iGNITE Technologies

A Detailed Guide on CRUNCH

WWW.HACKINGARTICLES.IN

Contents

Introduction

Often times attackers have the need to generate a wordlist based on certain criteria which is required for pentest scenarios like password spraying/brute-forcing. Other times it could be a trivial situation like directory enumeration. Crunch is a tool developed in C by **bofh28** that can create custom, highly modifiable wordlists that may aid an attacker in the situations mentioned above. It takes in min size, max size and alphanumeric character sets as input and generates any possible combination of words with or without meaning and writes it out in a text file. In this article, we'll demonstrate crunch filters in detail.

Installation and first run

Crunch is installed by default on Kali Linux but can be installed using apt package manager using

```
apt install crunch
```

After it is installed, we can run crunch to generate wordlist. When we input the min and max size of the word to be generated and just the output file, it automatically takes in small case alphabets as character sets and generates words.

For example, here 1 character to 3 characters per word is being generated in smallcase and stored in file dict.txt

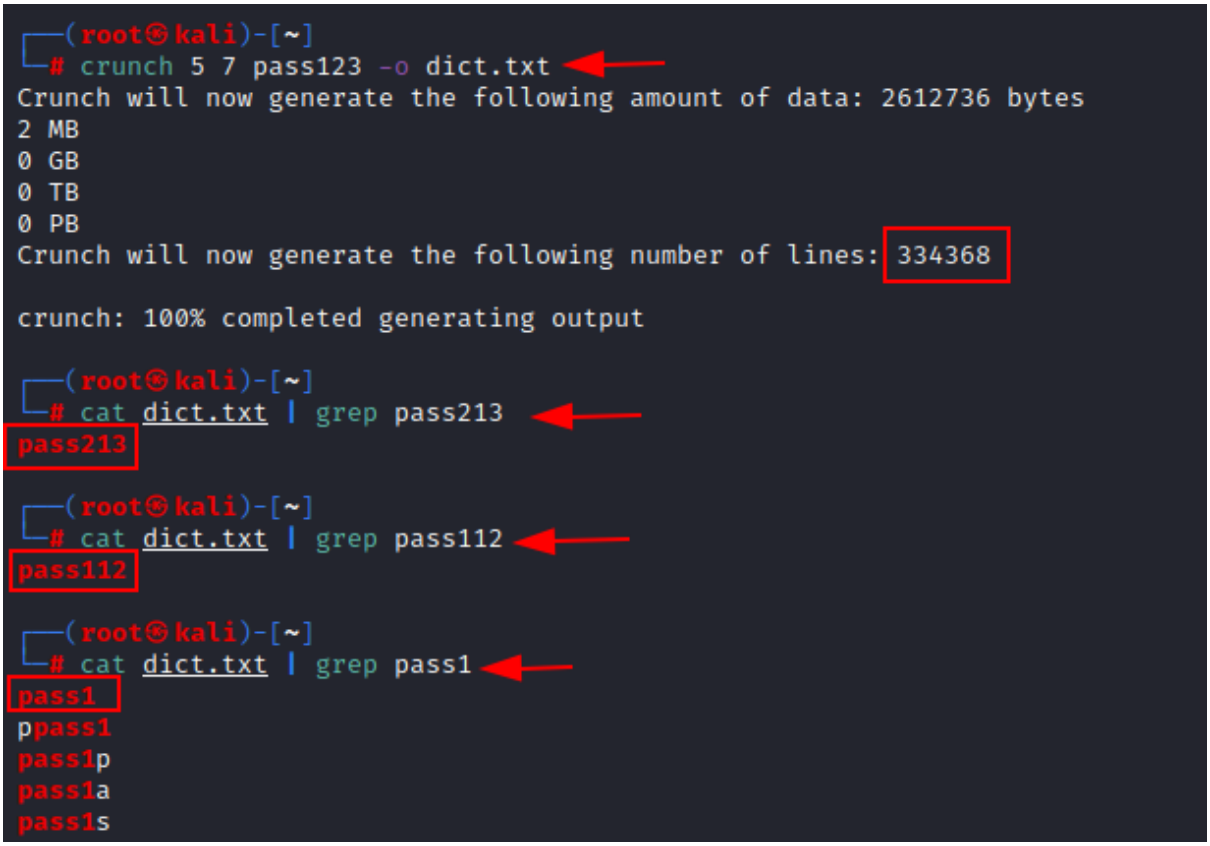
```
crunch 1 3 -o dict.txt
```

```
┌──(root㉿kali)-[~/crunch]
└─# crunch 1 3 -o dict.txt
Crunch will now generate the following amount of data: 72384 bytes
0 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 18278

crunch: 100% completed generating output

┌──(root㉿kali)-[~/crunch]
└─# ls
dict.txt

┌──(root㉿kali)-[~/crunch]
└─# head -n 10 dict.txt
a
b
c
d
e
f
g
h
i
j

┌──(root㉿kali)-[~/crunch]
└─# tail dict.txt
zzq
zzr
zzs
zzt
zzu
zzv
zzw
zzx
zzy
zzz
```

Defined Alphanumeric Characters

A user can also define the selected characters to be used while generating a wordlist. Here, min size 5 and max size 7 characters per words is being generated while using the characters “p, a, s, s, 1, 2, and 3” as input. Hence the dictionary starts with “ppppp, ppppa ….” And ends with “3333333” and contains combinations like pass213, pass1 etc.

crunch 5 7 pass123 -o dict.txt



Space character wordlist

One neat trick is to include space in the wordlist. Often times we need spaces in scenarios for passwords and many generic wordlists or tools don't have this feature. In crunch, we can define space as a character by putting space after the characterset to be used. For 1 to 3 characters per word including space we can do this:

```
crunch 1 3 "raj " -o space.txt
```

```
┌──(root㉿kali)-[~]
└─# crunch 1 3 "raj " -o space.txt
Crunch will now generate the following amount of data: 312 bytes
0 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 84

crunch: 100% completed generating output

┌──(root㉿kali)-[~]
└─# head -n 30 space.txt
r
a
j

rr
ra
rj
r
ar
aa
aj
a
jr
ja
jj
j
 r
 a
 j

rrr
rra
rrj
rr
rar
raa
raj
ra
rjr
rja
```

View character sets available

In the **/usr/share/crunch** directory, one may find a list file (charset.lst) mentioning all the different character sets supported by crunch. This is highly useful as a ready reference. One may manually specify character sets or can even use the codenames written on the left. It is quite simple to understand though. Description of each charset is given below:

Description:	charset code-name
Lowercase alphabets:	lalpha
Uppercase alphabets:	ualpha
Mixture of lowercase and uppercase alphabets:	mixalpha
Numbers:	numeric
Top 14 symbols:	symbol14
All symbols:	all
Then, we can make combinations of character sets too. It can be understood like the following:	
To include space with charset:	charset-space. For example, lalpha-space will have {abcdefghijklmnopqrstuvwxyz } (note space at the end)
To include numeric with charset:	charset-numeric
Combination of charset plus numeric plus space:	charset-numeric-space
To include top 14 symbols:	charset-symbol14
To include all symbols:	charset-all
To include all symbols and space:	charset-all-space

To view the charset file:

cat /usr/share/crunch/charset.lst

```
┌──(root㉿kali)-[/usr/share/crunch]
└─# cat charset.lst
# charset configuration file for winrtgen v1.2 by Massimiliano Montoro (mao@oxid.it)
# compatible with rainbowcrack 1.1 and later by Zhu Shuanglei <shuanglei@hotmail.com>


hex-lower                      = [0123456789abcdef]
hex-upper                      = [0123456789ABCDEF]

numeric                        = [0123456789]
numeric-space                  = [0123456789 ]

symbols14                      = [!@#$%^&*()-_+=]
symbols14-space                = [!@#$%^&*()-_+= ]

symbols-all                    = [!@#$%^&*()-_+=~`[]{}|\:;"'<>,.?/]
symbols-all-space              = [!@#$%^&*()-_+=~`[]{}|\:;"'<>,.?/ ]

ualpha                         = [ABCDEFGHIJKLMNOPQRSTUVWXYZ]
ualpha-space                   = [ABCDEFGHIJKLMNOPQRSTUVWXYZ ]
ualpha-numeric                 = [ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789]
ualpha-numeric-space           = [ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 ]
ualpha-numeric-symbol14        = [ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#$%^&*()-_+=]
ualpha-numeric-symbol14-space  = [ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#$%^&*()-_+= ]
ualpha-numeric-all             = [ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#$%^&*()-_+=~`[]{}|\:;"'<>,.?/]
ualpha-numeric-all-space       = [ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#$%^&*()-_+=~`[]{}|\:;"'<>,.?/ ]

lalpha                         = [abcdefghijklmnopqrstuvwxyz]
lalpha-space                   = [abcdefghijklmnopqrstuvwxyz ]
lalpha-numeric                 = [abcdefghijklmnopqrstuvwxyz0123456789]
lalpha-numeric-space           = [abcdefghijklmnopqrstuvwxyz0123456789 ]
lalpha-numeric-symbol14        = [abcdefghijklmnopqrstuvwxyz0123456789!@#$%^&*()-_+=]
lalpha-numeric-symbol14-space  = [abcdefghijklmnopqrstuvwxyz0123456789!@#$%^&*()-_+= ]
lalpha-numeric-all             = [abcdefghijklmnopqrstuvwxyz0123456789!@#$%^&*()-_+=~`[]{}|\:;"'<>,.?/]
lalpha-numeric-all-space       = [abcdefghijklmnopqrstuvwxyz0123456789!@#$%^&*()-_+=~`[]{}|\:;"'<>,.?/ ]
```

Using codename character sets

These codenames can be used while creating dictionary files. For example, to create a wordlist of 4 characters per word using mixture of alphabets, numerics and special characters, one can specify the charset.lst file using the "-f" option and then specify code word "mixalpha-numeric-all"

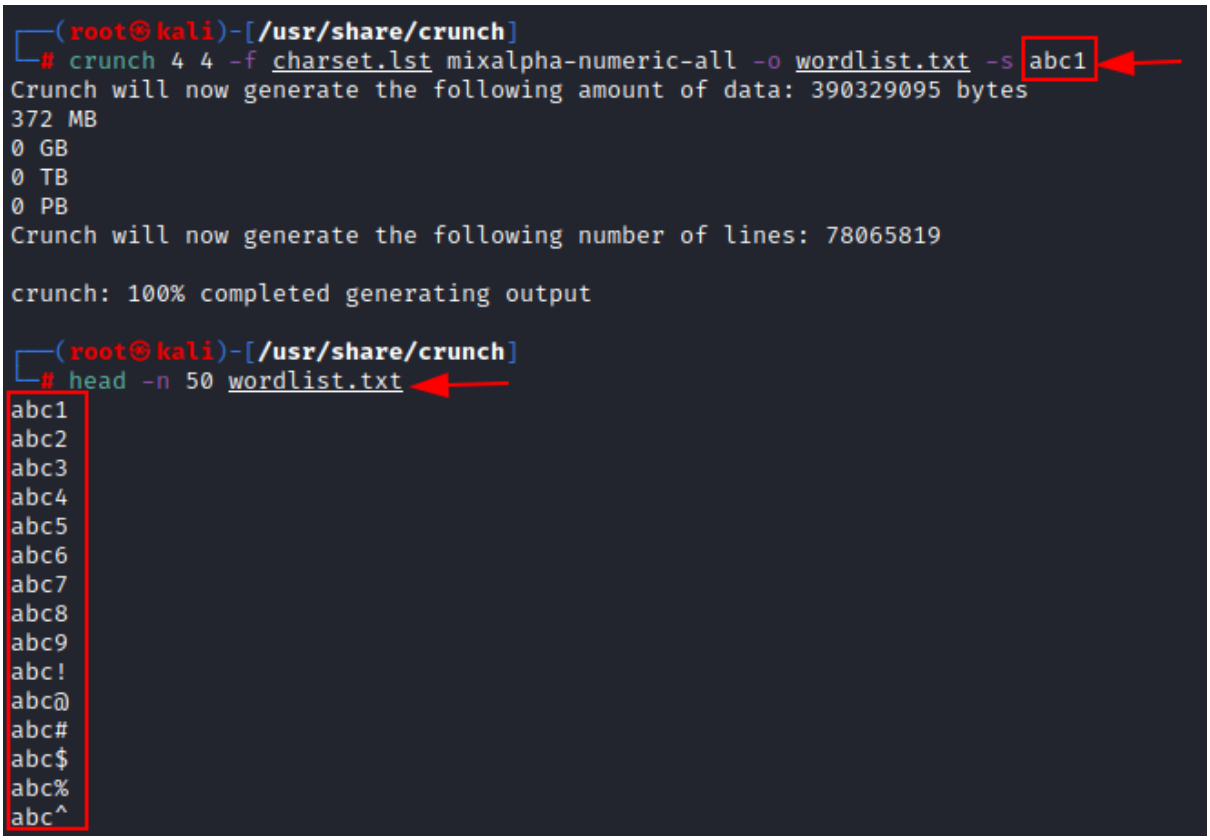
crunch 4 4 -f charset.lst mixalpha-numeric-all -o wordlist.txt

```
┌──(root㉿kali)-[/usr/share/crunch]
└─#  crunch 4 4 -f charset.lst mixalpha-numeric-all -o wordlist.txt
Crunch will now generate the following amount of data: 390374480 bytes
372 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 78074896

crunch: 100% completed generating output

┌──(root㉿kali)-[/usr/share/crunch]
└─# head -n 50 wordlist.txt
aaaa
aaab
aaac
aaad
aaae
aaaf
aaag
aaah
aaai
aaaj
aaak
aaal
aaam
```

Startblock in wordlists

A startblock can be defined using "-s" filter. By using this, we can define from where a wordlist should start generating. This is helpful in discarding unwanted combinations. For example, to start a wordlist from abc1, and having 4 characters per word including alphanumeric and special characters can be created like below. This way, dictionary starts with "abc1, abc2,..abd1, abd2…" and ends at "////"

crunch 4 4 -f charset.lst mixalpha-numeric-all -o wordlist.txt -s abc1



Creating Dictionary with various patterns

Please note that the following symbols when defined as input in character sets mean the following:

@ will insert lower case characters

, will insert upper case characters

% will insert numbers

^ will insert symbols

Now, if a user wants to create a word with 3 characters with first character lowercase, number as second character and symbol as third, he can specify this:

crunch -t @%^ -o dict.txt

With "-t" as the flag to provide the symbols. If you aren't going to use a particular character set you use a plus sign as a placeholder.

+ operator positioning: The + operator can be used where no specific character sets are used and any value can be replaced for the same. But this is in the following order:

Lowercase alphabets, uppercase alphabets, numbers, symbols

For example,

crunch 3 3 + + 123 +

This would take in the following input:

Lowercase: abcdefghijklmnopqrstuvwxyz

Uppercase: ABCDEFGHIJKLMNOPQRSTUVWXYZ

Numbers: 123

Symbols: !@#$%^&*()-_+=~`[]{}|\:;"'<>,.?/

Case 1: Fixed word + 3 numbers

Lets say if we want to fix first 3 letters as “raj” and insert random combinations of digits at the last 3 places in a 6 character per word wordlist, it can be done by specifying the pattern without the use of commas like above in “-t” filter.

crunch 6 6 -t raj%%% -o num.txt

```
┌──(root㉿kali)-[~/crunch]
└─# crunch 6 6 -t raj%%% -o num.txt

Crunch will now generate the following amount of data: 7000 bytes
0 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 1000

crunch: 100% completed generating output

┌──(root㉿kali)-[~/crunch]
└─# head num.txt
raj000
raj001
raj002
raj003
raj004
raj005
raj006
raj007
raj008
raj009
```

Case 2: Fixed word + 3 uppercase alphabets

Let's say if we want to fix first 3 letters as "raj" and insert random combinations of uppercase alphabets at the last 3 places in a 6 character per word wordlist, it can be done by

crunch 6 6 -t raj,,, -o upper.txt

```
┌──(root㉿kali)-[~/crunch]
└─# crunch 6 6 -t raj,,, -o upper.txt

Crunch will now generate the following amount of data: 123032 bytes
0 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 17576

crunch: 100% completed generating output

┌──(root㉿kali)-[~/crunch]
└─# head upper.txt
rajAAA
rajAAB
rajAAC
rajAAD
rajAAE
rajAAF
rajAAG
rajAAH
rajAAI
rajAAJ
```

Case 3: Fixed word + 3 lowercase alphabets

Let's say if we want to fix first 3 letters as "raj" and insert random combinations of smallcase alphabets at the last 3 places in a 6 character per word wordlist, it can be done by

crunch 6 6 -t raj@@@ -o lower.txt

```
┌──(root㉿kali)-[~/crunch]
└─# crunch 6 6 -t raj@@@ -o lower.txt

Crunch will now generate the following amount of data: 123032 bytes
0 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 17576

crunch: 100% completed generating output

┌──(root㉿kali)-[~/crunch]
└─# head lower.txt
rajaaa
rajaab
rajaac
rajaad
rajaae
rajaaf
rajaag
rajaah
rajaai
rajaaj
```

Case 4: Fixed word + 3 symbols

Let's say if we want to fix first 3 letters as "raj" and insert random combinations of special characters at the last 3 places in a 6 character per word wordlist, it can be done by

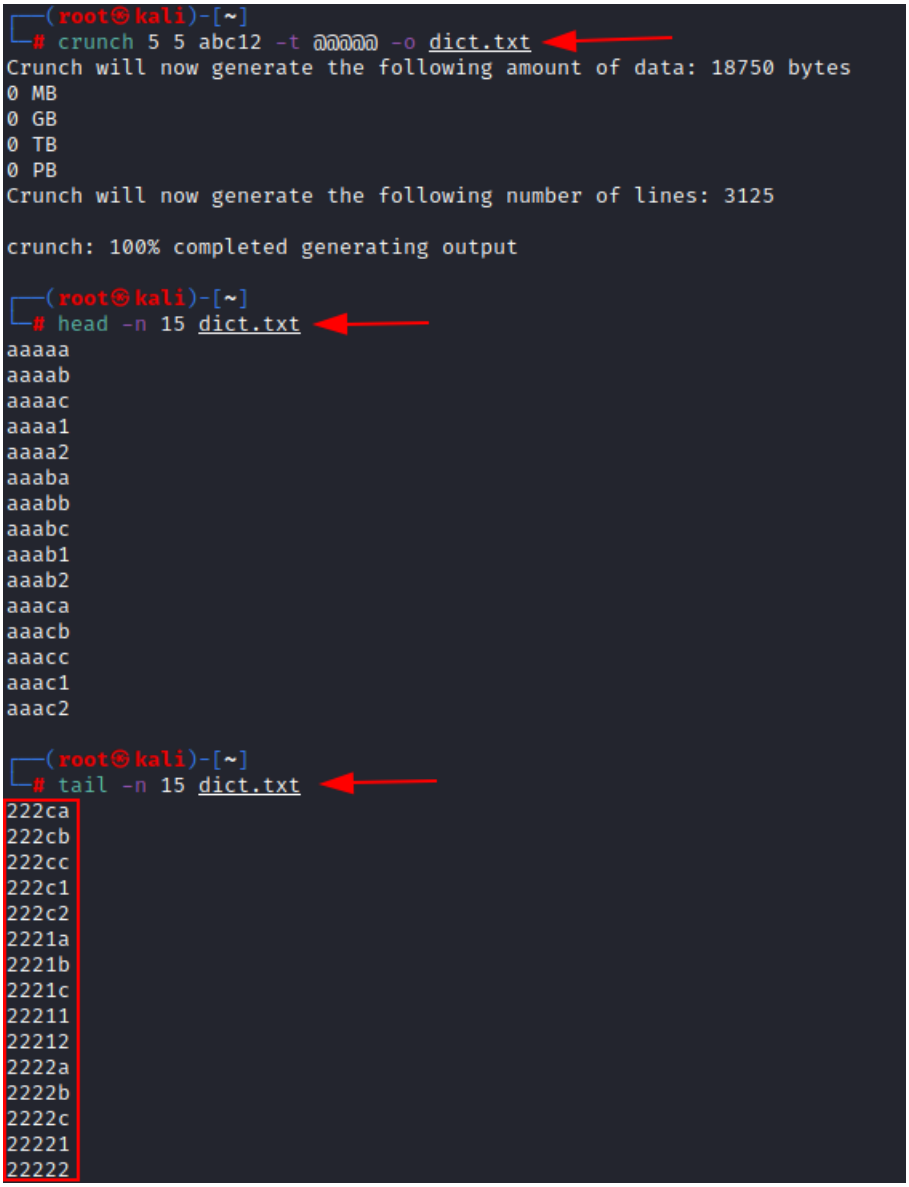
crunch 6 6 -t raj^^^ -o symbol.txt

```
┌──(root㉿kali)-[~/crunch]
└─# crunch 6 6 -t raj^^^ -o symbol.txt

Crunch will now generate the following amount of data: 251559 bytes
0 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 35937

crunch: 100% completed generating output

┌──(root㉿kali)-[~/crunch]
└─# head symbol.txt
raj!!!
raj!!@
raj!!#
raj!!$
raj!!%
raj!!^
raj!!&
raj!!*
raj!!(
raj!!)
```

Case 5: Placeholder fixed pattern

Let's say in place of the lowercase placeholder we input abc12 and with "-t" we supply in @ then the pattern shall also contain 1 and 2 even though we just gave "@" indicator. See the following example:

crunch 5 5 abc12 -t @@@@@ -o dict.txt



Case 6: Lowercase alphabet (a,b or c) + number (1,2 or 3) + symbol (ANY)

Now, a user can also provide character set from which a pattern is to be created. In the following example, abc and 123 have been used. A "+" operator is also used indicating that the pattern indicator for which charset is not supplied, shall be treated as "ANY" value.

So, if a user wants to create a dictionary with first character lowercase, number as second character and symbol as third but only "a,b or c" as characters, "1,2 or 3" as numbers and any random symbol on last position respectively, he can do the following:

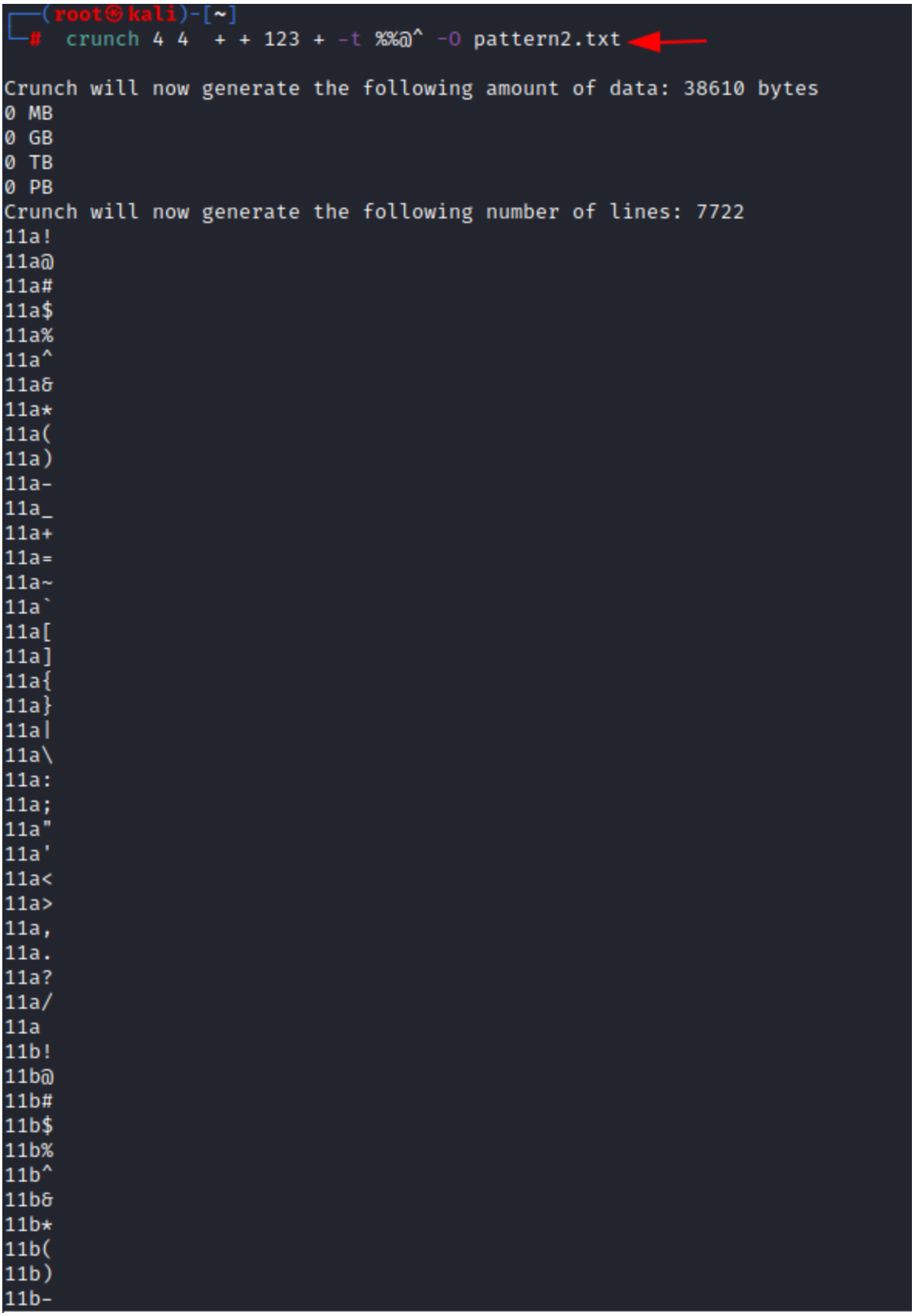
```
crunch 3 3 abc + 123 -t @%^ -o pattern.txt
```

```
┌──(root㉿kali)-[/usr/share/crunch]
└─# crunch 3 3 abc + 123 -t @%^ -o pattern.txt

Crunch will now generate the following amount of data: 1188 bytes
0 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 297

crunch: 100% completed generating output

┌──(root㉿kali)-[/usr/share/crunch]
└─# head -n 30 pattern.txt
a1!
a1@
a1#
a1$
a1%
a1^
a1&
a1*
a1(
a1)
a1-
a1_
a1+
a1=
a1~
a1`
a1[
a1]
a1{
a1}
a1|
a1\
a1:
a1;
a1"
a1'
a1<
a1>
a1,
a1.
```

Case 7: Two number (1,2 or 3) + lowercase alphabet (ANY) + symbol (ANY)

Similarly, to create a 4 character per word pattern of 2 digits (containing only 1,2, or 3)+lowercase alpha+symbol we can do this:

crunch 4 4 + + 123 + -t %%@^ -O pattern2.txt


┌──(root㉿kali)-[~]
└─# crunch 4 4 + + 123 + -t %%@^ -O pattern2.txt
Crunch will now generate the following amount of data: 38610 bytes
0 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 7722
11a!
11a@
11a#
11a$
11a%
11a^
11a&
11a*
11a(
11a)
11a-
11a_
11a+
11a=
11a~
11a`
11a[
11a]
11a{
11a}
11a|
11a\
11a:
11a;
11a"
11a'
11a<
11a>
11a,
11a.
11a?
11a/
11a
11b!
11b@
11b#
11b$
11b%
11b^
11b&
11b*
11b(
11b)
11b-

Case 8: Treating symbols as literals

When “-l” is used in accordance with the “-t” filter, it tells crunch which symbols should be treated as literals. For example, we know that @ is used to denote a lowercase letter. So, if we want to generate a 7 character per word wordlist using the word “p@ss” fixed, it will consider @ as a pattern indicator of a lowercase alphabets. Thereafter, -l filter can be used to define which character is to be treated as literal and not converted as pattern. This can be done like:

```
crunch 7 7 -t p@ss,%^ > dict.txt
crunch 7 7 -t p@ss,%^ -l a@aaaaa > 1.txt
```

```
┌──(root㉿kali)-[~/crunch]
└─# crunch 7 7 -t p@ss,%^ > dict.txt

Crunch will now generate the following amount of data: 1784640 bytes
1 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 223080

┌──(root㉿kali)-[~/crunch]
└─# head -n 15 dict.txt
passA0!
passA0@
passA0#
passA0$
passA0%
passA0^
passA0&
passA0*
passA0(
passA0)
passA0-
passA0_
passA0+
passA0=
passA0~

┌──(root㉿kali)-[~/crunch]
└─# crunch 7 7 -t p@ss,%^ -l a@aaaaa > 1.txt

Crunch will now generate the following amount of data: 68640 bytes
0 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 8580

┌──(root㉿kali)-[~/crunch]
└─# head -n 15 1.txt
p@ssA0!
p@ssA0@
p@ssA0#
p@ssA0$
p@ssA0%
p@ssA0^
p@ssA0&
p@ssA0*
p@ssA0(
p@ssA0)
p@ssA0-
p@ssA0_
p@ssA0+
p@ssA0=
p@ssA0~
```

Inverting Wordlist

A generated wordlist fixes, by default, first characters and creates combinations on the last character. For example, a wordlist containing “a,b and c” has

aaa
aab
aac
aba
abb
abc
aca
…

But this can be inverted using the “-i” option. Crunch would fix the last letter first and make combinations out of first letters. For example, a dictionary of 5 characters per word having 3 alphabets,2digits and inverted looks like following:

```
crunch 5 5 abc12 -t @@@%% -o dict.txt
crunch 5 5 abc12 -t @@@%% -i -o invert.txt
```

```
┌──(root㉿kali)-[~]
└─# crunch 5 5 abc12 -t @@@%% -o dict.txt
Crunch will now generate the following amount of data: 75000 bytes
0 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 12500

crunch: 100% completed generating output

┌──(root㉿kali)-[~]
└─# head -n 20 dict.txt
aaa00
aaa01
aaa02
aaa03
aaa04
aaa05
aaa06
aaa07
aaa08
aaa09
aaa10
aaa11
aaa12
aaa13
aaa14
aaa15
aaa16
aaa17
aaa18
aaa19

┌──(root㉿kali)-[~]
└─# crunch 5 5 abc12 -t @@@%% -i -o invert.txt
Crunch will now generate the following amount of data: 75000 bytes
0 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 12500

crunch: 100% completed generating output

┌──(root㉿kali)-[~]
└─# head -n 20 invert.txt
aaa00
baa00
caa00
1aa00
2aa00
aba00
bba00
cba00
```

Limit duplicate patterns

A user can place a limit on the number of characters that can occur together. For example, to create a wordlist of 5 characters per word using 3 lowercase alphabets,1 number and 1 symbol can be done like first command. But if a user wants to limit the occurrence of duplicate characters together to only 2 places he can use the “-d” operator. Note how in the first command 3 “a” occurred but in the second command duplicates are limited to only 2 and so only 2 “a”s have occurred.

```
crunch 5 5 abc + 123 -t @@@%^ -o 1.txt
crunch 5 5 abc + 123 -t @@@%^ -o 2.txt -d
```

```
┌──(root㉿kali)-[~/crunch]
└─# crunch 5 5 abc + 123 -t @@@%^ -o 1.txt
Crunch will now generate the following amount of data: 16038 bytes
0 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 2673

crunch: 100% completed generating output

┌──(root㉿kali)-[~/crunch]
└─# head 1.txt
aaa1!
aaa1@
aaa1#
aaa1$
aaa1%
aaa1^
aaa1&
aaa1*
aaa1(
aaa1)

┌──(root㉿kali)-[~/crunch]
└─# crunch 5 5 abc + 123 -t @@@%^ -o 2.txt -d 2@
Crunch will now generate the following amount of data: 14256 bytes
0 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 2376

crunch: 100% completed generating output

┌──(root㉿kali)-[~/crunch]
└─# head 2.txt
aab1!
aab1@
aab1#
aab1$
aab1%
aab1^
aab1&
aab1*
aab1(
aab1)
```

Putting early stops on wordlists

As per user requirements, there may also be a possibility when a user wants to cut short a list to certain combination. For example, if a user wants to create 3 characters per word wordlist

using “a,b and c” as characters but wants to cut it as soon as wordlist generates combination ”acc” it can be done like so:

```
crunch 3 3 abc -o 1.txt
crunch 3 3 abc -e acc -o 2.txt
```

```
┌──(root㉿kali)-[~/crunch]
└─# crunch 3 3 abc -o 1.txt
Crunch will now generate the following amount of data: 108 bytes
0 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 27

crunch: 100% completed generating output

┌──(root㉿kali)-[~/crunch]
└─# head -n 15 1.txt
aaa
aab
aac
aba
abb
abc
aca
acb
acc
baa
bab
bac
bba
bbb
bbc

┌──(root㉿kali)-[~/crunch]
└─# crunch 3 3 abc  -e acc -o 2.txt
Crunch will now generate the following amount of data: 36 bytes
0 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 9

crunch: 100% completed generating output

┌──(root㉿kali)-[~/crunch]
└─# cat 2.txt
aaa
aab
aac
aba
abb
abc
aca
acb
acc
```

Word permutations

In mathematics, permutations stand for non-repeating combinations of certain events. So, to generate non-repeating wordlists by permutations we can use the "-p" filter. Here, we supply 3 words as input none of which shall repeat even if the maximum size of the wordlist is 6.

crunch 3 6 -p raj chandel hackingarticles

```
┌──(root㉿kali)-[~/crunch]
└─# crunch 3 6 -p raj chandel hackingarticles
Crunch will now generate approximately the following amount of data: 156 bytes
0 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 6
chandelhackingarticlesraj
chandelrajhackingarticles
hackingarticleschandelraj
hackingarticlesrajchandel
rajchandelhackingarticles
rajhackingarticleschandel
```

Wordlist Permutations

Just like words can be permuted, wordlists can be permuted. Using the "-q" option, crunch can take input from a wordlist and do permutations on what is read in the file. For example, if the file list is:

A

B

C

Then, **crunch -q list.txt** would output:

ABC

ACB

BAC

BCA

CAB

CBA

Similarly, we can do permutations on 3 char per word wordlist like so:

crunch 3 3 abc -e acc -o 2.txt

crunch 3 3 abc -q 2.txt -o 3.txt

```
┌──(root㉿kali)-[~/crunch]
└─# crunch 3 3 abc -e acc -o 2.txt
Crunch will now generate the following amount of data: 36 bytes
0 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 9

crunch: 100% completed generating output

┌──(root㉿kali)-[~/crunch]
└─# cat 2.txt
aaa
aab
aac
aba
abb
abc
aca
acb
acc

┌──(root㉿kali)-[~/crunch]
└─# crunch 3 3 abc -q 2.txt -o 3.txt
Crunch will now generate approximately the following amount of data: 1016064
9 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 362880
^C
crunch: 100% completed generating output

┌──(root㉿kali)-[~/crunch]
└─# head -n 15  3.txt
aaaaabaacabaabbabcacaacbacc
aaaaabaacabaabbabcacaaccacb
aaaaabaacabaabbabcacbacaacc
aaaaabaacabaabbabcacbaccaca
aaaaabaacabaabbabcaccacaacb
aaaaabaacabaabbabcaccacbaca
aaaaabaacabaabbacaabcacbacc
aaaaabaacabaabbacaabcaccacb
aaaaabaacabaabbacaacbabcacc
aaaaabaacabaabbacaacbaccabc
aaaaabaacabaabbacaaccabcacb
aaaaabaacabaabbacaaccacbabc
aaaaabaacabaabbacbabcacaacc
aaaaabaacabaabbacbabcaccaca
aaaaabaacabaabbacbacaabcacc
```

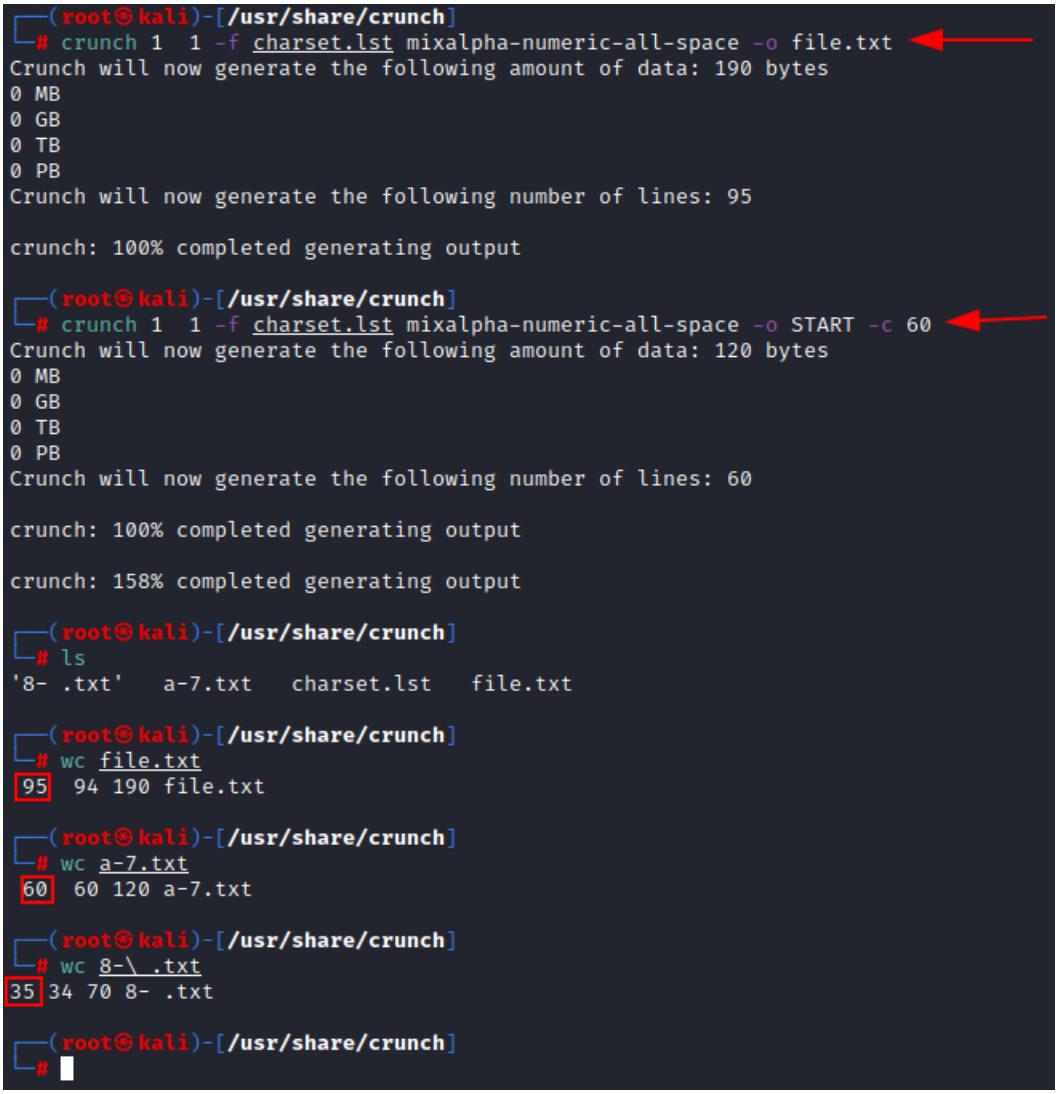
Splitting wordlist based on word count

A wordlist can be cut short using the "-c" option. Here, a file with 94 words has been generated. Now, to split that into multiple files each containing 60 words maximum can be done like so. Note, that this only works with "-o START" which will autoname the files in the format:

Starting character - Ending character.txt

Here, start and ending are a,7 and for next file, 8 and /(space)

```
crunch 1  1 -f charset.lst mixalpha-numeric-all-space -o file.txt
crunch 1  1 -f charset.lst mixalpha-numeric-all-space -o START -c 60
```



Splitting wordlist based on size

To cut short a file based on the size, we can use "-b" filter. For example, to split a wordlist in multiple files each of maximum 1 MB we can do:

```
crunch 4 7  Pass123 -b 1mb -o START
```

Remember, -o START is compulsory as it will automatically split the file in the format:

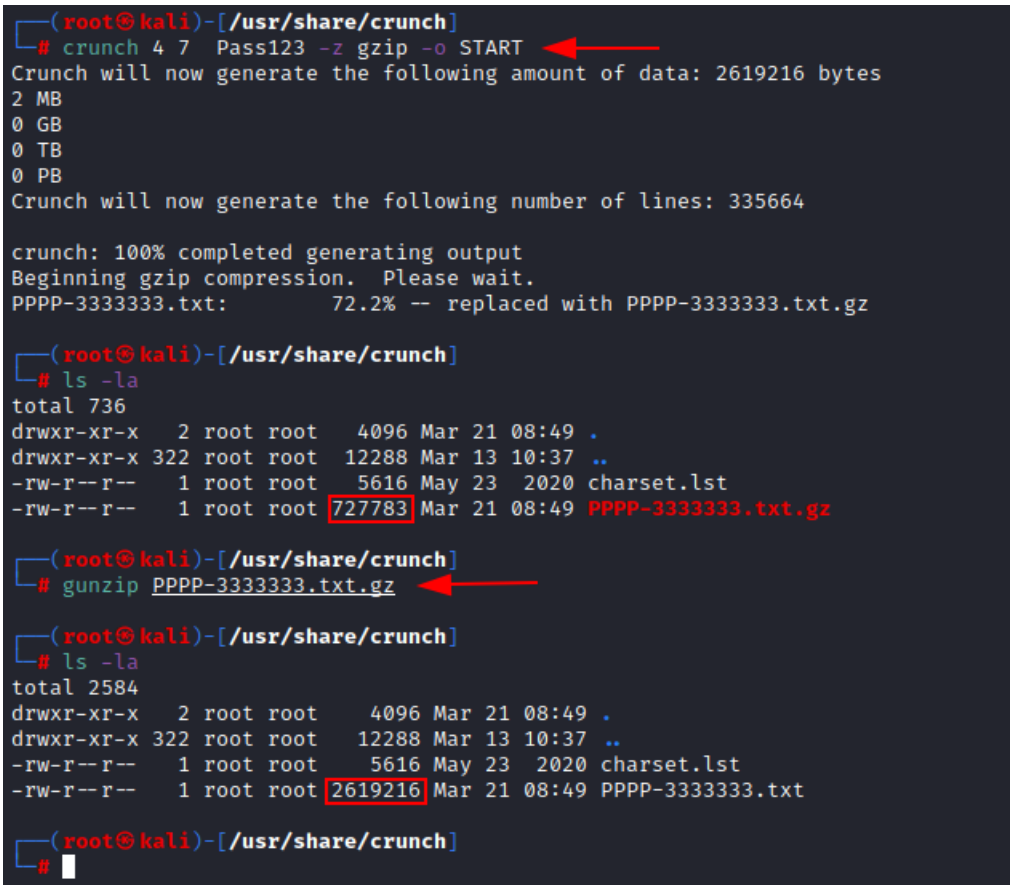
Starting character - Ending character.txt

```
┌──(root㉿kali)-[/usr/share/crunch]
└─# crunch 4 7  Pass123 -b 1mb -o START
Crunch will now generate the following amount of data: 2619216 bytes
2 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 335664

crunch:  39% completed generating output

crunch:  76% completed generating output

crunch: 100% completed generating output

┌──(root㉿kali)-[/usr/share/crunch]
└─# ls -la
total 2592
drwxr-xr-x   2 root root    4096 Mar 21 08:44 .
drwxr-xr-x 322 root root   12288 Mar 13 10:37 ..
-rw-r--r--   1 root root  619216 Mar 21 08:44 2sPa132-3333333.txt
-rw-r--r--   1 root root 1000000 Mar 21 08:44 a13232s-2sPa131.txt
-rw-r--r--   1 root root    5616 May 23  2020 charset.lst
-rw-r--r--   1 root root 1000000 Mar 21 08:44 PPPP-a13232a.txt

┌──(root㉿kali)-[/usr/share/crunch]
└─#
```

Compressing wordlist

Oftentimes, wordlists are too large in size while in text format and gzip can be used to compress them to over 60-70%. For example, to compress a file of max 7 mixalpha-numeric charset and autoname using START we can do this:

```
crunch 4 7  Pass123 -z gzip -o START
gunzip PPPP-3333333.txt.gz
```



Conclusion

The article is meant to be considered as a ready reference for quick and dirty wordlist generation using crunch. Crunch is a powerful and very fast tool written in C which is available by default in Kali Linux and is allowed to be used in competitive security certification exams. Hope you liked the article and thanks for reading.

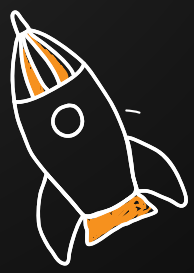
iGNITE Technologies

JOIN OUR TRAINING PROGRAMS

CLICK HERE

BEGINNER

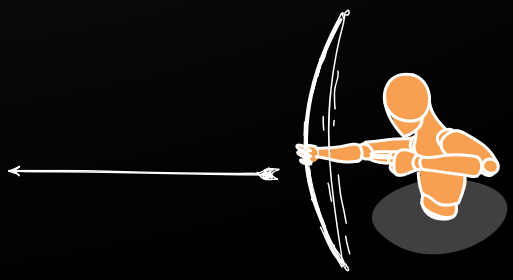
- Ethical Hacking
- Network Pentest
- Bug Bounty
- Wireless Pentest
- Network Security Essentials



ADVANCED

- Burp Suite Pro
- Android Pentest
- Web Services-API
- Advanced Metasploit
- Pro Infrastructure VAPT
- CTF
- Computer Forensics

EXPERT



- Red Team Operation
  - APT's - MITRE Attack Tactics
  - Active Directory Attack
  - MSSQL Security Assessment
- Privilege Escalation
  - Windows
  - Linux