCK=100 # and this. Works for Me™
export POWER=180 # power limit if you want one
export FAN=80 # trade off between temp and fan noise

XAUTHORITY=/run/user/131/gdm/Xauthority nvidia-settings \\ 
-a [gpu:0]/GpuPowerMizerMode=1 \\
-a [gpu:0]/GPUMemoryTransferRateOffset[3]=$MEMCLOCK \\
-a [gpu:0]/GPUFanControlState=1 -a [fan:0]/GPUTargetFanSpeed=$FAN \\
-a [gpu:0]/GPUGraphicsClockOffset[3]=$GFXCLOCK
nvidia-smi -pl $POWER
Tools – Hashcat overclocking
Tools – John the Ripper

JTR with all the bits and bobs, including UTF-8 support and GPUs.

$ git clone https://github.com/magnumripper/JohnTheRipper.git
$ cd JohnTheRipper/src
$ ./configure
# on mine – vi Makefile and delete –DJOHN_AVX – for some reason
$ make install

Ex: john hashlist.txt --wordlist=/usr/share/dict/rules --rules=Extra
Careful with non 7-bit ASCII

```
$ echo -n "möt" | md5sum
43191e523ba88fba40a6744b67b8f546
```

user:43191e523ba88fba40a6744b67b8f546

Crack this with cudaHashcat: ?b?b?b?b

43191e523ba88fba40a6744b67b8f546:$HEX[6dc3b674]

(depends on hashing scheme and pre-processing of input data)
Brief digression into UTF-8

Lower 7 bit ASCII chars stored as 1-byte – themselves

Top-bit set chars are stored as follows:

<table>
<thead>
<tr>
<th>Bits of code point</th>
<th>First code point</th>
<th>Last code point</th>
<th>Bytes in sequence</th>
<th>Byte 1</th>
<th>Byte 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>U+0000</td>
<td>U+007F</td>
<td>1</td>
<td>0xxxxxxx</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>U+0080</td>
<td>U+07FF</td>
<td>2</td>
<td>110xxxxx</td>
<td>10xxxxx</td>
</tr>
</tbody>
</table>

So 246 is 11110110 – stored as 1100 0011 10 110110 – which is C3 B6
Careful with non 7-bit ASCII

passwd is “mötorhead”

root: $6$1Zd0wvLZ$zb4lOouYxxx:::

hashcat64 -m 1800 sha512.txt motordict.txt --username

$6$1Zd0wvLZ$zb4lOouYxxx0pgJ90:$HEX[6dc3b6746f7268656164]

motordict.txt : 6d c3 b6 74 6f 72 68 65 61 64
Careful with non 7-bit ASCII

C:\> net user /add mot möt

Crack this with cudaHashcat: ?b?b?b?b

9e8ad77244a880f7f1f10d0b46693fce:$HEX[6df674]

It seems ö is coded as \xf6 and not the two-byte UTF-8 encoding. (NTLM)
Careful with non 7-bit ASCII

Compare LM/NTLM for the same account:

LM 759c0a91bxxxx8728d99f4:xxxxxxx$HEX[454250509c]
NTLM 7b0ee41fxxx6376a6aee3c:xxxxxxx$HEX[65627070a3]

xxxxxxxxebpp£

Coded in two different ways – depending on LM/NTLM.
NTLM: > Oxff with hashcat

Password is “Dŵr”. We need to use raw MD4.

# ./hashcat64.bin -m 900 dwr.txt -a3 -2 44 -1 00 --hex-charset ?2?1?b?b?b?1

5441d13...3fdb2e87:$HEX[4400 7501 7200]
0x0044 0x0175 0x0072
    Dŵr
Custom char maps with hashcat

We want some non-ASCII chars, but not all possible byte values.

```
$ perl -e '$i=32; while ($i<127) { print chr($i); $i++; }; $i=192; while ($i<255) { print chr($i); $i++; };' > win-ext.hcchr
```

```
hashcat64.exe -m 1000 bot.txt -a 3 -1 win-ext.hcchr ?1?1?1?1?1?1?1?1?1?1?1 --username --increment
```

```
9883fd245e9aee55ad39d31752eb4a4d:$HEX[62f674]
```
Connect as a DBA to Oracle 11g with sqlplus

```
set heading off
set feedback off
set pagesize 1000
set linesize 100

SELECT name || ':' || SUBSTR(spare4,3) FROM sys.user$ WHERE spare4 IS NOT NULL ORDER BY name;
```

DBSNMP:A092440872DEAA491B0F8C16F4A2C9304928238617DA4741CBF3683A469C
Oracle

DBSNMP:A092440872DEAA491B0F8C16F4A2C9304928238617DA4741CBF3683A469C

For hashcat, first 40 / last 20 hex chars and separate:

A092440872DEAA491B0F8C16F4A2C93049282386:17DA4741CBF3683A469C

hashcat64.exe -m 112 oracle.txt -a 3

a092440872deaa491b0f8c16f4a2c93049282386:17da4741cbf3683a469c:dbsnmp
Postgresql

```sql
select passwd, usename from pg_shadow
```

then remove "md5" from the front of passwd, and use passwd:username

Run with `-m 12`
Targetted Attacks - hashcat

# swap char - leetify ( word -> w0rd )
so0
# append ‘!’
$!
# Toggle case of first letter
T0
# Enclose in quotes
^"$"
# prepend 123
^3^2^1

https://hashcat.net/wiki/doku.php?id=rule_based_attack
Targetted Attacks - wordlists

Troy Hunt’s list – unpublished?

Breach compilation list
https://gist.github.com/scottlinux/9a3b11257ac575e4f71de811322ce6b3

Crackstation https://crackstation.net/

Probabilistic password lists https://github.com/berzerk0/Probable-Wordlists

SecLists Passwords
https://github.com/danielmiessler/SecLists/tree/master/Passwords

Your found passwords / --loopback option

Reference here: http://www.blacktraffic.co.uk/pw-dict-public/
Targetted Attacks - hashcat

Hybrid attack – dictionary and rules

```
hashcat64.exe -m 0 -a 6 642395xxxxx4863eca rock.txt
-r rules/InsidePro-PasswordsPro.rule
642395cef47664a970441d3c94863eca:powerslave1984
```

Hybrid attack – two word lists (cross-product)

```
hashcat64.exe -m 0 -a 1 642395xxxxx4863eca rock.txt found.txt
```
Troy Hunt’s List


SHA1, so it’s very quick. These, and 275 million others, were recovered on a Dell Precision 7510 / Quadro M2000M

tres metros sobre el cielo [ tr: three meters above the sky ]
john_a_ujo_firman@yahoo.co.id1 [ data issues? ]
S0metimearoundmidnight [ leetified phrase]
qwertyuiopasdfghjklzxcvbnm12332 [ keyboard pattern ]
danthemanfrombristolland
ironwhoironfuckingmaiden [ personal favourite ]
Greyarea – 1 x 1080 Ti

MD5..... : 32.8 GH/s (29.15ms)
SHA1... : 11.2 GH/s (83.96ms)
LM..... : 22.2 GH/s (84.00ms)
NTLM.. : 53.2 GH/s (17.69ms)

£1000 – basically bog standard components plus NVIDIA
Troy Hunt’s List - Density

`./hashcat-4.0.1/hashcat64.bin -O -m 100 ../uncracked-sha1.txt -a3 --increment-min=12 -i`

* Device #1: pthread-AMD A6-9500 RADEON R5, 8 COMPUTE CORES 2C+6G, skipped.
* Device #2: GeForce GTX 1080 Ti, 2792/11170 MB allocatable, 28MCU

Hash.Type...........: SHA1
Hash.Target...........: ../uncracked-sha1.txt
Speed.Dev.#2........: 9675.8 MH/s (11.57ms) [ ~ 10 bn/s ]

07bbb269b7f9b7bf2a649dc5d2472ad9058f5889:hpkhhp999999
Targetted Attacks - hashcat

Search a specific set of characters [a-z]{4-6}

hashcat64.exe -m 1800 -a 3 --increment-min=4 --increment-max=6 --increment $6$mZVuffPMxxxFF0 ?l?l?l?l?l?l

Hybrid attack – dictionary and mask

hashcat64.exe -m 0 -a 6 642395xxxxx4863eca rock.txt ?d?d?d?d

642395cef47664a970441d3c94863eca:powerslave1984
Targetted Attacks - hashcat

Hcmask files are series of ?d?l type clauses

hashcat64.exe -m 1800 -a 3 password.hcmask $6$mZVuffPMxxxFF0


Tries [Pp][@a4][s5$][s5$][oO]rd?d?d?d?s

e.g. P455w0rd123!
Hcmask files

Pp,@a4,s5$,o0,?1?2?3?3w?4rd?a?a
Pp,@a4,s5$,o0,?1?2?3?3w?4rd?a?a?a
Pp,@a4,s5$,o0,?1?2?3?3w?4rd?a?a?a
Pp,@a4,s5$,o0,?a?1?2?3?3W?4rd?a
Pp,@a4,s5$,o0,?a?1?2?3?3W?4rd?a
Pp,@a4,s5$,o0,?a?1?2?3?3W?4rd?a
... $ hashcat-4.0.1\hashcat64.exe -m 1000 users.ntlm -a3 password.hccmask
Tools

Hate_crack (trusted sec) [https://github.com/trustedsec/hate_crack](https://github.com/trustedsec/hate_crack)
(couldn’t get this one to go on Windows – can’t map NVIDIA cards through to Vmware workstation?)

Autohashcat [https://gitlab.com/pentest/autohashcat](https://gitlab.com/pentest/autohashcat)
(saves some keystrokes by running common / sensible params, and tries to identify hash type for you)

Ramdisks can be your friend when preprocessing
Too Long; Didn’t Read

Use PBKDF2, bcrypt, scrypt or Argon, hash server-side.

Try to stop users picking dumb passwords.

Try to stop brute-force attacks against the site. (e.g. present CAPTCHA after 3 failed logins per username)

If email is login, then worry about password stuffing. (e.g. present CAPTCHA after 3 failed logins per source IP)
Questions