



Wordlists and Rules

HACK THE BOX FRIDAY

Crunch

A very simple wordlist generator based on patterns

Syntax

```
crunch <minimum length> <maximum length> <charset> -t <pattern> -o  
      <output file>
```

Options

```
-t    Patterns  
    @ = lower case character  
    , = upper case character  
    % = numbers  
    ^ = symbols  
-o    Output
```

Example

```
crunch 12 12 -t 10031998@@@@ -d 1 -o crunch1.list
```



CUPP – Common User Password Profiler

```

cupp.py!
\
\
\      ' _ '
\      (oo)____
      ( _ )   )\
        ||--|| *

# Common
# User
# Passwords
# Profiler

[ Muris Kurgas | j0rgan@remote-exploit.org ]
[ Mebus | https://github.com/Mebus/ ]

[+] Insert the information about the victim to make a dictionary
[+] If you don't know all the info, just hit enter when asked! ;)

> First Name: Billy
> Surname: Bob
> Nickname: Bubba
> Birthdate (DDMMYYYY): 01011975

> Partners) name: Betty
> Partners) nickname: Boop
> Partners) birthdate (DDMMYYYY): 12121978

> Child's name: 
```

This is one of the more useful ones. You answer some questions, and it makes the list for you!

This typically isn't preinstalled on systems; it's a python script. We can clone it with this command:

```
git clone https://github.com/Mebus/cupp.git
  && cd cup
```

Usage

```
python3 cup.py -i
```

Kwprocessor - From the good folks at hashcat

Options	Short / Long	Type	Description
-V, --version			Print version
-h, --help			Print help
-o, --output-file		FILE	Output-file
-b, --keyboard-basic		BOOL	Include characters reachable without holding s
-s, --keyboard-shift		BOOL	Include characters reachable by holding shift
-a, --keyboard-altgr		BOOL	Include characters reachable by holding altgr
-z, --keyboard-all			Shortcut to enable all --keyboard-* modifier
-1, --keywalk-south-west		BOOL	Include routes heading diagonale south-west
-2, --keywalk-south		BOOL	Include routes heading straight south
-3, --keywalk-south-east		BOOL	Include routes heading diagonale south-east
-4, --keywalk-west		BOOL	Include routes heading straight west
-5, --keywalk-repeat		BOOL	Include routes repeating character
-6, --keywalk-east		BOOL	Include routes heading straight east
-7, --keywalk-north-west		BOOL	Include routes heading diagonale north-west
-8, --keywalk-north		BOOL	Include routes heading straight north
-9, --keywalk-north-east		BOOL	Include routes heading diagonale north-east
-c, --keywalk-cont			Shortcut to enable adjacent keys (continuous w
-0, --keywalk-all			Shortcut to enable all --keywalk-* directions
-n, --keywalk-distance-min		NUM	Minimum allowed distance between keys
-x, --keywalk-distance-max		NUM	Maximum allowed distance between keys



Least useful wordlist creator in my humble opinion. Creates keyboard walks using direction routes (Compass directions).



Not typically installed but can be cloned:


```
git clone
https://github.com/
hashcat/kwprocessor
.git && cd
kwprocessor && make
```



Usage: `kwp -s 1 <basechars> <keymap> <route>`

Kwprocessor - Routes

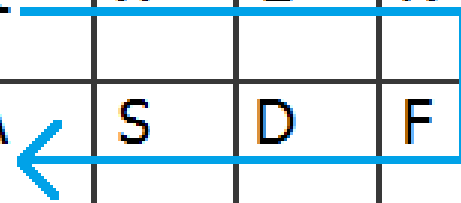
	#	\$	%	^	&	*	(
	3	4	5	6	7	8	9
V	E	R	T	Y	U	I	O
S	D	F	G	H	J	K	L
(C	V	B	N	M	<	>
						,	.



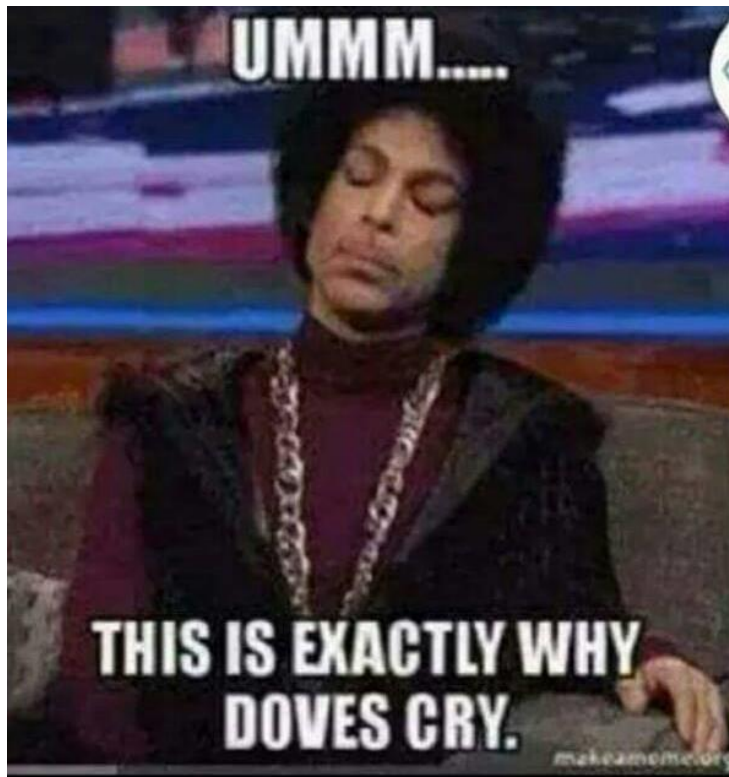
A route is a series of numbers representing the compass direction of the keyboard walk.

For instance, 313 represents go 3 spaces, then change direction and go 1 space, then change direction again and go 3 more spaces

!	@	#	\$	%	^	&
1	2	3	4	5	6	7
Q	W	E	R	T	Y	U
A	S	D	F	G	H	J
Z	X	C	V	B	N	M



Princeprocessor



Decent wordlist generator which takes in a wordlist and creates chains of words to create a new wordlist.

Installation

```
wget https://github.com/hashcat/princeprocessor/releases/download/v0.22/princeprocessor-0.22.7z && 7z x princeprocessor-0.22.7z && cd princeprocessor-0.22
```

Options

```
--pw-min=<integer>  Min length of password  
--pw-max=<integer>  Max length of password  
--keyspace           Check the keyspace
```

Usage

```
./pp64.bin -o <new wordlist> < <original wordlist>  
To check keyspace (to make sure you're not about to make a 15PB file)  
./pp64.bin -keyspace < <original wordlist>
```


Password Cracking with Rules



Hashcat's rules allow users to define specific modifications to a wordlist.



Modifications can include appending numbers, reversing words, or changing letter cases.



These rules significantly enhance the effectiveness of password “recovery” efforts.



Hashcat comes with a variety of built-in rules.



Rules enable sophisticated attacks that adapt to common password patterns and permutations.

Common Rules

Function	Description	Input	Output
l	Convert all letters to lowercase	InlaneFreight2020	inlanefreight2020
u	Convert all letters to uppercase	InlaneFreight2020	INLANEFREIGHT2020
c / C	capitalize / lowercase first letter and invert the rest	inlaneFreight2020 / Inlanefreight2020	Inlanefreight2020 / iNLANEFREIGHT2020
t / TN	Toggle case : whole word / at position N	InlaneFreight2020	iNLANEfREIGHT2020
d / q / zN / ZN	Duplicate word / all characters / first character / last character	InlaneFreight2020	InlaneFreight2020InlaneFreight2020 / llnnllaanneeFFreeeiigghtt22002200 / lInlaneFreight2020 / InlaneFreight20200
{ / }	Rotate word left / right	InlaneFreight2020	nlaneFreight2020l / 0InlaneFreight202
^X / \$X	Prepend / Append character X	InlaneFreight2020 (^! / \$!)	!InlaneFreight2020 / InlaneFreight2020!
r	Reverse	InlaneFreight2020	0202thgierFenaInl



There are a ton of rules available. These are some common ones that HTB lists.

A full list is available at https://hashcat.net/wiki/doku.php?id=rule_based_attack#implemented_compatible_functions

Rule example

c so0 si1 se3 ss5 sa@ \$2 \$0 \$1 \$9

Explanation

- c First letter is capitalized
- so0 s is the substitute function, it will replace `o` with `0`
- si1 Using the substitute function again, now it replaces `i` with `1`
- ss5 You got it, substitute again! Now `s` becomes `5`
- sa@ Once again, substituting. This time, `a` becomes `@`
- \$2 Appends `2` to the end
- \$0 Appends `0` after `2`
- \$1 Appends `1` after `0`
- \$9 Appends `9` after `1`
 - Gives us `2019` appended to the end

Great, how the heck do we use it now?

First, we'll need to create a file for the rule

```
echo 'c so0 si1 se3 ss5 sa@ /$2 /$0 /$1 /$9' > rule.txt
```

Then we can tell hashcat to use the rule

- Syntax

```
hashcat -a <attack mode> -m <hash type> <hash file> <wordlist> -r <rule file>
```

- Example

```
hashcat -a 0 -m 100 hash rockyou.txt -r rule.txt
```

- Troubleshooting (Debugging)

```
hashcat -r <rule file> <test wordlist> --stdout
```

Built-in Rules

MMMMMMMMMMMMMM



**GYROS WITH
BUILT IN FRENCH FRIES**

Fortunately, you don't have to make your own rules. In fact, you only want to do that in very specific situations where you need to customize your wordlist.

Hashcat has a ton of rules already created for you. They're located in `/usr/share/hashcat/rules/`

Hashcat also has the ability to generate random rules by using the `-g` flag.

In the following example, hashcat will generate 1000 random rules.

```
hashcat -a 0 -m 100 -g 1000 hash rockyou.txt
```

The HTB Challenge



Create the hash file

```
echo "46244749d1e8fb99c37ad4f14fccb601ed4ae283" > htb.hash
```

Identify the hash (they tell us it's SHA1, but it's a good practice to always identify the hash yourself)

```
hashid htb.hash -m
```

Create the rule

HTB tell you to append 2020 to the end of each attempt, but they're jerks. There are more rules that we'll need. We'll want to capitalize the first letter, change o to 0, l to 1, e to 3, s to 5, and a to @ in addition to appending 2020 to the end

```
echo 'c so0 si1 se3 ss5 sa@ /$2 /$0 /$2 /$0' > rule.txt
```

Now we can crack the hash

```
hashcat -a 0 -m 100 htb.hash  
/usr/share/wordlists/seclists/Passwords/Leaked-  
Databases/rockyou.txt.tar.gz -r rule.txt
```