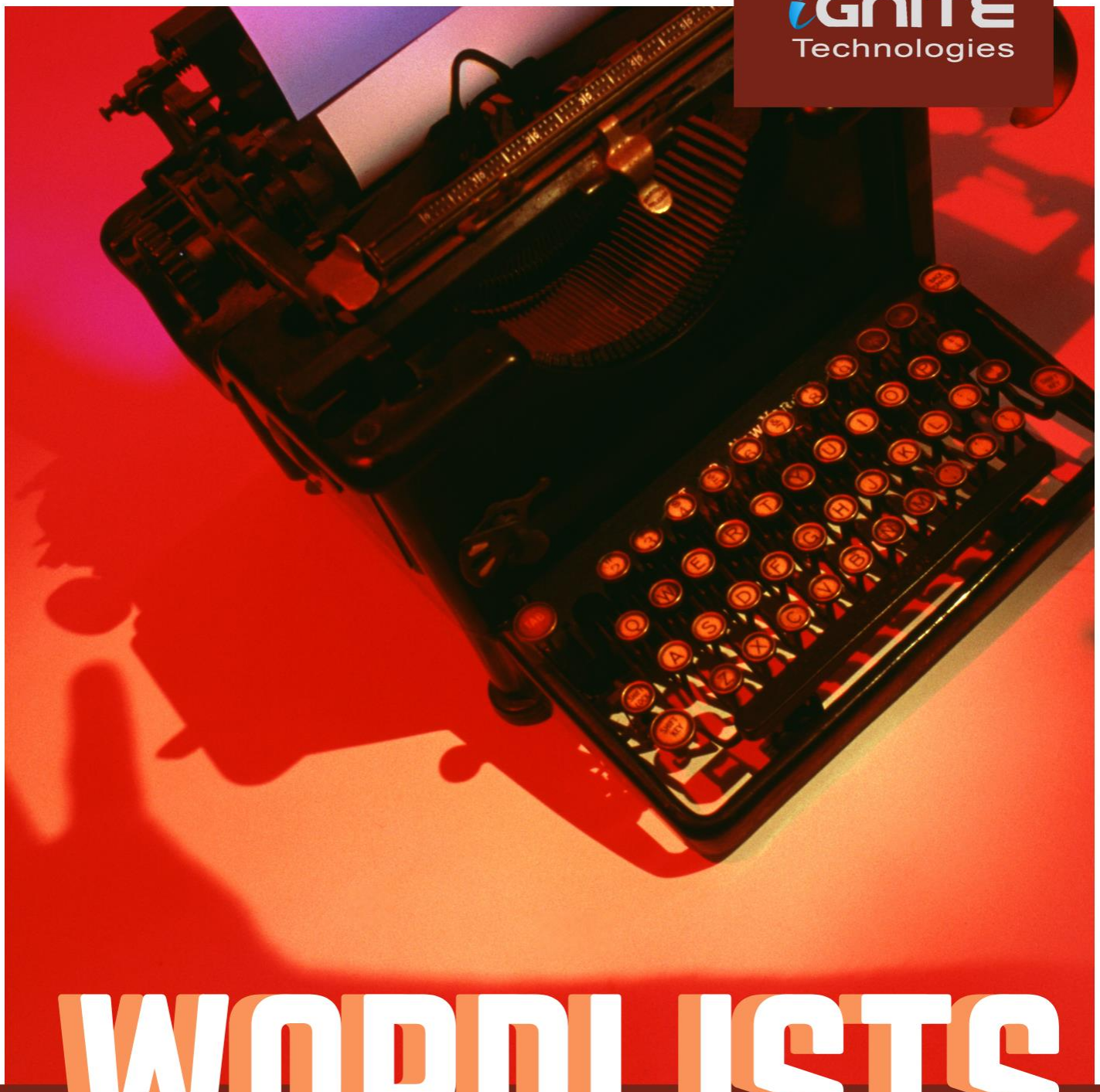
iGNITE Technologies



WORDLISTS FOR PENTESTER

WWW.HACKINGARTICLES.IN

Table of Contents

Abstract

A Pentester is as good as their tools and when it comes to cracking the password, stressing authentication panels or even a simple **directory Bruteforce** it all drills down to the wordlists that you use. Today we are going to understand wordlists, look around for some good wordlists, run some tools to manage the wordlists, and much more.

Ever since the evolution of Penetration Testers has begun, one of the things we constantly see is that the attacker cracks the password of the target and gets in! Well in most of the depictions of the attacks in movies and series often show this situation in detail as it is the simplest attack to depict. No matter how simple cracking passwords or performing **Credential Stuffing** were once a ban on the Web Applications. Today we somehow have got a bit of control over them with the use of **CAPTCHA** or Rate Limiting but still, they are one of the effective attacks. The soul of such attacks is the wordlist.

What are Wordlists?

A wordlist is a file (a text file in most cases but not limited to it) that contains a set of values that the attacker requires to provide to test a mechanism. This is a bit complex, let's dilute it a bit to understand better. Whenever an attacker is faced with an Authentication Mechanism, they can try to work around it but if that is not possible then the attacker has to try some well-known credentials into the Authentication Mechanism to try and guess. This list of well know credentials is a wordlist. And instead of manually entering the values one by one, the attacker uses a tool or script to automate this process. Similarly, in the case of cracking hash values, the tool uses the wordlists and encodes the entries of wordlists into the same hash and then uses a string compare function to match the hashes. If a match is found then the hash is deemed as cracked. It can be observed that the importance of wordlist is paramount in the Cyber Security World.

Wordlists in Kali Linux

Since Kali Linux was specially crafted to perform Penetration Testing, it is full of various kinds of wordlists. This is because of the various tools that are present in the Kali Linux to perform Bruteforce Attacks on Logins, Directories, etc. Let's go through some of the wordlists from the huge arsenal of wordlists Kali Linux contain.

Wordlists are located inside the /usr/share directory. Here, we have the dirb directory for the wordlists to be used while using the dirb tool to perform Directory Bruteforce. Then we have the dirbuster that is a similar tool that also performs Directory Bruteforce but with some additional options. Then we have a fern-wifi directory which helps to break the Wi-Fi Authentications. Then we have the Metasploit which uses wordlists for almost everything. Then there is a nmap wordlist that contains that can be used while scanning some specific services. Then we have the Rockstar of Wordlists: rockyou. This is compressed by default and you will have to extract it before using it. It is very large with 1,44,42,062 values that could be passwords for a lot of user accounts on the internet. At last, we have the wfuzz directory that has the wordlists that can be used clubbed with wfuzz.

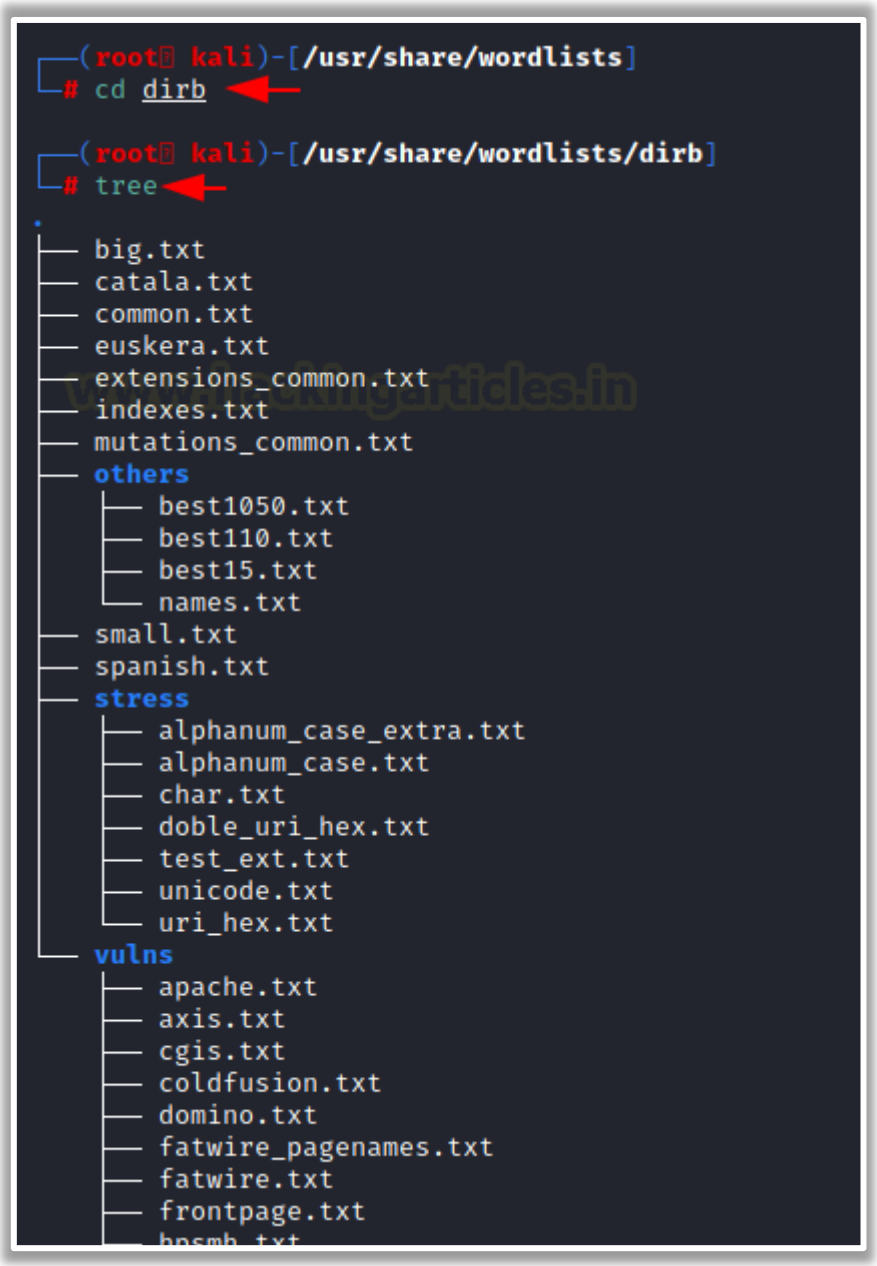
Location: /usr/share/wordlists

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

┌──(root㉿ kali)-[/usr/share/wordlists]
└─# ls -la
total 136660
drwxr-xr-x   2 root root      4096 Feb 26 14:57 .
drwxr-xr-x 351 root root     12288 Mar 11 12:42 ..
lrwxrwxrwx   1 root root        25 Nov 17 07:24 dirb → /usr/share/dirb/wordlists
lrwxrwxrwx   1 root root        30 Nov 17 07:24 dirbuster → /usr/share/dirbuster/wordlists
lrwxrwxrwx   1 root root        41 Nov 17 07:24 fasttrack.txt → /usr/share/set/src/fasttrack/wordlist.txt
lrwxrwxrwx   1 root root        45 Nov 17 07:24 fern-wifi → /usr/share/fern-wifi-cracker/extras/wordlists
lrwxrwxrwx   1 root root        46 Nov 17 07:24 metasploit → /usr/share/metasploit-framework/data/wordlists
lrwxrwxrwx   1 root root        41 Nov 17 07:24 nmap.lst → /usr/share/nmap/nselib/data/passwords.lst
-rw-r--r--   1 root root 139921507 Feb 26 14:50 rockyou.txt
lrwxrwxrwx   1 root root        25 Nov 17 07:24 wfuzz → /usr/share/wfuzz/wordlist
```

Dirb Wordlists

To take a closer look at one of the directories, we use the tree command to list all the wordlists inside the dirb directory. Here we have different wordlists that differ in size and languages. There is an extensions wordlist too so that the attacker can use that directory to perform a Directory Bruteforce. There are some application-specific wordlists such as apache.txt or sharepoint.txt as well.

Location: /usr/share/wordlists/dirb



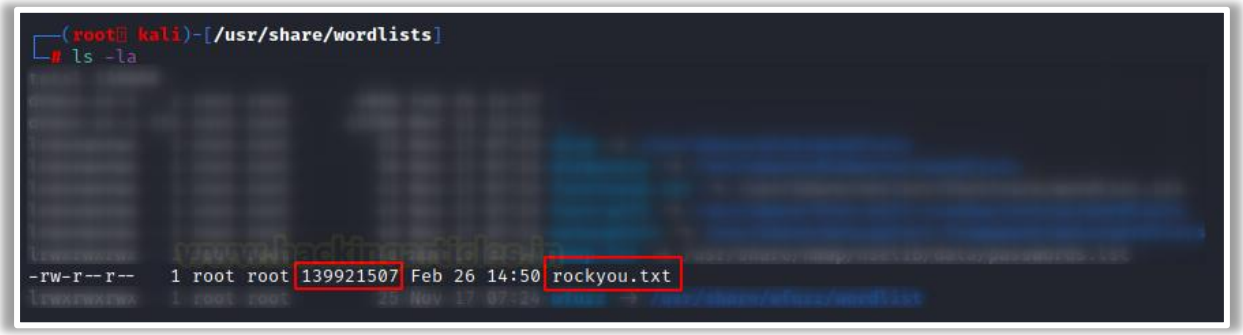
Rockyou wordist

Rockyou.txt is a set of compromised passwords from the social media application developer also known as RockYou. It developed widgets for the Myspace application. In December 2009, the company experienced a data breach resulting in the exposure of more than 32 million user accounts. It was mainly because of the company's policy of storing the passwords in cleartext.

Location: /usr/share/wordlists

When first booting Kali Linux, it will be compressed in a gz file. To unzip run the following command. It will decompress and ready for use on any kind of attack you want.

```
gzip -d /usr/share/wordlists/rockyou.txt.gz
```



Wfuzz Wordlists

Wfuzz tool was developed to perform Bruteforcing attacks on web applications. It can further be used to enumerate web applications as well. It can enumerate directories, files, and scripts, etc. It can change the request from GET to POST as well. That is helpful in a bunch of scenarios such as checking for SQL Injections. It comes with a set of predefined wordlists. These wordlists are designed to be used with wfuzz but they can be used anywhere you desire. The wordlists are divided into categories such as general, Injections, stress, vulns, web services, and others.

Location: /usr/share/wordlists/wfuzz

```
┌──(root㉿ kali)-[/usr/share/wordlists]
└─# tree
.
├── dirb → /usr/share/dirb/wordlists
├── dirbuster → /usr/share/dirbuster/wordlists
├── fasttrack.txt → /usr/share/set/src/fasttrack/wordlist.txt
├── fern-wifi → /usr/share/fern-wifi-cracker/extras/wordlists
├── metasploit → /usr/share/metasploit-framework/data/wordlists
├── nmap.lst → /usr/share/nmap/nselib/data/passwords.lst
├── rockyou.txt
└── wfuzz → /usr/share/wfuzz/wordlist

5 directories, 3 files

┌──(root㉿ kali)-[/usr/share/wordlists]
└─# cd wfuzz

┌──(root㉿ kali)-[/usr/share/wordlists/wfuzz]
└─# tree
.
├── general
│   ├── admin-panels.txt
│   ├── big.txt
│   ├── catala.txt
│   ├── common.txt
│   ├── euskera.txt
│   ├── extensions_common.txt
│   ├── http_methods.txt
│   ├── medium.txt
│   ├── megabeast.txt
│   ├── mutations_common.txt
│   ├── spanish.txt
│   └── test.txt
├── Injections
│   ├── All_attack.txt
│   ├── bad_chars.txt
│   ├── SQL.txt
│   ├── Traversal.txt
│   ├── XML.txt
│   └── XSS.txt
├── others
│   ├── common_pass.txt
│   └── names.txt
├── stress
│   ├── alphanum_case_extra.txt
│   ├── alphanum_case.txt
│   ├── char.txt
│   ├── doble_uri_hex.txt
│   ├── test_ext.txt
│   └── uri_hex.txt
└── vulns
    ├── apache.txt
    ├── cgis.txt
    ├── coldfusion.txt
    ├── dirTraversal-nix.txt
    ├── dirTraversal.txt
    ├── dirTraversal-win.txt
```

Looking into the Injections directory we see that we have an All_attack.txt that is a pretty generic wordlist for testing injections. Then we have a specific one for SQL, Directory Traversal, XML, XSS injections. Moving onto the general directory, we see that we have the big.txt that we discussed in the Dirb section. We have common.txt that also is the default wordlist in many tools due to its small size. Then we have the extensions_common.txt which contains like 25-ish extensions that might be enumerated some files that can be considered low-hanging fruits. Then we have the http_methods.txt wordlist. It contains the HTTP Methods such as POST, GET, PUT, etc. They can be used while testing if the target application has any misconfigured methods enabled or they forgot to disable them at the application and server level. mutations_common.txt also contains a bunch of uncommon extensions that could lead to the enumerations of rare artifacts.

```
┌──(root㉿kali)-[/usr/share/wordlists/wfuzz/Injections]
└─# cat XSS.txt
"><script>"
<script>alert("WXSS")</script>
<<script>alert("WXSS");//<</script>
<script>alert(document.cookie)</script>
'><script>alert(document.cookie)</script>
'><script>alert(document.cookie);</script>
\";alert('XSS');//
%3cscript%3ealert("WXSS");%3c/script%3e
%3cscript%3ealert(document.cookie);%3c%2fscript%3e
%3Cscript%3Ealert(%22X%20SS%22);%3C/script%3E
&ltscript&gtalert(document.cookie);</script>
&ltscript&gtalert(document.cookie);&ltscript&gtalert
<xss><script>alert('WXSS')</script></vulnerable>
<IMG%20SRC='javascript:alert(document.cookie)'>
<IMG%20SRC="javascript:alert('WXSS');">
<IMG%20SRC="javascript:alert('WXSS')"
<IMG%20SRC=javascript:alert('WXSS')>
<IMG%20SRC=JaVaScRiPt:alert('WXSS')>
<IMG%20SRC=javascript:alert(&quot;WXSS&quot;)>
<IMG%20SRC=`javascript:alert("'WXSS'")`>
<IMG%20"""><SCRIPT>alert("WXSS")</SCRIPT>">
<IMG%20SRC=javascript:alert(String.fromCharCode(88,83,83))>
<IMG%20SRC='javasc      ript:alert(document.cookie)'>
<IMG%20SRC="jav ascript:alert('WXSS');">
<IMG%20SRC="jav&#x09;ascript:alert('WXSS');">
<IMG%20SRC="jav&#x0A;ascript:alert('WXSS');">
<IMG%20SRC="jav&#x0D;ascript:alert('WXSS');">
<IMG%20SRC="%20&#14;%20javascript:alert('WXSS');">
<IMG%20DYNSRC="javascript:alert('WXSS')">
<IMG%20LOWSRC="javascript:alert('WXSS')">
```

Then we have the spanish.txt wordlist for the as you have guessed it for Spanish words/names/passwords. The other directory contains the common passwords and names that can be used to extract usernames or passwords at some forget password form where it responds with such messages that the user exists or it doesn't exist. Let's move onto the stress directory. It contains a wordlist designed to stress test the mechanism. It contains wordlists that contain the alphabets or numbers or special characters and hex codes for the same. Then we have the vulns directory, which contains the wordlists specially made for testing a particular vulnerability. We have the apache wordlist, CGI wordlist, directory wordlist, iis wordlist,

oracle9 wordlist, SharePoint wordlist, tomcat wordlist, and many more. Use these wordlists into a specific scenario where you are confirmed about the framework and versioning information and just use it to target a particular entry point.

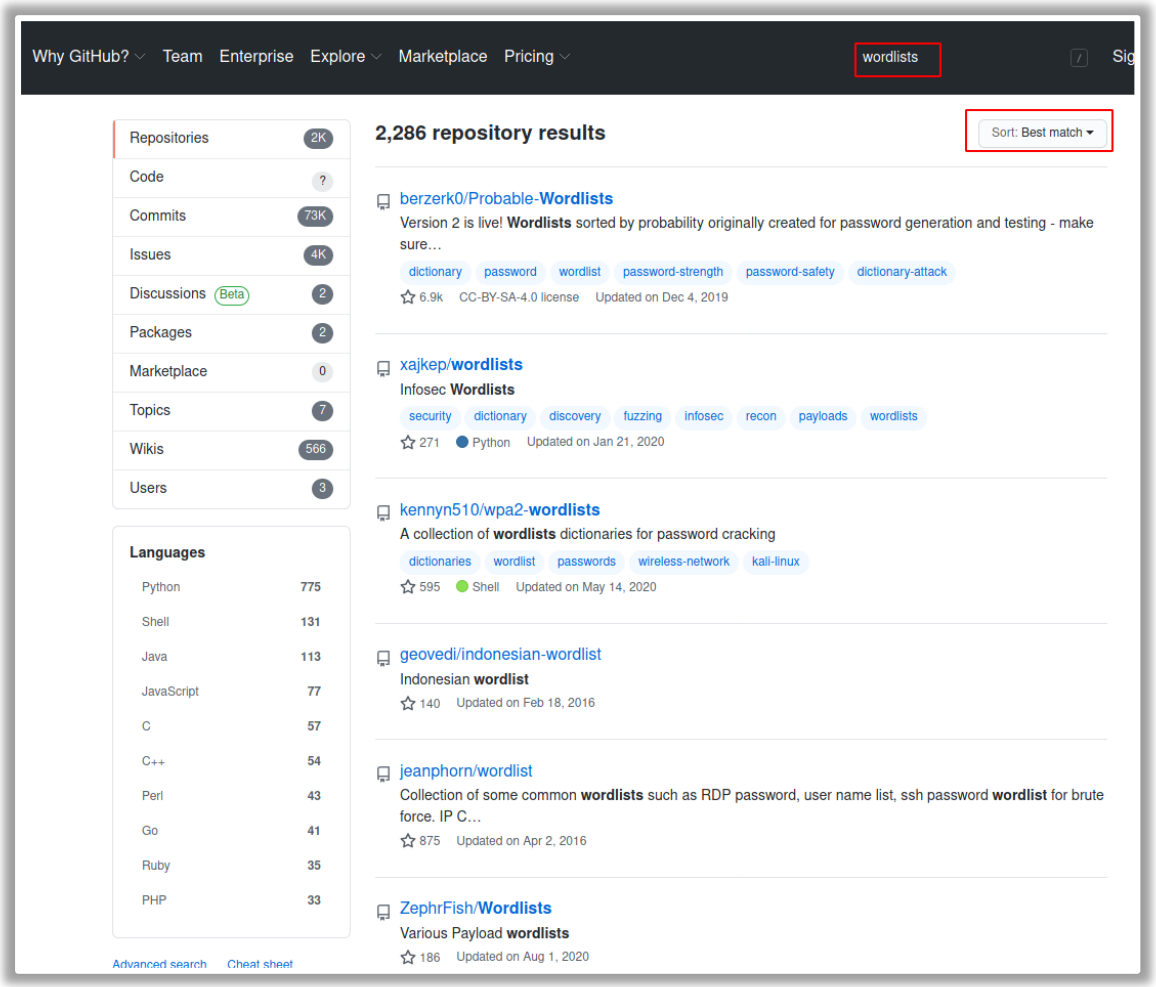
Online Wordlists

Github Wordlists

We learned about the huge collection that Kali Linux contains. But sometimes they tend to be not as latest as we require. This can happen in a scenario in which a new 0-day has been discovered. There will be no entry in those dictionaries. This is where we can go wild searching on the internet but it is vast and takes more time. This is where we can snoop in GitHub as many people might create such a dictionary. So, searching GitHub might give you those new and fresh dictionaries or it can help you find that specific dictionary that you require to fuzz a specific framework.

```
Location: /usr/share/wordlists/wfuzz
```

Link: GitHub Wordlists

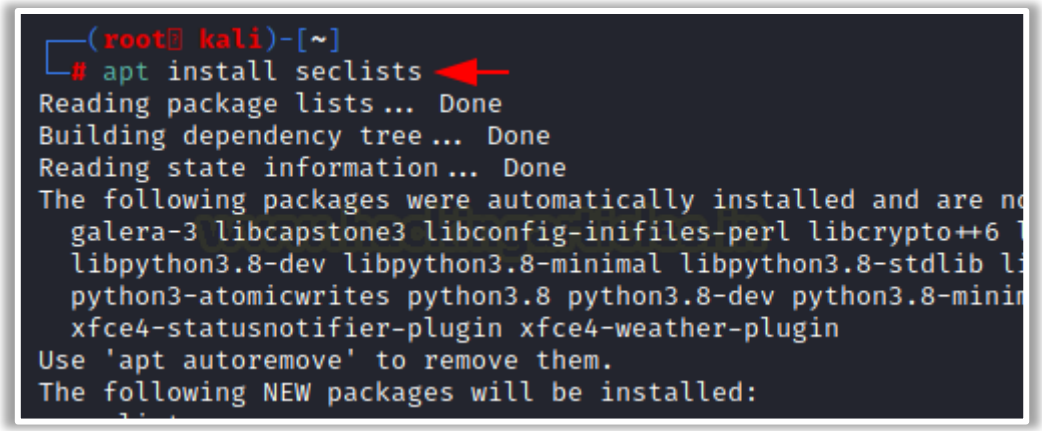


Seclists

Seclists are a collection of multiple types of wordlists that can be used during Penetration Testing or Vulnerability Assessment, all collected in one place. These wordlists can contain usernames, passwords, URLs, sensitive data patterns, fuzzing payloads, web shells, etc. To install on Kali Linux, we will use the apt command followed by the Seclists as shown in the image below.

GitHub: Seclists

```
gzip -d /usr/share/wordlists/rockyou.txt.gz
```



The installation will create a directory by the name of Seclists inside the /usr/share location. Going through we can see the different categories of wordlists such as Discovery, Fuzzing, IOCs, Misc, Passwords, Pattern Matching, Payloads, Usernames, and Web-Shells.

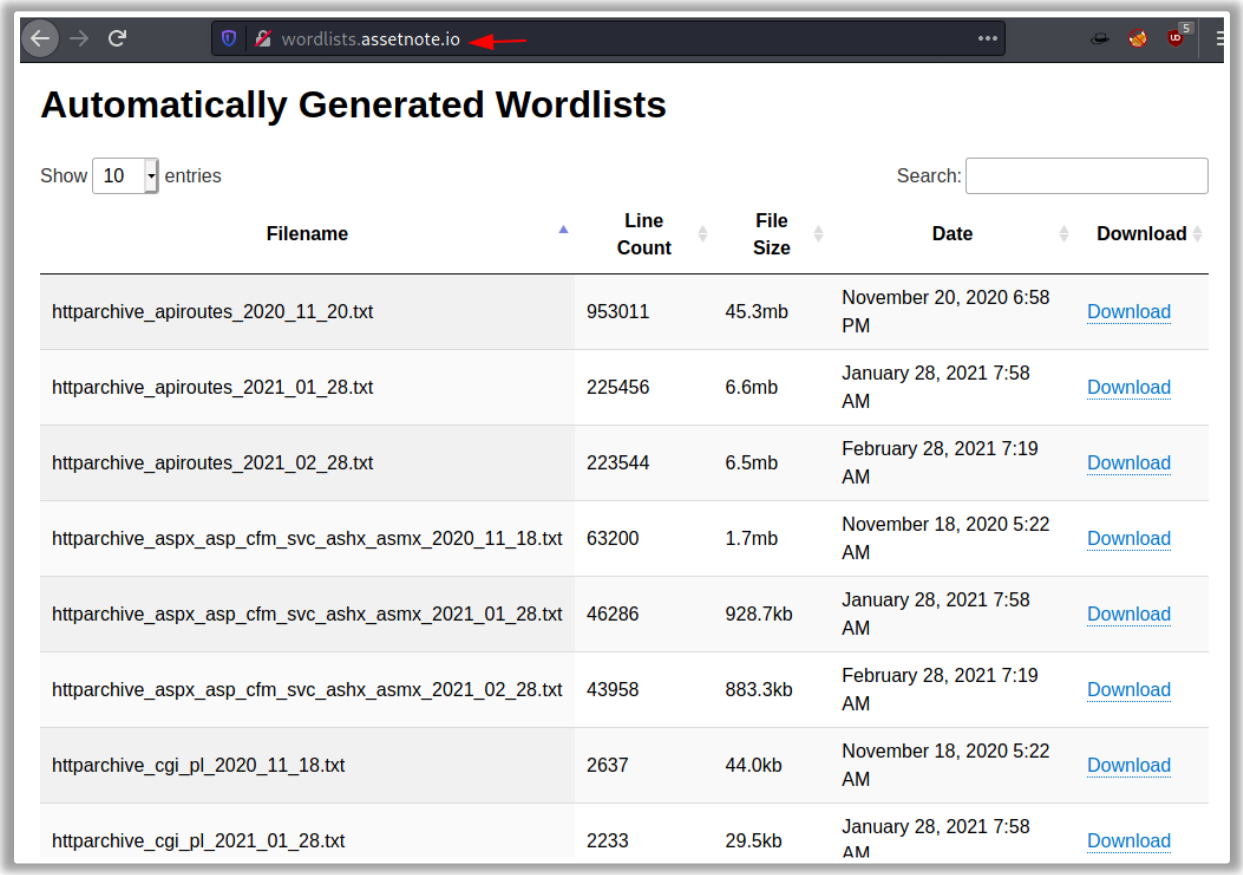
```
┌──(root㉿ kali)-[~]
└─# cd /usr/share/seclists

┌──(root㉿ kali)-[/usr/share/seclists]
└─# ls -la
total 56
drwxr-xr-x  11 root root  4096 Mar 17 14:07 .
drwxr-xr-x 352 root root 12288 Mar 17 14:06 ..
drwxr-xr-x   9 root root  4096 Mar 17 14:06 Discovery
drwxr-xr-x   8 root root  4096 Mar 17 14:07 Fuzzing
drwxr-xr-x   2 root root  4096 Mar 17 14:07 IOCs
drwxr-xr-x   5 root root  4096 Mar 17 14:07 Miscellaneous
drwxr-xr-x  12 root root  4096 Mar 17 14:07 Passwords
drwxr-xr-x   3 root root  4096 Mar 17 14:07 Pattern-Matching
drwxr-xr-x   9 root root  4096 Mar 17 14:07 Payloads
-rw-r--r--   1 root root  1953 Feb 11 16:59 README.md
drwxr-xr-x   4 root root  4096 Mar 17 14:07 Usernames
drwxr-xr-x   9 root root  4096 Mar 17 14:07 Web-Shells

┌──(root㉿ kali)-[/usr/share/seclists]
└─# cd Passwords

┌──(root㉿ kali)-[/usr/share/seclists/Passwords]
└─# tree
.
├── 2020-200_most_used_passwords.txt
├── BiblePass
│   ├── BiblePass_part01.txt
│   ├── BiblePass_part02.txt
│   ├── BiblePass_part03.txt
│   ├── BiblePass_part04.txt
│   ├── BiblePass_part05.txt
│   ├── BiblePass_part06.txt
│   ├── BiblePass_part07.txt
│   ├── BiblePass_part08.txt
│   ├── BiblePass_part09.txt
│   ├── BiblePass_part10.txt
│   ├── BiblePass_part11.txt
│   ├── BiblePass_part12.txt
│   ├── BiblePass_part13.txt
│   ├── BiblePass_part14.txt
│   ├── BiblePass_part15.txt
│   ├── BiblePass_part16.txt
│   └── BiblePass_part17.txt
├── bt4-password.txt
├── cirt-default-passwords.txt
├── clarkson-university-82.txt
├── Common-Credentials
│   ├── 100k-most-used-passwords-NCSC.txt
│   ├── 10k-most-common.txt
│   ├── 10-million-password-list-top-1000000.txt
│   ├── 10-million-password-list-top-100000.txt
│   ├── 10-million-password-list-top-10000.txt
│   ├── 10-million-password-list-top-1000.txt
│   ├── 10-million-password-list-top-100.txt
│   ├── 10-million-password-list-top-500.txt
│   ├── 500-worst-passwords.txt
│   ├── best1050.txt
```

Assetnode Wordlists

The Assetnode Wordlist releases a specially curated wordlist for a whole wide range of areas such as the subdomain discovery or special artifacts discovery. The best part is that it gets updated on the 28th of Each month as per their website. This is the next best thing that was released ever since the Seclists. To download all wordlists at once anybody can use the following wget command.

Website: [Assetnote Wordlists]

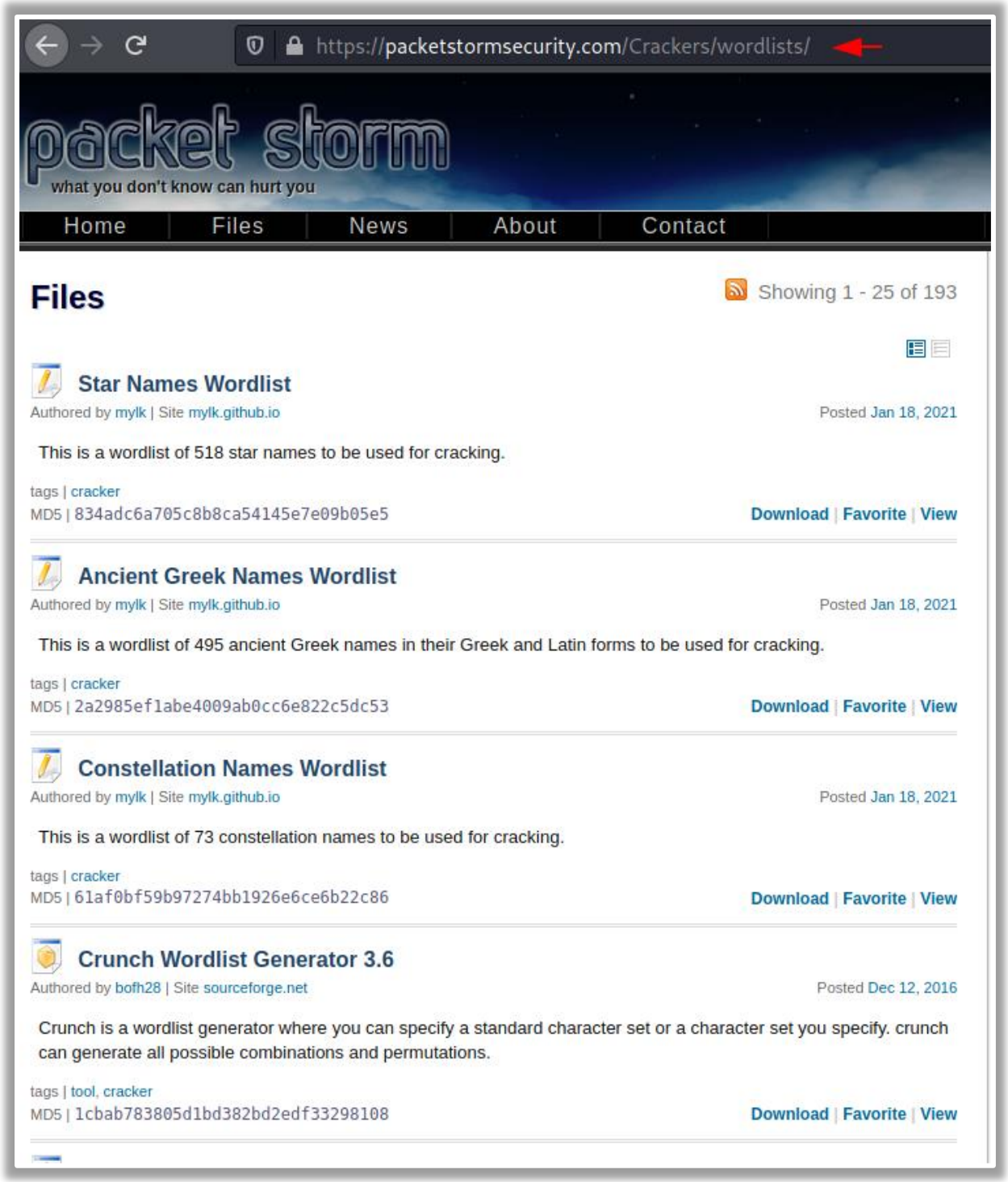
```
wget -r --no-parent -R "index.html*" https://wordlists-cdn.assetnote.io/ -nH
```



Packetstorm Wordlists

Packet Storm Security is an information security website that offers current and historical computer security tools, exploits, and security advisories. It is operated by a group of security enthusiasts that publish new security information and offer tools for educational and testing purposes. But much to our surprise, it also publishes wordlists. Any user that has crafter some specified wordlist can submit their wordlist on their website. So, if you are looking for a unique wordlist be sure to check it out.

Link: [Pack Strom Security Wordlists]



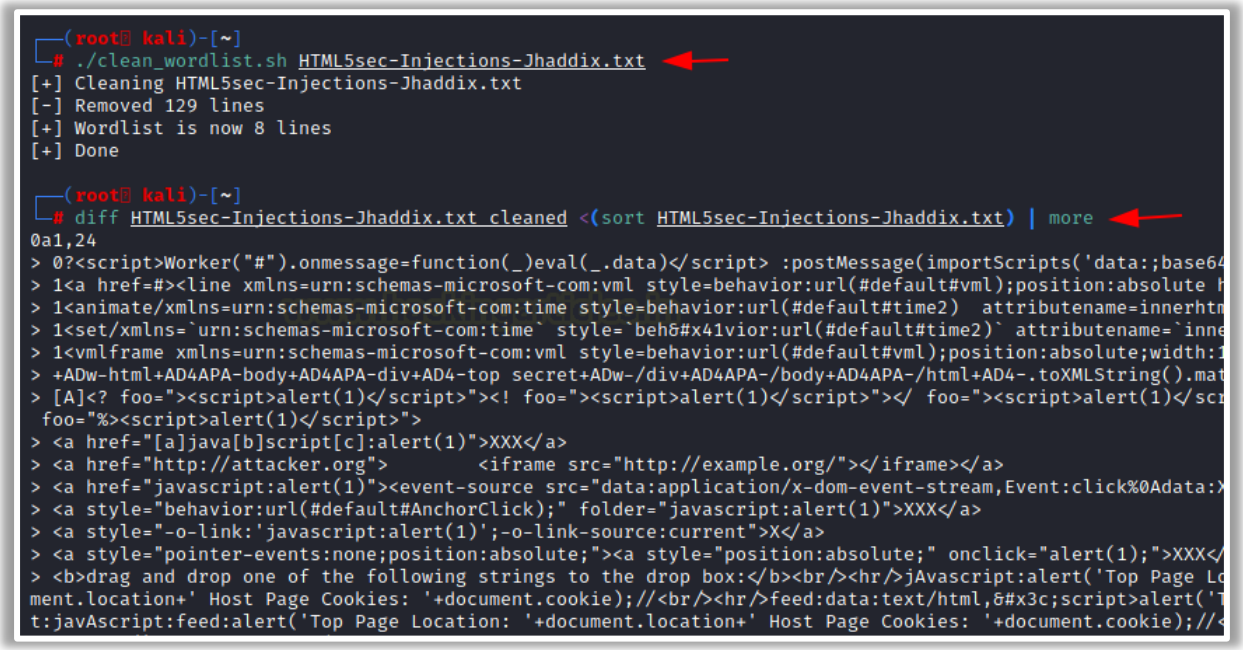
Cleaning Wordlists

Till now we saw multiple wordlists that contain thousands and thousands of entries inside them. Now during penetration testing on your vulnerable server or any CTF, it is possibly fine as they are designed to handle this kind of bruteforce but when we come to the real-life scenario things get a little complicated. As in real life, no development team or owner is going to permit you to perform a thousand after thousand wordlist bruteforce. This can hamper its quality of service to other customers. So, we should decrease the wordlist entries. I know it sounds

counterproductive but it is not. The wordlists might contain some payloads that might be exceeding 100 characters or even be too specific for them to extract anything directly. Then we do have some payloads that are the way to similar to each other that if we replace any one of them, the result remains the same**.** Jon Barber created a script that can remove noisy charters such as ! (, %. Furthermore, tidy the wordlist so that it can be more effective.

GitHub: [CleanWordlist.sh]

```
./clean_wordlists.sh HTML5sec-Injections-Jhaddix.txt
```



We can check the lines that were removed from the HTML5 Injection wordlist using the diff command as shown in the image above.

```
diff HTML5sec-Injections-Jhaddix.txt_cleaned < (sort
HTML5sec-Injections-Jhaddix.txt) | more
```

Crafting Wordlists

Cewl

CeWL is a Ruby application that spiders a given URL to a specified depth, optionally following external links, and returns a list of words that can then be used for password crackers such as John the Ripper. CeWL also has an associated command-line app, FAB (Files Already Bagged) which uses the same metadata extraction techniques to create author/creator lists from already downloaded. Here we are running CeWL against the tart URL and saving the output into a wordlist by the name of dict.txt.

GitHub: [CeWL – Custom Word List generator]

Learn More: [Comprehensive Guide on CeWL Tool]

```
┌──(root㉿ kali)-[~]
└─# cewl https://www.ignitetechnologies.in/ -w dict.txt
CeWL 5.4.8 (Inclusion) Robin Wood (robin@digi.ninja) (https://digi.ninja/)

┌──(root㉿ kali)-[~]
└─# head dict.txt
featured
end
the
icon
title
Testing
Penetration
ttm
Training
Security
```

Crafting Wordlist: Crunch

Crunch is a wordlist generator where you can specify a standard character set or a character set you specify. crunch can generate all possible combinations and permutations. Here, we used crunch to craft a wordlist with a minimum of 2 and a maximum of 3 characters and writing the output inside a wordlist by the name of dict.txt.

Learn More: [Comprehensive Guide on Crunch Tool]

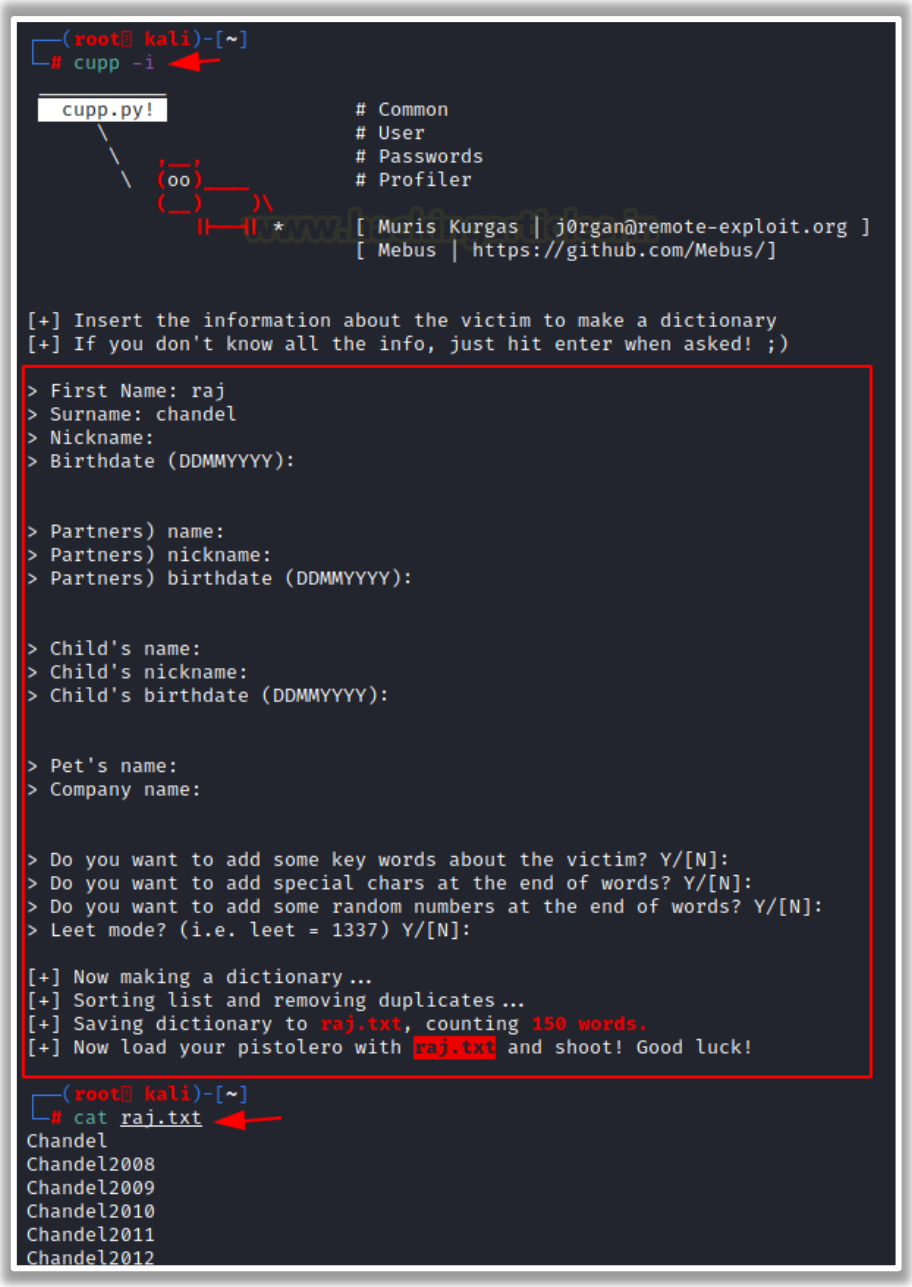
```
┌──(root㉿ kali)-[~]
└─# crunch 2 3 -o dict.txt
Crunch will now generate the following amount of data: 72332 bytes
0 MB
0 GB
0 TB
0 PB
Crunch will now generate the following number of lines: 18252

crunch: 100% completed generating output

┌──(root㉿ kali)-[~]
└─# head dict.txt
aa
ab
ac
ad
ae
af
ag
ah
ai
aj
```

Crafting Wordlists: Cupp

A weak password might be very short or only use alphanumeric characters, making decryption simple. A weak password can also be easily guessed by someone profiling the user, such as a birthday, nickname, address, name of a pet or relative, or a common word such as God, love, money, or password. This is where Cupp comes into use as it can be used in situations like legal penetration tests or forensic crime investigations. Here, we are creating a wordlist that is specific for a person named Raj. We enter the details and upon submission, we have a wordlist that is generated especially for this user.

GitHub: [CUPP – Common User Passwords Profiler]

Learn More: [Comprehensive Guide on Cupp– A wordlist Generating Tool]

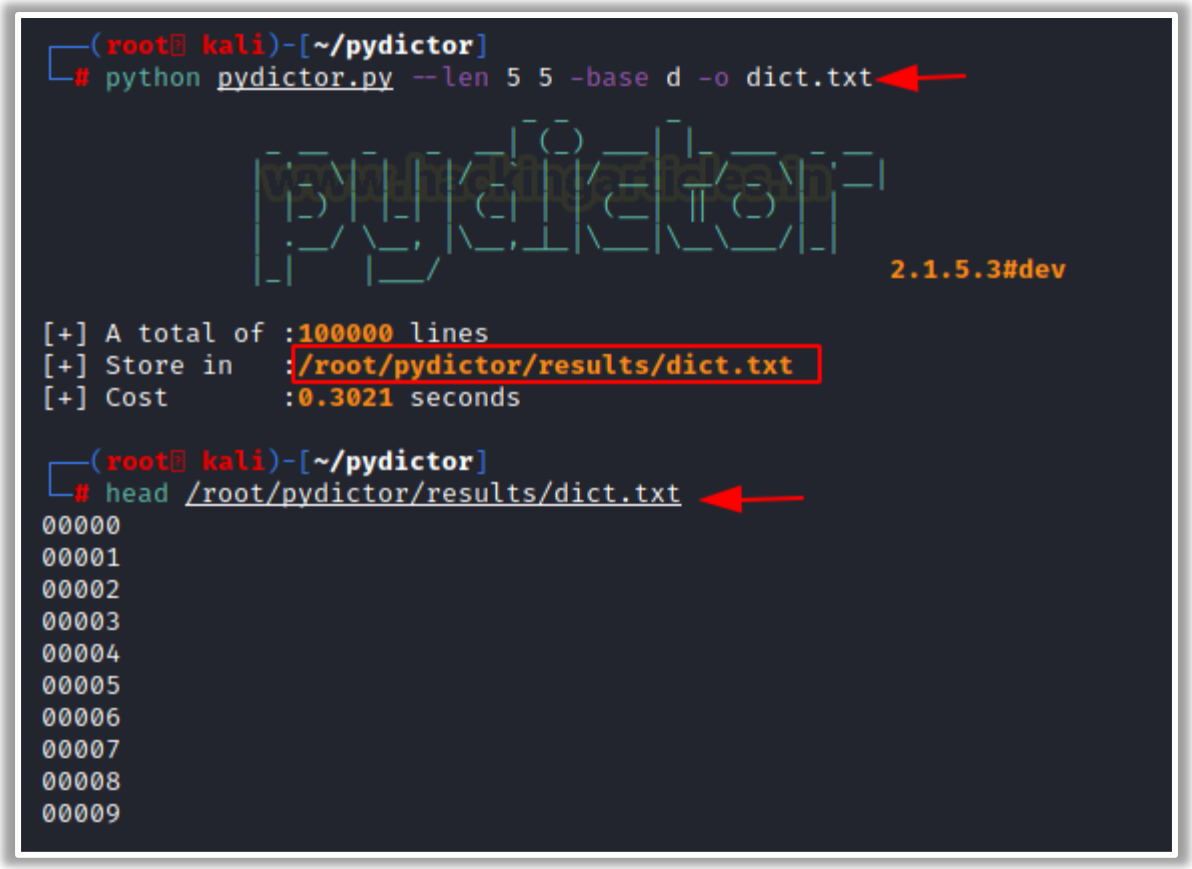


Crafting Wordlists: Pydictor

Pydictor is one of those tools that both novices and pro can appreciate. It is a dictionary-building tool that is great to have in your arsenal when dealing with password strength tests. The tool offers a plethora of features that can be used to create that perfect dictionary for pretty much any kind of testing situation. Here, we defined the base and length as 5 and then create a wordlist. The wordlist contains the numeric up to 5 digits.

GitHub: [pydictor]

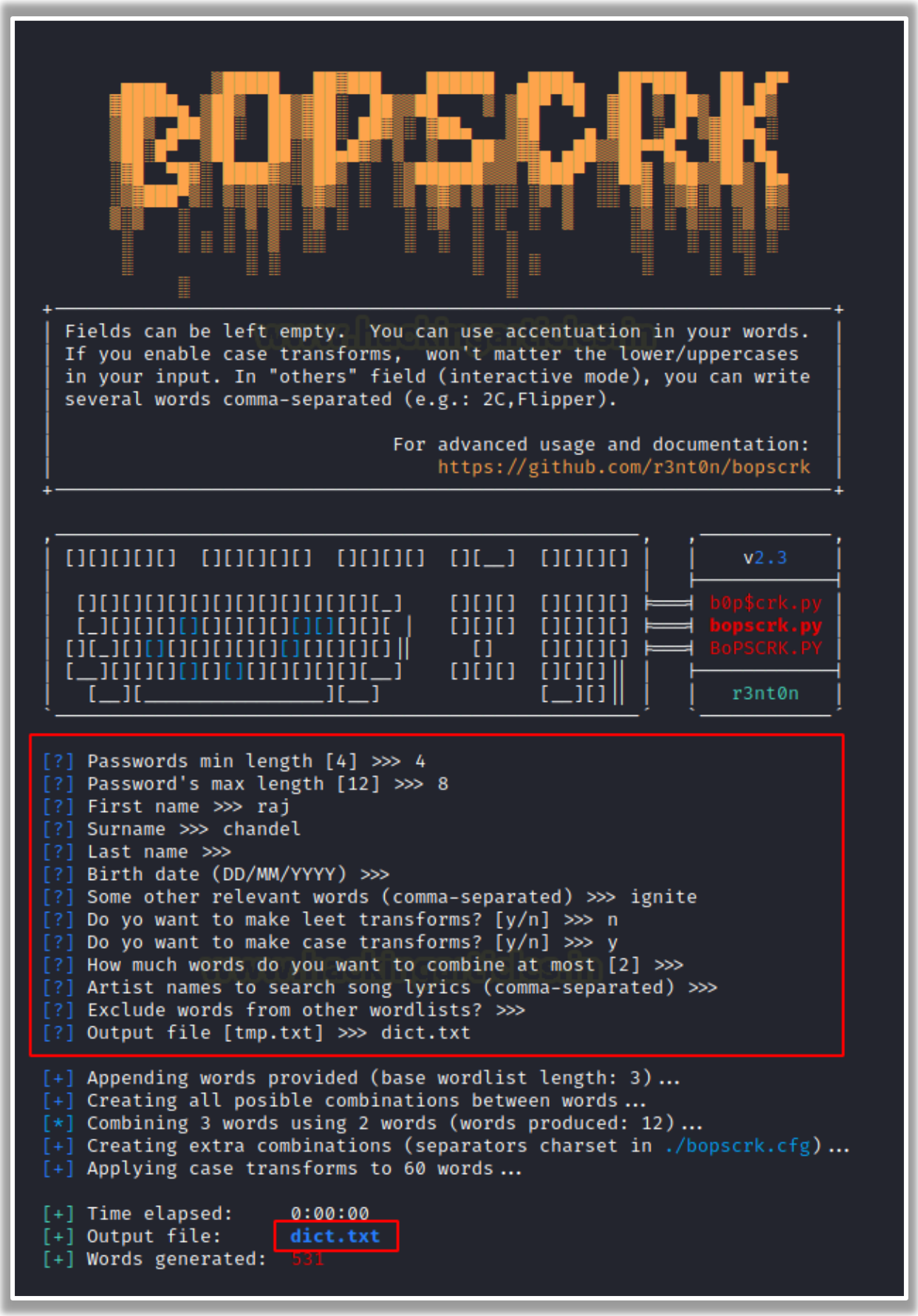
Learn More: [Comprehensive Guide on Pydictor – A wordlist Generating Tool]



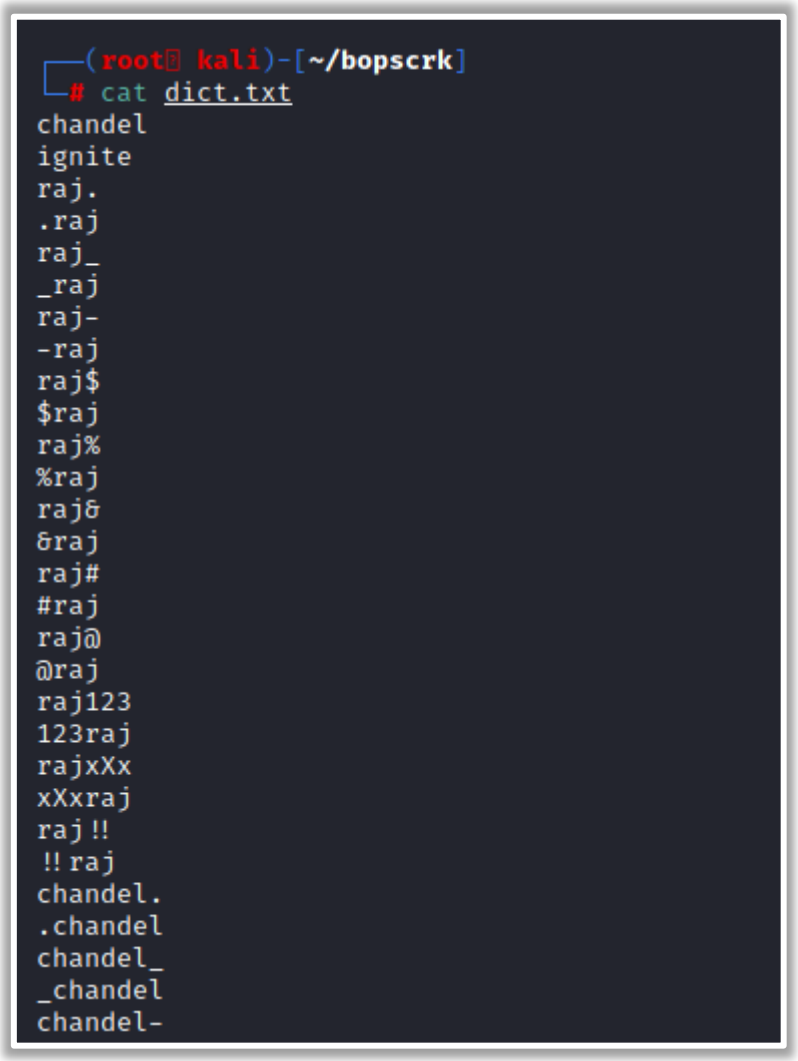
Crafting Wordlists: Bopscrk

Bopscrk (Before Outset PaSsword CRacKing) is a tool to generate smart and powerful wordlists for targeted attacks. It is part of Black Arch Linux for as long as we can remember. It introduces personal information related to the target and combines every word and transforms it into possible passwords. It also contains a lyric pass module which allows it to search lyrics related to the favourite artist of the target and then include them into the wordlists.

GitHub: [Bopscrk]


BOPSCRK
Fields can be left empty. You can use accentuation in your words.
If you enable case transforms, won't matter the lower/uppercases
in your input. In "others" field (interactive mode), you can write
several words comma-separated (e.g.: 2C,Flipper).
For advanced usage and documentation:
https://github.com/r3nt0n/bopscrk
v2.3
b0p$crk.py
bopscrk.py
BoPSCRK.PY
r3nt0n
[?] Passwords min length [4] >>> 4
[?] Password's max length [12] >>> 8
[?] First name >>> raj
[?] Surname >>> chandel
[?] Last name >>>
[?] Birth date (DD/MM/YYYY) >>>
[?] Some other relevant words (comma-separated) >>> ignite
[?] Do yo want to make leet transforms? [y/n] >>> n
[?] Do yo want to make case transforms? [y/n] >>> y
[?] How much words do you want to combine at most [2] >>>
[?] Artist names to search song lyrics (comma-separated) >>>
[?] Exclude words from other wordlists? >>>
[?] Output file [tmp.txt] >>> dict.txt
[+] Appending words provided (base wordlist length: 3)...
[+] Creating all posible combinations between words...
[*] Combining 3 words using 2 words (words produced: 12)...
[+] Creating extra combinations (separators charset in ./bopscrk.cfg)...
[+] Applying case transforms to 60 words...
[+] Time elapsed: 0:00:00
[+] Output file: dict.txt
[+] Words generated: 531

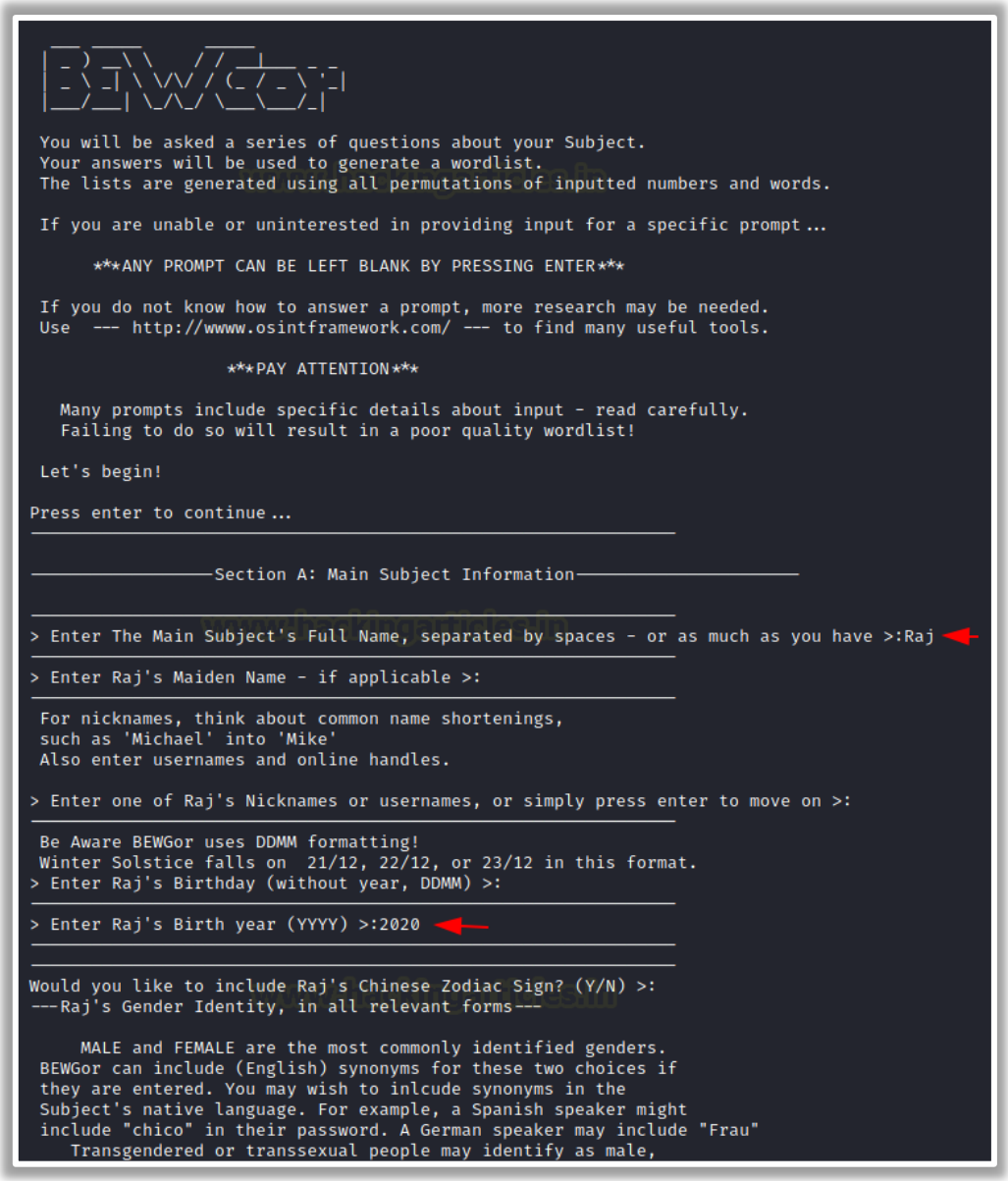
Here, we can see that the wordlist that was crafter from the details that were provided by us is neat and crafter with a high chance to be the actual password of the Raj user.



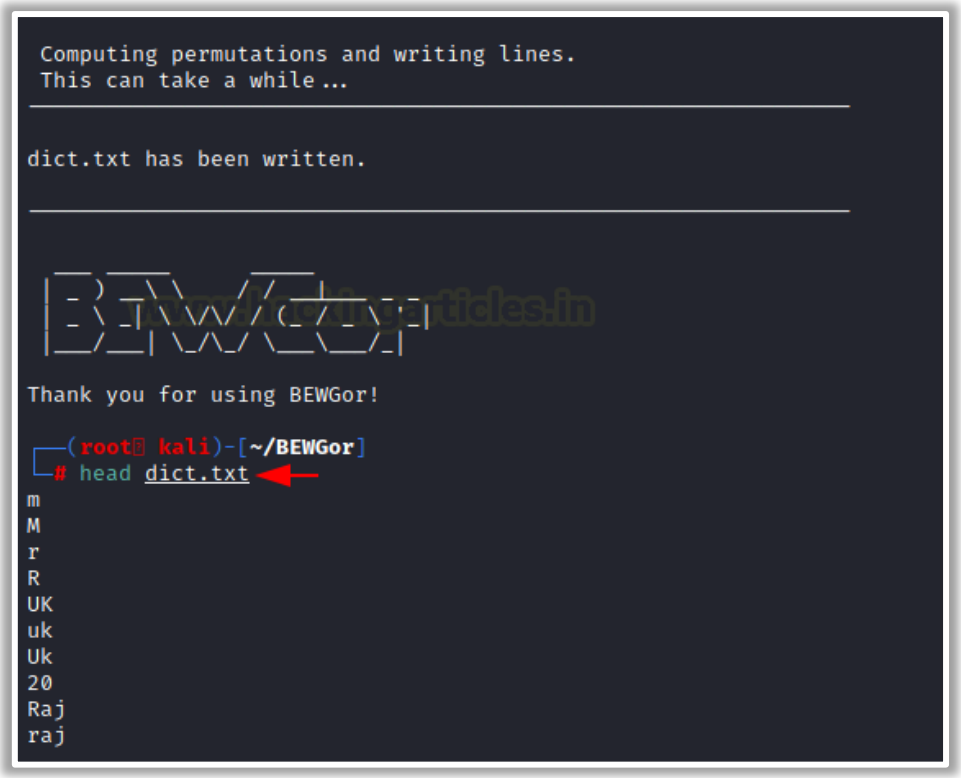
Crafting Wordlists: BEWGor

For starters, let's begin with the pronunciation. It is pronounced as Booger. I know not easy to wrap your head around it. BEWGor is designed to help with ensuring password security. It is a Python script that prompts the user for biographical data about a person, referred to as the Subject. This data is then used to create likely passwords for that Subject. BEWGor is heavily based on Cupp but they are different in some ways as It presents vastly Increased Information Detail on Main Subject, it includes support for an arbitrary number of family members and pets, Users can use permutations to generate possible passwords. Also, BEWGor can generate huge numbers of passwords, create Upper/Lower/Reverse variations of inputted values, save raw inputted values to a Terms file before variations are generated, set upper and lower limits on output line length, and check that an inputted Birthday is valid. Birthdays must not be the future, a false leap day, June 32nd, etc.

GitHub: [BEWGor – Bull's Eye Wordlist Generator]



After working for a while, we see that we have a refined wordlist for the user Raj. It can now be used to bruteforce the credentials of Raj.



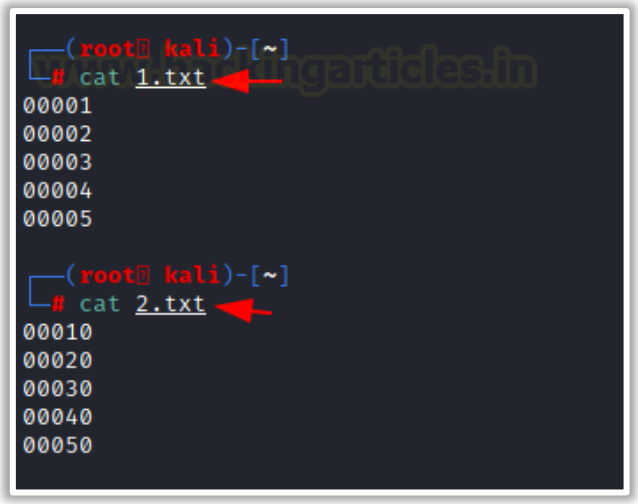
Merging Wordlists: DyMerge

A simple, yet powerful tool – written purely in python – takes given wordlists and merges them into one dynamic dictionary that can then be used as ammunition for a successful dictionary-based (or bruteforce) attack.

GitHub: [DyMerge – Dynamic Dictionary Merger]

Learn More: [Comprehensive Guide on Dymerge]

Here, we have two wordlists: 1.txt and 2.txt. Both containing 5 entries each. We will use DyMerge to combine both wordlists.



Running DyMerge, we provide result.txt as the wordlist to be created by merging 1.txt and 2.txt. This can be observed that the result.txt has 10 entries from both of the wordlists.

Crafting Wordlists: Mentalist

It is a GUI tool for crafting custom wordlists. It uses common human paradigms for creating password-based wordlists. It can craft the full wordlist with passwords but it can also create rules compatible to be cracked with Hashcat and John the Ripper.

It generates by joining nodes which in turn take a shape of a chain. The initial node in the chain is called the Base Words node. Each base word is then passed to the next node in the chain as

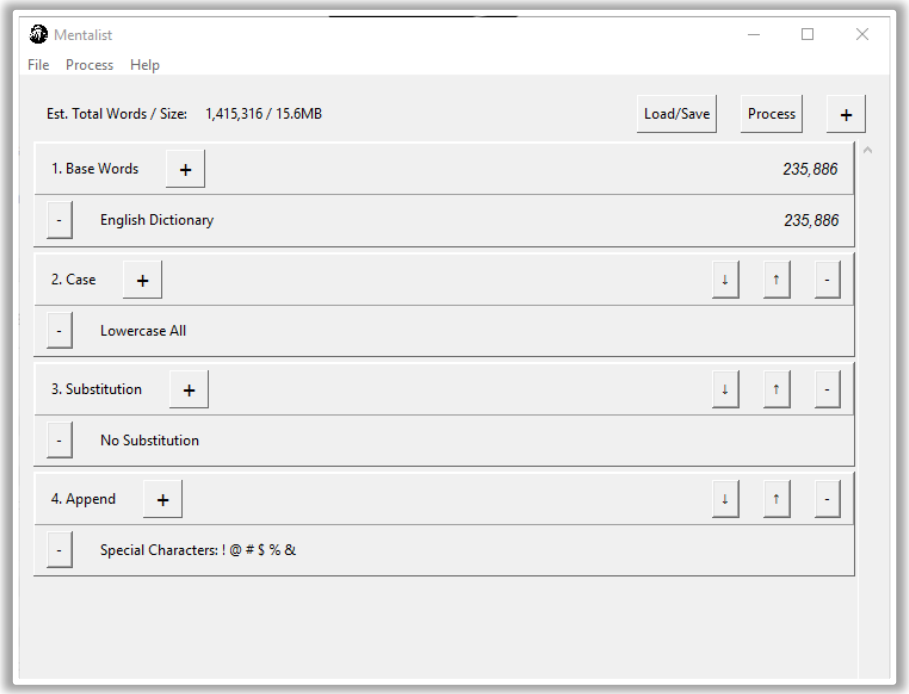
it is processed. That's how the words get modified throughout the wordlists. After working on the chain, it finally writes the result of the chain into the file specified or converts it into the rules as per the user request.

Hashcat/John Rules

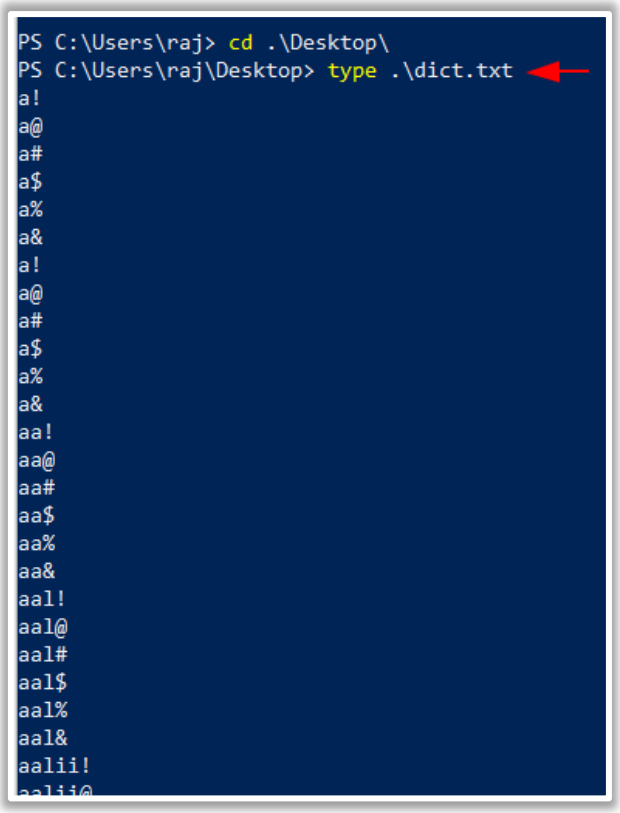
For offline cracking, there are times where the full wordlist is too large to output as a whole. In this case, it makes sense to output to rules so that Hashcat or John can programmatically generate the full wordlist. Download the release from GitHub.

GitHub: [Mentalist]

We are using Windows OS here to demonstrate the ability of Mentalist. We have chosen the English Dictionary as the Base Words. It calculates that 235,886 possible keywords can be manipulated into the passwords by taking English dictionaries as a base. Then we provide some additional options such as Case and if we want to substitute entries and If we want to add Special Character after each entry.



After running for a while, it has crafted a text file by the name of dict.txt. It contains all the passwords that were possible to craft as per our requirements.



Conclusion

The point that we are trying to convey through this article is that wordlist is one of the most important assets a penetration tester can have. There are multiple resources to get a wordlist and multiple tools to craft a wordlist of your own. We wanted this article to serve as your go-to guide whenever you are trying to learn or use a wordlist or any of the tools to craft a wordlist.

References

- https://www.hackingarticles.in/wordlists-for-pentester/

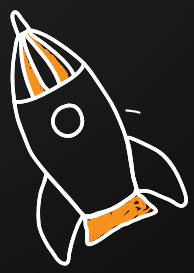
iGNITE Technologies

JOIN OUR TRAINING PROGRAMS

CLICK HERE

BEGINNER

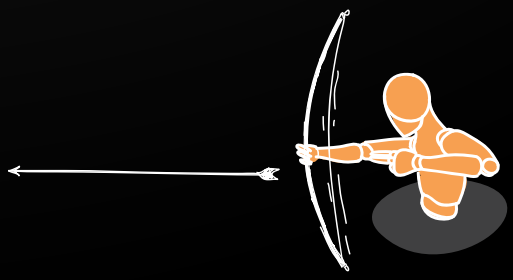
- Ethical Hacking
- Network Pentest
- Bug Bounty
- Wireless Pentest
- Network Security Essentials



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