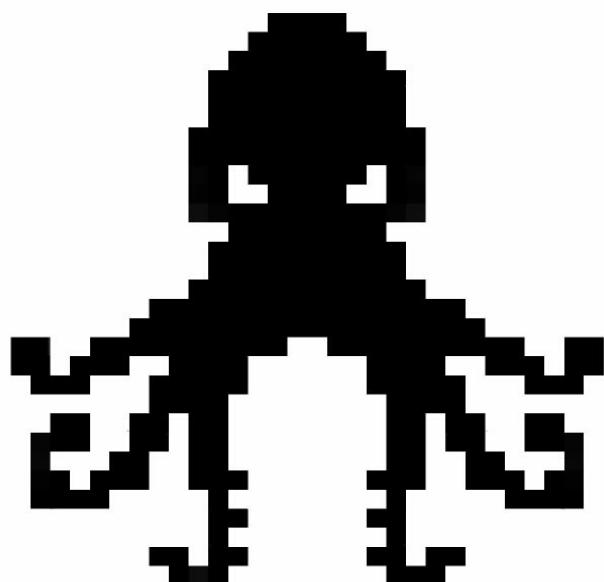


# HASH CRACK

PASSWORD CRACKING MANUAL



NETMIX

v2.0

**Hash Crack.** Copyright © 2017 Netmux LLC

ISBN-10: 1975924584

ISBN-13: 978-1975924584

# **Contents**

Intro

Required Software

Core Hash Cracking Knowledge

Cracking Methodology

Basic Cracking Playbook

Cheat Sheets

Extract Hashes

Password Analysis

Dictionary / Wordlist

Rules & Masks

Foreign Character Sets

Advanced Attacks

Cracking Concepts

Common Hash Examples

Appendix

-Terms

-Online Resources

-John The Ripper Menu

-Hashcat Menu

-Hash Cracking Benchmarks

-Hash Cracking Speed

# **INTRO**

This manual is meant to be a reference guide for cracking tool usage and supportive tools that assist network defenders and pentesters in password recovery (cracking). This manual will not be covering the installation of these tools, but will include references to their proper installation, and if all else fails, Google. Updates and additions to this manual are planned yearly as advancements in cracking evolve. Password recovery is a battle against math, time, cost, and human behavior; and much like any battle, the tactics are constantly evolving.

# **ACKNOWLEDGEMENTS**

This community would not enjoy the success and diversity without the following community members and contributors:

Alexander ‘Solar Designer’ Peslvak, John The Ripper Team, & Community  
Jens ‘atom’ Steube, Hashcat Team, & Devoted Hashcat Forum Community  
Jeremi ‘epixoip’ Gosney  
Korelogic & the Crack Me If You Can Contest  
Robin ‘DigiNinja’ Wood (Pipal & CeWL)  
CynoSure Prime Team  
Chris ‘Unix-ninja’ Aurelio  
Per Thorsheim (PasswordsCon)  
Blandyuk & Rurapenthe (HashKiller Contest)  
Peter ‘iphelix’ Kacherginsky (PACK)  
Royce ‘tychotithonus’ Williams  
‘Waffle’

And many, many, many more contributors. If a name was excluded from the above list please reach out and the next version will give them their due credit.

Lastly, the tools, research, and resources covered in the book are the result of people’s hard work. As such, I HIGHLY encourage all readers to DONATE to help assist in their efforts. A portion of the proceeds from this book will be distributed to

the various researchers/projects.

Suggestions or comments, send your message to [hashcrack@netmux.com](mailto:hashcrack@netmux.com)

## **REQUIRED SOFTWARE**

In order to follow many of the techniques in this manual, you will want to install the following software on your Windows or \*NIX host. This book does not cover how to install said software and assumes you were able to follow the included links and extensive support websites.

### **HASHCAT v3.6 (or newer)**

<https://hashcat.net/hashcat/>

### **JOHN THE RIPPER (v1.8.0 JUMBO)**

<http://www.openwall.com/john/>

### **PACK V0.0.4 (Password Analysis and Cracking Toolkit)**

<http://thesprawl.org/projects/pack/>

### **Hashcat-utils v1.7**

[https://hashcat.net/wiki/doku.php?id=hashcat\\_utils](https://hashcat.net/wiki/doku.php?id=hashcat_utils)

Additionally you will need dictionaries/wordlists and highly recommend the below sources:

### **WEAKPASS DICTIONARY**

<https://weakpass.com/wordlist>

### **CRACKSTATION DICTIONARY**

<https://crackstation.net/buy-crackstation-wordlist-password-cracking-dictionary.htm>

### **SKULL SECURITY WORDLISTS**

<https://wiki.skullsecurity.org/index.php?title=Passwords>

Throughout the manual, generic names have been given to the various inputs required in a cracking commands structure. Legend description is below:

### **COMMAND STRUCTURE LEGEND**

**hashcat** = Generic representation of the various Hashcat binary names

**john** = Generic representation of the John the Ripper binary names

**#type** = Hash type; which is an abbreviation in John or a number in Hashcat

**hash.txt** = File containing target hashes to be cracked

**dict.txt** = File containing dictionary/wordlist

**rule.txt** = File containing permutation rules to alter dict.txt input

**passwords.txt** = File containing cracked password results

**outfile.txt** = File containing results of some functions output

Lastly, as a good reference for testing various hash types to place into your “hash.txt” file, the below sites contain all the various hashing algorithms and example output tailored for each cracking tool:

## **HASHCAT HASH FORMAT EXAMPLES**

[https://hashcat.net/wiki/doku.php?id=example\\_hashes](https://hashcat.net/wiki/doku.php?id=example_hashes)

## **JOHN THE RIPPER HASH FORMAT EXAMPLES**

<http://pentestmonkey.net/cheat-sheet/john-the-ripper-hash-formats>

<http://openwall.info/wiki/john/sample-hashes>

# CORE HASH CRACKING KNOWLEDGE

## ENCODING vs HASHING vs ENCRYPTING

Encoding = transforms data into a publicly known scheme for usability

Hashing = one-way cryptographic function nearly impossible to reverse

Encrypting = mapping of input data and output data reversible with a key

## CPU vs GPU

CPU = 2-72 cores mainly optimized for sequential serial processing

GPU = 1000's of cores with 1000's of threads for parallel processing

## CRACKING TIME = KEYSPACE / HASHRATE

Keyspace: charset<sup>length</sup> (?a?a?a?a = 95<sup>4</sup> = 81,450,625)

Hashrate: hashing function / hardware power (bcrypt / GTX1080 = 13094 H/s)

Cracking Time: 81,450,625 / 13094 H/s = 6,220 seconds

\*Keyspace displayed and Hashrate vary by tool and hardware used

**SALT** = random data that's used as additional input to a one-way function

**ITERATIONS** = the number of times an algorithm is run over a given hash

**HASH IDENTIFICATION:** there isn't a foolproof method for identifying which hash function was used by simply looking at the hash, but there are reliable clues (i.e. \$6\$ sha512crypt). The best method is to know from where the hash was extracted and identify the hash function for that software.

**DICTIONARY/WORDLIST ATTACK** = straight attack uses a precompiled list of words, phrases, and common/unique strings to attempt to match a password.

**BRUTE-FORCE ATTACK** = attempts every possible combination of a given character set, usually up to a certain length.

**RULE ATTACK** = generates permutations against a given wordlist by modifying, trimming, extending, expanding, combining, or skipping words.

**MASK ATTACK** = a form of targeted brute-force attack by using placeholders for

characters in certain positions (i.e. ?a?a?a?l?d?d).

**HYBRID ATTACK** = combines a Dictionary and Mask Attack by taking input from the dictionary and adding mask placeholders (i.e. dict.txt ?d?d?d).

**CRACKING RIG** = from a basic laptop to a 64 GPU cluster, this is the hardware/platform on which you perform your password hash attacks.

## **EXPECTED RESULTS**

Know your cracking rig's capabilities by performing benchmark testing and don't assume you can achieve the same results posted by forum members without using the exact same dictionary, attack plan, or hardware setup. Cracking success largely depends on your ability to use resources efficiently and make calculated trade-offs based on the target hash.

## **DICTIONARY/WORDLIST vs BRUTE-FORCE vs ANALYSIS**

Dictionaries and brute-force are not the end all be all to crack hashes. They are merely the beginning and end of an attack plan. True mastery is everything in the middle, where analysis of passwords, patterns, behaviors, and policies affords the ability to recover that last 20%. Experiment with your attacks and research and compile targeted wordlists with your new knowledge. Do not rely heavily on dictionaries because they can only help you with what is "known" and not the unknown.

# **CRACKING METHODOLOGY**

Following is basic cracking methodology broken into steps, but the process is subject to change based on current/future target information uncovered during the cracking process.

## **1-EXTRACT HASHES**

Pull hashes from target, identify hashing function, and properly format output for your tool of choice.

## **2-FORMAT HASHES**

Format your hashes based on your tool's preferred method. See tool documentation for this guidance. Hashcat, for example, on each line takes <user>:<hash> OR just the plain <hash>.

## **3-EVALUATE HASH STRENGTH**

Using the [Appendix](#) table “Hash Cracking Speed (Slow-Fast)” assess your target hash and it’s cracking speed. If it’s a slow hash, you will need to be more selective at what types of dictionaries and attacks you perform. If it’s a fast hash, you can be more liberal with your attack strategy.

## **4-CALCULATE CRACKING RIG CAPABILITIES**

With the information from evaluating the hash strength, baseline your cracking rig’s capabilities. Perform benchmark testing using John The Ripper and/or Hashcat’s built-in benchmark ability on your rig.

**john --test**

**hashcat -b**

Based on these results you will be able to better assess your attack options by knowing your rigs capabilities against a specific hash. This will be a more accurate result of a hash’s cracking speed based on your rig. It will be useful to save these results for future reference.

## **5-FORMULATE PLAN**

Based on known or unknown knowledge begin creating an attack plan. Included on the next page is a “Basic Cracking Playbook” to get you started.

## **6-ANALYZE PASSWORDS**

After successfully cracking a sufficient amount of hashes analyze the results for any clues or patterns. This analysis may aid in your success on any remaining hashes.

## **7-CUSTOM ATTACKS**

Based on your password analysis create custom attacks leveraging those known clues or patterns. Examples would be custom mask attacks or rules to fit target users' behavior or preferences.

## **8-ADVANCED ATTACKS**

Experiment with Princeprocessor, custom Markov-chains, maskprocessor, or custom dictionary attacks to shake out those remaining stubborn hashes. This is where your expertise and creativity really come into play.

## **9-REPEAT**

Go back to STEP 4 and continue the process over again, tweaking dictionaries, mask, parameters, and methods. You're in the grind at this point and need to rely on skill and luck.

# BASIC CRACKING PLAYBOOK

This is only meant as a basic guide to processing hashes and each scenario will obviously be unique based on external circumstances. For this attack plan we will assume we know the password hashes are raw MD5 and assume we have already captured some plain text passwords of users. If we had no knowledge of plain text passwords we would most likely skip to DICTIONARY/WORDLIST attacks. Lastly, since MD5 is a “Fast” hash we can be more liberal with our attack plan.

## 1-CUSTOM WORDLIST

First compile your known plain text passwords into a custom wordlist file. Pass this to your tool of choice as a straight dictionary attack.

```
hashcat -a 0 -m 0 -w 4 hash.txt custom_list.txt
```

## 2-CUSTOM WORDLIST + RULES

Run your custom wordlist with permutation rules to crack slight variations.

```
hashcat -a 0 -m 0 -w 4 hash.txt custom_list.txt -r best64.rule --loopback
```

## 3 -DICTIONARY/WORDLIST

Perform a broad dictionary attack, looking for common passwords and leaked passwords in well known dictionaries/wordlists.

```
hashcat -a 0 -m 0 -w 4 hash.txt dict.txt
```

## 4-DICTIONARY/WORDLIST + RULES

Add rule permutations to the broad dictionary attack, looking for subtle changes to common words/phrases and leaked passwords.

```
hashcat -a 0 -m 0 -w 4 hash.txt dict.txt -r best64.rule --loopback
```

## 5-CUSTOM WORDLIST + RULES

Add any newly discovered passwords to your custom wordlist and run an attack again with permutation rules, looking any other subtle variations.

```
awk -F ":" '{print $2}' hashcat.potfile >> custom_list.txt  
hashcat -a 0 -m 0 -w 4 hash.txt custom_list.txt -r dive.rule --loopback
```

## **6-MASK**

Now we will use mask attacks included with Hashcat to search the keyspace for common password lengths and patterns, based on the RockYou dataset.

```
hashcat -a 3 -m 0 -w 4 hash.txt rockyou-1-60.hcmask
```

## **7-HYBRID DICTIONARY + MASK**

Using a dictionary of your choice, conduct hybrid attacks looking for larger variations of common words or known passwords by appending/prepending masks to those candidates.

```
hashcat -a 6 -m 0 -w 4 hash.txt dict.txt rockyou-1-60.hcmask
```

```
hashcat -a 7 -m 0 -w 4 hash.txt rockyou-1-60.hcmask dict.txt
```

## **8-CUSTOM WORDLIST + RULES**

Add any newly discovered passwords back to your custom wordlist and run an attack again with permutation rules looking any other subtle variations.

```
awk -F ":" '{print $2}' hashcat.potfile >> custom_list.txt
```

```
hashcat -a 0 -m 0 -w 4 hash.txt custom_list.txt -r dive.rule --loopback
```

## **9-COMBO**

Using a dictionary of your choice, perform a combo attack by individually combining the dictionary's password candidates together to form new candidates.

```
hashcat -a 1 -m 0 -w 4 hash.txt dict.txt dict.txt
```

## **10-CUSTOM HYBRID ATTACK**

Add any newly discovered passwords back to your custom wordlist and perform a hybrid attack against those new acquired passwords.

```
awk -F ":" '{print $2}' hashcat.potfile >> custom_list.txt
```

```
hashcat -a 6 -m 0 -w 4 hash.txt custom_list.txt rockyou-1-60.hcmask
```

```
hashcat -a 7 -m 0 -w 4 hash.txt rockyou-1-60.hcmask custom_list.txt
```

## **11-CUSTOM MASK ATTACK**

By now the easier, weaker passwords may have fallen to cracking, but still some remain. Using PACK (on [pg.51](#)) create custom mask attacks based on your currently cracked passwords. Be sure to sort out masks that match the previous rockyou-1-60.hcmask list.

```
hashcat -a 3 -m 0 -w 4 hash.txt custom_masks.hcmask
```

## **12-BRUTE-FORCE**

When all else fails begin a standard brute-force attack, being selective as to how large a keyspace your rig can adequately brute-force. Above 8 characters this is typically pointless due to hardware limitations and password entropy/ complexity.

```
hashcat -a 3 -m 0 -w 4 hash.txt -i ?a?a?a?a?a?a?a?a
```

# JOHN THE RIPPER CHEAT SHEET

## ATTACK MODES

BRUTEFORCE ATTACK

**john --format=#type hash.txt**

DICTIONARY ATTACK

**john --format=#type --wordlist=dict.txt hash.txt**

MASK ATTACK

**john --format=#type --mask=?l?l?l?l?l?l hash.txt -min-len=6**

INCREMENTAL ATTACK

**john --incremental hash.txt**

DICTIONARY + RULES ATTACK

**john --format=#type --wordlist=dict.txt --rules**

## RULES

--rules=Single

--rules=Wordlist

--rules=Extra

--rules=Jumbo

--rules=KoreLogic

--rules>All

## INCREMENT

--incremental=Digits

--incremental=Lower

--incremental=Alpha

--incremental=Alnum

## PARALLEL CPU or GPU

LIST OpenCL DEVICES

**john --list=opencl-devices**

LIST OpenCL FORMATS

**john --list=formats --format=opencl**

MULTI-GPU (example 3 GPU's)

```
john --format=<OpenCLformat> hash.txt --wordlist=dict.txt --rules --dev=<#> -  
-fork=3  
MULTI-CPU (example 8 cores)  
john --wordlist=dict.txt hash.txt --rules --dev=<#> --fork=8
```

## MISC

BENCHMARK TEST

**john --test**

SESSION NAME

**john hash.txt --session=example\_name**

SESSION RESTORE

**john --restore=example\_name**

SHOW CRACKED RESULTS

**john hash.txt --pot=<john potfile> --show**

WORDLIST GENERATION

**john --wordlist=dict.txt --stdout --external:[filter name] > out.txt**

## BASIC ATTACK METHODOLOGY

1- DEFAULT ATTACK

**john hash.txt**

2- DICTIONARY + RULES ATTACK

**john --wordlist=dict.txt --rules**

3- MASK ATTACK

**john --mask=?l?l?l?l?l?l hash.txt -min-len=6**

4- BRUTEFORCE INCREMENTAL ATTACK

**john --incremental hash.txt**

**HASH TYPES (SORTED ALPHABETICAL)**

7z	HMAC-SHA384	ntlmv2-opencl	Raw-SHA224
7z-opencl	HMAC-SHA512	o5logon	Raw-SHA256
AFS	hMailServer	o5logon-opencl	Raw-SHA256-ng
agilekeychain	hsrp	ODF	Raw-SHA256-opencl
agilekeychain-	IKE	ODF-AES-opencl	Raw-SHA384
opencl	ipb2	ODF-opencl	Raw-SHA512
aix-smd5	KeePass	Office	Raw-SHA512-ng
aix-ssha1	keychain	office2007-opencl	Raw-SHA512-opencl
aix-ssha256	keychain-opencl	office2010-opencl	ripemd-128
aix-ssha512	keyring	office2013-opencl	ripemd-160
asa-md5	keyring-opencl	oldoffice	rsvp
bcrypt	keystore	oldoffice-opencl	Salted-SHA1
bcrypt-opencl	known_hosts	OpenBSD-SoftRAID	sapb
bfegg	krb4	openssl-enc	sapg
Bitcoin	krb5	OpenVMS	scrypt
blackberry-es10	krb5-18	oracle	sha1-gen
Blockchain	krb5pa-md5	oracle11	sha1crypt
blockchain-opencl	krb5pa-md5-opencl	osc	sha1crypt-opencl
bsdicrypt	krb5pa-sha1	Panama	sha256crypt
chap	krb5pa-sha1-	PBKDF2-HMAC-SHA1	sha256crypt-
Citrix_NS10	opencl	PBKDF2-HMAC-	opencl
Clipperz	kwallet	SHA256	sha512crypt
cloudkeychain	LastPass	PBKDF2-HMAC-	sha512crypt-
cq	LM	SHA256-opencl	opencl
CRC32	lotus5	PBKDF2-HMAC-	Siemens-S7
crypt	lotus5-opencl	SHA512	SIP
dahua	lotus85	pbkdf2-hmac-	skein-256
decrypt	LUKS	sha512-opencl	skein-512
decrypt-opencl	MD2	PDF	skey
Django	md4-gen	PFX	Snefru-128
django-scrypt	md5crypt	phpass	Snefru-256
dmd5	md5crypt-opencl	phpass-opencl	SSH
dmg	md5ns	PHPS	SSH-ng
dmg-opencl	mdc2	pix-md5	ssha-opencl
dominosec	MediaWiki	PKZIP	SSHA512
dragonfly3-32	MongoDB	po	STRIP
dragonfly3-64	Mozilla	postgres	strip-opencl
dragonfly4-32	mscash	PST	SunMD5
dragonfly4-64	mscash2	PUTTY	sxc
Drupal7	mscash2-opencl	pwsafe	sxc-opencl
dummy	MSCHAPv2	pwsafe-opencl	Sybase-PROP
dynamic_n	mschapv2-naive	RACF	sybasease
eCryptfs	mssql	RAdmin	tc_aes_xts
EFS	mssql05	RAKP	tc_ripemd160
eigrp	mssql12	RAKP-opencl	tc_sha512
EncFS	mysql	rar	tc_whirlpool
encfs-opencl	mysql-sha1	rar-opencl	tcp-md5
EPI	mysql-sha1-opencl	RAR5	Tiger
EPiServer	mysqlna	RAR5-opencl	tripcode
fde	net-md5	Raw-Blake2	VNC
FormSpring	net-sha1	Raw-Keccak	vtp
Fortigate	nethalflm	Raw-Keccak-256	wbb3
gost	netlm	Raw-MD4	whirlpool
gpg	netlmv2	Raw-MD4-opencl	whirlpool0
gpg-opencl	netntlm	Raw-MD5	whirlpool1
HAVAL-128-4	netntlm-naive	Raw-MD5-opencl	WoWSRP
HAVAL-256-3	netntlmv2	Raw-MD5u	wpapsk
hdaa	nk	Raw-SHA	wpapsk-opencl
HMAC-MD5	nsldap	Raw-SHA1	xsha
HMAC-SHA1	NT	Raw-SHA1-Linkedin	xsha512
HMAC-SHA224	nt-opencl	Raw-SHA1-ng	XSHA512-opencl
HMAC-SHA256	nt2	Raw-SHA1-opencl	ZIP
			zip-opencl

# HASHCAT CHEAT SHEET

## ATTACK MODES

DICTIONARY ATTACK

**hashcat -a 0 -m #type hash.txt dict.txt**

DICTIONARY + RULES ATTACK

**hashcat -a 0 -m #type hash.txt dict.txt -r rule.txt**

COMBINATION ATTACK

**hashcat -a 1 -m #type hash.txt dict1.txt dict2.txt**

MASK ATTACK

**hashcat -a 3 -m #type hash.txt ?a?a?a?a?a?a**

HYBRID DICTIONARY + MASK

**hashcat -a 6 -m #type hash.txt dict.txt ?a?a?a?a**

HYBRID MASK + DICTIONARY

**hashcat -a 7 -m #type hash.txt ?a?a?a?a dict.txt**

## RULES

RULEFILE -r

**hashcat -a 0 -m #type hash.txt dict.txt -r rule.txt**

MANIPULATE LEFT -j

**hashcat -a 1 -m #type hash.txt left\_dict.txt right\_dict.txt -j <option>**

MANIPULATE RIGHT -k

**hashcat -a 1 -m #type hash.txt left\_dict.txt right\_dict.txt -k <option>**

## INCREMENT

DEFAULT INCREMENT

**hashcat -a 3 -m #type hash.txt ?a?a?a?a?a --increment**

INCREMENT MINIMUM LENGTH

**hashcat -a 3 -m #type hash.txt ?a?a?a?a?a --increment-min=4**

INCREMENT MAX LENGTH

**hashcat -a 3 -m #type hash.txt ?a?a?a?a?a --increment-max=5**

## MISC

BENCHMARK TEST (HASH TYPE)

**hashcat -b -m #type**

SHOW EXAMPLE HASH

**hashcat -m #type --example-hashes**

DISABLE PASSWORD LENGTH LIMIT (Max Length 256)

**hashcat -a 0 -m #type --length-limit-disable hash.txt dict.txt**

SESSION NAME

**hashcat -a 0 -m #type --session <uniq\_name> hash.txt dict.txt**

SESSION RESTORE

**hashcat -a 0 -m #type --restore --session <uniq\_name> hash.txt dict.txt**

SHOW KEYSpace

**hashcat -a 0 -m #type --keyspace hash.txt dict.txt -r rule.txt**

OUTPUT RESULTS FILE -o

**hashcat -a 0 -m #type -o results.txt hash.txt dict.txt**

CUSTOM CHARSET -1 -2 -3 -4

**hashcat -a 3 -m #type hash.txt -1 ?l?u -2 ?l?d?s ?l?2?a?d?u?l**

ADJUST PERFORMANCE -w

**hashcat -a 0 -m #type -w <1-4> hash.txt dict.txt**

## **BASIC ATTACK METHODOLOGY**

1- DICTIONARY ATTACK

**hashcat -a 0 -m #type hash.txt dict.txt**

2- DICTIONARY + RULES

**hashcat -a 0 -m #type hash.txt dict.txt -r rule.txt**

3- HYBRID ATTACKS

**hashcat -a 6 -m #type hash.txt dict.txt ?a?a?a?a**

4- BRUTEFORCE

**hashcat -a 3 -m #type hash.txt ?a?a?a?a?a?a?a**

## **HASH TYPES (SORTED ALPHABETICAL)**

6600 1Password, agilekeychain

8200 1Password, cloudkeychain

14100 3DES (PT = \$salt, key = \$pass)

11600 7-Zip

6300 AIX {smd5}

6400 AIX {ssha256}

6500 AIX {ssha512}

6700 AIX {ssha1}

5800 Android PIN

8800 Android FDE < v4.3

12900 Android FDE (Samsung DEK)

1600	Apache \$apr1\$
125	ArubaOS
12001	Atlassian (PBKDF2-HMAC-SHA1)
13200	AxCrypt
13300	AxCrypt in memory SHA1
3200	bcrypt \$2*\$, Blowfish(Unix)
600	BLAKE2-512
12400	BSDiCrypt, Extended DES
11300	Bitcoin/Litecoin wallet.dat
12700	Blockchain, My Wallet
15200	Blockchain, My Wallet, V2
15400	ChaCha20
2410	Cisco-ASA
500	Cisco-IOS \$1\$
5700	Cisco-IOS \$4\$
9200	Cisco-IOS \$8\$
9300	Cisco-IOS \$9\$
2400	Cisco-PIX
8100	Citrix Netscaler
12600	ColdFusion 10+
10200	Cram MD5
11500	CRC32
14000	DES (PT = \$salt, key = \$pass)
1500	descrypt, DES(Unix), Traditional DES
8300	DNSSEC (NSEC3)
124	Django (SHA-1)
10000	Django (PBKDF2-SHA256)
1100	Domain Cached Credentials (DCC), MS Cache
2100	Domain Cached Credentials 2 (DCC2), MS Cache 2
15300	DPAPI masterkey file v1 and v2
7900	Drupal7
12200	eCryptfs
141	EPiServer 6.x < v4
1441	EPiServer 6.x > v4
15600	Ethereum Wallet, PBKDF2-HMAC-SHA256
15700	Ethereum Wallet, PBKDF2-SCRYPT
15000	FileZilla Server >= 0.9.55

7000	Fortigate (FortiOS)
6900	GOST R 34.11-94
11700	GOST R 34.11-2012 (Streebog) 256-bit
11800	GOST R 34.11-2012 (Streebog) 512-bit
7200	GRUB 2
50	HMAC-MD5 (key = \$pass)
60	HMAC-MD5 (key = \$salt)
150	HMAC-SHA1 (key = \$pass)
160	HMAC-SHA1 (key = \$salt)
1450	HMAC-SHA256 (key = \$pass)
1460	HMAC-SHA256 (key = \$salt)
1750	HMAC-SHA512 (key = \$pass)
1760	HMAC-SHA512 (key = \$salt)
5100	Half MD5
5300	IKE-PSK MD 5
5400	IKE-PSK SHA1
2811	IPB (Invison Power Board)
7300	IPMI2 RAKP HMAC-SHA1
14700	iTunes Backup < 10.0
14800	iTunes Backup >= 10.0
4800	iSCSI CHAP authentication, MD5(Chap)
15500	JKS Java Key Store Private Keys (SHA1)
11	Joomla < 2.5.18
400	Joomla > 2.5.18
15100	Juniper/NetBSD shalcrypt
22	Juniper Netscreen/SSG (ScreenOS)
501	Juniper IVE
13400	Keepass 1 (AES/Twofish) and Keepass 2 (AES)
7500	Kerberos 5 AS-REQ Pre-Auth etype 23
13100	Kerberos 5 TGS-REP etype 23
6800	Lastpass + Lastpass sniffed
3000	LM
8600	Lotus Notes/Domino 5
8700	Lotus Notes/Domino 6
9100	Lotus Notes/Domino 8
14600	LUKS

900	MD4
0	MD5
10	md5(\$pass.\$salt)
20	md5(\$salt.\$pass)
30	md5(unicode(\$pass).\$salt)
40	md5(\$salt.unicode(\$pass))
3710	md5(\$salt.md5(\$pass))
3800	md5(\$salt.\$pass.\$salt)
3910	md5(md5(\$pass).md5(\$salt))
4010	md5(\$salt.md5(\$salt.\$pass))
4110	md5(\$salt.md5(\$pass.\$salt))
2600	md5(md5(\$pass))
4400	md5(sha1(\$pass))
4300	md5(strtoupper(md5(\$pass))))
500	md5crypt \$1\$, MD5(Unix)
9400	MS Office 2007
9500	MS Office 2010
9600	MS Office 2013
9700	MS Office <= 2003 \$0
9710	MS Office <= 2003 \$0
9720	MS Office <= 2003 \$0
9800	MS Office <= 2003 \$3
9810	MS Office <= 2003 \$3
9820	MS Office <= 2003 \$3
12800	MS-AzureSync PBKDF2-HMAC-SHA256
131	MSSQL(2000)
132	MSSQL(2005)
1731	MSSQL(2012)
1731	MSSQL(2014)
3711	Mediawiki B type
2811	MyBB
11200	MySQL CRAM (SHA1)
200	MySQL323
300	MySQL4.1/MySQL5
1000	NTLM
5500	NetNTLMv1
5500	NetNTLMv1 + ESS

5600	NetNTLMv2
101	nsldap, SHA-1(Base64), Netscape LDAP SHA
111	nsldaps, SSHA-1(Base64), Netscape LDAP SSHA
13900	OpenCart
21	osCommerce
122	OSX V10.4, OSX V10.5, OSX V10.6
1722	OSX V10.7
7100	OSX V10.8, OSX V10.9, OSX v10.10
112	Oracle S: Type (Oracle 11+)
3100	Oracle H: Type (Oracle 7+)
12300	Oracle T: Type (Oracle 12+)
11900	PBKDF2-HMAC-MD5
12000	PBKDF2-HMAC-SHA1
10900	PBKDF2-HMAC-SHA256
12100	PBKDF2-HMAC-SHA512
10400	PDF 1.1 - 1.3 (Acrobat 2 - 4)
10410	PDF 1.1 - 1.3 (Acrobat 2 - 4), collider #1
10420	PDF 1.1 - 1.3 (Acrobat 2 - 4), collider #2
10500	PDF 1.4 - 1.6 (Acrobat 5 - 8)
10600	PDF 1.7 Level 3 (Acrobat 9)
10700	PDF 1.7 Level 8 (Acrobat 10 - 11)
400	phpBB3
400	phpass
2612	PHPS
5200	Password Safe v3
9000	Password Safe v2
133	PeopleSoft
13500	PeopleSoft Token
99999	Plaintext
12	PostgreSQL
11100	PostgreSQL CRAM (MD5)
11000	PrestaShop
4522	PunBB
8500	RACF
12500	RAR3-hp
13000	RAR5

9900	Radmin2
7600	Redmine
6000	RipeMD160
7700	SAP CODVN B (BCODE)
7800	SAP CODVN F/G (PASSCODE)
10300	SAP CODVN H (PWDSALTEDHASH) iSSHA-1
8900	scrypt
1300	SHA-224
1400	SHA-256
1411	SSHA-256(Base64), LDAP {SSHA256}
5000	SHA-3(Keccak)
10800	SHA-384
1700	SHA-512
100	SHA1
14400	SHA1(CX)
110	sha1(\$pass.\$salt)
120	sha1(\$salt.\$pass)
130	sha1(unicode(\$pass).\$salt)
140	sha1(\$salt.unicode(\$pass))
4500	sha1(sh1(\$pass))
4520	sha1(\$salt.sha1(\$pass))
4700	sha1(md5(\$pass))
4900	sha1(\$salt.\$pass.\$salt)
1410	sha256(\$pass.\$salt)
1420	sha256(\$salt.\$pass)
1440	sha256(\$salt.unicode(\$pass))
1430	sha256(unicode(\$pass).\$salt)
7400	sha256crypt \$5\$, SHA256(Unix)
1710	sha512(\$pass.\$salt)
1720	sha512(\$salt.\$pass)
1740	sha512(\$salt.unicode(\$pass))
1730	sha512(unicode(\$pass).\$salt)
1800	sha512crypt \$6\$, SHA512(Unix)
11400	SIP digest authentication (MD5)
121	SMF (Simple Machines Forum)
1711	SSHA-512(Base64), LDAP {SSHA512}

10100	SipHash
14900	Skip32
23	Skype
8000	Sybase ASE
62XY	TrueCrypt
X	1 = PBKDF2-HMAC-RipeMD160
X	2 = PBKDF2-HMAC-SHA512
X	3 = PBKDF2-HMAC-Whirlpool
X	4 = PBKDF2-HMAC-RipeMD160 + boot-mode
Y	1 = XTS 512 bit pure AES
Y	1 = XTS 512 bit pure Serpent
Y	1 = XTS 512 bit pure Twofish
Y	2 = XTS 1024 bit pure AES
Y	2 = XTS 1024 bit pure Serpent
Y	2 = XTS 1024 bit pure Twofish
Y	2 = XTS 1024 bit cascaded AES-Twofish
Y	2 = XTS 1024 bit cascaded Serpent-AES
Y	2 = XTS 1024 bit cascaded Twofish-Serpent
Y	3 = XTS 1536 bit all
2611	vBulletin < V3.8.5
2711	vBulletin > V3.8.5
137XY	VeraCrypt
X	1 = PBKDF2-HMAC-RipeMD160
X	2 = PBKDF2-HMAC-SHA512
X	3 = PBKDF2-HMAC-Whirlpool
X	4 = PBKDF2-HMAC-RipeMD160 + boot-mode
X	5 = PBKDF2-HMAC-SHA256
X	6 = PBKDF2-HMAC-SHA256 + boot-mode
Y	1 = XTS 512 bit pure AES
Y	1 = XTS 512 bit pure Serpent
Y	1 = XTS 512 bit pure Twofish
Y	2 = XTS 1024 bit pure AES
Y	2 = XTS 1024 bit pure Serpent
Y	2 = XTS 1024 bit pure Twofish
Y	2 = XTS 1024 bit cascaded AES-Twofish
Y	2 = XTS 1024 bit cascaded Serpent-AES

Y	2 = XTS 1024 bit cascaded Twofish-Serpent
Y	3 = XTS 1536 bit all
8400	WBB3 (Woltlab Burning Board)
2500	WPA/WPA2
2501	WPA/WPA2 PMK
6100	Whirlpool
13600	WinZip
13800	Windows 8+ phone PIN/Password
400	Wordpress
21	xt:Commerce

## **TERMINAL COMMAND CHEAT SHEET**

**Ctrl + u**

delete everything from the cursor to the beginning of the line

**Ctrl + w**

delete the previous word on the command line before the cursor

**Ctrl + l**

clear the terminal window

**Ctrl + a**

jump to the beginning of the command line

**Ctrl + e**

move your cursor to the end of the command line

**Ctrl + r**

search command history in reverse, continue pressing key sequence to continue backwards search. Esc when done or command found.

## **FILE MANIPULATION CHEAT SHEET**

Extract all lowercase strings from each line and output to wordlist.

**sed 's/[a-z]\*//g' wordlist.txt > outfile.txt**

Extract all uppercase strings from each line and output to wordlist.

**sed 's/[A-Z]\*//g' wordlist.txt > outfile.txt**

Extract all lowercase/uppercase strings from each line and output to wordlist.

**sed 's/[a-zA-Z]\*//g' wordlist.txt > outfile.txt**

Extract all digits from each line in file and output to wordlist.

**sed 's/[0-9]\*//g' wordlist.txt > outfile.txt**

Watch hashcat potfile or designated output file live.

**watch -n .5 tail -50 <hashcat.potfile or outfile.txt>**

Pull 100 random samples from wordlist/passwords for visual analysis.

**shuf -n 100 file.txt**

Print statistics on length of each string and total counts per length.

**awk '{print length}' file.txt | sort -n | uniq -c**

Remove all duplicate strings and count how many times they are present; then sort by their count in descending order.

**uniq -c file.txt | sort -nr**

Command to create quick & dirty custom wordlist with length 1-15 character words from a designated website into a sorted and counted list.

**curl -s <http://www.netmux.com> | sed -e 's/<[^>]\*>//g' | tr " " "\n" | tr -dc '[:alnum:]\\n\\r' | tr '[:upper:]' '[:lower:]' | cut -c 1-15 | sort | uniq -c | sort -nr**

MD5 each line in a file (Mac OSX).

**while read line; do echo -n \$line | md5; done < infile.txt > outfile.txt**

MD5 each line in a file (\*Nix).

**while read line; do echo -n \$line | md5sum; done < infile.txt | awk -F " " '{print \$1}' > outfile.txt**

Remove lines that match from each file and only print remaining from file2.txt.

**grep -vwF -f file1.txt file2.txt**

Take two ordered files, merge and remove duplicate lines and maintain ordering.

**nl -ba -s ' : ' file1.txt >> outfile.txt**

**nl -ba -s ' : ' file2.txt >> outfile.txt**

**sort -n outfile.txt | awk -F ":" '{print \$2}' | awk '!seen[\$0]++ > final.txt**

Extract strings of a specific length into a new file/wordlist.

**awk 'length == 8' file.txt > 8len-out.txt**

Convert alpha characters on each line in file to lowercase characters.

**tr [A-Z] [a-z] < infile.txt > outfile.txt**

Convert alpha characters on each line in file to uppercase characters.

**tr [a-z] [A-Z] < infile.txt > outfile.txt**

Split a file into separate files by X number of lines per outfile.

```
split -d -l 3000 infile.txt outfile.txt
```

Reverse the order of each character of each line in the file.

```
rev infile.txt > outfile.txt
```

Sort each line in the file from shortest to longest.

```
awk '{print length,$0}' "" $0; }' infile.txt | sort -n | cut -d ' ' -f2-
```

Sort each line in the file from longest to shortest.

```
awk '{print length,$0}' "" $0; }' infile.txt | sort -r -n | cut -d ' ' -f2-
```

Substring matching by converting to HEX and then back to ASCII.

(Example searches for **5** character strings from file1.txt found as a substring in **20** character strings in file2.txt)

```
strings file1.txt | xxd -u -ps -c 5 | sort -u > out1.txt
strings file2.txt | xxd -u -ps -c 20 | sort -u > out2.txt
grep -Ff out1.txt out2.txt | xxd -r -p > results.txt
```

Clean dictionary/wordlist of newlines and tabs.

```
cat dict.txt | tr -cd "[[:print :]]/[n/t]\n" > outfile.txt
```

# **SYSTEM HASH EXTRACTION**

## **WINDOWS**

### **METERPRETER HASHDUMP**

Post exploitation dump local SAM database:

```
meterpreter> run post/windows/gather/hashdump
```

### **CREDDUMP**

<https://github.com/Neohapsis/creddump7>

Three modes of attack: **cachedump**, **lsadump**, **pwdump**

## **DUMP DOMAIN CACHED CREDENTIALS**

Save Windows XP/Vista/7 registry hive tables

```
C:\WIND0WS\system32>reg.exe save HKLM\SAM sam_backup.hiv
```

```
C:\WIND0WS\system32>reg.exe save HKLM\SECURITY sec_backup.hiv
```

```
C:\WIND0WS\system32>reg.exe save HKLM\system sys_backup.hiv
```

Run creddump tools against the saved hive files:

```
cachedump.py <system hive> <security hive> <Vista/7>
```

(Vista/7)

```
cachedump.py sys_backup.hiv sec_backup.hiv true
```

(XP)

```
cachedump.py sys_backup.hiv sec_backup.hiv false
```

## **DUMP LSA SECRETS**

```
lsadump.py sys_backup.hiv sec_backup.hiv
```

## **DUMP LOCAL PASSWORD HASHES**

```
pwdump.py sys_backup.hiv sec_backup.hiv
```

## **MIMIKATZ**

Post exploitation commands must be executed from SYSTEM level privileges.

```
mimikatz # privilege::debug  
mimikatz # token::whoami  
mimikatz # token::elevate  
mimikatz # lsadump::sam
```

Save Windows XP/Vista/7 registry tables

```
C:\WIND0WS\system32>reg.exe save HKLM\SAM sam_backup.hiv  
C:\WIND0WS\system32>reg.exe save HKLM\SECURITY  
C:\WIND0WS\system32>reg.exe save HKLM\system  
mimikatz # lsadump::sam SystemBkup.hiv SamBkup.hiv
```

## **\*NIX**

Requires root level privileges.

### **cat /etc/shadow**

Example \*NIX sha512crypt hash

```
root:$6$52450745$k5ka2p8bFuSmoVTltz0yyuaREkkKBcCNqoDKzYiDL9RaE:
```

## **MAC OSX 10.5-10.7**

Manual OSX Hash Extraction

```
dscl localhost -read /Search/Users/<username>|grep GeneratedUID|cut -c15-cat  
/var/db/shadow/hash/<GUID> | cut -c169-216 > osx_hash.txt
```

## **MAC OSX 10.8-10.12**

Manual OSX Hash Extraction

```
sudo defaults read /var/db/dslocal/nodes/Default/users/<username>.plist  
ShadowHashData|tr -dc '0-9a-f'|xxd -p -r|plutil -convert xml1 --o -
```

## **Scripted OSX Hash Extraction**

HASHCAT

<https://gist.github.com/nueh/8252572>

```
sudo plist2hashcat.py /var/db/dslocal/nodes/Default/users/<username>.plist
```

JOHN

<https://github.com/truongkma/ctf-tools/blob/master/John/run/ml2john.py>

```
sudo ml2john.py /var/db/dslocal/nodes/Default/users/<username>.plist
```

## PCAP HASH EXTRACTION

### LOCAL NETWORK AUTHENTICATION

#### PCREDZ

Extracts network authentication hashes from pcaps.

Single pcap file:

**Pcredz -f example.pcap**

Multiple pcap files in a directory:

**Pcredz -d /path/to/pcaps**

Interface to listen on and collect:

**Pcredz -i eth0**

### WPA/WPA2 PSK AUTHENTICATION

Capture the 4-way WPA/WPA2 authentication handshake.

#### AIRMON-NG / AIRODUMP-NG / AIREPLAY-NG

Step 1: Create monitoring interface mon0 Ex) interface wlan0

**airmon-ng start wlan0**

Step 2: Capture packets to file on target AP channel Ex) channel 11

**airodump-ng mon0 --write capture.cap -c 11**

Step 3: Start deauth attack against BSSID Ex) bb:bb:bb:bb:bb:bb

**aireplay-ng --deauth 0 -a bb:bb:bb:bb:bb:bb mon0**

Step 4: Wait for confirmation to appear at top of terminal:

**CH 11 ][ Elapsed: 25 s ][ <DATE / TIME) ][ WPA handshake: \*\***

Step 5: Extract handshake into JOHN or HASHCAT format:

#### JOHN FORMAT EXTRACT

Step1: cap2hccap.bin -e '<ESSID>' capture.cap capture\_out.hccap

Step2: hccap2john capture\_out.hccap > jtr\_capture

#### HASHCAT FORMAT EXTRACT

cap2hccapx.bin capture.cap capture\_out.hccapx

#### MISC WLAN TOOLS

-----  
**HCXTOOLS:** capture and convert packets from wlan devices for use with Hashcat.

## DATABASE HASH EXTRACTION

SQL queries require administrative privileges.

ORACLE 10g R2

```
SELECT username, password FROM dba_users WHERE  
username='<username>';
```

ORACLE 11g R1

```
SELECT name, password, spare4 FROM sys.user$ WHERE  
name='<username>';
```

MySQL4.1 / MySQL5+

```
SELECT User, Password FROM mysql.user INTO OUTFILE '/tmp/hash.txt' ;
```

MSSQL(2012), MSSQL(2014)

```
SELECT SL.name,SL.password_hash FROM sys.sql_logins AS SL;
```

POSTGRES

```
SELECT username, passwd FROM pg_shadow;
```

## MISCELLANEOUS HASH EXTRACTION

John The Ripper Jumbo comes with various programs to extract hashes:

NAME	DESCRIPTION
1password2john.py	1Password vault hash extract
7z2john.py	7zip encrypted archive hash extract
androidfde2john.py	Android FDE convert disks/images into JTR format
aix2john.py	AIX shadow file /etc/security/passwd
apex2john.py	Oracle APEX hash formating
bitcoin2john.py	Bitcoin old wallet hash extraction (check btcrecover)
blockchain2john.py	Blockchain wallet extraction
cisco2john.pl	Cisco config file ingestion/ extract

cracf2john.py	CRACF program crafc.txt files
dmg2john.py	Apple encrypted disk image
ecryptfs2john.py	eCryptfs disk encryption software
efs2john.py	Windows Encrypting File System (EFS) extract
encfs2john.py	EncFS encrypted filesystem userspace
gpg2john	PGP symmetrically encrypted files
hccap2john	Convert pcap capture WPA file to JTR format
htdigest2john.py	HTTP Digest authentication
ikescan2john.py	IKE PSK SHA256 authentication
kdcdump2john.py	Key Distribution Center (KDC) servers
keepass2john	Keepass file hash extract
keychain2john.py	Processes input Mac OS X keychain files
keyring2john	Processes input GNOME Keyring files
keystore2john.py	Output password protected Java KeyStore files
known_hosts2john.py	SSH Known Host file
kwallet2john.py	KDE Wallet Manager tool to manage the passwords
ldif2john.pl	LDAP Data Interchange Format (LDIF)
lion2john.pl	Converts an Apple OS X Lion plist file
lion2john-alt.pl	
lotus2john.py	Lotus Notes ID file for Domino
luks2john	Linux Unified Key Setup (LUKS) disk encryption
mcafee_epo2john.py	McAfee ePolicy Orchestrator password generator
ml2john.py	Convert Mac OS X 10.8 and later plist hash
mozilla2john.py	Mozilla Firefox, Thunderbird, SeaMonkey extract
odf2john.py	Processes OpenDocument Format ODF files
office2john.py	Microsoft Office [97-03, 2007, 2010, 2013) hashes
openbsd_softraid2john.py	OpenBSD SoftRAID hash
openssl2john.py	OpenSSL encrypted files
pcap2john.py	PCAP extraction of various protocols
pdf2john.py	PDF encrypted document hash extract
pfx2john	PKCS12 files
putty2john	PuTTY private key format
pwsafe2john	Password Safe hash extract
racf2john	IBM RACF binary database files
radius2john.pl	RADIUS protocol shared secret

rar2john	RAR 3.x files input into proper format
sap2john.pl	Converts password hashes from SAP systems
sipdump2john.py	Processes sipdump output files into JTR format
ssh2john	SSH private key files
sshng2john.py	SSH-ng private key files
strip 2john.py	Processes STRIP Password Manager database
sxc2john.py	Processes SXC files
truecrypt_volume2john	TrueCrypt encrypted disk volume
uaf2john	Convert OpenVMS SYSUAF file to unix-style file
vncpcap2john	TightVNC/RealVNCpcapsvs c3.3, 3.7 and 3.8 RFB
wpapcap2john	Converts PCAP or IVS2 files to JtR format
zip2john	Processes ZIP files extracts hash into JTR format

# PASSWORD ANALYSIS

## HISTORICAL PASSWORD ANALYSIS TIPS

- The average password length is 7-9 characters.
- The average English word is 5 characters long.
- The average person knows 50,000 to 150,000 words.
- 50% chance a user's password will contain one or more vowels.
- Women prefer personal names in their passwords, and men prefer hobbies.
- Most likely to be used symbols: ~, !, @, #, \$, %, &, \*, and ?
- If a number, it's usually a 1 or 2, sequential, and will likely be at the end.
- If more than one number it will usually be sequential or personally relevant.
- If a capital letter, it's usually the beginning, followed by a vowel.
- 66% of people only use 1 - 3 passwords for all online accounts.
- One in nine people have a password based on the common Top 500 list.
- Western countries use lowercase passwords and Eastern countries prefer digits.

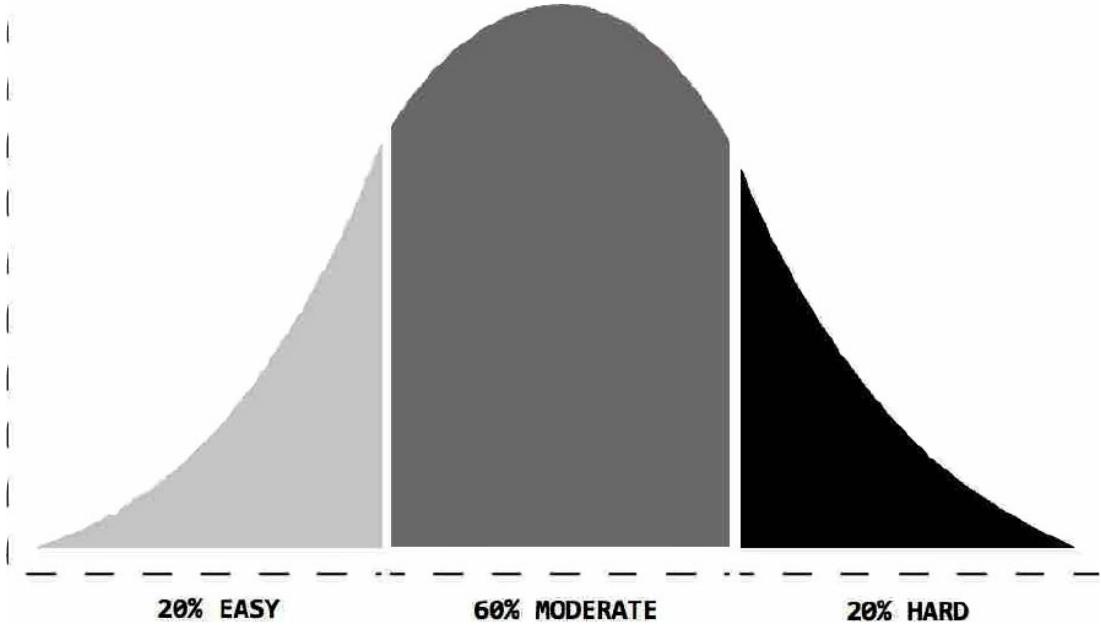
## 20-60-20 RULE

20-60-20 rule is a way to view the level of difficulty typically demonstrated by a large password dump, having characteristics that generally err on the side of a Gaussian Curve, mirroring the level of effort to recover said password dump.

**20%** of passwords are **easily** guessed dictionary words or known common passwords.

**60%** of passwords are **moderate** to slight variations of the earlier 20%.

**20%** of passwords are **hard**, lengthy, complex, or of unique characteristics.



## EXAMPLE HASHES & PASSWORDS

This is an example list of passwords to help convey the variation and common complexities seen with typical password creation. It also shows individual user biases to aid in segmenting your attacks to be tailored toward a specific user.

#	HASH (MySQL 323)	PASSWORD	MASK
1	24CA195A48D85A11	BlueParrot345	?u?1?1?1?u?1?1?1?1?1?d?d?d
2	020261361E63A3FE	r0b3rt2017!	?u?d?1?d?1?1?d?d?d?s
3	42DF901246D99098	Ralph@Netmux.com	?u?1?1?1?1?1?s?u?1?1?1?1?1?s?1?1?
4	7B6C1F5173EB4DD6	RedFerret789	?u?1?1?u?1?1?1?1?1?d?d?d
5	01085B1F3C3F49D2	Jennifer1981!	?u?1?1?1?1?1?1?1?1?d?d?d?s
6	080DBDB42AE6C3D7	7482Sacrifice	?d?d?d?d?u?1?1?1?1?1?1?1?
7	111C232F67BC52BB	234CrowBlack	?d?d?d?u?1?1?1?u?1?1?1?1?
8	1F3EF64F35878031	brownbooklamp	?1?1?1?1?1?1?1?1?1?1?1?1?
9	4A659CDA12E9F2F1	Solitaire7482	?u?1?1?1?1?1?1?1?1?d?d?d?
10	0B034EC713F89A68	password123	?1?1?1?1?1?1?1?1?d?d?d
11	28619CFE477235DE	5713063528	?d?d?d?d?d?d?d?d?d?
12	60121DFD757911C3	74821234WorldCup	?d?d?d?d?d?d?d?d?u?1?1?1?1?u?1?1?
13	6E84950D7FC8D13B	!q@w#e\$r%t^y	?s?1?s?1?s?1?s?1?s?1?
14	33F82FA23197EBD3	1qaz2wsx3edc!#@	?d?1?1?1?d?1?1?1?d?1?1?1?s?s?
15	544B4D3449C08787	X9z-2Qp-3qm-WGN	?u?d?1?-d?u?1?-d?1?1?-u?u?u?

ALICE PASSWORDS	#(2,5,8,11,14)
BOB PASSWORDS	#(1,4,7,10,13)
CRAIG PASSWORDS	#(3,6,9,12,15)

## CRACKING TIPS FOR EACH PASSWORD

### ALICE PASSWORDS

R0b3rt2017!	L33t Speak Name + Date !
Jennifer1981!	Simple Name + Date !
brownbooklamp	3 Common Words Phrase
5713063528	Possible Phone Number
1qaz2wsx3edc!@#	Vertical+ Horizontal Walk

### SIMPLE ANALYSIS

### BASIC ATTACK STRATEGY

Simple Dictionary + Rule Attack
Hybrid Attack Dict + Mask
Combinator3 + Simple Dictionary
Custom or Simple Digit Mask
Straight Dict or Kwprocessor

### BOB PASSWORDS

BlueParrot345	Color+Animal+3_Seq_digits
RedFerret789	Color+Animal+3_Seq_digits
234CrowBlack	3_Seq_digits+Animal+Color
password123	Lowercase_word+3_Seq_digits
!q@w#e\$r%t^y	Vertical Keyboard Walk

### SIMPLE ANALYSIS

### BASIC ATTACK STRATEGY

Combo Dictionary + Rule Attack
Combo Dictionary + 3 Digit Mask
3 Digit Mask + Combo Dictionary
Straight Dictionary Attack
Straight Dictionary Attack

### CRAIG PASSWORDS

Ralph@Netmux.com	Name + Domain Name
7482Sacrifice	4 Digits + Simple Word
Solitaire7482	Simple Word + 4 Digits
74821234WorldCup	7482+ 4_Seq_digits+ Dict
X9z-2Qp-3qm-WGN	Structured Random Pattern

### SIMPLE ANALYSIS

### BASIC ATTACK STRATEGY

Hybrid Attack Dict + @Netmux.com
Hybrid Attack Mask + Dict
Hybrid Attack Dict + Mask
Hybrid Attack 7482?d?d?d?d + Dict
Custom Mask X9z-2Qp-?d?l?l-?u?u?u

\*This List of passwords will be referenced throughout the book and the List can also be found online at: <https://github.com/netmux/HASH-CRACK>

## PASSWORD PATTERN ANALYSIS

A password can contain many useful bits of information related to it's creator and their tendencies/patterns, but you have to break down the structure to decipher the meaning. This analysis process could be considered a sub-category of *Text Analytics*' and split into three pattern categories I'm calling:

### **Basic Pattern, Macro-Pattern, & Micro-Pattern.**

\*Refer to *EXAMPLE HASH & PASSWORDS* chapter (pg.29) for numbered examples.

**Basic Pattern** : visually obvious when compared to similar groupings (i.e. language and base word/words & digits). Let's look at Alice's passwords (2,5):

-Each password uses a name: R0b3rt & Jennifer

-Ending in a 4 digit date with common special character: **2017!** & **1981!**

**!TIP!** This type of basic pattern lends itself to a simple dictionary and L33T speak rule appending dates or hybrid mask attack appending Dict+ ?d?d?d?d?s

**Macro-Pattern** : statistics about the passwords underlying structure such as length and character set. Let's look at Craig's passwords (6,9):

7482Sacrifice

Solitaire7482

-Length structure can be summed up as: **4 Digits + 7 Alpha & 7 Alpha + 4 Digits**

-Uses charsets ?l?u?d , so we may be able to ignore special characters.

-Basic Pattern preference for the numbers **7482** and Micro-Pattern for capitalizing words beginning in "S".

**!TIP!** You can assume this user is 'unlikely' to have a password less than 12 characters (+1 char) and the 4 digit constant lowers the work to 8 chars. These examples lend themselves to a Hybrid Attack (Dict + 7482) or (7482 + Dict).

**Micro-Pattern** : subtlety and context which expresses consistent case changes, themes, and personal data/interest. Let's look at Bob's passwords (1,4)

BlueParrot345

RedFerret789

-Each password begins with a color: Blue & Red

-Second word is a type of animal: **Parrot** & **Ferret**

-Consistent capitalization of all words

-Lastly, ending in a 3 digit sequential pattern: **345** & **789**

**!TIP!** This pattern lends itself to a custom combo dictionary and rule or hybrid mask attack appending sequential digits ?d?d?d

So when analyzing passwords be sure to group passwords and look for patterns such as language, base word/digit, length, character sets, and subtle themes with possible contextual meaning or password policy restrictions.

# WESTERN COUNTRY PASSWORD ANALYSIS

Password Length Distribution based on large corpus of English website dumps:

7=15%    8=27%    9=15%    10=12%    11=4.8%    12=4.9%    13=.6%

Character frequency analysis of a large corpus of English texts:

etaoinshrdlcumwfgypbvkjxqz

Character frequency analysis of a large corpus of English password dumps:

aeionrlstmc dyhubkgpjvfwzxq

Top Western password masks out of a large corpus of English website dumps:

?1?1?1?1?1?1?1	6-Lowercase
?1?1?1?1?1?1?1	7-Lowercase
?1?1?1?1?1?1?1?1	8-Lowercase
?d?d?d?d?d?d	6-Digits
?1?1?1?1?1?1?1?1?1?1?1?1?1?1?1	12-Lowercase
?1?1?1?1?1?1?1?1?1?1?1?1	9-Lowercase
?1?1?1?1?1?1?1?1?1?1?1?1	10-Lowercase
?1?1?1?1?1	5-Lowercase
?1?1?1?1?1?1?1?d?d?1?1?1?1?1	6-Lowercase + 2-Digits + 4-Lowercase
?d?d?d?d?d?d?d?d?1?1?1?1?1	8-Digits + 4-Lowercase
?1?1?1?1?1?1?d?d	5-Lowercase + 2-Digits
?d?d?d?d?d?d?d?d?	8-Digits
?1?1?1?1?1?1?1?d?d	6-Lowercase + 2-Digits
?1?1?1?1?1?1?1?1?1?d?d	8-Lowercase + 2-Digits

# EASTERN COUNTRY PASSWORD ANALYSIS

Password Length Distribution based on large corpus of Chinese website dumps:

7=21%    8=22%    9=12%    10=12%    11=4.2%    12=.9%    13=.5%

Character frequency analysis of a large corpus of Chinese texts:

aineo hglw uyszx qcdj mbtfrkpv

Character frequency analysis of a large corpus of Chinese password dumps:

inauhegoyszdjmxwqbctlpfrkv

Top Eastern password masks out of a large corpus of Chinese website dumps:

?d?d?d?d?d?d?d?d	8-Digits
?d?d?d?d?d?d	6-Digits
?d?d?d?d?d?d	7-Digits
?d?d?d?d?d?d?d?d	9-Digits
?d?d?d?d?d?d?d?d?d	10-Digits
?1?1?1?1?1?1?1?1?	8-Digits
?d?d?d?d?d?d?d?d?d	11-Digits
?1?1?1?1?1?1?	6-Lowercase
?1?1?1?1?1?1?1?1?	9-Lowercase
?1?1?1?1?1?1?	7-Lowercase
?1?1?1?d?d?d?d?d?	3-Lowercase + 6-Digits
?1?1?d?d?d?d?d?	2-Lowercase + 6-Digits
?1?1?1?1?1?1?1?1?1?	10-Lowercase
?d?d?d?d?d?d?d?d?d	12-Digits

## PASSWORD MANAGER ANALYSIS

### Apple Safari Password Generator

-default password 15 characters with “-” & four groups of three random  
u=ABCDEFHJKLMNPQRSTU VWXYZ l=abcdefghijklmnoprstuvwxyz and  
d=3456789

Example) X9z-2Qp-3qm-WGN

XXX-XXX-XXX-XXX where X = ?u?l?d

### Dashlane

-default password 12 characters using just letters and digits.

Example) Up0k9ZAj54Kt

XXXXXXXXXXXXXX where X = ?u?l?d

### KeePass

-default password 20 characters using uppercase, lowercase, digits, and special.

Example) \$Zt={EcgQ.Umf)R,C7XF

XXXXXXXXXXXXXXXXXXXX where X = ?u?l?d?s

### LastPass

-default password 12 characters using at least one digit, uppercase and lowercase.

Example) msfNdkG29n38

XXXXXXXXXXXX where X = ?u?l?d

## **RoboForm**

-default password 15 characters using uppercase, lowercase, digits, and special with a minimum of 5 digits.

Example) **871v2%%4F0w31zJ**

XXXXXXXXXXXXXXXXXXXX where X = ?u?l?d?s

## **Symantec Norton Identity Safe**

-default password 8 characters using uppercase, lowercase, and digits.

Example) **Ws81f0Zg**

XXXXXXXX where X = ?u?l?d

## **True Key**

-default password 16 characters using uppercase, lowercase, digits, and special.

Example) **1B1H:9N+@>+sgWs**

XXXXXXXXXXXXXXXXXXXX where X = ?u?l?d?s

## **1Password v6**

-default password 24 characters using uppercase, lowercase, digits, and special.

Example) **cTmM7Tzm6iPhCdpMu. \* V ] , VP**

XXXXXXXXXXXXXXXXXXXXXXXXXXXX where X = ?u?l?d?s

# **PACK (Password Analysis and Cracking Kit)**

<http://thesprawl.org/projects/pack/>

## **STATSGEN**

---

Generate statistics about the most common length, percentages, character-set and other characteristics of passwords from a provided list.

**python statsgen.py passwords.txt**

### **STATSGEN OPTIONS**

<b>-o &lt;file.txt&gt;</b>	output stats and masks to file
<b>--hiderare</b>	hide stats of passwords with less than 1% of occurrence
<b>--minlength=</b>	minimum password length for analysis
<b>--maxlength=</b>	maximum password length for analysis

--charset=	password char filter: loweralpha, upperalpha, numeric, special
--simplemask=	password mask filter: string, digit, special

## **STATSGEN EXAMPLES**

**Output stats of passwords.txt to file example.mask:**

python statsgen.py passwords.txt -o example.mask

**Hide less than 1% occurrence; only analyze passwords 7 characters and greater:**

python statsgen.py passwords.txt --hiderare --minlength=7 -o example.mask

**Stats on passwords with only numeric characters:**

python statsgen.py passwords.txt --charset=numeric

## **ZXCVBN (LOW-BUDGET PASSWORD STRENGTH ESTIMATION)**

A realistic password strength (entropy) estimator developed by Dropbox.

<https://github.com/dropbox/zxcvbn>

## **PIPAL (THE PASSWORD ANALYSER)**

Password analyzer that produces stats and pattern frequency analysis.

<https://digi.ninja/projects/pipal.php>

**pipal.rb -o outfile.txt passwords.txt**

## **PASSPAT (PASSWORD PATTERN IDENTIFIER)**

Keyboard pattern analysis tool for passwords. <https://digi.ninja/projects/passpat.php>

**passpat.rb --layout us passwords.txt**

# CHARACTER FREQUENCY ANALYSIS

Character frequency analysis is the study of the frequency of letters or groups of letters in a corpus/text. This is the basic building block of Markov chains.

## Character-Frequency-CLI-Tool

Tool to analyze a large list of passwords and summarize the character frequency.  
<https://github.com/jcchurch/Character-Frequency-CLI-Tool>

**charfreq.py <options> passwords.txt**

Options:    -w Window size to analyze, default=1  
              -r Rolling window size  
              -s Skip spaces, tabs, newlines

# ONLINE PASSWORD ANALYSIS RESOURCES

## WEAKPASS

Analyzes public password dumps and provides efficient dictionaries for download.

<http://weakpass.com/>

## PASSWORD RESEARCH

Important password security and authentication research papers in one place.

<http://www.passwordresearch.com/>

## THE PASSWORD PROJECT

Compiled analysis of larger password dumps using PIPAL and PASSPAL tools.

[http://www.thepasswordproject.com/leaked\\_password\\_lists\\_and\\_dictionaries](http://www.thepasswordproject.com/leaked_password_lists_and_dictionaries)

# **DICTIONARY / WORDLIST**

## **DOWNLOAD RESOURCES**

### **WEAKPASS**

<http://weakpass.com/wordlist>

### **CRACKSTATION DICTIONARY**

<https://crackstation.net/buy-crackstation-wordlist-password-cracking-dictionary.htm>

### **HAVE I BEEN PWNED**

\*You'll have to crack the SHA1's

<https://haveibeenpwned.com/passwords>

### **SKULL SECURITY WORDLISTS**

<https://wiki.skullsecurity.org/index.php?title=Passwords>

### **CAPSOP**

<https://wordlists.capsop.com/>

### **UNIX-NINJA DNA DICTIONARY**

\*Dictionary link at bottom of article\*

[https://www.unix-ninja.com/p>Password\\_DNA](https://www.unix-ninja.com/p>Password_DNA)

### **PROBABLE-WORDLIST**

<https://github.com/berzerk0/Probable-Wordlists>

### **EFF-WORDLIST**

Long-list (7,776 words) & Short-list (1,296 words)

[https://www.eff.org/files/2016/07/18/eff\\_large\\_wordlist.txt](https://www.eff.org/files/2016/07/18/eff_large_wordlist.txt)

[https://www.eff.org/files/2016/09/08/eff\\_short\\_wordlist\\_1.txt](https://www.eff.org/files/2016/09/08/eff_short_wordlist_1.txt)

## **RAINBOW TABLES**

\*Rainbow Tables are for the most part obsolete but provided here for reference\*

<http://project-rainbowcrack.com/table.htm>

## **WORDLIST GENERATION**

### **JOHN THE RIPPER**

**Generate wordlist that meets complexity specified in the complex filter.**

```
john --wordlist=dict.txt --stdout --external : [filter name] > outfile.txt
```

### **STEMMING PROCESS**

Stripping characters from a password list to reach the “stem” or base word/words of the candidate password. Commands are from “File Manipulation Cheat Sheet”.

Extract all lowercase strings from each line and output to wordlist.

```
sed 's/[a-z]*//g' passwords.txt > outfile.txt
```

Extract all uppercase strings from each line and output to wordlist.

```
sed 's/[A-Z]*//g' passwords.txt > outfile.txt
```

Extract all lowercase/uppercase strings from each line and output to wordlist.

```
sed 's/[a-zA-Z]*//g' passwords.txt > outfile.txt
```

Extract all digits from each line in file and output to wordlist.

```
sed 's/[0-9]*//g' passwords.txt > outfile.txt
```

## **HASHCAT UTILS**

[https://hashcat.net/wiki/doku.php?id=hashcat\\_utils](https://hashcat.net/wiki/doku.php?id=hashcat_utils)

### **COMBINATOR**

**Combine multiple wordlists with each word appended to the other.**

```
combinator.bin dict1.txt dict2.txt > combined_dict.txt
```

```
combinator3.bin dict1.txt dict2.txt dict3.txt > combined_dict.txt
```

## **CUTB**

**Cut the specific length off the existing wordlist and pass it to STDOUT.**

cutb.bin offset [length] < infile.txt > outfile.txt

Example to cut first 4 characters in a wordlist and place into a file:

cutb.bin 0 4 < dict.txt > outfile.txt

## **RLI**

**Compares a file against another file or files and removes all duplicates.**

rli dict1.txt outfile.txt dict2.txt

## **REQ**

**Dictionary candidates are passed to stdout if it matches an specified password group criteria/requirement. Groups can be added together (i.e. 1 + 2 = 3 )**

1 = LOWER (abcdefghijklmnopqrstuvwxyz)

2 = UPPER (ABCDEFGHIJKLMNOPQRSTUVWXYZ)

4 = DIGIT (012345789)

8 = OTHER (All other characters not matching 1,2, or 4)

This example would stdout all candidates matching upper and lower characters

req.bin 3 < dict.txt

## **COMBIPOW**

**Creates “unique combinations” of a custom dictionary; dictionary cannot be greater than 64 lines; option -1 limits candidates to 15 characters.**

combipow.bin dict.txt

combipow.bin -1 dict.txt

## **EXPANDER**

**Dictionary into stdin is parsed and split into all its single chars (up to 4) and sent to stdout.**

expander.bin < dict.txt

## **LEN**

**Each candidate in a dictionary is checked for length and sent to stdout.**

len.bin <min len> <max len> < dict.txt

This example would send to stdout all candidates 5 to 10 chars long.

```
len.bin 5 10 < dict.txt
```

## MORPH

**Auto generates insertion rules for the most frequent chains of characters**

```
morph.bin dict.txt depth width pos_min pos_max
```

## PERMUTE

**Dictionary into stdin parsed and run through “The Countdown QuickPerm Algorithm”**

```
permute.bin < dict.txt
```

## CRUNCH

**Wordlist generator can specify a character set and generate all possible combinations and permutations.**

<https://sourceforge.net/projects/crunch-wordlist/>

```
crunch <min length> <max length> <character set> -o outfile.txt
```

```
crunch 8 8 0123456789ABCDEF -o crunch_wordlist.txt
```

## TARGETED WORDLISTS

### CeWL

**Custom wordlist generator scrapes & compiles keywords from websites.**

<https://digi.ninja/projects/cewl.php>

Example scan depth of 2 and minimum word length of 5 output to wordlist.txt

```
cewl -d 2 -m 5 -w wordlist.txt http://<target website>
```

### SMEEGESCAPE

Text file and website scraper which generates custom wordlists from content.

<http://www.smeegesec.com/2014/01/smeegescape-text-scraper-and-custom.html>

Compile unique keywords from text file and output into wordlist.

**S meegeScrape.py -f file.txt -o wordlist.txt**

Scrape keywords from target website and output into wordlist.

**S meegeScrape.py -u http://<target website> -si -o wordlist.txt**

## GENERATE PASSWORD HASHES

Use the below methods to generate hashes for specific algorithms.

**HASHCAT**

<https://github.com/hashcat/hashcat/tree/master/tools>

**test.pl passthrough <#type> <#> dict.txt**

**MDXFIND**

<https://hashes.org/mdxfind.php>

**echo | mdxfind -z -h '<#type>' dict.txt**

## LYRICPASS (Song Lyrics Password Generator)

<https://github.com/initstring/lyricpass>

**Generator using song lyrics from chosen artist to create custom dictionary.**

**python lyricpass.py “Artist Name” artist-dict.txt**

## CONVERT WORDLIST ENCODING

**HASHCAT**

Force internal wordlist encoding from X

**hashcat -a 0 -m #type hash.txt dict.txt --encoding-from=utf-8**

Force internal wordlist encoding to X

```
hashcat -a 0 -m #type hash.txt dict.txt --encoding-to=iso-8859-15
```

## ICONV

### Convert wordlist into language specific encoding

```
iconv -f <old_encode> -t <new_encode> < dict.txt | sponge dict.txt.enc
```

## CONVERT HASHCAT \$HEX OUTPUT

### Example of converting \$HEX[] entries in hashcat.potfile to ASCII

```
grep '$HEX' hashcat.pot | awk -F ":" '{print $2}' | perl -ne ' if($_ =~ m/^$HEX\\([A-Fa-f0-9]+)\\])/ {print pack ("H*", $1), "\n"}'
```

## EXAMPLE CUSTOM DICTIONARY CREATION

1-Create a custom dictionary using CeWL from [www.netmux.com](http://www.netmux.com) website:

```
cewl -d 2 -m 5 -w custom_dict.txt http://www.netmux.com
```

2-Combine the new custom\_dict.txt with the Google 10,000 most common English words: <https://github.com/first20hours/google-10000-english>

```
cat google-1000.txt >> custom_dict.txt
```

3-Combine with Top 196 passwords from “Probable Wordlists”:

```
github.com/berzerk0/Probable-Wordlists/blob/master/Real-Passwords
```

```
cat Topl96-probable.txt >> custom_dict.txt
```

4-Combo the Topl96-probable.txt together using Hashcat-util “combinator.bin” and add it to our custom\_dict.txt

```
combinator.bin Topl96-probable.txt Topl96-probable.txt >> custom_dict.txt
```

5-Run the best64.rule from Hashcat on Top196-probable.txt and send that output

into our custom dictionary:

```
hashcat -a 0 Top196-probable.txt -r best64.rule --stdout >> custom_dict.txt
```

Can you now come up with an attack that can crack this hash?

**e4821d16a298092638ddb7cadc26d32f**

*\*Answer in the Appendix*

## **RULES & MASKS**

## **RULE FUNCTIONS**

Following are compatible between Hashcat, John The Ripper, & PasswordPro

[https://hashcat.net/wiki/doku.php?id=rule\\_based\\_attack](https://hashcat.net/wiki/doku.php?id=rule_based_attack)

NAME	FUNCTION	DESCRIPTION
Nothing	:	Do nothing
Lowercase	l	Lowercase all letters
Uppercase	u	Uppercase all letters
Capitalize	c	Capitalize the first letter and lower the rest
Invert Capitalize	C	Lowercase first character, uppercase rest
Toggle Case	t	Toggle the case of all characters in word.
Toggle @	TN	Toggle the case of characters at position N
Reverse	r	Reverse the entire word
Duplicate	d	Duplicate entire word
Duplicate N	pN	Append duplicated word N times
Reflect	f	Duplicate word reversed
Rotate Left	{	Rotates the word left.
Rotate Right	}	Rotates the word right
AppendChar	\$X	Append character X to end
PrependChar	^X	Prepend character X to front
Truncate left	[	Deletes first character
Truncate right	]	Deletes last character
Delete @ N	DN	Deletes character at position N
Extract range	xNM	Extracts M characters, starting at position N
Omit range	ONM	Deletes M characters, starting at position N
Insert @ N	iNX	Inserts character X at position N
Overwrite @ N	oNX	Overwrites character at position N with X
Truncate @ N	'N	Truncate word at position N
Replace	sXY	Replace all instances of X with Y
Purge	@X	Purge all instances of X
Duplicate first N	zN	Duplicates first character N times
Duplicate last N	ZN	Duplicates last character N times
Duplicate all	q	Duplicate every character
Extract memory	XNMI	Insert substring of length M starting at position N of word in memory at position I
Append memory	4	Append word in memory to current word
Prepend memory	6	Prepend word in memory to current word
Memorize	M	Memorize current word

## RULES TO REJECT PLAINS

[https://hashcat.net/wiki/doku.php?id=rule\\_based\\_attack](https://hashcat.net/wiki/doku.php?id=rule_based_attack)

NAME	FUNCTION	DESCRIPTION
Reject less	<N	Reject plains of length greater than N
Reject greater	>N	Reject plains of length less than N
Reject contain	!X	Reject plains which contain char X
Reject not contain	/X	Reject plains which do not contain char X
Reject equal first	(X	Reject plains which do not start with X
Reject equal last	)X	Reject plains which do not end with X
Reject equal at	=NX	Reject plains which do not have char X at position N
Reject contains	%NX	Reject plains which contain char X less than N times
Reject contains	Q	Reject plains where the memory saved matches current word

## IMPLEMENTED SPECIFIC FUNCTIONS

Following functions are not compatible with John The Ripper & PasswordPro

NAME	FUNCTION	DESCRIPTION
Swap front	k	Swaps first two characters
Swap back	K	Swaps last two characters
Swap @ N	*XY	Swaps character X with Y
Bitwise shift left	LN	Bitwise shift left character @ N
Bitwise shift right	RN	Bitwise shift right character @ N
Ascii increment	+N	Increment character @ N by 1 ascii value
Ascii decrement	-N	Decrement character @ N by 1 ascii value
Replace N + 1	.N	Replaces character @ N with value at @ N plus 1
Replace N - 1	,N	Replaces character @ N with value at @ N minus 1
Duplicate block front	yN	Duplicates first N characters
Duplicate block back	YN	Duplicates last N characters
Upper Lower	E	Lower case the whole line, then upper case the first letter and every letter after a space

# RULE ATTACK CREATION

## EXAMPLE RULE CREATION & OUTPUT

Below we apply basic rules to help explain the expected output when using rules.

<u>WORD</u>	<u>RULE</u>	<u>OUTPUT</u>
password	\$1	password1
password	^!^1	1!password
password	so0 sa@	p@ssw0rd
password	c so0 sa@ \$1	P@ssw0rd1
password	u r	DROWSSAP

## MASKPROCESSOR HASHCAT-UTIL

<https://github.com/hashcat/maskprocessor>

Maskprocessor can be used to generate a long list of rules very quickly.

Example rule creation of prepend digit and special char to dictionary candidates (i.e. ^1^! , ^2^@ , . . .) :

```
mp64.bin '^?d ^?s' -o rule.txt
```

Example creating rule with custom charset appending lower, uppercase chars and all digits to dictionary candidates (i.e. \$a \$Q \$1 , \$e \$ A \$2, . . .) :

```
mp64.bin -1 aeiou -2 QAZWSX '$?1 $?2 $?d'
```

## GENERATE RANDOM RULES ATTACK (i.e. “Raking”)

```
hashcat -a 0 -m #type -g <#rules> hash.txt dict.txt
```

## GENERATE RANDOM RULES FILE USING HASHCAT-UTIL

```
generate-rules.bin <#rules> <seed> | ./cleanup-rules.bin [1=CPU,2=GPU] > out.txt
```

```
generate-rules.bin 1000 42 | ./cleanup-rules.bin 2 > out.txt
```

## SAVE SUCCESSFUL RULES/METRICS

```
hashcat -a 0 -m #type --debug-mode=l --debug-file=debug.txt hash.txt -r rule.txt
```

## SEND RULE OUTPUT TO STDOUT / VISUALLY VERIFY RULE OUTPUT

```
hashcat dict.txt -r rule.txt --stdout
```

```
john --wordlist=dict.txt --rules=example --stdout
```

## PACK (Password Analysis and Cracking Kit) RULE CREATION

<http://thesprawl.org/projects/pack/>

### RULEGEN

---

Advanced techniques for reversing source words and word mangling rules from already cracked passwords by continuously recycling/expanding generated rules and words. Outputs rules in Hashcat format.

<http://thesprawl.org/research/automatic-password-rule-analysis-generation/>

\*\*Ensure you install ‘AppleSpell’ ‘aspell\* module using packet manager\*\*

```
python rulegen.py --verbose --password P@ssw0rdl23
```

### RULEGEN OPTIONS

-b rockyou

Output base name. The following files will be generated: basename.words, basename.rules and basename.stats

-w wiki.dict

Use a custom wordlist for rule analysis.

-q, --quiet

Don’t show headers.

--threads=THREADS

Parallel threads to use for processing.

#### Fine tune source word generation::

--maxworddist=10

Maximum word edit distance (Levenshtein)

--maxwords=5

Maximum number of source word candidates to consider

--morewords

Consider suboptimal source word candidates

--simplewords

Generate simple source words for given passwords

#### Fine tune rule generation::

--maxrulelen=10

Maximum number of operations in a single rule

--maxrules=5

Maximum number of rules to consider

--morerules

Generate suboptimal rules

--simplerules	Generate simple rules insert,delete,replace
--bruterules	Bruteforce reversal and rotation rules (slow)

Fine tune spell checker engine::

--providers=aspell,myspell	Comma-separated list of provider engines
----------------------------	--

Debugging options::

-v, --verbose	Show verbose information.
-d, --debug	Debug rules.
--password	Process the last argument as a password not a file.
--word=Password	Use a custom word for rule analysis
--hashcat	Test generated rules with hashcat-cli

## **RULEGEN EXAMPLES**

**Analysis of a single password to automatically detect rules and potential source word used to generate a sample password:**

python rulegen.py --verbose --password P@ssw0rdl23

**Analyze passwords.txt and output results:**

python rulegen.py passwords.txt -q

analysis.word - unsorted and non-unique source words  
 analysis-sorted.word - occurrence sorted and unique source words  
 analysis.rule - unsorted and non-unique rules  
 analysis-sorted.rule - occurrence sorted and unique rules

## **HASHCAT INCLUDED RULES**

## **Approx # Rules**

Incisive-leetspeak.rule	15,487
InsidePro-HashManager.rule	6,746
InsidePro-PasswordsPro.rule	3,254
T0XlC-insert_00-99_1950-2050_toprules_0_F.rule	4,019
T0XlC-insert_space_and_special_0_F.rule	482
T0XlC-insert_top_100_passwords_1_G.rule	1,603
T0XlC.rule	4,088
T0XlCv1.rule	11,934
best64.rule	77

combinator.rule	59
d3ad0ne.rule	34,101
dive.rule	99,092
generated.rule	14,733
generated2.rule	65,117
leetspeak.rule	29
oscommerce.rule	256
rockyou-30000.rule	30,000
specific.rule	211
toggles1.rule	15
toggles2.rule	120
toggles3.rule	575
toggles4.rule	1,940
toggles5.rule	4,943
unix-ninja-leetspeak.rule	3,073

---

<b>JOHN INCLUDED RULES</b>	<b>Approx # Rules</b>
All (Jumbo + KoreLogic)	7,074,300
Extra	17
Jumbo (Wordlist + Single + Extra + NT + OldOffice)	226
KoreLogic	7,074,074
Loopback (NT + Split)	15
NT	14
OldOffice	1
Single	169
Single-Extra (Single + Extra + OldOffice)	187
Split	1
Wordlist	25

<http://www.openwall.com/john/doc/RULES.shtml>

## CUSTOM RULE PLANS

<u>L33TSP3@K RULES</u>	<u>2 DIGIT APPEND</u>	<u>\$APPEND/^PREPEND DATE</u>
so0	\$0 \$0	\$1 \$9 \$9 \$5
si1	\$0 \$1	^5 ^9 ^9 ^1
se3	\$0 \$2	\$2 \$0 \$0 \$0
ss5	\$1 \$1	^0 ^0 ^0 ^2
sa@	\$1 \$2	\$2 \$0 \$1 \$0
s00	\$1 \$3	^0 ^1 ^0 ^2
sI1	\$2 \$1	\$2 \$0 \$1 \$7
sE3	\$2 \$2	^7 ^1 ^0 ^2
sS5	\$6 \$9	\$2 \$0 \$1 \$8
sA@	\$9 \$9	^8 ^1 ^0 ^2

<u>TOP 10 dive.rule</u>	<u>TOP 10 best64.rule</u>	<u>TOP 10 rockyou.rule</u>
c	:	:
l	r	\$1
u	u	r
T0	T0	\$2
\$1	\$0	\$1 \$2 \$3
} } } }	\$1	\$1 \$2
p3	\$2	\$3
[	\$3	\$7
\$.	\$4	^1
]	\$5	\$1 \$3

## MASK ATTACK CREATION

### **DEBUG / VERIFY MASK OUTPUT**

hashcat -a 3 ?a?a?a?a --stdout

john --mask=?a?a?a?a --stdout

### **HASHCAT MASK ATTACK CREATION**

**Example usage:**

hashcat -a 3 -m #type hash.txt <mask>

**Example brute-force all possible combinations 7 characters long:**

hashcat -a 3 -m #type hash.txt ?a?a?a?a?a?a?a

**Example brute-force all possible combinations 1 - 7 characters long:**

hashcat -a 3 -m #type hash.txt -i ?a?a?a?a?a?a?a

**Example brute-force uppercase first letter, 3 unknown middle characters, and ends in 2 digits (i.e. Pass12):**

```
hashcat -a 3 -m #type hash.txt ?u?a?a?a?d?d
```

**Example brute-force known first half word “secret” and unknown ending:**  
hashcat -a 3 -m #type hash.txt secret?a?a?a?a

**Example hybrid mask (leftside) + wordlist (rightside) (i.e. 123!Password)**  
hashcat -a 7 -m #type hash.txt ?a?a?a?a dict.txt

**Example wordlist (leftside) + hybrid mask (rightside) (i.e. Password123!)**  
hashcat -a 6 -m #type hash.txt dict.txt ?a?a?a?a

## HASHCAT CUSTOM CHARSETS

Four custom buffer charsets to create efficient targeted mask attacks defined as: **-1 -2 -3 -4**

**Example custom charset targeting passwords that only begin in a,A,b,B,or c,C , 4 unknown middle characters, and end with a digit (i.e. al7z#q7):** hashcat -a 3 -m #type hash.txt -1 abcABC ?l?a?a?a?a?d

**Example custom charset targeting passwords that only begin in uppercase or lowercase, 4 digits in the middle, and end in special character !,@,\$ (i.e. W7462! or f1234\$):**

```
hashcat -a 3 -m #type hash.txt -1 ?u?l -2 !@$ ?l?d?d?d?d?2
```

**Example using all four custom charsets at once (i.e. pow!12er):**

```
hashcat -a 3 -m #type hash.txt -1 qwer -2 poiu -3 123456 -4 !@#$% ?2?2?1?4?3?3?1?1
```

## JOHN MASK ATTACK CREATION

**Example usage:**

```
john --format=#type hash.txt --mask=<mask>
```

**Example brute-force all possible combinations up to 7 characters long:**

```
john --format=#type hash.txt --mask=?a?a?a?a?a?a?a
```

**Example brute-force uppercase first letter, 3 unknown middle characters, and ends in 2 digits (i.e. Pass12):**

```
john --format=#type hash.txt --mask=?u?a?a?a?d?d
```

**Example brute-force known first half word “secret” and unknown ending:**

```
john --format=#type hash.txt --mask=secret?a?a?a?a
```

**Example mask (leftside) + wordlist (rightside) (i.e. 123!Password)**  
john --format=#type hash.txt --wordlist=dict.txt --mask=?a?a?a?a?w

**Example wordlist (leftside) + mask (rightside) (i.e. Password123!)**  
john --format=#type hash.txt --wordlist=dict.txt --mask=?w?a?a?a?a

## JOHN CUSTOM CHARSETS

Nine custom buffer charsets to create efficient targeted mask attacks defined as: **-1 -2 -3 -4 -5 -6 -7 -8 -9**

**Example custom charset targeting passwords that only begin in a,A,b,B,or c,C , 4 unknown middle characters, and end with a digit (i.e. a17z#q7):**  
john --format=#type hash.txt -1=abcABC --mask=?l?a?a?a?a?d

**Example custom charset targeting passwords that only begin in uppercase or lowercase, 4 digits in the middle, and end in special character !,@,\$ (i.e. W7462! or f1234\$):**

john --format=#type hash.txt -1=?u?l -2=!@\$ --mask=?l?d?d?d?d?2

**Example using four custom charsets at once (i.e. pow!12er):**

john --format=#type hash.txt -1=qwer -2=poiu -3=123456 -4=!@#\$%, --mask=?2?2?l?4?3?3?1?1

## HASHCAT MASK CHEAT SHEET

?l = lowercase	= 26 chars	= abcdefghijklmnopqrstuvwxyz
?u = uppercase	= 26 chars	= ABCDEFGHIJKLMNOPQRSTUVWXYZ
?d = digits	= 10 chars	= 0123456789
?s = special	= 33 chars	= «space»!"#\$%&'()*+, -./:;=>?@[\\]^_`{ }~
?a = all	= 95 chars	= lowercase+uppercase+digits+special
?h = hex	= 16 chars	= 0123456789abcdef
?H = HEX	= 16 chars	= 0123456789ABCDEF
?b = byte	= 256 byte	= 0x00 - 0xff

## JOHN MASK CHEAT SHEET

**?l** = lowercase = 26 chars = abcdefghijklmnopqrstuvwxyz  
**?u** = uppercase = 26 chars = ABCDEFGHIJKLMNOPQRSTUVWXYZ  
**?d** = digits = 10 chars = 0123456789  
**?s** = special = 33 chars = «space»!"#\$%&'()\*+,-./:;=>?@[\\]^\_`{|}~  
**?a** = all = 95 chars = lowercase+uppercase+digits+special  
**?h** = hex = 0x80 - 0xff  
**?A** = all valid characters in the current code page  
**?h** = all 8-bit (0x80-0xff)  
**?H** = all except the NULL character  
**?L** = non-ASCII lower-case letters  
**?U** = non-ASCII upper-case letters  
**?D** = non-ASCII "digits"  
**?S** = non-ASCII "specials"  
**?w** = Hybrid mask mode placeholder for the original word

## MASK FILES

Hashcat allows for the creation of mask files by placing custom masks, one per line, in a text file with “.hcmask” extension.

HASHCAT BUILT-IN MASK FILES	Approx # Masks
8char-11-1u-1d-1s-compliant.hcmask	40,824
8char-11-1u-1d-1s-noncompliant.hcmask	24,712
rockyou-1-60.hcmask	836
rockyou-2-1800.hcmask	2,968
rockyou-3-3600.hcmask	3,971
rockyou-4-43200.hcmask	7,735
rockyou-5-86400.hcmask	10,613
rockyou-6-864000.hcmask	17,437
rockyou-7-2592000.hcmask	25,043

## WESTERN COUNTRY TOP MASKS

?1?1?1?1?1?1	6-Lowercase
?1?1?1?1?1?1?1	7-Lowercase
?1?1?1?1?1?1?1?1	8-Lowercase
?d?d?d?d?d?d	6-Digits
?1?1?1?1?1?1?1?1?1	12-Lowercase
?1?1?1?1?1?1?1?1	9-Lowercase
?1?1?1?1?1?1?1?1?1	10-Lowercase
?1?1?1?1?1?1	5-Lowercase
?1?1?1?1?1?1?d?d?1?1?1?1	6-Lowercase + 2-Digits + 4-Lowercase
?d?d?d?d?d?d?d?1?1?1?1	8-Digits + 4-Lowercase
?1?1?1?1?1?1?d?d	5-Lowercase + 2-Digits
?d?d?d?d?d?d?d?d	8-Digits
?1?1?1?1?1?1?d?d	6-Lowercase + 2-Digits
?1?1?1?1?1?1?d?d	8-Lowercase + 2-Digits

## EASTERN COUNTRY TOP MASKS

?d?d?d?d?d?d?d?d	8-Digits
?d?d?d?d?d?d	6-Digits
?d?d?d?d?d?d?d	7-Digits
?d?d?d?d?d?d?d?d	9-Digits
?d?d?d?d?d?d?d?d?d	10-Digits
?1?1?1?1?1?1?1?1?1	8-Digits
?d?d?d?d?d?d?d?d?d	11-Digits
?1?1?1?1?1?1	6-Lowercase
?1?1?1?1?1?1?1?1?1	9-Lowercase
?1?1?1?1?1?1?1	7-Lowercase
?1?1?1?d?d?d?d?d?d	3-Lowercase + 6-Digits
?1?1?d?d?d?d?d?d	2-Lowercase + 6-Digits
?1?1?1?1?1?1?1?1?1?1	10-Lowercase
?d?d?d?d?d?d?d?d?d?d	12-Digits

## PACK (Password Analysis and Cracking Kit) MASK CREATION

<http://thesprawl.org/projects/pack/>

## MASKGEN

---

MaskGen allows you to automatically generate pattern-based mask attacks from

known passwords and filter by length and desired cracking time.

## **python maskgen.py example.mask**

### **MASKGEN OPTIONS**

```
-t          target cracking time of all masks combined (seconds)
-o <file.hcmask> output masks to a file
--showmasks    show matching masks
```

#### **Individual Mask Filter Options:**

```
--minlength=8      Minimum password length
--maxlength=8      Maximum password length
--mintime=3600      Minimum mask runtime (seconds)
--maxtime=3600      Maximum mask runtime (seconds)
--mincomplexity=1   Minimum complexity
--maxcomplexity=100 Maximum complexity
--minoccurrence=1   Minimum occurrence
--maxoccurrence=100 Maximum occurrence
```

#### **Mask Sorting Options:**

```
--optindex        sort by mask optindex (default)
--occurrence      sort by mask occurrence
--complexity      sort by mask complexity
```

#### **Check mask coverage:**

```
--checkmasks=?u?1?1?1?1?1?d,?1?1?1?1?1?d?d  check mask coverage
--checkmasksfile=masks.hcmask                  check mask coverage in a file
```

#### **Miscellaneous options:**

```
--pps=1000000000  Passwords per Second
```

## **MASKGEN EXAMPLES**

**Gather stats about cracked passwords.txt and hide the less than 1% results:**

```
python statsgen.py --hiderare passwords.txt
```

**Save masks stats to a .mask file for further analysis:**

```
python statsgen.py --hiderare passwords.txt -o example.mask
```

**Analyze example.mask results, number of masks, estimated time to crack, etc...**

```
python maskgen.py example.mask
```

**Create 24 hour (86400 seconds) mask attack based on cracking speed of a single  
GTX 1080 against MD5 hashes 24943.1 MH/s(based on appendix table).**

**!Substitute your GPU's cracking speed against MD5 (c/s)!!**

```
python maskgen.py example.mask --targettime=86400 --optindex --
```

pps=24943000000 -q

## **Output 24 hour mask attack to a .hcmask file for use in Hashcat:**

python maskgen.py example.mask --targettime=86400 --optindex --  
pps=24943000000 -q -o example.hcmask

## **Use your new example.hcmask file with Hashcat in mask attack mode:**

hashcat -a 3 -m #type hash.txt example.hcmask

### **TIME TABLE CHEAT SHEET**

60 seconds	1 minute
3,600 seconds	1 hour
86,400 seconds	1 day
604,800 seconds	1 week
1,209,600 seconds	1 fortnight
2,419,200 seconds	1 month (30days)
31,536,000 seconds	1 year

### **POLICYGEN**

---

Generate a collection of masks following the password complexity in order to significantly reduce the cracking time.

**python policygen.py [options] -o example.hcmask**

### **POLICYGEN OPTIONS**

-o masks.hcmask	Save masks to a file
--pps=1000000000	Passwords per Second
--showmasks	Show matching masks
--noncompliant	Generate masks for noncompliant passwords
-q, --quiet	Don't show headers.

### **Password Policy:**

Define the minimum (or maximum) password strength policy that you would like to test

```
--minlength=8      Minimum password length
--maxlength=8      Maximum password length
--mindigit=1       Minimum number of digits
--minlower=1        Minimum number of lower-case characters
--minupper=1        Minimum number of upper-case characters
--minspecial=1     Minimum number of special characters
--maxdigit=3        Maximum number of digits
--maxlower=3        Maximum number of lower-case characters
--maxupper=3        Maximum number of upper-case characters
--maxspecial=3     Maximum number of special characters
```

## **POLICYGEN EXAMPLES**

**Generate mask attack for password policy 8 character length requiring at least 1 lowercase, 1 uppercase, 1 digit, and 1 special character:**

```
python policygen.py --minlength 8 --maxlength 8 --minlower 1 --minupper 1 --
mindigit 1 --minspecial 1 -o example.hcmask
```

**Generate mask attack and estimate time of completion based on GTX 1080 against MD5 hashes 24943.1 MH/s(based on appendix table) for password policy 8 character length requiring at least 1 lowercase, 1 uppercase, 1 digit, and 1 special character:**

```
python policygen.py --minlength 8 --maxlength 8 --minlower 1 --minupper 1 --
mindigit 1 --minspecial 1 -o example.hcmask --pps=24943000000
```

## **CUSTOM MASK PLANS**

### **DATE YYMMDD MASK**

```
hashcat -a 3 -m #type hash.txt -1 12 -2 90 -3 01 -4 123 ?l?2?3?d?4?d
```

### **DATE YYYYMMDD MASK**

```
hashcat -a 3 -m #type hash.txt -1 12 -2 90 -3 01 -4 123 ?l?2?d?d?3?d? 4?d
```

### **3 SEQUENTIAL NUMBERS MASK + SPECIAL**

```
hashcat -a 3 -m #type hash.txt -1 147 -2 258 -3 369 ?l?2?3?s
```

# FOREIGN CHARACTER SETS

## UTF8 POPULAR LANGUAGES

\**Incremental four character password examples*

### Arabic

UTF8 (d880-ddbf)

```
hashcat -a 3 -m #type hash.txt --hex charset -1 d8d9dadbdccdd -2 80818283848586  
8788898a8b8c8d8e8f909192939495969798999a9b9c9d9e9fa0ala2a3a4a5a6a7a8a9a:  
afb0blb2b3b4b5b6b7b8b9babbbcbdbebf -i ?1?2?1?2?1?2?1?2
```

### Bengali

UTF8 (e0a680-e0adb)

```
hashcat -a 3 -m #type hash.txt --hex charset -1 e0 -2 a6a7a8a9aaabacad -3 8081  
82838485868788898a8b8c8d8e8f909192939495969798999a9b9c9d9e9fa0ala2a3a4a  
aaabacadaefb0blb2b3b4b5b6b7b8b9babbbcbdbebf -i ?1?2?3?1?2?3?1?2?3?1?2?3
```

### Chinese (Common Characters)

UTF8 (e4b880-e4bbbf)

```
hashcat -a 3 -m #type hash.txt --hex charset -1 e4 -2 b8b9babb -3 808182838485  
868788898a8b8c8d8e8f909192939495969798999a9b9c9d9e9fa0ala2a3a4a5a6a7a8a:  
aeafb0blb2b3b4b5b6b7b8b9babbbcbdbebf -i ?1?2?3?1?2?3?1?2?3?1?2?3
```

### Japanese (Katakana & Hiragana)

UTF8 (e38180-e3869f)

```
hashcat -a 3 -m #type hash.txt --hex charset -1 e3 -2 818283848586 -3 80818283  
8485868788898a8b8c8d8e8f909192939495969798999a9b9c9d9e9fa0ala2a3a4a5a6a  
acadaefb0blb2b3b4b5b6b7b8b9babbbcbdbebf -i ?1?2?3?1?2?3?1?2?3?1?2?3
```

### Russian

UTF8 (d080-d4bf)

```
hashcat -a 3 -m #type hash.txt --hex charset -1 d0d1d2d3d4 -2 8081828384858687
```

88898a8b8c8d8e8f909192939495969798999a9b9c9d9e9fa0ala2a3a4a5a6a7a8a9aaab0b1b2b3b4b5b6b7b8b9babbbcbdbef -i ?1?2?1?2?1?2?1?2

## HASHCAT BUILT-IN CHARSETS

### German

hashcat -a 3 -m #type hash.txt -1 charsets/German.hcchr -i ?1?1?1?1

### French

hashcat -a 3 -m #type hash.txt -1 charsets/French.hcchr -i ?1?1?1?1

### Portuguese

hashcat -a 3 -m #type hash.txt -1 charsets/Portuguese.hcchr -i ?1?1?1?1

### SUPPORTED LANGUAGE ENCODINGS

**hashcat -a 3 -m #type hash.txt -1 charsets/<language>.hcchr -i ?1?1?1?1**

Bulgarian, Castilian, Catalan, English, French, German, Greek, Greek Polytonic, Italian, Lithuanian, Polish, Portuguese, Russian, Slovak, Spanish

## JOHN UTF8 & BUILT-IN CHARSETS

### OPTIONS:

<b>--encoding=NAME</b>	input encoding (eg. UTF-8, ISO- 8859-1).
<b>--input-encoding=NAME</b>	input encoding (alias for --encoding)
<b>--internal-encoding=NAME</b>	encoding used in rules/masks (see doc/ENCODING)
<b>- -target-encoding=NAME</b>	output encoding (used by format)

**Example LM hashes from Western Europe, using a UTF-8 wordlist:**  
john --format=lm hast.txt --encoding=utf8 --target:cp850 --wo:spanish.txt

**Example using UTF-8 wordlist with internal encoding for rules processing:**  
john --format=#type hash.txt --encoding=utf8 --internal=CP1252 --wordlist=french.1st --rules

## **Example mask mode printing all possible “Latin-1” words of length 4:**

```
john --stdout --encoding=utf8 --internal=8859-1 --mask:?1?1?1?1
```

## **SUPPORTED LANGUAGE ENCODINGS**

UTF-8, ISO-8859-1 (Latin), ISO-8859-2 (Central/Eastern Europe), ISO-8859-7 (Latin/Greek), ISO-8859-15 (Western Europe), CP437 (Latin), CP737 (Greek), CP850 (Western Europe), CP852 (Central Europe), CP858 (Western Europe), CP866 (Cyrillic), CP1250 (Central Europe), CP1251 (Russian), CP1252 (Default Latin1), CP1253 (Greek) and KOI8-R (Cyrillic).

## **HASHCAT ?b BYTE CHARSET**

If you’re unsure as to position of a foreign character set contained within your target password, you can attempt the ?b byte charset in a mask using a sliding window. For example if we have a password 6 characters long:

**?b = 256 byte = 0x00 - 0xff**

hashcat -a 3 -m #type hash.txt

```
?b?a?a?a?a?a  
?a?b?a?a?a?a  
?a?a?b?a?a?a  
?a?a?a?b?a?a  
?a?a?a?a?b?a  
?a?a?a?a?a?b
```

## **CONVERT ENCODING**

### **HASHCAT**

Force internal wordlist encoding from X

```
hashcat -a 0 -m #type hash.txt dict.txt --encoding-from=utf-8
```

Force internal wordlist encoding to X

```
hashcat -a 0 -m #type hash.txt dict.txt --encoding-to=iso-8859-15
```

### **ICONV**

**Convert wordlist into language specific encoding**

```
iconv -f <old_encode> -t <new_encode> < dict.txt | sponge dict.txt.enc
```

# **CONVERT HASHCAT \$HEX OUTPUT**

**Example of converting \$HEX[] entries in hashcat.pot file to ASCII**

```
grep '$HEX' hashcat.pot | awk -F ":" {"print$2"} |perl -ne 'if ($_ =~ m/^$HEX\[([A-Fa-f0-9]+)\]/) {print pack("H*", $1), "\n"}'
```

## **ADVANCED ATTACKS**

### **PRINCE ATTACK**

PRINCE (PRobability INfinite Chained Elements) Attack takes one input wordlist and builds “chains” of combined words automatically.

#### **HASHCAT PRINCEPROCESSOR**

<https://github.com/hashcat/princeprocessor>

##### **Attack slow hashes:**

```
pp64.bin dict.txt | hashcat -a 0 -m #type hash.txt
```

##### **Amplified attack for fast hashes:**

```
pp64.bin --case-permute dict.txt | hashcat -a 0 -m #type hash.txt -r rule.txt
```

**Example PRINCE attack producing minimum 8 char candidates with 4 elements piped directly into Hashcat with rules attack.**

```
pp64.bin --pw-min=8 --limit=4 dict.txt|hashcat -a 0 -m # hash.txt -r best64.rule
```

#### **PRINCECEPTION ATTACK (epixoip)**

Piping the output of one PRINCE attack into another PRINCE attack.

```
pp64.bin dict.txt | pp64.bin | hashcat -a 0 -m #type hash.txt
```

#### **JOHN BUILT-IN PRINCE ATTACK**

```
john --prince=dict.txt hash.txt
```

# MASK PROCESSOR

Mask attack generator with a custom configurable charset and ability to limit consecutive and repeating characters to decrease attack keyspace.

<https://github.com/hashcat/maskprocessor>

**Limit 4 consecutive identical characters in the password string “-q” option:**

```
mp64.bin -q 4 ?d?d?d?d?d?d?d | hashcat -a 0 -m #type hash.txt
```

**Limit 4 identical characters in the password string “-r” option:**

```
mp64.bin -r 4 ?d?d?d?d?d?d?d | hashcat -a 0 -m #type hash.txt
```

**Limit 2 consecutive and 2 identical characters in the password string:**

```
mp64.bin -r 2 -q 2 ?d?d?d?d?d?d?d | hashcat -a 0 -m #type hash.txt
```

**Custom charset limiting 2 consecutive and 2 identical characters in the password string:**

```
mp64.bin -r 2 -q 2 -1 aeiuo -2 TGBYHN ?1?2?l?2?d?d?d?d | hashcat -a 0 -m #type hash.txt
```

# CUSTOM MARKOV MODEL / STATSPROCESSOR

Word-generator based on the per-position markov-attack.

<https://github.com/hashcat/statsprocessor>

## HCSTATGEN

Create custom Markov models using hashcat-util hcstatgen.bin based on cracked target passwords. The util hcstatgen makes a 32MB file each time no matter how small/large the password list provided. Highly recommended you make custom Markov models for different target sets.

```
hcstatgen.bin outfile.hcstat < passwords.txt
```

## STATSPROCESSOR

Is a high-performance word-generator based on a user supplied per-position Markov

model (hcstat file) using mask attack notation.

## **Step 1: Create your custom Markov model**

```
hcstatgen.bin out.hcstat < passwords.txt
```

## **Step 2.1: Supply your new Markov model to Hashcat as mask or rule attack.**

```
hashcat -a 3 -m #type hash.txt --markov-hcstat=out.hcstat ?a?a?a?a?a?a
```

```
hashcat -a 0 -m #type hash.txt dict.txt -r rule.txt --markov-hcstat=out.hcstat
```

## **Step 2.2: OR Supply your new Markov model with sp64 and pipe into Hashcat.**

```
sp64.bin --pw-min 3 --pw-max 5 out.hcstat ?1?1?1?1?1?1 | hashcat -a 0 -m #type hash.txt
```

# KEYBOARD WALK PROCESSOR

Keyboard-walk generator with configurable base chars, keymappings and routes.  
<https://github.com/hasheat/kwprocessor>

Example keyboard walk with tiny charset in english mapping and with 2-10 adjacent keys piping out results into a hashcat attack:

```
kwp.bin basechar/tiny.base keymaps/en.keymap routes/2-to-10-max-3 -0 -z |  
hashcat -a 0 -m #type hash.txt
```

Example keyboard walk with full charset in english mapping and with 3x3 adjacent keys piping out results into a hashcat attack:

```
./kwp basechars/full.base keymaps/en.keymap routes/3-to-3-exhaustive.route |  
hashcat -a 0 -m #type hash.txt
```

## [FULL LIST OF OPTIONS]

./kwp [options]...	basechars-file keymap-file routes-file
-V, --version	Print version
-h, --help	Print help
-o, --output-file	Output-file
-b, --keyboard-basic	Characters reachable without holding shift or altgr
-s, --keyboard-shift	Characters reachable by holding shift
-a, --keyboard-altgr	Characters reachable by holding altgr (non-english)
-z, --keyboard-all	Shortcut to enable all --keyboard-* modifier
-1, --keywalk-south-west	Routes heading diagonale south-west
-2, --keywalk-south	Routes heading straight south
-3, --keywalk-south-east	Routes heading diagonale south-east
-4, --keywalk-west	Routes heading straight west
-5, --keywalk-repeat	Routes repeating character
-6, --keywalk-east	Routes heading straight east
-7, --keywalk-north-west	Routes heading diagonale north-wes
-8, --keywalk-north	Routes heading straight north
-9, --keywalk-north-east	Routes heading diagonale north-east
-0, --keywalk-all	Shortcut to enable all --keywalk-* directions
-n, --keywalk-distance-min	Minimum allowed distance between keys
-x, --keywalk-distance-max	Maximum allowed distance between keys

# MDXFIND / MDSPLIT

<https://hashes.org/mdxfind.php>

(credit ‘Waffle’)

MDXFIND is a program which allows you to run large numbers of unsolved hashes of any type, using many algorithms concurrently, against a large number of plaintext words and rules, very quickly. It’s main purpose was to deal with large lists (20 million, 50 million, etc) of unsolved hashes and run them against new dictionaries as you acquire them.

So when would you use MDXFIND on a pentest? If you dump a database tied to website authentication and the hashes are not cracking by standard attack plans. The hashes may be generated in a unique nested hashing series. If you are able to view the source code of said website to view the custom hashing function you can direct MDXFIND to replicate that hashing series. If not, you can still run MDXFIND using some of the below ‘Generic Attack Commands’. MDXFIND is tailored toward intermediate to expert level password cracking but is extremely powerful and flexible.

Example website SHA1 custom hashing function performing multiple iterations:

```
$hash = sha1($password . $salt);
for ($i = 1; $i <= 65000; ++$i)
{
    $hash = sha1($hash . $salt);
}
```

## **MDXFIND**

---

### **COMMAND STRUCTURE THREE METHODS 1-STDOUT 2-STDIN 3-File**

**1-** Reads hashes coming from cat (or other) commands stdout.

```
cat hash.txt | mdxfind -h <regex #type> -i <#iterations> dict.txt > out.txt
```

**2-** Takes stdin from outside attack sources in place of dict.txt when using the options variable ‘-f’ to specify hash.txt file location and variable ‘stdin’.

```
mp64.bin ?d?d?d?d?d?d | mdxfind -h <regex #type> -i <#iterations> -f hash.txt
stdin > out.txt
```

**3-** Specify file location ‘-f’ with no external stdout/stdin sources.

```
mdxfind -h <regex #type> -i <#iterations> -f hash.txt dict.txt > out.txt
```

## [FULL LIST OF OPTIONS]

- a Do email address munging
- b Expand each word into unicode, best effort
- c Replace each special char (<>&, etc) with XML equivalents
- d De-duplicate wordlists, best effort...but best to do ahead of time
- e Extended search for truncated hashes
- p Print source (filename) of found plain-texts
- Internal iteration counts for SHA1MD5X, and others. For example, if you have a hash that is SHA1(MD5(MD5(MD5(MD5(\$pass))))), you would set -q to 5.
- g Rotate calculated hashes to attempt match to input hash
- s File to read salts from
- u File to read Userid/Usernames from
- k File to read suffixes from
- n Number of digits to append to passwords. Other options, like: -n 6x would append 6 digit hex values, and 8i would append all ipv4 dotted-quad IP-addresses.
- i The number of iterations for each hash
- t The number of threads to run
- f file to read hashes from, else stdin
- l Append CR/LF/CRLF and print in hex
- r File to read rules from
- v Do not mark salts as found.
- w Number of lines to skip from first wordlist
- y Enable directory recursion for wordlists
- z Enable debugging information/hash results
- h The hash types: 459 TOTAL HASHES SUPPORTED

## GENERIC ATTACK PLANS

This is a good general purpose MDXFIND command to run your hashes against if you suspect them to be “non-standard” nested hashing sequences. This command says “Run all hashes against dict.txt using 10 iterations except ones having a salt, user, or md5x value in the name.” It’s smart to skip salted/user hash types in MDXFIND unless you are confident a salt value has been used.

```
cat hash.txt | mdxfind -h ALL -h '!salt,!user,!md5x' -i 10 dict.txt > out.txt
```

The developer of MDXFIND also recommends running the below command options as a good general purpose attack:

```
cat hash.txt | mdxfind -h <^md5$,^sha1$,^md5md5pass$,^md5sha1$' -i 5  
dict.txt > out.txt
```

And you could add a rule attack as well:

```
cat hash.txt | mdxfind -h <^md5$,^sha1$,^md5md5pass$,^md5sha1$' -i 5  
dict.txt -r best64.rule > out.txt
```

### ***GENERAL NOTES ABOUT MDXFIND***

-Can do multiple hash types/files all during a single attack run.

```
cat sha1/*.txt sha256/*.txt md5/*.txt salted/*.txt | mdxfind
```

-Supports 459 different hash types/sequences

-Can take input from special 'stdin' mode

-Supports VERY large hashlists (100mil) and 10kb character passwords

-Supports using hashcat rule files to integrate with dictionary

-Option '-z' outputs ALL viable hashing solutions and file can grow very large

-Supports including/excluding hash types by using simple regex parameters

-Supports multiple iterations (up to 4 billion times) by tweaking -i parameter for instance:

MD5X01 is the same as md5(\$Pass)

MD5X02 is the same as md5(md5(\$pass))

MD5X03 is the same as md5(md5(md5(\$pass)))

...

MD5x10 is the same as

md5(md5(md5(md5(md5(md5(md5(md5(\$pass))))))))

-Separate out -usernames -email -ids -salts to create custom attacks

-If you are doing brute-force attacks, then hashcat is probably better route

-When MDXfind finds any solution, it outputs the kind of solution found, followed by the hash, followed by the salt and/or password. For example:

Solution	HASH	:PASSWORD
----------	------	-----------

MD5X01 000012273bc5cab48bf3852658b259ef:lEb0TBK3

MD5X05 033blll073e5f64ee59f0be9d6b8a561:08061999

MD5X09 aadb9dlb23729a3e403d7fc62d507df7:1140

MD5X09 326d921d591162eed302ee25a09450ca:1761974

# MDSPLIT

---

When cracking large lists of hashes from multiple file locations, MDSPLIT will help match which files the cracked hashes were found in, while also outputting them into separate files based on hash type. Additionally it will remove the found hashes from the original hash file.

## COMMAND STRUCTURE TWO METHODS 1-STDOUT 2-STDIN 3-File

1- Matching MDXFIND results files with their original hash\_orig.txt files.

```
cat hashes_out/out_results.txt | mdsplit hashes_orig/hash_orig.txt
```

OR perform matching against a directory of original hashes and their results.

```
cat hashes_out/* | mdsplit hashes_orig/*
```

2- Piping MDXFIND directly into MDSPLIT to sort in real-time results.

```
cat *.txt | mdxfind -h ALL -h '!salt,!user,!md5x' -i 10 dict.txt | mdsplit *.txt
```

3- Specifying a file location in MDXFIND to match results in real-time.

```
mdxfind -h ALL -f hashes.txt -i 10 dict.txt | mdsplit hashes.txt
```

## ***GENERAL NOTES ABOUT MDSPLIT***

-MDSPLIT will append the final hash solution to the end of the new filename. For example, if we submitted a ‘hashes.txt’ and the solution to the hashes was “MD5x01” then the results file would be ‘hashes.MD5x01’. If multiple hash solutions are found then MDSPLIT knows how to deal with this, and will then remove each of the solutions from hashes.txt, and place them into ‘hashes.MD5x01’, ‘hashes.MD5x02’, ‘hashes.SHA1’... and so on.

-MDSPLIT can handle sorting multiple hash files, types, and their results all at one time. Any solutions will be automatically removed from all of the source files by MDSPLIT, and tabulated into the correct solved files. For example:

```
cat dir1/*.txt dir2/*.txt dir3/*.txt | mdxfind -h '^md5$|^sha1$|^sha256$' -i 10 dict.txt | mdsplit dir1/*.txt dir2/*.txt dir3/*.txt
```

# DISTRIBUTED / PARALLELIZATION CRACKING

## HASHCAT

---

<https://hashcat.net/forum/thread-3047.html>

### Step 1: Calculate keyspace for attack (Example MD5 Brute Force x 3nodes)

hashcat -a 3 -m 0 ?a?a?a?a?a?a --keyspace

81450625

### Step 2: Distribute work through keyspace division (s)kip and (l)imit

81450625 / 3 = 27150208.3

*Node1#* hashcat -a 3 -m 0 hash.txt ?a?a?a?a?a?a -s 0 -l 27150208

*Node2#* hashcat -a 3 -m 0 hash.txt ?a?a?a?a?a?a -s 27150208 -l 27150208

*Node3#* hashcat -a 3 -m 0 hash.txt ?a?a?a?a?a?a -s 54300416 -l 27150209

## JOHN

---

<http://www.openwall.com/john/doc/OPTIONS.shtml>

### Manual distribution using Options --node & --fork to 3 similar CPU nodes utilizing 8 cores:

*Node#* john --format=<#> hash.txt --wordlist=dict.txt --rules=All --fork=8 --node=1-8/24

*Node2#* john --format=<#> hash.txt --wordlist=dict.txt --rules=All --fork=8 --node=9-16/24

*Node3#* john --format=<#> hash.txt --wordlist=dict.txt --rules=All --fork=8 --node=17-24/24

Other John Options for parallelization:

Option 1:Enable OpenMP through uncommenting in Makefile

Option 2:Create additional incremental modes in john.conf

Option 3:Utilize built-in MPI parallelization

## PASSWORD GUESSING FRAMEWORK

<https://github.com/RUB-SysSec/Password-Guessing-Framework>

<https://www.password-guessing.org/>

Password Guessing Framework is an open source tool to provide an automated and reliable way to compare password guessers. It can help to identify individual strengths and weaknesses of a guesser, it's modes of operation, or even the underlying guessing strategies. Therefore, it gathers information about how many passwords from an input file (password leak) have been cracked in relation to the amount of generated guesses. Subsequent to the guessing process an analysis of the cracked passwords is performed.

## **OTHER CREATIVE ADVANCED ATTACKS**

Random creative password attacks using the power of stdin and stdout. Not implying they're useful but to demonstrate the power of mixing and matching. Go forth and create something useful.

### **PRINCE-MDXFIND ATTACK**

```
pp64.bin dict.txt | mdxfind -h ALL -f hash.txt -i 10 stdin > out.txt
```

### **HASHCAT-UTIL COMBONATOR PRINCE**

```
combinator.bin dict.txt dict.txt | pp64.bin | hashcat -a 0 -m #type hash.txt -r best64.rule
```

```
combinator3.bin dict.txt dict.txt dict.txt | pp64.bin | hashcat -a 0 -m #type hash.txt -r rockyou-30000.rule
```

### **HASHCAT STDOUT ATTACKS PRINCE**

```
hashcat -a 0 dict.txt -r dive.rule --stdout | pp64.bin | hashcat -a 0 -m #type hash.txt
```

```
hashcat -a 6 dict.txt ?a?a?a?a --stdout | pp64.bin --pw-min=8 | hashcat -a 0 -m #type hash.txt
```

```
hashcat -a 7 ?a?a?a?a dict.txt --stdout | pp64.bin --pw-min=8 | hashcat -a 0 -m #type hash.txt
```

```
hashcat -a 6 dict.txt rockyou-1-60.hmask --stdout | pp64.bin --pw-min=8 --pw-max=14 I hashcat -a 0 -m #type hash.txt
```

```
hashcat -a 7 rockyou-1-60.hcmask dict.txt --stdout | pp64.bin --pw-min=8 --pw-max=14 I hashcat -a 0 -m #type hash.txt
```

## DISTRIBUTED CRACKING SOFTWARE

### HASHTOPUSSY

<https://bitbucket.org/seinlc/hashtopussy/>

### HASHSTACK

<https://sagitta.pw/software/>

### DISTHC

<https://github.com/unix-ninja/disthc>

### CRACKLORD

<http://jmmcatee.github.io/cracklord/>

### HASHTOPUS

<http://hashtopus.org/Site/>

### HASHVIEW

<http://www.hashview.io/>

### CLORTHO

<https://github.com/ccdes/clortho>

## ONLINE HASH CRACKING SERVICES

### GPUHASH

<https://gpuhash.me/>

### CRACKSTATION

<https://crackstation.net/>

### ONLINE HASH CRACK

<https://www.onlinehashcrack.com/>

**HASH HUNTERS**

<http://www.hashhunters.net/>

Information in this chapter is an attempt to summarize a few of the basic and more complex concepts in password cracking. This allows all skill levels to grasp these concepts without needing a Linguistics or Mathematics Degree. It's an almost impossible task to condense into one paragraph, but the following is an attempt. For a deeper understanding, I highly encourage you to read the Resource links included below each section.

## PASSWORD ENTROPY vs CRACK TIME

Password entropy is a measure of how random/unpredictable a password could have been, so it does not really relate to the password itself, but to a selection process. When judging human generated passwords for entropy, it frankly isn't an accurate measurement. This is true mainly because humans like to use memorable words/sequences and thus a myriad of attacks account for that behavior. However, entropy is good for measuring randomly generated passwords from password managers, such as 1Password or KeePass, in that each default character set used can be calculated. Password entropy is measured in bits and uses the following formula where C=Size of Character set & L=Length of password:  $\log(C) / \log(2) * L$ . To calculate the time to crack, just use the benchmarking function on your favorite cracking software against your mode of hash to obtain cracks per second. The table below estimates password length using an MD4 hashing function against an 8 GPU x Nvidia GTX1080 system:

Length	Alphanumeric 0-9, a-z, A-Z	Time to Crack (350 GH/s)
8	47 bits	~15 Mins
9	59 bits	
10	65 bits	~457 Hours
11	65 bits	~3.3 Years
12	71 bits	~214 Years
13	77 bits	~13,690 Years
14	80 bits	~109,500 Years
15	89 bits	~56,080,000 Years
20	119 bits	~Doesn't Matter

*\*Table only truly matters for randomly generated passwords*

## Resources

### Password Complexity versus Password Entropy

<https://blogs.technet.microsoft.com/msftcam/2015/05/19/password-complexity-versus-password-entropy/>

## WHAT IS A CRYPTOGRAPHIC HASH?

A cryptographic hash function is a subclass of the general hash function which possesses properties lending its use in cryptography. Cryptographic hash functions are mathematical algorithms which map data of any size to a string containing a fixed length, and should make it infeasible to reverse. For instance, the string “password,” when mapped using the MD5 hash function, returns a fixed length 32 character string “5f4dcc3b5aa765d61d8327deb882cf99”. The 32 character string cannot theoretically be reversed with any other mapped input data except “password”. The current method of recreating this input data “password” is through a dictionary/mask/brute-force attack of all possible inputs matching the hashed value; also called a pre-image attack. Generally speaking, hash functions should possess the below characteristics:

- Be computationally infeasible to find two different sets of input data with the same hash value (also called a collision).
- The hash value should be “quick” to compute (i.e.  $> \sim 1$  second).
- It should be difficult to generate the input data Just by looking at the hash value.
- One simple change to the input data should drastically change the resultant hash value.

## Resources

### How Hash Algorithms Work

<http://www.metamorphosite.com/one-way-hash-encryption-sha1-data-software>

## MARKOV CHAINS

Markov Chains are created, for our password cracking purposes, by statistical analysis of a large list of passwords/words (i.e. the RockYou password dataset). The

resultant analysis of these words and their per-position character frequency/probability are stored in a table. This table is referenced when performing brute-force/mask attacks to prevent having to generate password candidates in a sequential order, which is very inefficient. Instead, the most common characters are attempted first in order of preceding character probability. So let's see sequential brute-force ?a?a?a?a with out Markov Chains applied:

aaaa	aaad	aaag
aaab	aaae	aaah
aaac	aaaf	....

Now the same brute-force attack with Markov Chains applied:

sari	aari	pari
mari	cari	2ari
1ari	bari	....

Markov Chains predict the probability of the next character in a password based on the previous characters, or context characters. It's that simple.

## Resources

**Fast Dictionary Attacks on Passwords Using Time-Space Tradeoff**  
[http://www.cs.utexas.edu/~shmat/shmat\\_ccs05pwd.pdf](http://www.cs.utexas.edu/~shmat/shmat_ccs05pwd.pdf)

**OMEN: Faster Password Guessing Using an Ordered Markov Enumerator**  
<https://hal.inria.fr/hal-01112124/document>

## PROBABILISTIC CONTEXT-FREE GRAMMARS (PCFG)

A Probabilistic Context Free Grammar (PCFG) consists of terminal and nonterminal variables. Each feature to be modeled has a production rule that is assigned a probability, estimated from a training set of RNA structures. Production rules are recursively applied until only terminal residues are left. The notion supporting PCFGs is that passwords are constructed with template structures and terminals that fit into those structures. For example, the password candidate ‘passwordl23!’ is 8 letters, 3 digits, 1 special and would be noted as ‘L<sub>8</sub>D<sub>3</sub>S<sub>1</sub>’. A password’s probability of occurring is the probability of its structure, multiplied by those of its underlying terminals.

## **Resources**

### **Password Cracking Using Probabilistic Context-Free Grammars**

[https://sites.google.com/site/reusablesec/Home/password-cracking-tools/probablistic\\_cracker](https://sites.google.com/site/reusablesec/Home/password-cracking-tools/probablistic_cracker)

### **Next Gen PCFG Password Cracking**

[https://github.com/lakiw/pcfg\\_cracker](https://github.com/lakiw/pcfg_cracker)

# **NEURAL NETWORKS**

Artificial Neural Networks or Neural Networks (NN) is a machine-learning technique composed of nodes called Artificial Neurons, just like the brain possesses. Such systems use Machine Learning to approximate highly dimensional functions and progressively learn through examples of training set data, or in our case a large password dump. They have shown initial promise to be effective at generating original yet representative password candidates. Advantages to NN's for password cracking are the low overhead for storing the final NN model, approximately 500kb, and the ability to continually learn over time through retraining or transfer learning.

## **Resources**

### **Fast, Lean, and Accurate: Modeling Password Guessability Using Neural Networks (USENIX '16)**

[https://www.usenix.org/system/files/conference/usenixsecurity16/sec16\\_paper\\_melic](https://www.usenix.org/system/files/conference/usenixsecurity16/sec16_paper_melic)  
[https://github.com/cupslab/neural\\_network\\_cracking](https://github.com/cupslab/neural_network_cracking)

## COMMON HASH EXAMPLES

**MD5, NTLM, NTLMv2, LM, MD5crypt, SHA1, SHA256, bcrypt, PDF 1.4 - 1.6 (Acrobat 5-8), Microsoft OFFICE 2013, RAR3-HP, Winzip, 7zip, Bitcoin/Litecoin, MAC OSX v10.5-v10.6, MySQL 4.1-5+, Postgres, MSSQL(2012)-MSSQL(2014), Oracle 11g, Cisco TYPE 4 5 8 9, WPA PSK / WPA2 PSK**

## MDS

HASHCAT

**HASH FORMAT**

8743b52063cd84097a65dl633f5c74f5

**BRUTE FORCE ATTACK**

hashcat -m 0 -a 3 hash.txt ?a?a?a?a?a?a

**WORDLIST ATTACK**

hashcat -m 0 -a 0 hash.txt dict.txt

**WORDLIST + RULE ATTACK**

hashcat -m 0 -a 0 hash.txt dict.txt -r rule.txt

JOHN

**HASH FORMAT**

8743b52063cd84097a65dl633f5c74f5

**BRUTE FORCE ATTACK**

john --format=raw-md5 hash.txt

**WORDLIST ATTACK**

john --format=raw-md5 wordlist=dict.txt hash.txt

**WORDLIST + RULE ATTACK**

john --format=raw-md5 wordlist=dict.txt --rules hash.txt

# **NTLM (PWDUMP)**

HASHCAT

**HASH FORMAT**

b4b9b02e6f09a9bd760f388b67351e2b

**BRUTE FORCE ATTACK**

hashcat -m 1000 -a 3 hash.txt ?a?a?a?a?a?a

**WORDLIST ATTACK**

hashcat -m 1000 -a 0 hash.txt dict.txt

**WORDLIST + RULE ATTACK**

hashcat -m 1000 -a 0 hash.txt dict.txt -r rule.txt

JOHN

**HASH FORMAT**

b4b9b02e6f09a9bd760f388b67351e2b

**BRUTE FORCE ATTACK**

john --format=nt hash.txt

**WORDLIST ATTACK**

john --format=nt wordlist=dict.txt hash.txt

**WORDLIST + RULE ATTACK**

john --format=nt wordlist=dict.txt --rules hash.txt

# **NTLM V2**

HASHCAT

**HASH FORMAT**

username::N46iSNekpT:08ca45b7d7ea58ee:88dcbe4446168966a153a0064958dac6:5C78303100000000000000b45c67103d07d7b95acdl2ffall230e0000000052920b85f78d3b92f5d765c783030

**BRUTE FORCE ATTACK**

hashcat -m 5600 -a 3 hash.txt ?a?a?a?a?a?a

**WORDLIST ATTACK**

hashcat -m 5600 -a 0 hash.txt dict.txt

## **WORDLIST + RULE ATTACK**

hashcat -m 5600 -a 0 hash.txt dict.txt -r rule.txt

### JOHN

#### **HASH FORMAT**

username:\$NETNTLMv2\$NTLMV2TESTWORKGROUP\$1122334455667788\$076  
5C87EC1D5\$01010000000000006CF6385B74CA01B3610B02D99732DD000000  
0052004B00470052004F00550050000100200044004100540041002E00420049004F  
004500430055005200490000000000

#### **BRUTE FORCE ATTACK**

john --format=netntlmv2 hash.txt

#### **WORDLIST ATTACK**

john --format=netntlmv2 wordlist=dict.txt hash.txt

#### **WORDLIST + RULE ATTACK**

john --format=netntlmv2 wordlist=dict.txt --rules hash.txt

## **LM**

### HASHCAT

#### **HASH FORMAT**

299bd128c1101fd6

#### **BRUTE FORCE ATTACK**

hashcat -m 3000 -a 3 hash.txt ?a?a?a?a?a?a

#### **WORDLIST ATTACK**

hashcat -m 3000 -a 0 hash.txt dict.txt

#### **WORDLIST + RULE ATTACK**

hashcat -m 3000 -a 0 hash.txt dict.txt -r rule.txt

### JOHN

#### **HASH FORMAT**

\$LM\$a9c604d244c4e99d

#### **BRUTE FORCE ATTACK**

john --format=lm hash.txt

#### **WORDLIST ATTACK**

john --format=lm wordlist=dict.txt hash.txt

## **WORDLIST + RULE ATTACK**

john --format=lm wordlist=dict.txt --rules hash.txt

# **MD5CRYPT**

## HASHCAT

### **HASH FORMAT**

\$1\$28772684\$iEwNOgGugq09.bIz5sk8k/

### **BRUTE FORCE ATTACK**

hashcat -m 500 -a 3 hash.txt ?a?a?a?a?a?a

### **WORDLIST ATTACK**

hashcat -m 500 -a 0 hash.txt dict.txt

### **WORDLIST + RULE ATTACK**

hashcat -m 500 -a 0 hash.txt dict.txt -r rule.txt

## JOHN

### **HASH FORMAT**

\$1\$28772684\$iEwNOgGugq09.bIz5sk8k/

### **BRUTE FORCE ATTACK**

john --format=md5crypt hash.txt

### **WORDLIST ATTACK**

john --format=md5crypt wordlist=dict.txt hash.txt

### **WORDLIST + RULE ATTACK**

john --format=md5crypt wordlist=dict.txt --rules hash.txt

# **SHA1**

## HASHCAT

### **HASH FORMAT**

b89eaac7e61417341b710b727768294d0e6a277b

### **BRUTE FORCE ATTACK**

hashcat -m 100 -a 3 hash.txt ?a?a?a?a?a?a

### **WORDLIST ATTACK**

hashcat -m 100 -a 0 hash.txt dict.txt

## **WORDLIST + RULE ATTACK**

hashcat -m 100 -a 0 hash.txt dict.txt -r rule.txt

### JOHN

#### **HASH FORMAT**

b89eaac7e61417341b710b727768294d0e6a277b

#### **BRUTE FORCE ATTACK**

john --format=raw-sha1 hash.txt

#### **WORDLIST ATTACK**

john --format=raw-sha1 wordlist=dict.txt hash.txt

#### **WORDLIST + RULE ATTACK**

john --format=raw-sha1 wordlist=dict.txt --rules hash.txt

## **SHA256**

### HASHCAT

#### **HASH FORMAT**

127e6fbfe24a750e72930c220a8e138275656b8e5d8f48a98c3c92df2caba935

#### **BRUTE FORCE ATTACK**

hashcat -m 1400 -a 3 hash.txt ?a?a?a?a?a?a

#### **WORDLIST ATTACK**

hashcat -m 1400 -a 0 hash.txt dict.txt

#### **WORDLIST + RULE ATTACK**

hashcat -m 1400 -a 0 hash.txt dict.txt -r rule.txt

### JOHN

#### **HASH FORMAT**

127e6fbfe24a750e72930c220a8e138275656b8e5d8f48a98c3c92df2caba935

#### **BRUTE FORCE ATTACK**

john --format=raw-sha256 hash.txt

#### **WORDLIST ATTACK**

john --format=raw-sha256 wordlist=dict.txt hash.txt

#### **WORDLIST + RULE ATTACK**

john --format=raw-sha256 wordlist=dict.txt --rules hash.txt

# BCRYPT

## HASHCAT

## HASH FORMAT

\$2a\$05\$LhayLxezLhK1LhWvKxCyLOj0jlu.Kj0jZ0pEmml34uzrQlFvQDLF6

## BRUTE FORCE ATTACK

```
hashcat -m 3200 -a 3 hash.txt ?a?a?a?a?a?a
```

# WORDLIST ATTACK

```
hashcat -m 3200 -a 0 hash.txt dict.txt
```

## WORDLIST + RULE ATTACK

```
hashcat -m 3200 -a 0 hash.txt dict.txt -r rule.txt
```

JOHN

## HASH FORMAT

\$2a\$05\$LhayLxezLhK1LhWvKxCyLOj0jlu.Kj0jZ0pEmml34uzrQ1FvQDLF6

## BRUTE FORCE ATTACK

```
john --format=bcrypt hash.txt
```

# WORDLIST ATTACK

```
john --format=bcrypt wordlist=dict.txt hash.txt
```

# WORDLIST + RULE ATTACK

```
john --format=bcrypt wordlist=dict.txt --rules hash.txt
```

## **PDF 1.4 - 1.6 (ACROBAT 5-8)**

## HASHCAT

## HASH FORMAT

## EXTRACT HASH

```
pdf2hashcat.py example.pdf > hash.txt
```

## BRUTE FORCE ATTACK

hashcat -m 10500 -a 3 hash.txt ?a?a?a?a?a?a

## **WORDLIST ATTACK**

hashcat -m 10500 -a 0 hash.txt dict.txt

## **WORDLIST + RULE ATTACK**

hashcat -m 10500 -a 0 hash.txt dict.txt -r rule.txt

## JOHN

### **HASH FORMAT**

\$pdf\$Standard\*badadle86442699427116d3e5d5271bc80a27814fc5e80f815feef8393  
89ece9b5ce451a5d7064693dab3badf101112131415161718191alblcldlef\*16\*34lblb6e  
af681a9b63fa8bf563b\*1\*1\*0\*1\*4\*128\*-4\*3\*2

### **EXTRACT HASH**

pdf2john.py example.pdf > hash.txt

### **BRUTE FORCE ATTACK**

john --format=pdf hash.txt

## **WORDLIST ATTACK**

john --format=pdf wordlist=dict.txt hash.txt

## **WORDLIST + RULE ATTACK**

john --format=pdf wordlist=dict.txt --rules hash.txt

# **MICROSOFT OFFICE 2013**

## HASHCAT

### **HASH FORMAT**

example.docx:\$office\$\*2013\*100000\*256\*16\*7dd611d7eb4c899f74816dldec817b31  
b2c2c6c32f14b5995a543ad037\*0b7ee0e48e935f937192a59de48a7d561ef2691d5c8a1  
04402a94895

### **EXTRACT HASH**

office2hashcat.py example.docx > hash.txt

### **BRUTE FORCE ATTACK**

hashcat -m 9600 -a 3 --username hash.txt ?a?a?a?a?a?a

## **WORDLIST ATTACK**

hashcat -m 9600 -a 0 --username hash.txt dict.txt

## **WORDLIST + RULE ATTACK**

```
hashcat -m 9600 -a 0 --username hash.txt dict.txt -r rule.txt
```

## JOHN

### **HASH FORMAT**

```
example.docx:$office$*2013*100000*256*16*7dd611d7eb4c899f74816dldec817b31  
04402a94895
```

### **EXTRACT HASH**

```
office2john.py example.docx > hash.txt
```

### **BRUTE FORCE ATTACK**

```
john --format=office2013 hash.txt
```

### **WORDLIST ATTACK**

```
john --format=office2013 wordlist=dict.txt hash.txt
```

### **WORDLIST + RULE ATTACK**

```
john --format=office2013 wordlist=dict.txt --rules hash.txt
```

## **RAR3-HP (ENCRYPTED HEADER)**

## HASHCAT

### **HASH FORMAT**

```
$RAR3$*0*45109af8ab5f297a*adbf6c5385d7a40373e8f77d7b89d317  
#!Ensure to remove extraneous rar2john output to match above hash!#
```

### **EXTRACT HASH**

```
rar2john.py example.rar > hash.txt
```

### **BRUTE FORCE ATTACK**

```
hashcat -m 12500 -a 3 hash.txt ?a?a?a?a?a?a
```

### **WORDLIST ATTACK**

```
hashcat -m 12500 -a 0 hash.txt dict.txt
```

### **WORDLIST + RULE ATTACK**

```
hashcat -m 12500 -a 0 hash.txt dict.txt -r rule.txt
```

## JOHN

### **HASH FORMAT**

```
example.rar:$RAR3$*1*20e041a232b4b7f0*5618c5f0*1472*2907*0*/Path/To/  
example.rar*138*33:1::example.txt
```

## **EXTRACT HASH**

```
rar2john.py example.rar > hash.txt
```

## **BRUTE FORCE ATTACK**

```
john --format=rar hash.txt
```

## **WORDLIST ATTACK**

```
john --format=rar wordlist=dict.txt hash.txt
```

## **WORDLIST + RULE ATTACK**

```
john --format=rar wordlist=dict.txt --rules hash.txt
```

# **WINZIP**

## **HASHCAT**

### **HASH FORMAT**

```
$zip2$*0*3*0*b5d2b7bf57ad5e86a55c400509c672bd*d218*0**ca3d736d03a34165  
zip2$
```

```
#!Ensure to remove extraneous zip2john output to match above hash!#
```

## **EXTRACT HASH**

```
zip2john.py example.zip > hash.txt
```

## **BRUTE FORCE ATTACK**

```
hashcat -m 13600 -a 3 hash.txt ?a?a?a?a?a?a
```

## **WORDLIST ATTACK**

```
hashcat -m 13600 -a 0 hash.txt dict.txt
```

## **WORDLIST + RULE ATTACK**

```
hashcat -m 13600 -a 0 hash.txt dict.txt -r rule.txt
```

## **JOHN**

### **HASH FORMAT**

```
example.zip:$zip2$*0*3*0*5b0a8b153fb94bf719abb81a80e90422*8e91*9*0b76bf50  
938ce9c*3f37001e241el96195al*$zip2$:: ::example.zip
```

## **EXTRACT HASH**

```
zip2john.py example.zip > hash.txt
```

## **BRUTE FORCE ATTACK**

```
john --format=ZIP hash.txt
```

## **WORDLIST ATTACK**

john --format=ZIP wordlist=dict.txt hash.txt

## **WORDLIST + RULE ATTACK**

john --format=ZIP wordlist=dict.txt --rules hash.txt

# **7-ZIP**

## HASHCAT

### **HASH FORMAT**

\$7z\$0\$19\$0\$salt\$8\$f6196259a7326e3f000000000000000\$185065650\$112\$98\$f3b  
419a25acd40c0c2d75421cf23263f69c51bl3f9blaada41a8a09f9adeae45d67c60b56aac  
0dcc5eb811c7a61128ee0746f922cdb9c59096869f341c7a9cblac7bb7d771f546b82cf4  
Cd4b61751e4d8de66dd6e2dfb5b7dl022d2211e2d66eal703f96

#!Ensure to remove extraneous 7zip2john output to match above hash!#

### **EXTRACT HASH**

7z2john.py example.7z > hash.txt

## **BRUTE FORCE ATTACK**

hashcat -m 11600 -a 3 hash.txt ?a?a?a?a?a?a

### **WORDLIST ATTACK**

hashcat -m 11600 -a 0 hash.txt dict.txt

### **WORDLIST + RULE ATTACK**

hashcat -m 11600 -a 0 hash.txt dict.txt -r rule.txt

## JOHN

### **HASH FORMAT**

example.7z:\$7z\$0\$19\$0\$salt\$8\$f6196259a7326e3f000000000000000\$185065650\$  
3bc2a88062c419a25acd40c0c2d75421cf23263f69c51bl3f9blaada41a8a09f9adeae45c  
6aad338f20c0dcc5eb811c7a61128ee0746f922cdb9c59096869f341c7a9cblac7bb7d77  
Cf4e6fla5ecd4b61751e4d8de66dd6e2dfb5b7dl022d2211e2d66eal703f96

### **EXTRACT HASH**

7z2john.py example.7z > hash.txt

## **BRUTE FORCE ATTACK**

john --format=7z hash.txt

### **WORDLIST ATTACK**

john --format=7z wordlist=dict.txt hash.txt

### **WORDLIST + RULE ATTACK**

```
john --format=7z wordlist=dict.txt --rules hash.txt
```

## BITCOIN / LITECOIN

### HASHCAT

#### **HASH FORMAT**

```
$bitcoin$96$d011alb6a8d675b7a36d0cd2efaca32a9f8d1d57d6d01a58399ea04e703e  
99039326f7a00f171a7bbc854a54$16$1563277210780230$158555$96$62883542681  
5704485715363525107408232305571584532274162540768587307602723386534  
25882875480513751851333441623702852811440775888122046360561760525
```

#### **EXTRACT HASH**

```
bitcoin2john.py wallet.dat > hash.txt
```

#### **BRUTE FORCE ATTACK**

```
hashcat -m 11300 -a 3 hash.txt ?a?a?a?a?a?a
```

#### **WORDLIST ATTACK**

```
hashcat -m 11300 -a 0 hash.txt dict.txt
```

#### **WORDLIST + RULE ATTACK**

```
hashcat -m 11300 -a 0 hash.txt dict.txt -r rule.txt
```

### JOHN

#### **HASH FORMAT**

```
$bitcoin$96$d011alb6a8d675b7a36d0cd2efaca32a9f8d1d57d6d01a58399ea04e703e  
99039326f7a00f171a7bbc854a54$16$1563277210780230$158555$96$62883542681  
5704485715363525107408232305571584532274162540768587307602723386534  
25882875480513751851333441623702852811440775888122046360561760525
```

#### **EXTRACT HASH**

```
bitcoin2john.py wallet.dat > hash.txt
```

#### **BRUTE FORCE ATTACK**

```
john --format=bitcoin hash.txt
```

#### **WORDLIST ATTACK**

```
john --format=bitcoin wordlist=dict.txt hash.txt
```

#### **WORDLIST + RULE ATTACK**

```
john --format=bitcoin wordlist=dict.txt --rules hash.txt
```

## HASHCAT

### **HASH FORMAT**

```
username:$ml$35714$50973de90d336b5258f01e48ab324aa9ac81ca7959ac470d3d9c  
398$631a0ef84081b37cf3a63173cd2ec25047b89457ed300f2b41b30a07  
C5f3f7be8f74b7269ee3713172642de96ee482432a8d12bf291a
```

### **EXTRACT HASH**

```
sudo plist2hashcat.py /var/db/dslocal/nodes/Default/users/<username>.plist
```

### **BRUTE FORCE ATTACK**

```
hashcat -m 122 -a 3 hash.txt ?a?a?a?a?a?a
```

### **WORDLIST ATTACK**

```
hashcat -m 122 -a 0 hash.txt dict.txt
```

### **WORDLIST + RULE ATTACK**

```
hashcat -m 122 -a 0 hash.txt dict.txt -r rule.txt
```

## JOHN

### **HASH FORMAT**

```
username:$pbkdf2-hmac-  
sha512$31724.019739e90d326b5258f01e483b124aa9ac81ca7959acb  
70c3d9c4297af924398.631a0bf84081b37dae594a5468cf3a63183cd2ec25047b89457.  
1b40a0793a39512fc5a3f7ae8f74b7269ee3723172642de96eee82432a8dllbf365e:  
501:20 : HOST NAME:/bin/bash:/var/db/dslocal/nodes/Default/users/username.plist
```

### **EXTRACT HASH**

```
sudo ml2john.py /var/db/dslocal/nodes/Default/users/<username>.plist
```

### **BRUTE FORCE ATTACK**

```
john --format=xsha hash.txt
```

### **WORDLIST ATTACK**

```
john --format=xsha wordlist=dict.txt hash.txt
```

### **WORDLIST + RULE ATTACK**

```
john --format=xsha wordlist=dict.txt --rules hash.txt
```

## **MYSQL4.1 / MYSQL5+ (DOUBLE SHA1)**

## HASHCAT

### **HASH FORMAT**

```
FCF7C1B8749CF99D88E5F34271D636178FB5D130
```

### **EXTRACT HASH**

```
SELECT user,password FROM mysql.user INTO OUTFILE '/tmp/hash.txt';
```

## **BRUTE FORCE ATTACK**

hashcat -m 300 -a 3 hash.txt ?a?a?a?a?a?a

## **WORDLIST ATTACK**

hashcat -m 300 -a 0 hash.txt dict.txt

## **WORDLIST + RULE ATTACK**

hashcat -m 300 -a 0 hash.txt dict.txt -r rule.txt

## JOHN

### **HASH FORMAT**

\*FCF7C1B8749CF99D88E5F34271D636178FB5D130

### **EXTRACT HASH**

SELECT user,password FROM mysql.user INTO OUTFILE '/tmp/hash.txt';

## **BRUTE FORCE ATTACK**

john --format=mysql-sha1 hash.txt

## **WORDLIST ATTACK**

john --format=mysql-sha1 wordlist=dict.txt hash.txt

# **POSTGRESQL**

## HASHCAT

### **HASH FORMAT**

a6343a68d964ca596d9752250d54bb8a:postgres

### **EXTRACT HASH**

SELECT username, passwd FROM pg\_shadow;

## **BRUTE FORCE ATTACK**

hashcat -m 12 -a 3 hash.txt ?a?a?a?a?a?a

## **WORDLIST ATTACK**

hashcat -m 12 -a 0 hash.txt dict.txt

## **WORDLIST + RULE ATTACK**

hashcat -m 12 -a 0 hash.txt dict.txt -r rule.txt

## JOHN

### **HASH FORMAT**

a6343a68d964ca596d9752250d54bb8a:postgres

### **EXTRACT HASH**

SELECT username, passwd FROM pg\_shadow;

## **BRUTE FORCE ATTACK**

john --format=postgres hash.txt

## **WORDLIST ATTACK**

john --format=postgres wordlist=dict.txt hash.txt

## **WORDLIST + RULE ATTACK**

john --format=postgres wordlist=dict.txt --rules hash.txt

# **MSSQL(2012), MSSQL(2014)**

## HASHCAT

### **HASH FORMAT**

0x02000102030434ealbl7802fd95ea6316bd61d2c94622ca3812793e8fb1672487b5c91  
2ab4a78890d563d2fcf5663e46fe797d71550494be50cf4915d3f4d55ec375

### **EXTRACT HASH**

SELECT SL.name,SL.password\_hash FROM sys.sql\_logins AS SL;

### **BRUTE FORCE ATTACK**

hashcat -m 1731 -a 3 hash.txt ?a?a?a?a?a?a

### **WORDLIST ATTACK**

hashcat -m 1731 -a 0 hash.txt dict.txt

### **WORDLIST + RULE ATTACK**

hashcat -m 1731 -a 0 hash.txt dict.txt -r rule.txt

## JOHN

### **HASH FORMAT**

0x02000102030434ealbl7802fd95ea6316bd61d2c94622ca3812793e8fb1672487b5c91  
2ab4a78890d563d2fcf5663e46fe797d71550494be50cf4915d3f4d55ec375

### **EXTRACT HASH**

SELECT SL.name,SL.password\_hash FROM sys.sql\_logins AS SL;

### **BRUTE FORCE ATTACK**

john --format=mssql12 hash.txt

### **WORDLIST ATTACK**

john --format=mssql12 wordlist=dict.txt hash.txt

### **WORDLIST + RULE ATTACK**

john --format=mssql12 wordlist=dict.txt --rules hash.txt

# **ORACLE 11G**

## HASHCAT

### **HASH FORMAT**

ac5fle62d21fd0529428b84d42e8955b04966703:38445748184477378130

### **EXTRACT HASH**

SELECT SL.name,SL.password\_hash FROM sys.sql\_logins AS SL;

### **BRUTE FORCE ATTACK**

hashcat -m 112 -a 3 hash.txt ?a?a?a?a?a?a

### **WORDLIST ATTACK**

hashcat -m 112 -a 0 hash.txt dict.txt

### **WORDLIST + RULE ATTACK**

hashcat -m 112 -a 0 hash.txt dict.txt -r rule.txt

## JOHN

### **HASH FORMAT**

ac5fle62d21fd0529428b84d42e8955b04966703:38445748184477378130

### **EXTRACT HASH**

SELECT SL.name,SL.password\_hash FROM sys.sql\_logins AS SL;

### **BRUTE FORCE ATTACK**

john --format=oracle11 hash.txt

### **WORDLIST ATTACK**

john --format=oracle11 wordlist=dict.txt hash.txt

### **WORDLIST + RULE ATTACK**

john --format=oracle11 wordlist=dict.txt --rules hash.txt

## **CISCO TYPE 4 (SHA256)**

## HASHCAT

### **HASH FORMAT**

2btjjy78REtmYkkW0csHUbDZ0stRXoWdX1mGrmmfeHI

### **BRUTE FORCE ATTACK**

hashcat -m 5700 -a 3 hash.txt ?a?a?a?a?a?a

### **WORDLIST ATTACK**

hashcat -m 5700 -a 0 hash.txt dict.txt

### **WORDLIST + RULE ATTACK**

hashcat -m 5700 -a 0 hash.txt dict.txt -r rule.txt

## CISCO TYPE 5 (MD5)

HASHCAT

**HASH FORMAT**

\$1\$28772684\$iEwN0gGugq09.bIz5sk8k/

**BRUTE FORCE ATTACK**

hashcat -m 500 -a 3 hash.txt ?a?a?a?a?a?a

**WORDLIST ATTACK**

hashcat -m 500 -a 0 hash.txt dict.txt

**WORDLIST + RULE ATTACK**

hashcat -m 500 -a 0 hash.txt dict.txt -r rule.txt

JOHN

**HASH FORMAT**

\$1\$28772684\$iEwN0gGugq09.bIz5sk8k/

**BRUTE FORCE ATTACK**

john --format=md5crypt hash.txt

**WORDLIST ATTACK**

john --format=md5crypt wordlist=dict.txt hash.txt

**WORDLIST + RULE ATTACK**

john --format=md5crypt wordlist=dict.txt --rules hash.txt

## CISCO TYPE 8 (PBKDF2+SHA256)

HASHCAT

**HASH FORMAT**

\$8\$TnGX/fE4KGH0VU\$pEhnEvxrvaynpi8j4f.EMHr6M.FzU8xnZnBr/tJdFWk

**BRUTE FORCE ATTACK**

hashcat -m 9200 -a 3 hash.txt ?a?a?a?a?a?a

**WORDLIST ATTACK**

hashcat -m 9200 -a 0 hash.txt dict.txt

**WORDLIST + RULE ATTACK**

hashcat -m 9200 -a 0 hash.txt dict.txt -r rule.txt

## JOHN

### **HASH FORMAT**

```
$8$TnGX/fE4KGH0VU$pEhnEvxrvaynpi8j4f.EMHr6M.FzU8xnZnBr/tJdFWk
```

### **BRUTE FORCE ATTACK**

```
john --format=pbkdf2-hmac-sha256 hash.txt
```

### **WORDLIST ATTACK**

```
john --format=pbkdf2-hmac-sha256 wordlist=dict.txt hash.txt
```

### **WORDLIST + RULE ATTACK**

```
john --format=pbkdf2-hmac-sha256 wordlist=dict.txt --rules hash.txt
```

## **CISCO TYPE 9 (SCRYPT)**

## HASHCAT

### **HASH FORMAT**

```
$9$2MJBozw/9R3UsU$21FhcKvpghcyw8deP25G0fyZaagyU0GBymkryv0dfo6
```

### **BRUTE FORCE ATTACK**

```
hashcat -m 9300 -a 3 hash.txt ?a?a?a?a?a?a
```

### **WORDLIST ATTACK**

```
hashcat -m 9300 -a 0 hash.txt dict.txt
```

### **WORDLIST + RULE ATTACK**

```
hashcat -m 9300 -a 0 hash.txt dict.txt -r rule.txt
```

## JOHN

### **HASH FORMAT**

```
$9$2MJBozw/9R3UsU$21FhcKvpghcyw8deP25G0fyZaagyU0GBymkryv0dfo6
```

### **BRUTE FORCE ATTACK**

```
john --format=scrypt hash.txt
```

### **WORDLIST ATTACK**

```
john --format=scrypt wordlist=dict.txt hash.txt
```

### **WORDLIST + RULE ATTACK**

```
john --format=scrypt wordlist=dict.txt --rules hash.txt
```

## **WPA PSK / WPA2 PSK**

## HASHCAT

### **HASH FORMAT**

\*Capture 4-way authentication handshake > capture.cap  
cap2hccapx.bin capture.cap capture\_out.hccapx

### **BRUTE FORCE ATTACK**

hashcat -m 2500 -a 3 capture\_out.hccapx ?a?a?a?a?a?a

### **WORDLIST ATTACK**

hashcat -m 2500 -a 3 capture\_out.hccapx dict.txt

### **WORDLIST + RULE ATTACK**

hashcat -a 0 capture\_out.hccapx dict.txt -r rule.txt

## JOHN

### **HASH FORMAT**

\*Capture 4-way authentication handshake > capture.cap  
cap2hccap.bin -e '<ESSID>' capture.cap capture\_out.hccap  
hccap2john capture\_out.hccap > jtr\_capture

### **BRUTE FORCE ATTACK**

john --format=wpapsk jtr\_capture

### **WORDLIST ATTACK**

john --format=wpapsk wordlist=dict.txt jtr\_capture

### **WORDLIST + RULE ATTACK**

john --format=wpapsk wordlist=dict.txt --rules jtr\_capture

# **APPENDIX**

## **TERMS**

**BRUTE-FORCE ATTACK** - the act of trying every possible combination of a given keyspace or character set for a given length

**DICTIONARY** - a collection of commons words, phrases, keyboard patterns, generated passwords, or leaked passwords, also known as a wordlist

**DICTIONARY ATTACK** - using a file containing common or known password combinations or words in an attempt to match a given hashing function's output by running said words through the same target hashing function

**HASH** - the fixed bit result of a hash function

**HASH FUNCTION** - maps data of arbitrary size to a bit string of a fixed size (a hash function) which is designed to also be a one-way function, that is, a function which is infeasible to invert

**ITERATIONS** - the number of times an algorithm is run over a given hash

**KEYSPACE** - the number of possible combinations for a given character set to the power of it's length (i.e. charset<sup>length</sup>)

**MASK ATTACK** - using placeholder representations to try all combinations of a given keyspace, similar to brute-force but more targeted and efficient

**PASSWORD ENTROPY** - an estimation of how difficult a password will be to crack given its character set and length

**PLAINTEXT** - unaltered text that hasn't been obscured or algorithmically

**altered through a hashing function**

**RAKING - generating random password rules/candidates in an attempt to discover a previously unknown matching password pattern**

**RAINBOW TABLE - a precomputed table of a targeted cryptographic hash function of a certain minimum and maximum character length**

**RULE ATTACK - similar to a programming language for generating candidate passwords based on some input such as a dictionary**

**SALT - random data that used as additional input to a one-way function**

**WORDLIST - a collection of commons words, phrases, keyboard patterns, generated passwords, or leaked passwords, also known as a dictionary**

## **TIME TABLE**

60 seconds	1 minute
3,600 seconds	1 hour
86,400 seconds	1 day
604,800 seconds	1 week
1,209,600 seconds	1 fortnight
2,419,200 seconds	1 month (30days)
31,536,000 seconds	1 year

## **ONLINE RESOURCES**

**JOHN**

<http://openwall.info/wiki/john>

<http://openwall.info/wiki/john/sample-non-hashes>

<http://pentestmonkey.net/cheat-sheet/john-the-ripper-hash-formats>

<https://countuponsecurity.com/2015/06/14/john-the-ripper-cheat-sheet/>

<https://xinn.org/blog/JtR-AD-Password-Auditing.html>

<https://www.owasp.org/images/a/af/2011-Supercharged-Slides-Redman-OWASP-Feb.pdf>

## HASHCAT

<https://hashcat.net/wiki/>

[https://hashcat.net/wiki/doku.php?id=hashcat\\_utils](https://hashcat.net/wiki/doku.php?id=hashcat_utils)

<https://hashcat.net/wiki/doku.php?id=statsprocessor>

<http://www.netmux.com/blog/ultimate-guide-to-cracking-foreign-character-passwords-using-has>

<http://www.netmux.com/blog/cracking-12-character-above-passwords>

## CRACKING RIGS

<http://www.netmux.com/blog/how-to-build-a-password-cracking-rig>

[https://www.unix-ninja.com/p/Building\\_a\\_Password\\_Cracking\\_Rig\\_for\\_Hashcat\\_-\\_Part\\_III](https://www.unix-ninja.com/p/Building_a_Password_Cracking_Rig_for_Hashcat_-_Part_III)

## EXAMPLE HASH GENERATION

<https://www.onlinehashcrack.com/hash-generator.php>

<https://www.tobtu.com/tools.php>

<http://hash.online-convert.com/>

[https://www.tools4noobs.com/online\\_tools/hash/](https://www.tools4noobs.com/online_tools/hash/)

<https://quickhash.com/>

<http://bitcoinvalued.com/tools.php>

<http://www.sha1-online.com/>

<http://www.freeformatter.com/hmac-generator.html>

<http://openwall.info/wiki/john/Generating-test-hashes>

## OTHER

<http://blog.thireus.com/cracking-story-how-i-cracked-over-122-million-sha1-and-md5-hashed-passwords/>

<http://www.utf8-chartable.de/>

<http://thesprawl.org/projects/pack/>

<https://blog.gotmilk.com/2011/06/dictionaries-wordlists/>

<http://wpengine.com/unmasked/>

[https://www.unix-ninja.com/p/A\\_cheat-sheet\\_for\\_password\\_crackers](https://www.unix-ninja.com/p/A_cheat-sheet_for_password_crackers)

<https://room362.com/post/2017/05-06-2017-password-magic-numbers/>

<http://www.netmux.com/blog/how-to-build-a-password-cracking-rig>

<http://passwordchart.com/>

<http://www.vigilante.pw>

## **NETMUX**

<http://www.netmux.com>

<http://www.hashcrack.io>

<https://github.com/netmux>

<https://twitter.com/netmux>

<https://www.instagram.com/netmux/>

**\*\*\*ANSWER TO CUSTOM DICTIONARY CREATION HASH:**

**e4821d16a298092638ddb7cadcc26d32f = letmein123456Netmux**

## **10 CRACK COMMANDMENTS**

1. Thou shalt know hash types and their origin/function
2. Thou shalt know cracking software strengths & weaknesses
3. Thou shalt study & apply password analysis techniques
4. Thou shalt be proficient at hash extraction methods
5. Thou shalt create custom/targeted dictionaries
6. Thou shalt know thy cracking rigs capabilities
7. Thou shalt understand basic human psychology/behavior
8. Thou shalt create custom masks, rules, and Markov chains
9. Thou shalt continually experiment with new techniques
10. Thou shalt support thy fellow cracking community members

## **JOHN THE RIPPER HELP MENU**

John the Ripper password cracker, version 1.8.0-jumbo-1 [darwin15.6.0 64-bit AVX2-autoconf]

Usage: john [OPTIONS] [PASSWORD-FILES]	
--single[=SECTION]	“single crack” mode
--wordlist[=FILE] --stdin	wordlist mode, read words from FILE or stdin
--pipe	like --stdin, but bulk reads, and allows rules
--loopback[=FILE]	like --wordlist, but fetch words from a .pot file
--dupe-suppression	suppress all dupes in wordlist (and force preload)
--encoding=NAME	input encoding (eg. UTF-8, ISO-8859-1). See also doc/ENCODING and --list=hidden-options.
--rules[=SECTION]	enable word mangling rules for wordlist modes
--incremental[=MODE]	“incremental” mode [using section MODE]
--mask=MASK	mask mode using MASK
--markov[=OPTIONS]	“Markov” mode (see doc/MARKOV)
--external=MODE	external mode or word filter
--stdout[=LENGTH]	just output candidate passwords [cut at LENGTH]
--restore[=NAME]	restore an interrupted session [called NAME]
--session=NAME	give a new session the NAME
--status[=NAME]	print status of a session [called NAME]
--make-charset=FILE	make a charset file. It will be overwritten
--show[=LEFT]	show cracked passwords [if =LEFT, then uncracked]
--test[=TIME]	run tests and benchmarks for TIME seconds each
--users=[-]LOGIN UID[...]	[do not] load this (these) user(s) only
--groups=[-]GID[...]	load users [not] of this (these) group(s) only
--shells=[-]SHELL[...]	load users with[out] this (these) shell(s)

--salts=[-]COUNT[:MAX] only  
load salts with [out] COUNT [to MAX] hashes

--save-memory=LEVEL enable memory saving, at LEVEL 1..3  
this node's number range out of TOTAL count

--node=MIN[-MAX]/TOTAL fork N processes

--fork=N pot file to use

--pot=NAME list capabilities, see --list=help or doc/OPTIONS

--list=WHAT set OpenCL device(s) (list using --list=opencl-

--devices=N[,...] devices) force hash type NAME:

--format=NAME

7z 7z-opencl AFS agilekeychain agilekeychain-opencl aix-smd5 aix-ssha1 aix-ssha256 aix-ssha512 asa-md5 bcrypt bcrypt-opencl bfegg Bitcoin blackberry-es10 Blockchain blockchain-opencl bsdicrypt chap Citrix\_NS10 Clipperz cloudkeychain cq CRC32 crypt dahua decrypt decrypt-opencl Django django-scrypt dmd5 dmg dmg-opencl dominosec dragonfly3-32 dragonfly3-64 dragonfly4-32 dragonfly4-64 Drupal7 dummy dynamic\_n eCryptfs EFS eigrp EncFS encfs-opencl EPI EPiServer fde FormSpring Fortigate gost gpg gpg-opencl HAVAL-128-4 HAVAL-256-3 hdAA HMAC-MD5 HMAC-SHA1 HMAC-SHA224 HMAC-SHA256 HMAC-SHA384 HMAC-SHA512 hMailServer hsrp IKE ipb2 KeePass keychain keychain-opencl keyring keyring-opencl keystore known\_hosts krb4 krb5 krb5-18 krb5pa-md5 krb5pa-md5-opencl krb5pa-sha1 krb5pa-sha1-opencl kwallet LastPass LM lotus5 lotus5-opencl lotus85 LUKS MD2 md4-gen md5crypt md5crypt-opencl md5ns mdc2 MediaWiki MongoDB Mozilla mscash mscash2 mscash2-opencl MSCHAPv2 mschapv2-naive mssql mssql05 mssql12 mysql mysql-sha1 mysql-sha1-opencl mysqlna net-md5 net-sha1 nethalflm netlm netlmv2 netntlm netntlm-naive netntlmv2 nk nsldap NT nt-opencl nt2 ntlmv2-opencl o51ogon o51ogon-opencl ODF ODF-AES-opencl ODF-opencl Office office2007-opencl office2010-opencl office2013-opencl oldoffice oldoffice-opencl OpenBSD-SoftRAID openssl-enc OpenVMS oracle oraclell osc Panama PBKDF2-HMAC-SHA1 PBKDF2-HMAC-SHA1-opencl PBKDF2-HMAC-SHA256 PBKDF2-HMAC-SHA256-opencl PBKDF2-HMAC-SHA512 pbkdf2-hmac-sha512-opencl PDF PFX phpass phpass-opencl PHPS pix-md5 PKZIP po postgres PST PuTTY pwsafe pwsafe-opencl RACF RAdmin RAKP RAKP-opencl rar rar-opencl RAR5 RAR5-opencl Raw-Blake2 Raw-Keccak Raw-Keccak-256 Raw-MD4 Raw-MD4-opencl Raw-MD5 Raw-MD5-opencl Raw-MD5u Raw-SHA Raw-SHA1 Raw-SHA1-Linkedin Raw-SHA1-ng Raw-SHA1-opencl Raw-SHA224 Raw-SHA256 Raw-SHA256-ng Raw-SHA256-opencl Raw-SHA384

Raw-SHA512 Raw-SHA512-ng Raw-SHA512-opencl ripemd-128 ripemd-160 rsvp  
Salted-SHA1 sapb sapg scrypt sha1-gen sha1crypt sha1crypt-opencl sha256crypt  
sha256crypt-opencl sha512crypt sha512crypt-opencl Siemens-S7 SIP skein-256  
skein-512 skey Snelfru-128 Snelfru-256 SSH SSH-ng ssha-opencl SSHA512 STRIP  
strip-opencl SunMD5 sxc sxc-opencl Sybase-PROP sybasease tc\_aes\_xts  
tc\_ripemd160 tc\_sha512 tc\_whirlpool tcp-md5 Tiger tripcode VNC vtp wbb3  
whirlpool whirlpool0 whirlpool0l WoWSRP wpapsk wpapsk-opencl xsha xsha512  
XSHA512-opencl ZIP zip-opencl

## **HASHCAT HELP MENU**

hashcat 3.6 - advanced password recovery

Usage: hashcat [options]... hash[hash-file|hccapxfile] [dictionary|mask| directory]...

- [ Options ] -

Options Short / Long	Description
-m, --hash-type	Hash-type, see references below
-a, --attack-mode	Attack-mode, see references below
-V, --version	Print version
-h, --help	Print help
--quiet	Suppress output
--hex-charset	Assume charset is given in hex
--hex-salt	Assume salt is given in hex
--hex-wordlist	Assume words in wordlist are given in hex
--force	Ignore warnings
--status	Enable automatic update of the status screen
--status-timer	Sets seconds between status screen updates to X
--machine-readable	Display status view in machine-readable format
--keep-guessing	Keep guessing hash after it has been cracked
--loopback	Add new plains to induct directory
--weak	Threshold X to stop checking for weak hashes
--markov-hcstat	Specify hcstat file to use
--markov-disable	Disables markov-chains, classic brute-force
--markov-classic	Enables classic markov-chains, no per-position
-t, --markov-threshold	Threshold X to stop accepting new markov-chains
--runtime	Abort session after X seconds of runtime
--session	Define specific session name
--restore	Restore session from --session
--restore-disable	Do not write restore file
--restore-file-path	Specific path to restore file
-o, --outfile	Define outfile for recovered hash
--outfile-format	Define outfile-format X for recovered hash
--outfile-autohex-disable	Disable the use of \$HEX[] in output plains
--outfile-check-timer	Sets seconds between outfile checks to X
-p, --separator	Separator char for hashlists and outfile
--stdout	Do not crack a hash, print candidates only
--show	Compare hashlist with potfile; show cracked
--left	Compare hashlist with potfile; show uncracked
--username	Enable ignoring of usernames in hashfile
--remove	Enable removal of hashes once they are cracked
--remove-timer	Update input hash file each X seconds
--potfile-disable	Do not write potfile
--potfile-path	Specific path to potfile
--encoding-from	Force internal wordlist encoding from X
--encoding-to	Force internal wordlist encoding to X
--debug-mode	Defines the debug mode 1-4
--debug-file	Output file for debugging rules
--induction-dir	Specify induction directory to use for loopback
--outfile-check-dir	Specify outfile directory to monitor for plains
--logfile-disable	Disable the logfile
--hccapx-message-pair	Load only message pairs from hccapx matching X
--nonce-error-corrections	BF size range to replace AP's nonce last bytes
--truecrypt-keyfiles	Keyfiles to use, separated with commas
--veracrypt-keyfiles	Keyfiles to use, separated with commas
--veracrypt-pim	VeraCrypt personal iterations multiplier
-b, --benchmark	Run benchmark

--speed-only	Return expected speed of the attack, then quit
--progress-only	Return ideal step size and time to process   X
-c,	Sets size in MB to cache from the wordfile to X
--segment-size	Sets minimum bits allowed for bitmaps to X
--bitmap-min	Sets maximum bits allowed for bitmaps to X
--bitmap-max	Locks to CPU devices, separated with commas
--cpu-affinity	Show info on detected OpenCL platforms/devices
-I, --opencl-info	OpenCL platforms to use, separated with commas
--opencl-platforms	OpenCL devices to use, separated with commas
-d, --opencl-devices	OpenCL device-types to use, separate with commas
-D, --opencl-device-types	Manually override OpenCL vector-width to X
--opencl-vector-width	Enable a specific workload profile, 1-4
-w, --workload-profile	Workload tuning, set outerloop step size to X
-n, --kernel-accel	Workload tuning, set innerloop step size to X
-u, --kernel-loops	Workaround NVIDIA's CPU burning loop bug
--nvidia-spin-damp	Disable temperature, fanspeed reads, triggers
--gpu-temp-disable	Abort if GPU temperature reaches X degrees C
--gpu-temp-abort	Try to retain GPU temperature at X degrees C
--gpu-temp-retain	Enable power tuning. Restores settings
--powertune-enable	Manually override TMTO value for scrypt to X
--scrypt-tmto	Skip X words from the start
-s, --skip	Limit X words from the start + skipped words
-l, --limit	Show keyspace base:mod values and quit
--keyspace	Single rule applied to word from left wordlist
-j, --rule-left	Single rule applied to word from right wordlist
-k, --rule-right	Multiple rules applied to word from wordlists
-r, --rules-file	Generate X random rules
-g, --generate-rules	Force min X functions per rule
--generate-rules-func-min	Force max X functions per rule
--generate-rules-func-max	Force RNG seed set to X
--generate-rules-seed	User-defined charset ?1
-1, --custom-charset1	User-defined charset ?2
-2, --custom-charset2	User-defined charset ?3
-3, --custom-charset3	User-defined charset ?4
-4, --custom-charset4	Enable mask increment mode
-i, --increment	Start mask incrementing at X
--increment-min	Stop mask incrementing at X
--increment-max	

## - [ Hash modes ] -

#	Name	Category
<hr/>		
900	MD4	Raw Hash
0	MD5	Raw Hash
5100	Half MD5	Raw Hash
100	SHA1	Raw Hash
1300	SHA-224	Raw Hash
1400	SHA-256	Raw Hash
10800	SHA-384	Raw Hash
1700	SHA-512	Raw Hash
5000	SHA-3 (Keccak)	Raw Hash
600	BLAKE2b-512	Raw Hash
10100	SipHash	Raw Hash
6000	RIPEMD-160	Raw Hash
6100	Whirlpool	Raw Hash
6900	GOST R 34.11-94	Raw Hash
11700	GOST R 34.11-2012 (Streebog) 256-bit	Raw Hash
11800	GOST R 34.11-2012 (Streebog) 512-bit	Raw Hash
10	md5(\$pass.\$salt)	Raw Hash, Salted and/or Iterated
20	md5(\$salt.\$pass)	Raw Hash, Salted and/or Iterated
30	md5(utf16le(\$pass).\$salt)	Raw Hash, Salted and/or Iterated
40	md5(\$salt.utf16le(\$pass))	Raw Hash, Salted and/or Iterated
3800	md5(\$salt.\$pass.\$salt)	Raw Hash, Salted and/or Iterated
3710	md5(\$salt.md5(\$pass))	Raw Hash, Salted and/or Iterated

4010	md5(\$salt.md5(\$salt.\$pass))	Raw Hash, Salted and/or Iterated
4110	md5(\$salt.md5(\$pass.\$salt))	Raw Hash, Salted and/or Iterated
2600	md5(md5(\$pass))	Raw Hash, Salted and/or Iterated
3910	md5(md5(\$pass).md5(\$salt))	Raw Hash, Salted and/or Iterated
4300	md5(strtoupper(md5(\$pass)))	Raw Hash, Salted and/or Iterated
4400	md5(shai(\$pass))	Raw Hash, Salted and/or Iterated
110	sha1(\$pass.\$salt)	Raw Hash, Salted and/or Iterated
120	sha1(\$salt.\$pass)	Raw Hash, Salted and/or Iterated
130	sha1(utf16le(\$pass).\$salt)	Raw Hash, Salted and/or Iterated
140	sha1(\$salt.utf16le(\$pass))	Raw Hash, Salted and/or Iterated
4500	sha1(shai(\$pass))	Raw Hash, Salted and/or Iterated
4520	sha1(\$salt.sha1(\$pass))	Raw Hash, Salted and/or Iterated
4700	sha1(md5(\$pass))	Raw Hash, Salted and/or Iterated
4900	sha1(\$salt.\$pass.\$salt)	Raw Hash, Salted and/or Iterated
14400	sha1(CX)	Raw Hash, Salted and/or Iterated
1410	sha256(\$pass.\$salt)	Raw Hash, Salted and/or Iterated
1420	sha256(\$salt.\$pass)	Raw Hash, Salted and/or Iterated
1430	sha256(utf16le(\$pass).\$salt)	Raw Hash, Salted and/or Iterated
1440	sha256(\$salt.utf16le(\$pass))	Raw Hash, Salted and/or Iterated
1710	sha512(\$pass.\$salt)	Raw Hash, Salted and/or Iterated
1720	sha512(\$salt.\$pass)	Raw Hash, Salted and/or Iterated
1730	sha512(utf16le(\$pass).\$salt)	Raw Hash, Salted and/or Iterated
1740	sha512(\$salt.utf16le(\$pass))	Raw Hash, Salted and/or Iterated
50	HMAC-MD5 (key = \$pass)	Raw Hash, Authenticated
60	HMAC-MD5 (key = \$salt)	Raw Hash, Authenticated
150	HMAC-SHA1 (key = \$pass)	Raw Hash, Authenticated
160	HMAC-SHA1 (key = \$salt)	Raw Hash, Authenticated
1450	HMAC-SHA256 (key = \$pass)	Raw Hash, Authenticated
1460	HMAC-SHA256 (key = \$salt)	Raw Hash, Authenticated
1750	HMAC-SHA512 (key = \$pass)	Raw Hash, Authenticated
1760	HMAC-SHA512 (key = \$salt)	Raw Hash, Authenticated
14000	DES (PT = \$salt, key = \$pass)	Raw Cipher, Known-Plaintext
14100	3DES (PT = \$salt, key = \$pass)	Raw Cipher, Known-Plaintext
14900	Skip32 (PT = \$salt, key = \$pass)	Raw Cipher, Known-Plaintext
15400	ChaCha20	Raw Cipher, Known-Plaintext
400	phpass	Generic KDF
8900	scrypt	Generic KDF
11900	PBKDF2-HMAC-MDS	Generic KDF
12000	PBKDF2-HMAC-SHA1	Generic KDF
10900	PBKDF2-HMAC-SHA256	Generic KDF
12100	PBKDF2-HMAC-SHA512	Generic KDF
23	Skype	Network Protocols
2500	WPA/WPA2	Network Protocols
2501	WPA/WPA2 PMK	Network Protocols
4800	iSCSI CHAP authentication, MDS(CHAP)	Network Protocols
5300	IKE-PSK MDS	Network Protocols
5400	IKE-PSK SHA1	Network Protocols
5500	NetNTLMv1	Network Protocols
5500	NetNTLMv1+ESS	Network Protocols
5600	NetNTLMv2	Network Protocols
7300	IPMI2 RAKP HMAC-SHA1	Network Protocols
7500	Kerberos 5 AS-REQ Pre-Auth etype 23	Network Protocols
8300	DNSSEC (NSEC3)	Network Protocols
10200	CRAM-MDS	Network Protocols
11100	PostgreSQL CRAM (MD5)	Network Protocols
11200	MySQL CRAM (SHA1)	Network Protocols
11400	SIP digest authentication (MD5)	Network Protocols
13100	Kerberos 5 TGS-REP etype 23	Network Protocols
121	SMF (Simple Machines Forum) > v1.1	Forums,CMS,E-Commerce
400	phpBB3 (MD5)	Forums, CMS, E-Commerce
2611	vBulletin < v3.8.5	Forums, CMS, E-Commerce
2711	vBulletin >= v3.8.5	Forums, CMS, E-Commerce
2811	MyBB 1.2+	Forums, CMS, E-Commerce
2811	IPB2+ (Invision Power Board)	Forums, CMS, E-Commerce
8400	WBB3 (Woltlab Burning Board)	Forums, CMS, E-Commerce
11	Joomla < 2.5.18	Forums, CMS, E-Commerce
400	Joomla >= 2.5.18 (MD5)	Forums, CMS, E-Commerce
400	WordPress (MD5)	Forums, CMS, E-Commerce
2612	PHPS	Forums, CMS, E-Commerce
7900	Drupal7	Forums, CMS, E-Commerce
21	osCommerce	Forums, CMS, E-Commerce
21	xt:Commerce	Forums, CMS, E-Commerce

11000	PrestaShop	Forums, CMS, E-Commerce
124	Django (SHA-1)	Forums, CMS, E-Commerce
10000	Django (PBKDF2-SHA256)	Forums, CMS, E-Commerce
3711	MediaWiki B type	Forums, CMS, E-Commerce
13900	OpenCart	Forums, CMS, E-Commerce
4521	Redmine	Forums, CMS, E-Commerce
4522	PunBB	Forums, CMS, E-Commerce
12001	Atlassian (PBKDF2-HMAC-SHA1)	Forums, CMS, E-Commerce
12	PostgreSQL	Database Server
131	MSSQL (2000)	Database Server
132	MSSQL (2005)	Database Server
1731	MSSQL (2012, 2014)	Database Server
200	MySQL323	Database Server
300	MySQL4.1/MySQL5	Database Server
3100	Oracle H: Type (Oracle 7+)	Database Server
112	Oracle S: Type (Oracle 11+)	Database Server
12300	Oracle T: Type (Oracle 12+)	Database Server
8000	Sybase ASE	Database Server
141	Episerver 6.x < .NET 4	HTTP, SMTP, LDAP Server
1441	Episerver 6.x >= .NET 4	HTTP, SMTP, LDAP Server
1600	Apache \$apr1\$ MD5, md5apr1, MD5 (APR)	HTTP, SMTP, LDAP Server
12600	ColdFusion 10+	HTTP, SMTP, LDAP Server
1421	hMailServer	HTTP, SMTP, LDAP Server
101	nsldap, SHA-1(Base64), Netscape LDAP SHA	HTTP, SMTP, LDAP Server
111	nsldaps, SSHA-1(Base64), Netscape LDAP SSHA	HTTP, SMTP, LDAP Server
1411	SSHA-256(Base64), LDAP {SSHA256}	HTTP, SMTP, LDAP Server
1711	SSHA-512(Base64), LDAP {SSHA512}	HTTP, SMTP, LDAP Server
15000	FileZilla Server >= 0.9.55	FTP Server
11500	CRC32	Checksums
3000	LM	Operating Systems
1000	NTLM	Operating Systems
1100	Domain Cached Credentials (DCC), MS Cache	Operating Systems
2100	Domain Cached Credentials 2 (DCC2), MS Cache 2	Operating Systems
15300	DPAPI masterkey file v1 and v2	Operating Systems
12800	MS-AzureSync PBKDF2-HMAC-SHA256	Operating Systems
1500	decrypt, DES (Unix), Traditional DES	Operating Systems
12400	BSDi Crypt, Extended DES	Operating Systems
500	md5crypt, MD5 (Unix), Cisco-IOS \$1\$ (MD5)	Operating Systems
3200	bcrypt \$2*\$, Blowfish (Unix)	Operating Systems
7400	sha256crypt \$5\$, SHA256 (Unix)	Operating Systems
1800	sha512crypt \$6\$, SHA512 (Unix)	Operating Systems
122	OSX v10.4, OSX v10.5, OSX v10.6	Operating Systems
1722	OSX v10.7	Operating Systems
7100	OSX v10.8+ (PBKDF2-SHA512)	Operating Systems
6300	AIX {smd5}	Operating Systems
6700	AIX {ssha1}	Operating Systems
6400	AIX {ssha256}	Operating Systems
6500	AIX {ssha512}	Operating Systems
2400	Cisco-PIX MD5	Operating Systems
2410	Cisco-ASA MD5	Operating Systems
500	Cisco-IOS \$1\$ (MD5)	Operating Systems
5700	Cisco-IOS type 4 (SHA256)	Operating Systems
9200	Cisco-IOS \$8\$ (PBKDF2-SHA256)	Operating Systems
9300	Cisco-IOS \$9\$ (scrypt)	Operating Systems
22	Juniper NetScreen/SSG (ScreenOS)	Operating Systems
501	Juniper IVE	Operating Systems
15100	Juniper/NetBSD sha1crypt	Operating Systems
7000	FortiGate (FortiOS)	Operating Systems
5800	Samsung Android Password/PIN	Operating Systems
13800	Windows Phone 8+ PIN/password	Operating Systems
8100	Citrix NetScaler	Operating Systems
8500	RACF	Operating Systems
7200	GRUB 2	Operating Systems
9900	Radmin2	Operating Systems
125	ArubaOS	Operating Systems
7700	SAP CODVN B (BCODE)	Enterprise Application Software
7800	SAP CODVN F/G (PASSCODE)	Enterprise Application Software
10300	SAP CODVN H (PWDSALTEDHASH) iSSHA-1	Enterprise Application Software
8600	Lotus Notes/Domino 5	Enterprise Application Software
8700	Lotus Notes/Domino 6	Enterprise Application Software
9100	Lotus Notes/Domino 8	Enterprise Application Software
133	PeopleSoft	Enterprise Application Software

13500	PeopleSoft PS_TOKEN	Enterprise Application Software
11600	7-Zip	Archives
12500	RAR3-hp	Archives
13000	RAR5	Archives
13200	AxCrypt	Archives
13300	AxCrypt in-memory SHA1	Archives
13600	WinZip	Archives
14700	iTunes backup < 10.0	Backup
14800	iTunes backup >= 10.0	Backup
62XY	TrueCrypt	Full-Disk Encryption (FDE)
	X 1 = PBKDF2-HMAC-RIPEMD160	Full-Disk Encryption (FDE)
	X 2 = PBKDF2-HMAC-SHA512	Full-Disk Encryption (FDE)
	X 3 = PBKDF2-HMAC-Whirlpool	Full-Disk Encryption (FDE)
	X 4 = PBKDF2-HMAC-RIPEMD160 + boot-mode	Full-Disk Encryption (FDE)
	Y 1 = XTS 512 bit pure AES	Full-Disk Encryption (FDE)
	Y 1 = XTS 512 bit pure Serpent	Full-Disk Encryption (FDE)
	Y 1 = XTS 512 bit pure Twofish	Full-Disk Encryption (FDE)
	Y 2 = XTS 1024 bit pure AES	Full-Disk Encryption (FDE)
	Y 2 = XTS 1024 bit pure Serpent	Full-Disk Encryption (FDE)
	Y 2 = XTS 1024 bit pure Twofish	Full-Disk Encryption (FDE)
	Y 2 = XTS 1024 bit cascaded AES-Twofish	Full-Disk Encryption (FDE)
	Y 2 = XTS 1024 bit cascaded Serpent-AES	Full-Disk Encryption (FDE)
	Y 2 = XTS 1024 bit cascaded Twofish-Serpent	Full-Disk Encryption (FDE)
	Y 3 = XTS 1536 bit all	Full-Disk Encryption (FDE)
8800	Android FDE <= 4.3	Full-Disk Encryption (FDE)
12900	Android FDE (Samsung DEK)	Full-Disk Encryption (FDE)
12200	eCryptfs	Full-Disk Encryption (FDE)
137XY	VeraCrypt	Full-Disk Encryption (FDE)
	X 1 = PBKDF2-HMAC-RIPEMD160	Full-Disk Encryption (FDE)
	X 2 = PBKDF2-HMAC-SHA512	Full-Disk Encryption (FDE)
	X 3 = PBKDF2-HMAC-Whirlpool	Full-Disk Encryption (FDE)
	X 4 = PBKDF2-HMAC-RIPEMD160 + boot-mode	Full-Disk Encryption (FDE)
	X 5 = PBKDF2-HMAC-SHA256	Full-Disk Encryption (FDE)
	X 6 = PBKDF2-HMAC-SHA256 + boot-mode	Full-Disk Encryption (FDE)
	Y 1 = XTS 512 bit pure AES	Full-Disk Encryption (FDE)
	Y 1 = XTS 512 bit pure Serpent	Full-Disk Encryption (FDE)
	Y 1 = XTS 512 bit pure Twofish	Full-Disk Encryption (FDE)
	Y 2 = XTS 1024 bit pure AES	Full-Disk Encryption (FDE)
	Y 2 = XTS 1024 bit pure Serpent	Full-Disk Encryption (FDE)
	Y 2 = XTS 1024 bit pure Twofish	Full-Disk Encryption (FDE)
	Y 2 = XTS 1024 bit cascaded AES-Twofish	Full-Disk Encryption (FDE)
	Y 2 = XTS 1024 bit cascaded Serpent-AES	Full-Disk Encryption (FDE)
	Y 2 = XTS 1024 bit cascaded Twofish-Serpent	Full-Disk Encryption (FDE)
	Y 3 = XTS 1536 bit all	Full-Disk Encryption (FDE)
14600	LUKS	Full-Disk Encryption (FDE)
9700	MS Office <= 2003 \$0/\$1, MD5 + RC4	Documents
9710	MS Office <= 2003 \$0/\$1, MD5 + RC4, collider #1	Documents
9720	MS Office <= 2003 \$0/\$1, MD5 + RC4, collider #2	Documents
9800	MS Office <= 2003 \$3/\$4, SHA1 + RC4	Documents
9810	MS Office <= 2003 \$3, SHA1 + RC4, collider #1	Documents
9820	MS Office <= 2003 \$3, SHA1 + RC4, collider #2	Documents
9400	MS Office 2007	Documents
9500	MS Office 2010	Documents
9600	MS Office 2013	Documents
10400	PDF 1.1 - 1.3 (Acrobat 2 - 4)	Documents
10410	PDF 1.1 - 1.3 (Acrobat 2 - 4), collider #1	Documents
10420	PDF 1.1 - 1.3 (Acrobat 2 - 4), collider #2	Documents
10500	PDF 1.4 - 1.6 (Acrobat 5 - 8)	Documents
10600	PDF 1.7 Level 3 (Acrobat 9)	Documents
10700	PDF 1.7 Level 8 (Acrobat 10 - 11)	Documents
9000	Password Safe v2	Password Managers
5200	Password Safe v3	Password Managers
6800	LastPass + LastPass sniffed	Password Managers
6600	1Password, agilekeychain	Password Managers
8200	1Password, cloudkeychain	Password Managers
11300	Bitcoin/Litecoin wallet.dat	Password Managers
12700	Blockchain, My Wallet	Password Managers
15200	Blockchain, My Wallet, V2	Password Managers
13400	KeePass 1 (AES/Twofish) and KeePass 2 (AES)	Password Managers
15500	JKS Java Key Store Private Keys (SHA1)	Password Managers
15600	Ethereum Wallet, PBKDF2-HMAC-SHA256	Password Managers
15700	Ethereum Wallet, SCRYPT	Password Managers

## - [ Outfile Formats ] -

#	Format
1	hash[:salt]
2	plain
3	hash[:salt]:plain
4	hex_plain
5	hash[:salt]:hex_plain
6	plain:hex_plain
7	hash[:salt]:plain:hex_plain
8	crackpos
9	hash[:salt]:crack_pos
10	plain:crack_pos
11	hash[:salt]:plain:crack_pos
12	hex_plain:crack_pos
13	hash[:salt]:hex_plain:crack_pos
14	plain:hex_plain:crack_pos
15	hash[:salt]:plain:hex_plain:crack_pos

## - [ Rule Debugging Modes ] -

#	Format
1	Finding-Rule
2	Original-Word
3	Original-Word:Finding-Rule
4	Original-Word:Finding-Rule:Processed-Word

## - [ Attack Modes ] -

#	Mode
0	Straight
1	Combination
3	Brute-force
6	Hybrid Wordlist + Mask
7	Hybrid Mask + Wordlist

## - [ Built-in Charsets ] -

?	Charset
l	abcdefghijklmnopqrstuvwxyz
U	ABCDEFGHIJKLMNOPQRSTUVWXYZ
d	0123456789
h	0123456789abcdef
H	0123456789ABCDEF
s	!"#\$%&(')*+, - ./ : ; <=> ? @ [ \ ] ^ _ ` {   } ~
a	?1?u?d?s
b	0x00 - 0xff

## - [ OpenCL Device Types ] -

#	Device Type
1	CPU
2	GPU
3	FPGA, DSP, Co-Processor

## - [ Workload Profiles ] -

#	Performance	Runtime	Power Consumption	Desktop Impact
1	Low	2 ms	Low	Minimal
2	Default	12 ms	Economic	Noticeable
3	High	96 ms	High	Unresponsive
4	Nightmare	480 ms	Insane	Headless

- [ Basic Examples ] -

Attack-Mode	Hash-Type	Example command
Wordlist	\$P\$	hashcat -a 0 -m 400 example400.hash example.dict
Wordlist + Rules rules/best64.rule	MD5	hashcat -a 0 -m 0 example0.hash example.dict -r
Brute-Force Combinator	MD5	hashcat -a 3 -m 0 example0.hash ?a?a?a?a?a? hashcat -a 1 -m 0 example0.hash example.dict example.dict

If you still have no idea what just happened, try the following pages:

- \* [https://hashcat.net/wiki/#howtos\\_videos\\_papers\\_articles\\_etc\\_in\\_the\\_wild](https://hashcat.net/wiki/#howtos_videos_papers_articles_etc_in_the_wild)
- \* <https://hashcat.net/faq/>

*\*\*\*HASH CRACKING BENCHMARK tables are meant to be a reference to enable users to gauge how SLOW or FAST a hashing algorithm is before formulating an attack plan. Nvidia GTX2080 was chosen as the default due to its prevalence among the cracking community and it's position as a top performing GPU card.*

## HASH CRACKING BENCHMARKS (ALPHABETICAL)

1Password, agilekeychain	3319.2 kH/s
1Password, cloudkeychain	10713 H/s
3DES (PT = \$salt, key = \$pass)	594.3 MH/s
7-Zip	7514 H/s
AIX	14937.2 kH/s
AIX	44926.1 kH/s
AIX	6359.3 kH/s
AIX	9937.1 kH/s
Android FDE (Samsung DEK)	291.8 kH/s
Android FDE <= 4.3	803.0 kH/s
Android PIN	5419.4 kH/s
ArubaOS	6894.7 MH/s
Atlassian (PBKDF2-HMAC-SHA1)	283.6 kH/s
AxCrypt	113.9 kH/s
AxCrypt in memory SHA1	7503.3 MH/s
bcrypt, Blowfish(OpenBSD)	13094 H/s
BSDiCrypt, Extended DES	1552.5 kH/s
Bitcoin/Litecoin wallet.dat	4508 H/s
BLAKE2-512	1488.9 MH/s
Blockchain, My Wallet	50052.3 kH/s
Blockchain, My Wallet, V2	305.2 kH/s
ChaCha20	3962.0 MH/s
Cisco \$8\$	59950 H/s
Cisco \$9\$	22465 H/s
Cisco-ASA MD5	17727.2 MH/s
Cisco-IOS SHA256	2864.3 MH/s

Cisco-PIX MD5	16407.2 MH/s
Citrix NetScaler	7395.3 MH/s
ColdFusion 10+	1733.6 MH/s
DES (PT = \$salt, key = \$pass)	19185.7 MH/s
decrypt, DES(Unix), Traditional DES	906.7 MH/s
DNSSEC (NSEC3)	3274.6 MH/s
Django (PBKDF2-SHA256)	59428 H/s
Django (SHA-1)	6822.6 MH/s
Domain Cached Credentials (DCC), MS Cache	11195.8 MH/s
Domain Cached Credentials 2 (DCC2), MS Cache 2	317.5 kH/s
DPAPI masterkey file v1 and v2	73901 H/s
Drupal7	56415 H/s
eCryptfs	13813 H/s
Ethereum Wallet, PBKDF2-HMAC-SHA256	4518 H/s
Ethereum Wallet, SCRYPT	29 H/s
EPiServer 6.x < v4	6818.5 MH/s
EPiServer 6.x > v4	2514.4 MH/s
FileZilla Server >= 0.9.55	565.2 MH/s
FortiGate (FortiOS)	6386.2 MH/s
GOST R 34.11-2012 (Streebog) 256-bit	50018.8 kH/s
GOST R 34.11-2012 (Streebog) 512-bit	49979.4 kH/s
GOST R 34.11-94	206.2 MH/s
GRUB 2	43235 H/s
Half MD5	15255.8 MH/s
hMailServer	2509.6 MH/s
IKE-PSK MD5	1834.0 MH/s
IKE-PSK SHA1	788.2 MH/s
IPB2+, MyBB1.2+	5011.8 MH/s
IPMI2 RAKP HMAC-SHA1	1607.3 MH/s
iTunes backup < 10.0	140.2 kH/s
iTunes backup >= 10.0	94 H/s
JKS Java Key Store Private Keys (SHA1)	7989.4 MH/s
Joomla < 2.5.18	25072.2 MH/s
Juniper IVE	9929.1 kH/s
Juniper/NetBSD sha1crypt	144.1 kH/s
Juniper Netscreen/SSG (ScreenOS)	12946.8 MH/s
Keepass 1 (AES/Twofish) and Keepass 2 (AES)	139.8 kH/s

Kerberos 5 AS-REQ Pre-Auth etype 23	291.5 MH/s
Kerberos 5 TGS-REP etype 23	291.1 MH/s
LM	18382.7 MH/s
Lastpass	2331.2 kB/s
Lotus Notes/Domino 5	205.2 MH/s
Lotus Notes/Domino 6	69673.5 kB/s
Lotus Notes/Domino 8	667.2 kB/s
LUKS	8703 H/s
MD4	43722.9 MH/s
MD5	24943.1 MH/s
md5(md5(\$pass).md5(\$salt))	4291.9 MH/s
md5(\$salt.md5(\$salt.\$pass))	5037.7 MH/s
md5(\$salt.md5(\$pass.\$salt))	5401.6 MH/s
md5apr1, MD5(APR), Apache MD5	9911.5 kB/s
md5crypt, MD5(Unix), FreeBSD MD5, Cisco-IOS MD5	9918.1 kB/s
MS Office <= 2003 MD5+RC4,collision-mode #1	339.9 MH/s
MS Office <= 2003 MD5+RC4,oldoffice\$0, oldoffice\$l	219.6 MH/s
MS Office <= 2003 SHA1+RC4,collision-mode #1	330.8 MH/s
MS Office <= 2003 SHA1+RC4,oldoffice\$3, oldoffice\$4	296.7 MH/s
MS-AzureSync PBKDF2-HMAC-SHA256	10087.9 kB/s
MSSQL(2000)	8609.7 MH/s
MSSQL(2005)	8636.4 MH/s
MSSQL(2012)	1071.3 MH/s
Mediawiki B type	6515.8 MH/s
MySQL Challenge-Response Authentication (SHA1)	2288.0 MH/s
MySQL323	51387.0 MH/s
MySQL4.1/MySQL5	3831.5 MH/s
NTLM	41825.0 MH/s
NetNTLMv1-VANILLA / NetNTLMv1+ESS	22308.5 MH/s
NetNTLMv2	1634.9 MH/s
osCommerce, xt	12883.7 MH/s
OSX V10.4, V10.5, V10.6	6831.3 MH/s
OSX V10.7	834.1 MH/s
OSX V10.8+	12348 H/s
Office 2007	134.5 kB/s
Office 2010	66683 H/s

Office 2013	8814 H/s
OpenCart	2097.0 MH/s
Oracle H	851.6 MH/s
Oracle S	8565.0 MH/s
Oracle T	104.7 kH/s
Password Safe v2	332.0 kH/s
Password Safe v3	1233.4 kH/s
PBKDF2-HMAC-MD5	7408.3 kH/s
PBKDF2-HMAC-SHA1	3233.9 kH/s
PBKDF2-HMAC-SHA256	1173.1 kH/s
PBKDF2-HMAC-SHA512	431.4 kH/s
PDF 1.1 - 1.3 (Acrobat 2 - 4)	345.0 MH/s
PDF 1.1 - 1.3 (Acrobat 2 - 4) + collider-mode #1	373.4 MH/s
PDF 1.4 - 1.6 (Acrobat 5 - 8)	16048.0 kH/s
PDF 1.7 Level 3 (Acrobat 9)	2854.1 MH/s
PDF 1.7 Level 8 (Acrobat 10 - 11)	30974 H/s
PeopleSoft	8620.3 MH/s
PeopleSoft PS_TOKEN	3226.5 MH/s
phpass, MD5(Wordpress), MD5/phpBB3), MD5(Joomla)	6917.9 kH/s
PHPS	6972.6 MH/s
Plaintext	37615.5 MH/s
PostgreSQL	25068.0 MH/s
PostgreSQL Challenge-Response Auth (MD5)	6703.0 MH/s
PrestaShop	8221.3 MH/s
PunBB	2837.7 MH/s
RACF	2528.4 MH/s
RAR3-hp	29812 H/s
RAR5	36473 H/s
Radmin2	8408.3 MH/s
Redmine Project Management Web App	2121.3 MH/s
RipeMD160	4732.0 MH/s
SAP CODVN B (BCODE)	1311.2 MH/s
SAP CODVN F/G (PASSCODE)	739.3 MH/s
SAP CODVN H (PWDSALTEDHASH) iSSHA-1	6096.6 kH/s
scrypt	435.1 kH/s
SHA-1(Base64), nsldap, Netscape LDAP SHA	8540.0 MH/s

SHA-3(Keccak)	769.8 MH/s
SHA1	8538.1 MH/s
SHA1(CX)	291.8 MH/s
sha1(\$salt.sha1(\$pass))	2457.6 MH/s
SHA-224	3076.6 MH/s
SHA256	2865.2 MH/s
sha256crypt, SHA256(Unix)	388.8 kH/s
SHA384	1044.8 MH/s
SHA512	1071.1 MH/s
sha512crypt, SHA512(Unix)	147.5 kH/s
SIP digest authentication (MD5)	2004.3 MH/s
SKIP32	4940.9 MH/s
SMF > v1.1	6817.7 MH/s
SSHA-1(Base64), nsldaps, Netscape LDAP SSHA	8584.5 MH/s
SSHA-256(Base64), LDAP {SSHA256}	3216.9 MH/s
SSHA-512(Base64), LDAP	1072.2 MH/s
SipHash	28675.1 MH/s
Skype	12981.9 MH/s
Sybase ASE	398.1 MH/s
TrueCrypt PBKDF2-HMAC-RipeMD160+XTS512bit+boot-mode	512.4 kH/s
TrueCrypt PBKDF2-HMAC-RipeMD160+XTS512 bit	277.0 kH/s
TrueCrypt PBKDF2-HMAC-SHA512+XTS512 bit	376.2 kH/s
TrueCrypt PBKDF2-HMAC-Whirlpool+XTS512 bit	36505 H/s
vBulletin < V3.8.5	6947.7 MH/s
vBulletin > V3.8.5	4660.5 MH/s
VeraCrypt PBKDF2-HMAC-RipeMD160+XTS 512bit	907 H/s
VeraCrypt PBKDF2-HMAC-RipeMD160+XTS 512bit+boot-mode	1820 H/s
VeraCrypt PBKDF2-HMAC-SHA256+XTS 512bit	1226 H/s
VeraCrypt PBKDF2-HMAC-SHA256+XTS 512bit+boot-mode	3012 H/s
VeraCrypt PBKDF2-HMAC-SHA512+XTS 512bit	830 H/s
VeraCrypt PBKDF2-HMAC-Whirlpool+XTS 512bit	74 H/s
WBB3, Woltlab Burning Board 3	1293.3 MH/s
WPA/WPA2	396.8 kH/s

Whirlpool

253.9 MH/s

WinZip

1054.4 kH/s

*\*CRACKING SPEED BASED ON NVIDIA GTX 1080 & HASHCAT v3.6*

## HASH CRACKING SPEED (SLOW - FAST)

Ethereum Wallet, SCRYPT	29 H/s
VeraCrypt PBKDF2-HMAC-Whirlpool+XTS 512bit	74 H/s
iTunes backup >= 10.0	94 H/s
VeraCrypt PBKDF2-HMAC-SHA512+XTS 512bit	830 H/s
VeraCrypt PBKDF2-HMAC-RipeMD160+XTS 512bit	907 H/s
VeraCrypt PBKDF2-HMAC-SHA256+XTS 512bit	1226 H/s
VeraCrypt PBKDF2-HMAC-RipeMD160+XTS 512bit+boot-mode	1820 H/s
VeraCrypt PBKDF2-HMAC-SHA256+XTS 512bit+boot- mode	3012 H/s
Bitcoin/Litecoin wallet.dat	4508 H/s
Ethereum Wallet, PBKDF2-HMAC-SHA256	4518 H/s
7-Zip	7514 H/s
LUKS	8703 H/s
Office 2013	8814 H/s
1Password, cloudkeychain	10713 H/s
OSX V10.8+	12348 H/s
bcrypt, Blowfish(OpenBSD)	13094 H/s
eCryptfs	13813 H/s
Cisco \$9\$	22465 H/s
RAR3-hp	29812 H/s
PDF 1.7 Level 8 (Acrobat 10 - 11)	30974 H/s
RAR5	36473 H/s
TrueCrypt PBKDF2-HMAC-Whirlpool+XTS512 bit	36505 H/s
GRUB 2	43235 H/s
Drupal7	56415 H/s
Django (PBKDF2-SHA256)	59428 H/s
Cisco \$8\$	59950 H/s
Office 2010	66683 H/s

DPAPI masterkey file v1 and v2	73901 H/s
Oracle T	104.7 kB/s
AxCrypt	113.9 kB/s
Office 2007	134.5 kB/s
Keepass 1 (AES/Twofish) and Keepass 2 (AES)	139.8 kB/s
iTunes backup < 10.0	140.2 kB/s
Juniper/NetBSD sha1crypt	144.1 kB/s
sha512crypt, SHA512(Unix)	147.5 kB/s
TrueCrypt PBKDF2-HMAC-RipeMD160+XTS512 bit	277.0 kB/s
Atlassian (PBKDF2-HMAC-SHA1)	283.6 kB/s
Android FDE (Samsung DEK)	291.8 kB/s
Blockchain, My Wallet, V2	305.2 kB/s
Domain Cached Credentials 2 (DCC2), MS Cache 2	317.5 kB/s
Password Safe v2	332.0 kB/s
TrueCrypt PBKDF2-HMAC-SHA512+XTS512 bit	376.2 kB/s
sha256crypt, SHA256(Unix)	388.8 kB/s
WPA/WPA2	396.8 kB/s
PBKDF2-HMAC-SHA512	431.4 kB/s
scrypt	435.1 kB/s
TrueCrypt PBKDF2-HMAC-RipeMD160+XTS 512bit+boot-mode	512.4 kB/s
Lotus Notes/Domino 8	667.2 kB/s
Android FDE <= 4.3	803.0 kB/s
WinZip	1054.4 kB/s
PBKDF2-HMAC-SHA256	1173.1 kB/s
Password Safe v3	1233.4 kB/s
BSDiCrypt, Extended DES	1552.5 kB/s
Lastpass	2331.2 kB/s
PBKDF2-HMAC-SHA1	3233.9 kB/s
1Password, agilekeychain	3319.2 kB/s
Android PIN	5419.4 kB/s
SAP CODVN H (PWDSALTEDHASH) iSSHA-1	6096.6 kB/s
AIX	6359.3 kB/s
phpass, MD5(Wordpress), MD5/phpBB3, MD5(Joomla)	6917.9 kB/s
PBKDF2-HMAC-MD5	7408.3 kB/s
md5apr1, MD5(APR), Apache MD5	9911.5 kB/s
md5crypt, MD5(Unix), FreeBSD MD5, Cisco-IOS MD5	9918.1 kB/s

Juniper IVE

9929.1 kH/s

AIX	9937.1 kH/s
MS-AzureSync PBKDF2-HMAC-SHA256	10087.9 kH/s
AIX	14937.2 kH/s
PDF 1.4 - 1.6 (Acrobat 5 - 8)	16048.0 kH/s
AIX	44926.1 kH/s
GOST R 34.11-2012 (Streebog) 512-bit	49979.4 kH/s
GOST R 34.11-2012 (Streebog) 256-bit	50018.8 kH/s
Blockchain, My Wallet	50052.3 kH/s
Lotus Notes/Domino 6	69673.5 kH/s
Lotus Notes/Domino 5	205.2 MH/s
GOST R 34.11-94	206.2 MH/s
MS Office <= 2003 MD5+RC4,oldoffice\$0, oldoffice\$\l	219.6 MH/s
Whirlpool	253.9 MH/s
Kerberos 5 TGS-REP etype 23	291.1 MH/s
Kerberos 5 AS-REQ Pre-Auth etype 23	291.5 MH/s
SHA1(CX)	291.8 MH/s
MS Office <= 2003 SHA1+RC4,oldoffice\$3, oldoffice\$4	296.7 MH/s
MS Office <= 2003 SHA1+RC4,collision-mode #1	330.8 MH/s
MS Office <= 2003 MD5+RC4,collision-mode #1	339.9 MH/s
PDF 1.1 - 1.3 (Acrobat 2 - 4)	345.0 MH/s
PDF 1.1 - 1.3 (Acrobat 2 - 4) + collider-mode #1	373.4 MH/s
Sybase ASE	398.1 MH/s
FileZilla Server >= 0.9.55	565.2 MH/s
3DES (PT = \$salt, key = \$pass)	594.3 MH/s
SAP CODVN F/G (PASSCODE)	739.3 MH/s
SHA-3(Keccak)	769.8 MH/s
IKE-PSK SHA1	788.2 MH/s
OSX V10.7	834.1 MH/s
Oracle H	851.6 MH/s
decrypt, DES(Unix), Traditional DES	906.7 MH/s
SHA384	1044.8 MH/s
SHA512	1071.1 MH/s
MSSQL(2012)	1071.3 MH/s
SSHA-512(Base64), LDAP	1072.2 MH/s
WBB3, Woltlab Burning Board 3	1293.3 MH/s

SAP CODVN B (BCODE)	1311.2 MH/s
BLAKE2-512	1488.9 MH/s
IPMI2 RAKP HMAC-SHA1	1607.3 MH/s
NetNTLMv2	1634.9 MH/s
ColdFusion 10+	1733.6 MH/s
IKE-PSK MD5	1834.0 MH/s
SIP digest authentication (MD5)	2004.3 MH/s
OpenCart	2097.0 MH/s
Redmine Project Management Web App	2121.3 MH/s
MySQL Challenge-Response Authentication (SHA1)	2288.0 MH/s
sha1(\$salt.sha1(\$pass))	2457.6 MH/s
hMailServer	2509.6 MH/s
EPiServer 6.x > v4	2514.4 MH/s
RACF	2528.4 MH/s
PunBB	2837.7 MH/s
PDF 1.7 Level 3 (Acrobat 9)	2854.1 MH/s
Cisco-IOS SHA256	2864.3 MH/s
SHA256	2865.2 MH/s
SHA-224	3076.6 MH/s
SSHA-256(Base64), LDAP {SSHA256}	3216.9 MH/s
PeopleSoft PS_TOKEN	3226.5 MH/s
DNSSEC (NSEC3)	3274.6 MH/s
MySQL4.1/MySQL5	3831.5 MH/s
ChaCha20	3962.0 MH/s
md5(md5(\$pass).md5(\$salt))	4291.9 MH/s
vBulletin > V3.8.5	4660.5 MH/s
RipeMD160	4732.0 MH/s
SKIP32	4940.9 MH/s
IPB2+, MyBB1.2+	5011.8 MH/s
md5(\$salt.md5(\$salt.\$pass))	5037.7 MH/s
md5(\$salt.md5(\$pass.\$salt))	5401.6 MH/s
FortiGate (FortiOS)	6386.2 MH/s
Mediawiki B type	6515.8 MH/s
PostgreSQL Challenge-Response Authentication (MD5)	6703.0 MH/s
SMF > v1.1	6817.7 MH/s
EPiServer 6.x < v4	6818.5 MH/s

Django (SHA-1)	6822.6 MH/s
OSX V10.4, V10.5, V10.6	6831.3 MH/s
ArubaOS	6894.7 MH/s
vBulletin < V3.8.5	6947.7 MH/s
PHPS	6972.6 MH/s
Citrix NetScaler	7395.3 MH/s
AxCrypt in memory SHA1	7503.3 MH/s
JKS Java Key Store Private Keys (SHA1)	7989.4 MH/s
PrestaShop	8221.3 MH/s
Radmin2	8408.3 MH/s
SHA1	8538.1 MH/s
SHA-1(Base64), nsldap, Netscape LDAP SHA	8540.0 MH/s
SSHA-1(Base64), nsldaps, Netscape LDAP SSHA	8584.5 MH/s
MSSQL(2000)	8609.7 MH/s
PeopleSoft	8620.3 MH/s
MSSQL(2005)	8636.4 MH/s
Oracle S	8565.0 MH/s
Domain Cached Credentials (DCC), MS Cache	11195.8 MH/s
osCommerce, xt	12883.7 MH/s
Juniper Netscreen/SSG (ScreenOS)	12946.8 MH/s
Skype	12981.9 MH/s
Half MD5	15255.8 MH/s
Cisco-PIX MD5	16407.2 MH/s
Cisco-ASA MD5	17727.2 MH/s
LM	18382.7 MH/s
DES (PT = \$salt, key = \$pass)	19185.7 MH/s
NetNTLMv1-VANILLA / NetNTLMv1+ESS	22308.5 MH/s
MD5	24943.1 MH/s
PostgreSQL	25068.0 MH/s
Joomla < 2.5.18	25072.2 MH/s
SipHash	28675.1 MH/s
Plaintext	37615.5 MH/s
MD4	43722.9 MH/s
NTLM	41825.0 MH/s
MySQL323	51387.0 MH/s

*\*Speed based on NVIDIA GTX 1080 Running Hashcat v3.6*