Web Application - ShopMore

Penetration Test Report

Application Security

Sepp Eyckmans
2CSS

2024 - 2025

Campus Geel, Kleinhoefstraat 4, BE-2440 Geel

Table of Contents

Document version

Version	Comments	Date	Author
1.0	Initial report to client	31/12/2024	Sepp Eyckmans

1 OVERVIEW

1.1 Scope

The scope of this test is limited to a single server hosting the “ShopMore” web application at the IP address *192.168.1.100*. The application includes three user accounts, each with a unique username and password for login. Everything else surrounding the web application is out-of-scope. This security assessment will be conducted over a period of 2 to 3 months.

1.1.1 Time Scope

Name	Start Date	End Date	Executed by	Reviewed by
1ste Assessment	07/10/2024	31/12/2024	Sepp Eyckmans	Alexander Hensels & Michaël Cloots

1.1.2 Material Scope

- “ShopMore” Web Application
  - 192.168.1.100

1.1.3 Out-Of-Scope

- “ShopMore” Web Application
  - Containerized (Docker) application container
  - Direct access to server OS & infrastructure
  - HTTP (Request & Responds) content

- MySQL RDBMS Database
  - Direct access to database & database data
  - Containerized (Docker) database container
  - Direct access to server OS & infrastructure

1.1.4 User Accounts

The following user accounts listed below will be utilized during this security assessment. These placeholder accounts will be either modified or deleted following the completion of the assessment period.

Username	Password	Role
user	user	Normal User
test	test	Admin
admin	admin	Admin

1.1.5 Methodology

We will utilize Burp Suite Community Edition, a comprehensive penetration testing tool developed by PortSwigger, to conduct this assessment. The tool will be employed for both manual testing processes.

The security testing tool ZAP (Zed Attack Proxy) will be employed to conduct automated scans for identifying common vulnerabilities in the web application. Most of the vulnerability testing will be automated using ZAP's robust scanning capabilities.

the penetration testing tool SQLmap will be used to automate the detection of SQL injection vulnerabilities in the web application. In conjunction with SQLmap, manual testing will also be performed to thoroughly review and validate potential SQL injection issues. This combination of automated and manual testing ensures comprehensive coverage and accurate identification of security flaws.

For more information about Burp Suite, ZAP and SQLmap, visit:

https://portswigger.net/burp

https://www.zaproxy.org/

https://sqlmap.org/

Our testing approach is grounded in the OWASP Top 10, the WSTG (Web Security Testing Guide), the CVSS scoring system, and the CWE vulnerability classification database. We begin by aligning the WSTG tests with each of the OWASP Top 10 categories, ensuring that relevant tests are grouped under their corresponding OWASP classification.

Following a structured approach, we work through the OWASP Top 10 list, performing all pertinent WSTG tests for each category. When a vulnerability is identified, we link it to one or more relevant CWE entries for precise classification. We then evaluate the severity of the vulnerability using the CVSS scoring system, calculated with the CVSS calculator tool.

Each finding will include the WSTG test number, the corresponding CWE classification(s), the calculated CVSS score, and detailed information about the finding.

For more information OWASP top 10, visit :

https://owasp.org/Top10/

For more information about WSTG, visit :

https://owasp.org/www-project-web-security-testing-guide/latest/

For more information about CVSS and the CVSS calculator, visit :

https://www.first.org/cvss/

https://www.first.org/cvss/calculator/3.1

For more information about CWE, visit :

https://cwe.mitre.org/

Here's a clear and concise table presenting the OWASP Top 10 with its associated WSTG tests:

OWASP TOP 10	WSTG tests
1. Broken Access Control	▪ WSTG-CONF (4.2) : Configuration and Deployment Management Testing ▪ WSTG-ATHN (4.4) : Authentication Testing ▪ WSTG-ATHZ (4.5) : Authorization Testing
2. Cryptographic Failures	▪ WSTG-INFO (4.1) : Information Disclosure ▪ WSTG-CONF (4.2) : Configuration and Deployment Management Testing ▪ WSTG-CRYPT (4.9) : Testing for Weak Cryptography ▪ WSTG-ERRH (4.8) : Testing for Error Handling
3. Injection	▪ WSTG-ATHN (4.4) : Authentication Testing ▪ WSTG-INPV (4.7) : Input Validation Testing
4. Insecure Design	▪ WSTG-CONF (4.2) : Configuration and Deployment Management Testing ▪ WSTG-ERRH (4.8) : Testing for Error Handling ▪ WSTG-BUSL (4.10) : Business Logic Testing
5. Security Misconfiguration	▪ WSTG-CLNT (4.11) : Client-side Testing
6. Vulnerable and Outdated Components	▪ WSTG-CLNT (4.11) : Client-side Testing
7. Identification and Authentication Failures	▪ WSTG-IDNT (4.3) : Identity Management Testing ▪ WSTG-ATHN (4.4) : Authentication Testing ▪ WSTG-SESS (4.6) : Session Management Testing ▪ WSTG-CLNT (4.11) : Client-side Testing
8. Software and Data Integrity Failures	▪ WSTG-ATHN (4.4) : Authentication Testing ▪ WSTG-ATHZ (4.5) : Authorization Testing ▪ WSTG-SESS (4.6) : Session Management Testing ▪ WSTG-INPV (4.7) : Input Validation Testing ▪ WSTG-CLNT (4.11) : Client-side Testing
9. Security Logging and Monitoring Failures	▪ WSTG-CONF (4.2) : Configuration and Deployment Management Testing ▪ WSTG-ATHN (4.4) : Authentication Testing ▪ WSTG-CLNT (4.11) : Client-side Testing
10. Server-Side Request Forgery (SSRF)	▪ WSTG-CONF (4.2) : Configuration and Deployment Management Testing ▪ WSTG-SESS (4.6) : Session Management Testing ▪ WSTG-INPV (4.7) : Input Validation Testing ▪ WSTG-CLNT (4.11) : Client-side Testing

2 EXECUTIVE SUMMARY

2.1 Findings

CRITICAL	HIGH	MEDIUM	LOW	INFO
2 Findings	1 Findings	2 Findings	2 Findings	0 Findings

2.2 Context

Feestberg Inc. has requested Möbius Security to conduct an penetration test in the period between 7 October 2024 and 31 December 2024.

The goal of this test is to identify any potential security vulnerabilities in Feestberg Inc.'s new "ShopMore" web shop. This web application will serve as a platform for Feestberg Inc.'s customers to browse and purchase products from their online platform.

This report will provide Feestberg Inc. with a comprehensive overview of all identified vulnerabilities, along with recommended solutions for each, to enhance and strengthen the security of their application before its full production launch.

2.3 Observations

2.3.1 Positive

The “ShopMore” web shop platform demonstrates a clear effort to implement proper security measures and comply to industry best practices. The platform’s HTTP methods are well-structured, incorporating essential security features such as session IDs, anti-CSRF tokens, and secure cookie management, ensuring robust protection against common web vulnerabilities.

One of the key security practices employed by ShopMore is the use of parameterized queries across most pages, which effectively mitigates the risk of various injection attacks. This approach ensures that user input is treated as data, not executable code, preventing attackers from injecting malicious commands into the application.

Additionally, the database is configured with appropriate access controls, ensuring that sensitive information is protected from unauthorized access. The use of strict access permissions strengthens the database's defense mechanisms, safeguarding against potential breaches.

2.3.2 Negative

Despite its strengths, our assessment has identified 2 critical vulnerabilities, 1 high severity vulnerability, 2 moderate severity vulnerabilities and 2 low severity vulnerabilities that must be addressed immediately before the platform's full production launch.

First, Forced Browsing was discovered, which allows attackers to bypass access controls by directly navigating to a user creation page. This vulnerability enables unauthorized users to create admin accounts, posing a significant security risk.

Second, an OS Command Injection vulnerability was found, allowing any user to inject commands directly into the host system. When combined with a critical Path Traversal vulnerability, which enables an attacker to navigate to any directory and access any file through the command injection point, this creates a severe exposure of the system. Together, these vulnerabilities provide an attacker with full control over the system, compromising not only the application but also the integrity of the company's infrastructure and certificates. These vulnerabilities must be mitigated and resolved immediately.

Additionally, the lack of a lockout mechanism and rate limiter on the login page makes the system vulnerable to brute force attacks, putting user accounts and sensitive data at risk. Coupled with a security misconfiguration on the login page that fails to block SQL injection payloads, this vulnerability could lead to a Denial of Service (DoS) attack, potentially disrupting access to the platform and compromising its availability.

Furthermore, a minor issue was also discovered where sensitive and restricted pages store their state in the cache. This could result in other users gaining access to sensitive information if they access these cached pages, potentially exposing confidential data.

Finally, the presence of an active debug page poses a serious risk of information disclosure, potentially exposing sensitive system details to unauthorized users. This must be addressed to prevent the leakage of confidential information.

2.3.3 Conclusion & Recommendations

The application has several significant issues that must be addressed before going into production. We strongly recommend implementing proper user input validation and sanitization to ensure that malicious inputs are rejected and not processed. Additionally, a more robust access control solution should be implemented to restrict access to sensitive pages, allowing only authorized users.

To prevent brute-force attacks and resource exhaustion, we recommend adding a lockout mechanism and rate limiter on the login page. Proper cache security must also be implemented to ensure that sensitive data is not exposed through cached pages. Finally, the debug page should be disabled to enhance overall security and prevent potential information leaks. These measures are essential to ensure the platform is secure and ready for launch.

The conclusion of this report is that the "ShopMore" web shop is only ready for production when all these issues described above are resolved. Then and only then should the web application be ready for production and should otherwise stay in development.

3 LIST OF FINDINGS

Title	Status	Severity	Recommendation	Page
Broken Access Control via Forced Browsing	Unresolved	Critical	• User Input Validation & Sanitization • Robust Access Control Mechanisms • Whitelisting	9
Full System Compromise via Directory Traversal	Unresolved	Critical	• User Input Validation & Sanitization • Robust Access Control Mechanisms • Whitelisting • Parameterized queries	12
Webserver Compromise via OS Command Injection	Unresolved	High	• User Input Validation & Sanitization • Robust Access Control Mechanisms • Whitelisting • Parameterized queries	16
Authentication failure by absence of an lockout mechanism	Unresolved	Medium	• Lockout Mechanism • Rate limiter on login attempts • Unpredictable usernames and stronger passwords	20
Denial of Service by Database Connection Pool Exhaustion	Unresolved	Medium	• Lockout Mechanism • Rate limiter on login attempts • Whitelisting • User Input Validation & Sanitization	23
Information Disclosure via active Debug page	Unresolved	Low	• Disable Debug Page • Proper Error Handling • Logging • Remove Unnecessary Information In HTTP Headers	26
Authentication failure via Browser Cache Weakness	Unresolved	Low	• Proper Cache-Control headers • HTTPS	29

3.1 Broken Access Control via Forced Browsing

Critical

Category : A01:2021 – Broken Access Control

Score : CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H

Description

Forced Browsing Vulnerability occurs when attackers gain unauthorized access to restricted resources by manipulating URLs that lack proper authorization mechanisms. This vulnerability allows attackers to guess or brute-force their way into restricted pages without needing valid credentials.

An unauthorized user can navigate to /Admin/Create and create their own admin account, granting themselves administrative privileges. This flaw enables unauthorized individuals to gain full admin-level access to the entire web shop, posing a severe security risk.

This issue is further compounded by the complete lack of authorization across the application. Users browsing the application without logging in are granted full CRUD (Create, Read, Update, Delete) permissions. This allows unauthorized individuals to create or delete products and categories at will, posing a significant security risk.

Risk

This critical vulnerability can be exploited by anyone capable of brute-forcing the unprotected admin page or using automated web application scanners to identify restricted pages. Such tools make it trivial to locate and exploit this flaw. Even inexperienced attackers can easily compromise the entire application if they know the restricted pages name.

The vulnerability represents a significant security lapse, as it allows for straightforward vertical privilege escalation, enabling attackers to gain admin-level access without proper authorization. Furthermore, an attacker could exploit this access by deleting all active user accounts, remove products or categories, or use the application as a springboard for further attacks, leading to widespread disruption, significant data loss and total compromise of all user accounts and their credentials.

Recommendation

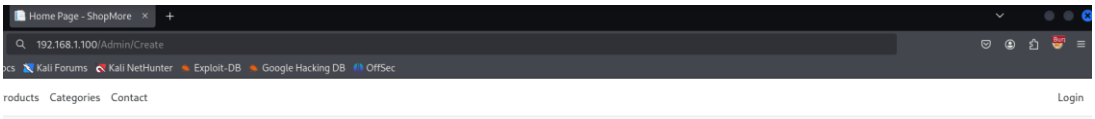
Mitigation of this issue requires a combination of robust security measures. Proper user validation should be enforced, alongside comprehensive access control mechanisms. Applying these mitigations to all sensitive endpoints can effectively prevent this and similar risks, rather than solely relying on obfuscation. Additionally, implementing whitelisting can restrict access to specific pages, ensuring that only authorized individuals and machines are permitted entry.

Proof

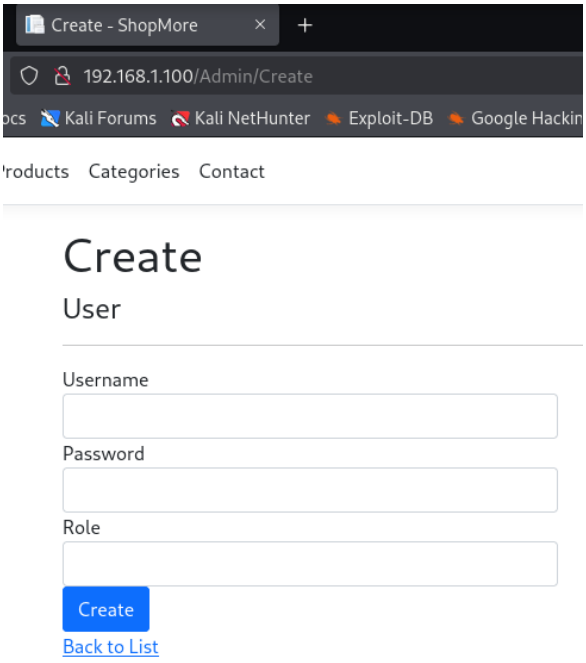
Remark: This is only one instance of this finding. Investigating and mitigating all instances and variations across the entire application is the responsibility of the client.

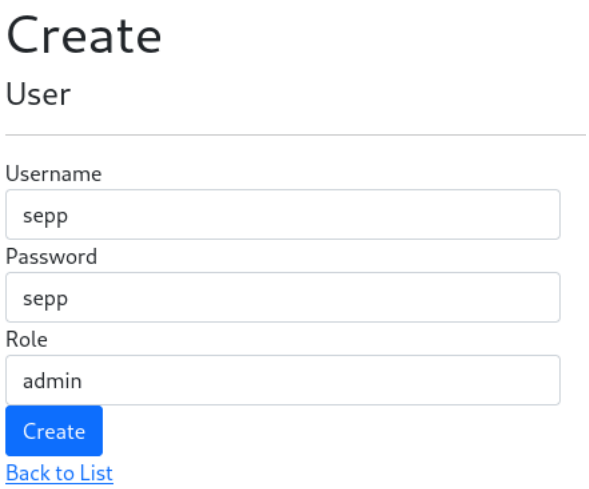
The application fails to authorize any individual browsing to the */Admin/Create* restricted page.

The screenshots below demonstrates that even as a non-logged-in user, you can access this restricted page and create a new admin-level user account :

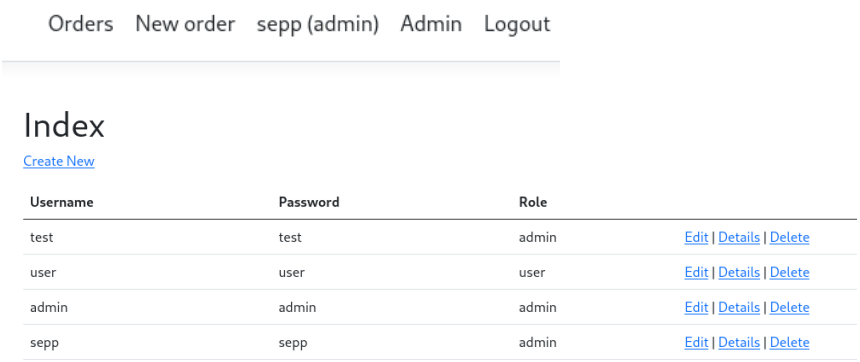


From the Home page, by navigating to */Admin/Create*, any user can access this restricted page, as shown here.





After user creation, login as this user and you will have an own admin-level user account.



Classification

CWE-200 : Exposure of Sensitive Information to an Unauthorized Actor
CWE-425: Direct Request ('Forced Browsing')
CWE-284: Improper Access Control

Found testing

WSTG-CONF-02 (4.2.2) : Test Application Platform Configuration
WSTG-CONF-05 (4.2.5) : Enumerate Infrastructure and Application Admin Interfaces
WSTG-ATHZ-03 (4.5.3) : Testing for Privilege Escalation
WSTG-ATHN-04 (4.4.4) : Testing for Insecure Direct Object References

3.2 Full System Compromise via Directory Traversal

Critical

Category : A05:2021-Security Misconfiguration

Score : CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H

Description

Directory traversal injection are vulnerabilities where an attacker uses an injection point (e.g., OS command injection found in previous finding) to gain unauthorized access to the system, allowing them to fully compromise it. As a result, an attacker can navigate through the system's directories and open any file with full access.

Directory traversal can be done in various ways and is often effective at bypassing security filters or protections in place. Simple filters can be evaded through techniques like nested traversal, while more advanced filters can be bypassed using URL encoding.

When exploiting the OS command injection point found in the previous finding, an attacker can manipulate the command by adding URL encoding to bypass security measures, enabling them to execute more complex and varied commands. This grants them the ability to navigate through all directories and access any file on the system. Sensitive files, such as the passwd file, shadow file, certificates, and configuration files containing database connection strings and credentials, can be exposed.

Risk

With unrestricted access to all directories and files on the system, an attacker can gather every credential and certificate present. This represents a critical vulnerability that goes beyond compromising the application and extends to a severe threat to the entire organization, especially due to the potential for complete certificate compromise. In addition, this breach could provide attackers with the necessary credentials to access the database, amplifying the attack's scope.

The attacker can also leverage this backdoor for extensive reconnaissance of the network, enabling further attacks. Furthermore, with full server access, an Denial-of-Service (DoS) attack can be easily executed, potentially crippling or completely destroying the system.

Recommendation

Directory traversal mitigation is similar to the OS command injection mitigation techniques.

Proper input validation and sanitization will result in even the most complex bypassing technique to be rendered ineffective. Implementing a robust access control system with limited permissions ensures that users and applications have only the minimum necessary access to directories and sensitive files, reducing the impact of any potential attack

Proof

Remark: This is only one instance of this finding. Investigating and mitigating all instances and variations across the entire application is the responsibility of the client.

The application contains an injection vulnerability on the */Export* page, where users can manipulate the value of the *format* parameter to inject any OS command. Any user can exploit this, and its results of the command are displayed for the attacker to view.

Instead of using a standard command, encoding our command in a URL results in :

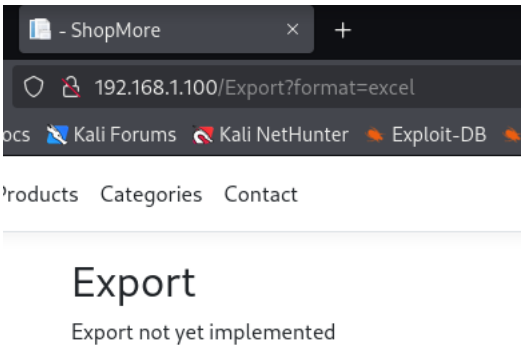
http://192.168.1.100/Export?format=excel%22%26cat+%2Fetc%2Fpasswd%26%22

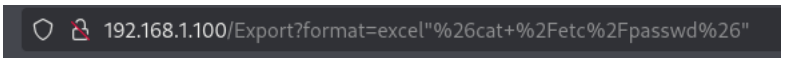
Encode version of :

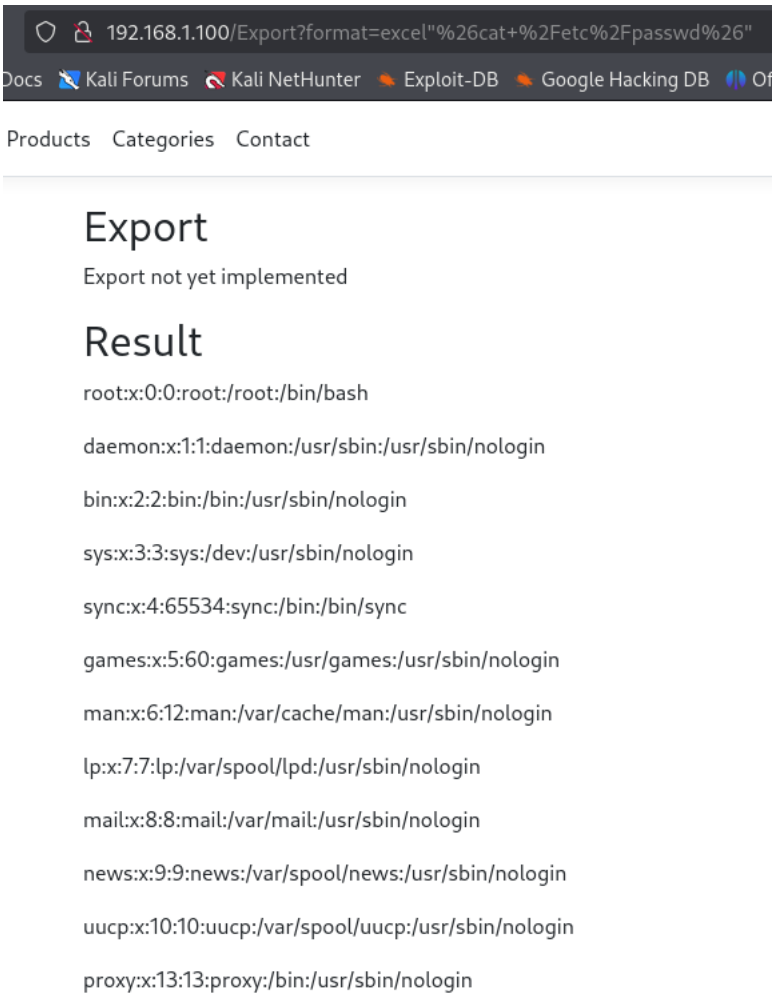
cat etc/passwd

This will result in all basic user info in the server being exposed to the attacker. The attacker will of course not stop here and expose the shadow file also, containing all user passwords.

The exploitation of this vulnerability is demonstrated in the screenshots below :







Some example command are :

http://192.168.1.100/Export?format=excel"%26cat+%2Fetc%2Fshadow%26"

Encoded version of :

Cat /etc/shadow
(all user passwords)

http://192.168.1.100/Export?format=excel"%26cat+%2Fusr%2Fshare%2Fca-certificates%2Fmozilla%2FAmazon_Root_CA_1.crt%26"

Encoded version of :

cat usr/share/certificates/mozilla/Amazon_Root_CA_1.crt
(one of the many certificates to open and view)

http://192.168.1.100/Export?format=excel"%26cat+appsettings.json%26"

Encoded version of :

cat appsettings.json
(contains database connection string with plaintext credentials

Classification

CWE-78 : OS Command Injection

Found testing

WSTG-INPV-12 (4.7.12) : Testing for Command Injection

3.3 Webserver Compromise via OS Command Injection

High

Category : A03:2021 – Injection

Score : CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

Description

OS command injection are vulnerabilities where an attacker can inject OS commands into the application. These commands are then processed and executed by the host system running the application. In some cases, the results of the execution may even be returned to the attacker.

This happens when the application improperly passes user input to a system shell or command-line interface without proper user validation or sanitization. As a result, the attacker can manipulate the application's behavior, execute malicious commands on the host operating system, and potentially gain unauthorized access to the system, data, or resources.

The most common injection target is a URL parameter. By altering the parameter values to include a command, an attacker can easily trigger an OS command injection.

Risk

The result of a command injection is that an attacker gains unauthorized access to the web application's files. Exposing these files provides the attackers with critical info that can be used as a springboard for other and more sophisticated attacks. In this case, only the root application file is directly compromised. However, further exploitation requires additional vulnerabilities (such as directory traversal) to access other sensitive parts of the system.

Additionally, the attacker can use this access to gain deeper insight into the system and its users, paving the way for further reconnaissance and attacks. Malicious commands can also be executed to launch a Denial of Service (DoS) attack, disrupting the application and causing it to become unavailable to legitimate users.

Recommendation

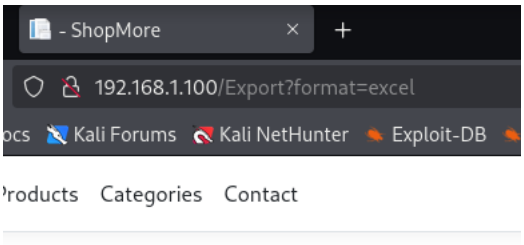
Mitigation for command injection can be implemented in various ways. Robust and proper user input validation and sanitization ensure that all input is strictly validated, and critical characters commonly used in injections are rejected. Whitelisting inputs, where only predefined, safe values are accepted, is an effective approach to minimize the risk of executing harmful commands by ensuring that only trusted data is processed. Using Parameterized queries ensures only proper parameter values are executed. It can do this by restricting the value to only integers or enforcing a specific maximum length (e.g., 5 characters). This can prevent not only command injection, but any other injection type.

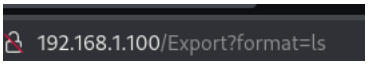
Proof

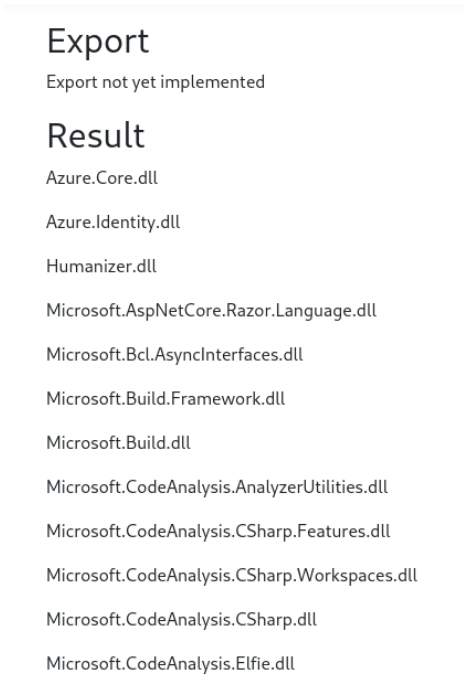
Remark: This is only one instance of this finding. Investigating and mitigating all instances and variations across the entire application is the responsibility of the client.

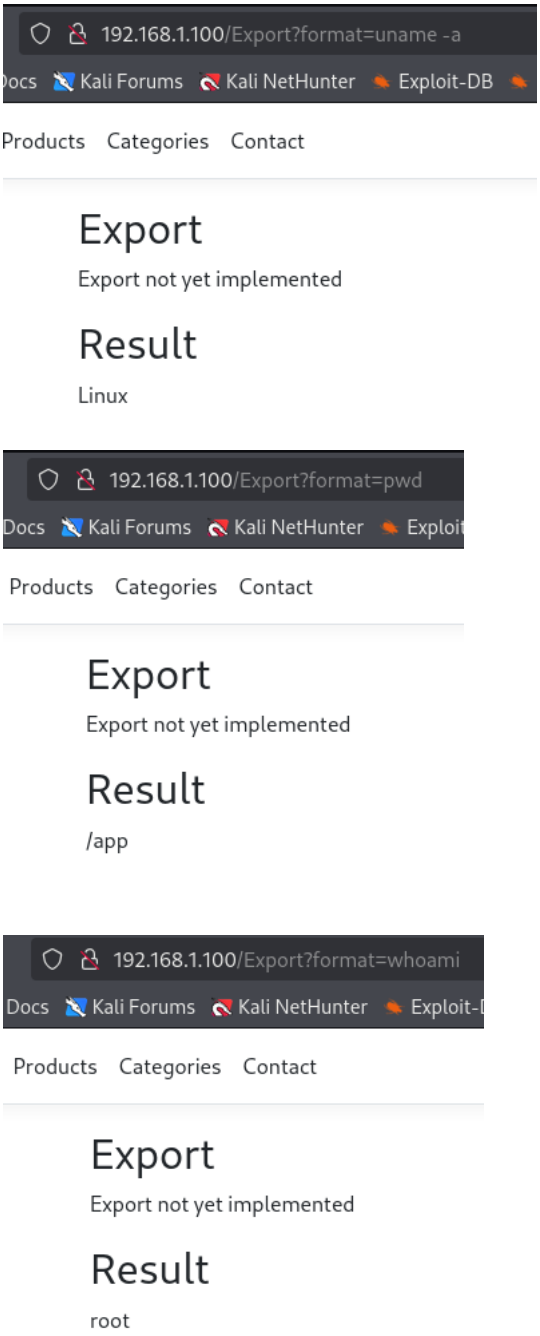
The application contains an injection vulnerability on the */Export* page, where users can manipulate the value of the *format* parameter to inject any OS command. Any user can exploit this, and its results of the command are displayed for the attacker to view.

The exploitation of this vulnerability is demonstrated in the screenshots below:









By replacing *excel* in the parameter *?format=excel*, the following example command could successfully be executed :

- whoami
- ls
- pwd
- uname -a

Classification

CWE-78 : OS Command Injection

Found testing

WSTG-INPV-12 (4.7.12) : Testing for Command Injection

3.4 Authentication failure by absence of an lockout mechanism

Medium

Category : A07:2021-Identification and Authentication Failures

Score : CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:N

Description

During authentication testing, it was identified that the system lacks a lockout mechanism to restrict repeated login attempts. This mechanism is essential to deter brute-force attacks. In its absence, a malicious actor could systematically attempt multiple username and password combinations without encountering any countermeasures.

Combined with weak passwords, an malicious actor is able to brute-force any username and password in a matter of minutes. This causes a breach in authentication and privacy.

Risk

A malicious actor can easily brute-force access to various user accounts. The use of default credential naming conventions, combined with weak passwords, significantly lowers the barrier for attackers to take control of accounts.

This vulnerability allows attackers to compromise accounts within minutes, resulting in a critical breach of authentication protocols and the erosion of user privacy.

Recommendation

Implementing an account lockout mechanism should be mandatory. Locking out accounts after five failed login attempts for a couple of hours can effectively mitigate brute-force attacks by significantly slowing down repeated attempts.

Additionally, enforcing a strong password policy is critical. This includes requiring passwords to be sufficiently complex and unique, with a combination of uppercase and lowercase letters, numbers, and special characters. Combining this with the use of unpredictable default and admin usernames will further strengthen the system's resistance to unauthorized access.

Finally, implementing Multi-Factor Authentication (MFA) will add an additional layer of security ensuring only authorized users can access their account.

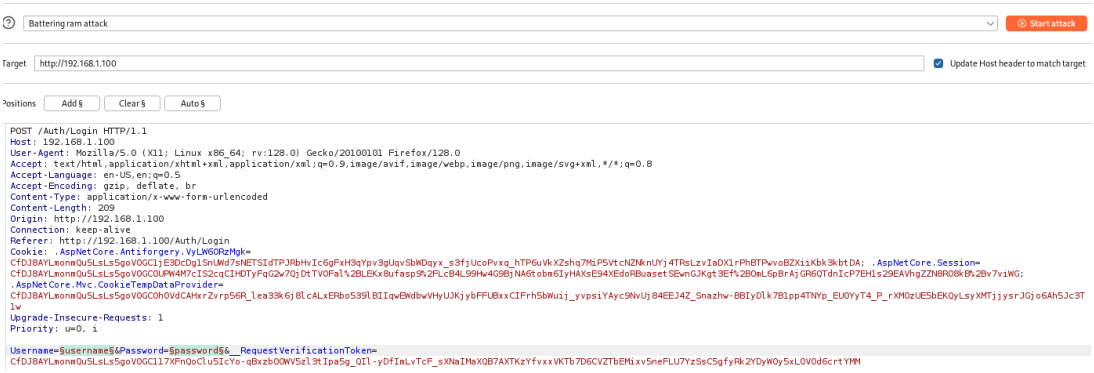
Proof

Remark: This is only one instance of this finding. Investigating and mitigating all instances and variations across the entire application is the responsibility of the client.

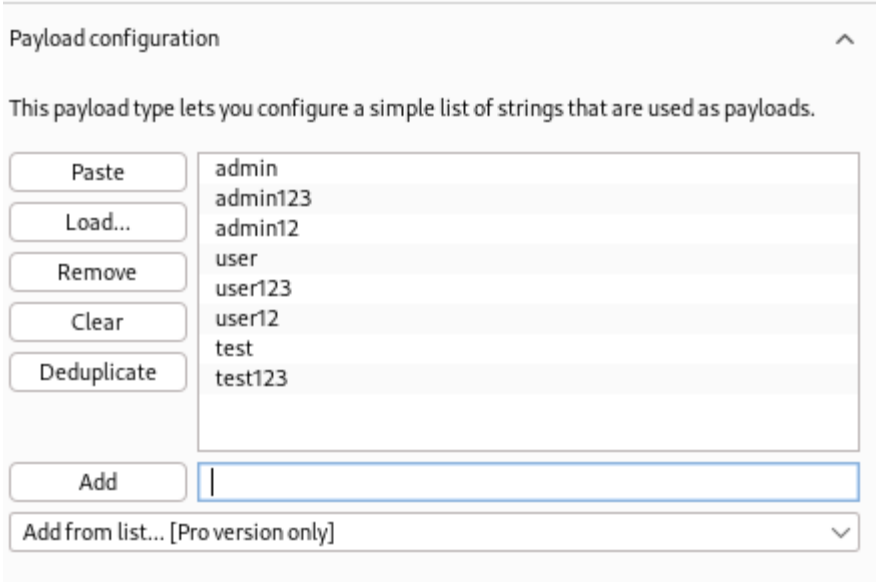
To execute a brute-force attack on the */Auth/Login* endpoint, we begin by capturing an HTTP request to use as a template for automation. (Even with HTTPS in place, attackers can still perform brute-force attacks without needing to intercept the HTTP request. Either manually or with another tool.)

Next we activate our brute-force tool to try every username and password in the list of values we give. This contains variety of different common credentials. Ours is self-made, but attackers can easily find a more robust and sophisticated list online.

The exploitation of this vulnerability is demonstrated in the screenshots below :



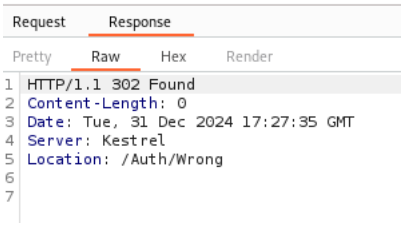
A captured HTTP login request modified for rapid brute-force attacks.



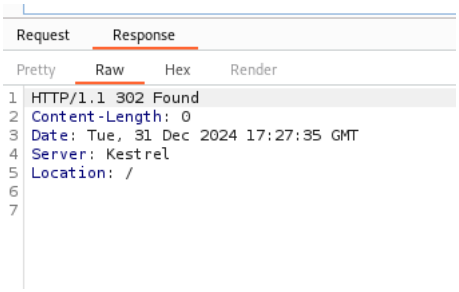
Our self-made payload.

Request	Payload	Status code	Response received	Error	Timeout	Length	Comment
0		302	9			118	
1	admin	302	2			108	
2	admin123	302	21			118	
3	admin12	302	2			118	
4	user	302	2			108	
5	user123	302	20			118	
6	user12	302	2			118	
7	test	302	3			108	
8	test123	302	16			118	

A normal wrong authentication HTTP response has the following content :



Each successful HTTP login response has the following content :



Instead of referring us to the /Auth/Wrong page, it refers to Home, meaning an successful login.

Classification

CWE-307: Improper Restriction of Excessive Authentication Attempts

Found testing

WSTG-ATHN-03 (4.4.3) : Testing for Weak Lock Out Mechanism
WSTG-ATHN-07 (4.4.7) : Testing for Weak Authentication Methods

3.5 Denial of Service by Database Connection Pool Exhaustion

Medium

Category : A05:2021-Security Misconfiguration

Score : CVSS:3.0/AV:L/AC:L/PR:L/UI:N/S:U/C:L/I:L/A:H

Description

To manage and establish multiple database connections simultaneously, a database connection pool is used. It acts as a cache where pre-established, reusable connections are stored, allowing the application to quickly access them when needed. When properly configured, if all connections in the pool are in use, the application will wait for an available connection without overwhelming the server. However, if the connection pool is not properly configured, connection pool exhaustion can prevent connections from being reused, making the database-accessing parts of the application unusable, and potentially disrupting the entire system.

By running a large-scale SQL injection attack using tools like sqlmap or manually performing numerous SQL injection attempts simultaneously, a malicious actor can exhaust the database connection pool. This causes the application to run out of available database connections, effectively leading to a Denial of Service (DoS) and rendering the application unusable.

On the ShopMore application, targeting known database features, like the login page, with a high volume of SQL injection payloads can consume all available connections in the pool. Resulting in a DoS attack and making the application unusable.

Risk

Causing a Denial of Service (DoS) attack through connection pool exhaustion can lead to the complete or partial unavailability of the application. When all available database connections are consumed, legitimate users are unable to access critical features, causing widespread disruption. This can affect everything from logging in and browsing the site to performing transactions or accessing user accounts. As a result, the application becomes unresponsive, significantly impacting its functionality. Prolonged downtime can lead to significant financial loss and reputational damage.

Recommendation

This issue can be mitigated by implementing an account lockout mechanism, which helps prevent brute-force attacks by significantly slowing down repeated login attempts after a set number of failed attempts. Additionally, whitelisting should be used to block any requests that contain patterns indicative of SQL injection, stopping malicious input before it can reach the system. Moreover, robust user input validation is crucial for safeguarding against future DoS attacks, ensuring that all incoming data is properly sanitized and checked for security risks.

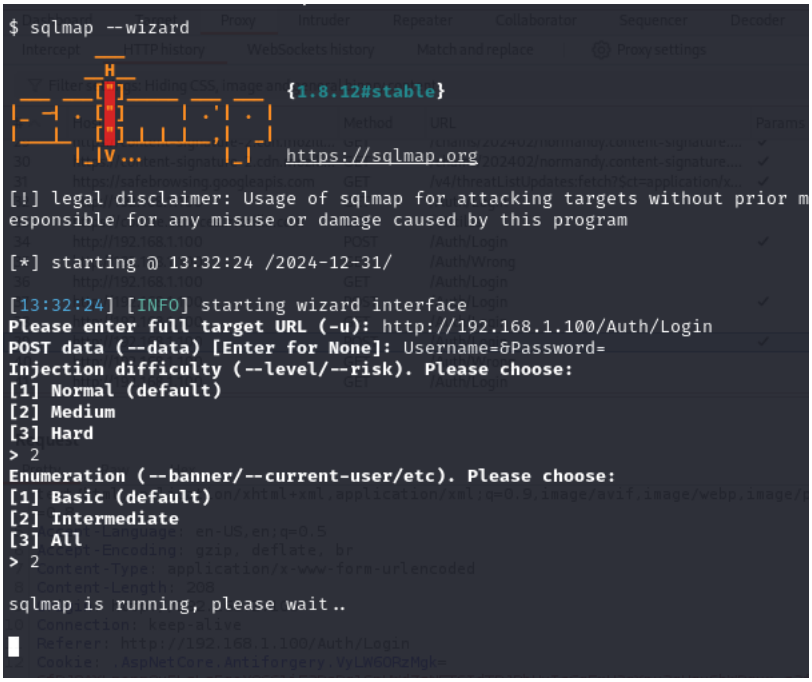
Proof

Remark: This is only one instance of this finding. Investigating and mitigating all instances and variations across the entire application is the responsibility of the client.

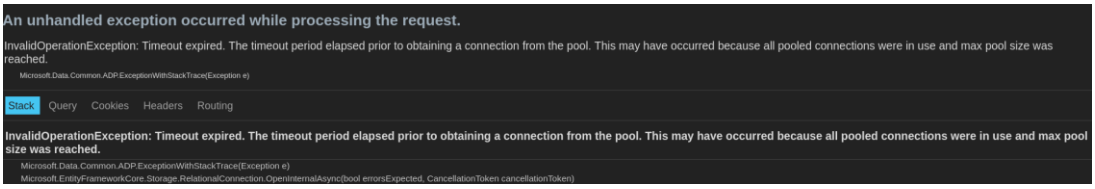
By using the sqlmap tool on the */Auth/Login* page, an attacker can sever all active connections, resulting in a denial of service (DoS) attack.

(Even with HTTPS enabled, the attacker can easily identify the login page endpoint and infer the required parameters for the POST request body (username and password). These parameters don't even need to contain valid values for the attack to be successful. These parameter field don't have to be guessed if info was collected using vulnerabilities from previous findings.)

The exploitation of this vulnerability is demonstrated in the screenshots below :



If you now browse to any page on the application involving database connections, the following shows :



This error renders all database-related pages permanently unusable, clearly demonstrating that the DoS attack was successful.

Classification

CWE-400: Uncontrolled Resource Consumption
CWE-770: Allocation of Resources Without Limits or Throttling

Found testing

WSTG-INPV-05 (4.7.5) : Testing for SQL injection
WSTG-ATHN-07 (4.4.7) : Testing for Weak Authentication Methods

3.6 Information Disclosure via active Debug page

Low

Category : A02:2021-Cryptographic Failures

Score : CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N

Description

Information disclosure occurs when sensitive, private, or confidential information is unintentionally or unauthorizedly exposed to individuals who should not have access to it. This often happens when vulnerabilities are exploited, allowing attackers to access details that are not meant for regular users.

For instance, an attacker might trigger errors that reveal hidden information about the system. This data, such as server configurations, database details, or debugging information, can serve as a springboard for further malicious activities.

Throughout the application, there are various points where errors or improper handling results in the application displaying a full debug page, explaining in detail the error and revealing hidden information about the webserver and database in use and its content. Malicious actors can use this sensitive and critical info for more targeted attacks.

Risk

The risk of exposing hidden information is that it reveals critical system details, such as server configurations, database information, cookies, and session IDs in use. This sensitive information, when accessed by a malicious actor, can enable them to create and execute more targeted attacks tailored to the exposed system.

Revealing software versions or frameworks in use can highlight unpatched vulnerabilities, opening the door to exploitation of critical weaknesses.

Additionally, exposing the existence of the MySQL database in use, combined with revealing some of its database content, allows malicious actors to plan specific attacks such as Denial of Service (DoS) attacks, SQL injection attacks, or other database-specific exploits to manipulate, extract or destroy data and/or services.

Recommendation

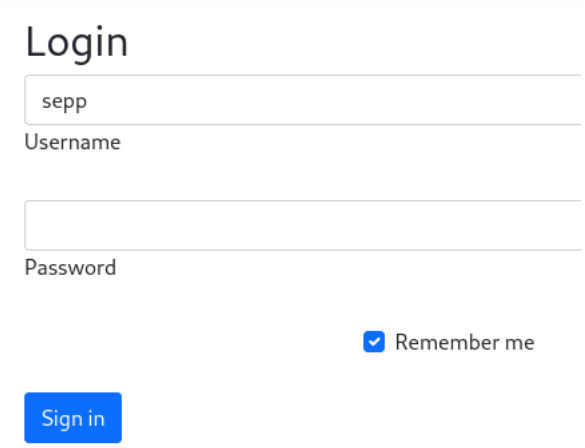
To mitigate information disclosure begins with disabling the debug page to prevent the exposure of critical information to unauthorized users. Secure error handling is essential—replacing detailed error messages with generic, user-friendly messages ensures sensitive information is not inadvertently revealed. Implementing detailed server-side logging allows developers to access the necessary diagnostic detailed information without exposing it to users.

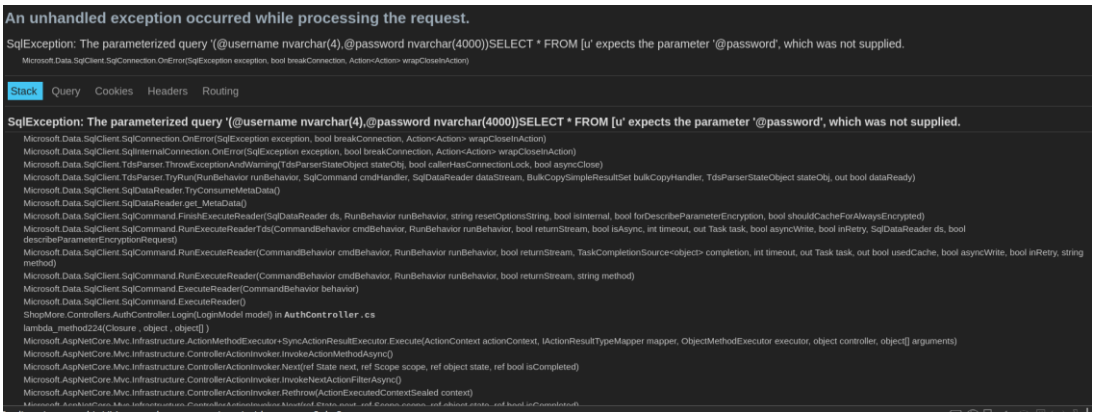
Additionally, removing unnecessary information from HTTP headers, API responses, and other application outputs significantly reduces the attack surface. Finally, enforcing robust user input validation prevents attackers from causing errors that could lead to sensitive information being disclosed.

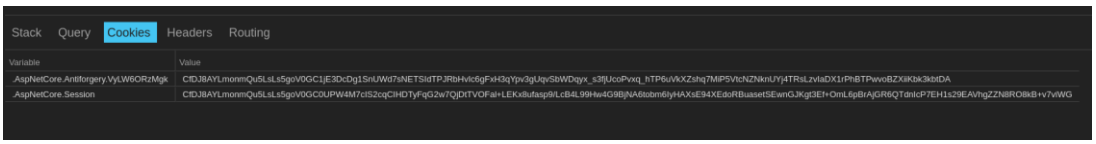
Proof

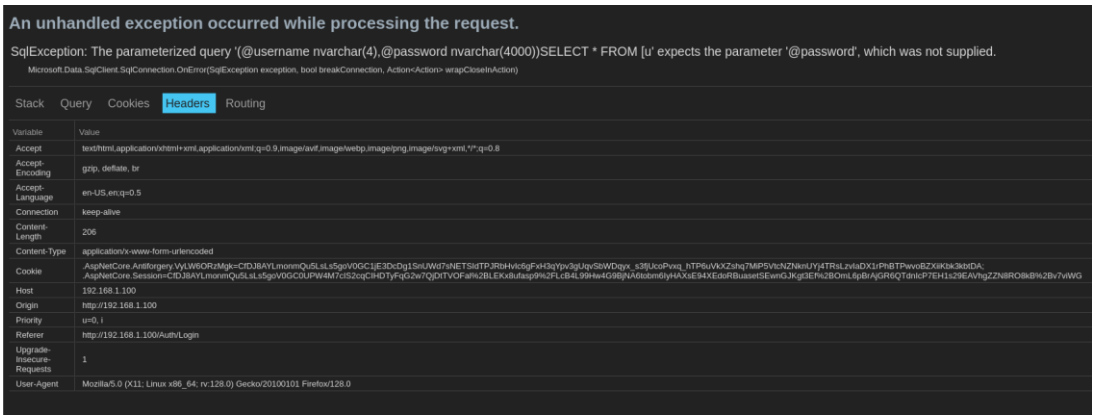
Remark: This is only one instance of this finding. Investigating and mitigating all instances and variations across the entire application is the responsibility of the client.

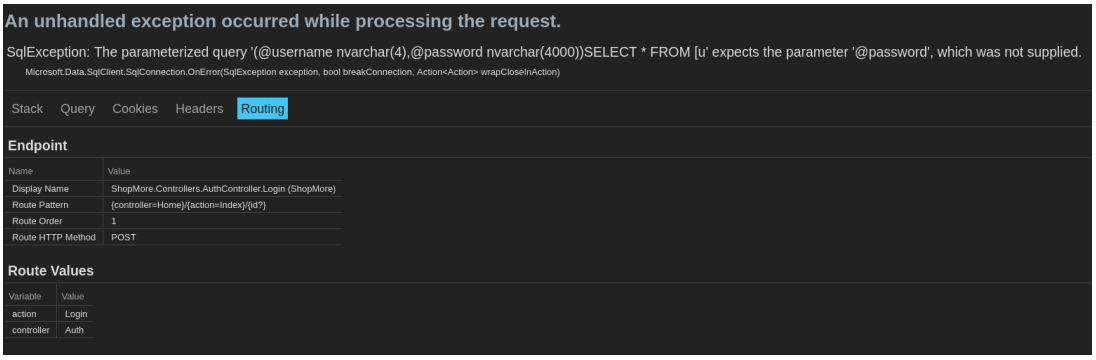
Simply by attempting to login at */Auth/Login* with either the username, password or both fields blank triggers an error, resulting in an error displaying an debug page. Demonstrated in the screenshots below :











Creating different kinds of errors will result in different and more system and database info being exposed.

Classification

[CWE-489 : Active Debug Code]
[CWE-209: Information Exposure Through an Error Message]

Found testing

[WSTG-ERRH-01 (4.8.1) : Testing for Improper Error Handling]
[WSTG-INFO-04 (4.1.4) : Enumerate Applications on Webserver]

3.7 Authentication failure via Browser Cache Weakness

Low

Category : A07:2021 - Identification and Authentication Failures

Score : CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:L

Description

Aside from the login page, where proper browser cache security is implemented, all other pages store browser state in the cache without adequate protection. The browser caches certain data to improve speed and reduce the server's workload, avoiding the need to retrieve the same information on every refresh. However, this cached data can include sensitive information such as session data, form inputs, and cookies. Well-configured applications prevent caching of pages containing sensitive data, but when not properly configured, this can be exploited by malicious actors to retrieve a victim's session state and gain unauthorized access to their account.

This vulnerability enables attackers to use the browser’s back button to view a previous user’s order page or even place new orders in their name, even after the user has logged out. The issue is only dangerous if session IDs are fixed or intercepted, or if the application is accessed from a public device (such as in a library or internet café). In these cases, unauthorized individuals can exploit cached session data to impersonate users and perform actions like placing orders on their behalf.

In the ShopMore application, any individual can use the browser's back button to access another customer's session state, allowing attackers to submit orders on behalf of other users.

Risk

This vulnerability enables unauthorized individuals to exploit cached session data to impersonate users and perform actions on their behalf, such as placing orders. The risk is only dangerous if session IDs are fixed, intercepted, or if the application is accessed from a public device (e.g., in a library or internet café).

This vulnerability could result in financial losses, unauthorized purchases, and breaches of user privacy, ultimately undermining the trust and security of the application.

Additionally, this weakness can serve as a gateway for other attacks, such as cache poisoning, CSRF, and other similar vulnerabilities.

Recommendation

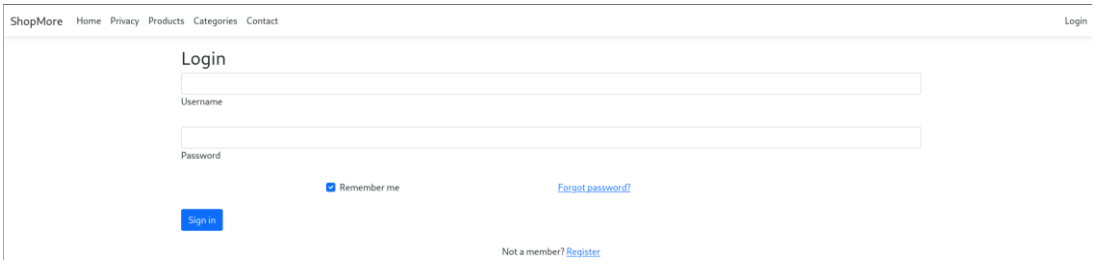
The browser cache vulnerability can be mitigated by implementing proper Cache-Control headers on pages containing sensitive information or restricted access. These headers ensure that such pages do not store browser state in the cache upon refresh, effectively preventing the issue. When combined with HTTPS implementation, this provides a robust solution to secure sensitive data and mitigate the risk of unauthorized access.

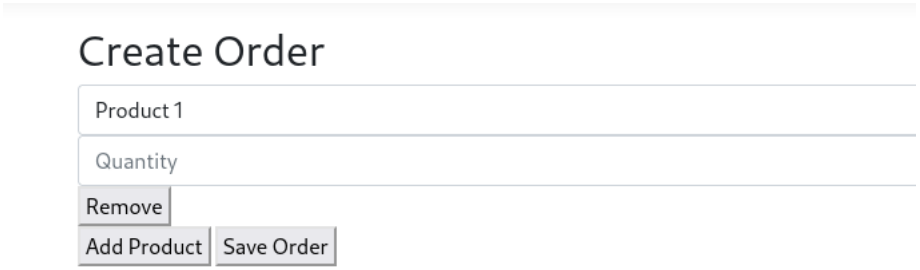
Proof

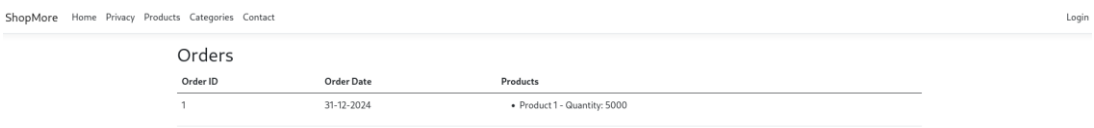
Remark: This is only one instance of this finding. Investigating and mitigating all instances and variations across the entire application is the responsibility of the client.

If the last active user on the device was a normal user, using the back button could potentially allow an unauthorized user to access that user's product order page and place orders on their behalf.

The exploitation of this vulnerability is demonstrated in the screenshots below :







Classification

CWE-524: Use of Cache Containing Sensitive Information

Found testing

WSTG-ATHN-06 (4.4.6) : Testing for Browser Cache Weakness

4 APPENDIX A: TEST CASES

All WSTG tests evaluated during this penetration test. If a specific test was deemed inapplicable or out-of-scope, it is indicated accordingly in the status column.

Test Name	Description	Status
Conduct Search Engine Discovery Reconnaissance for Information Leakage (WSTG-INFO-01)	▪ Identify what sensitive design and configuration information of the application, system, or organization is exposed directly (on the organization's site) or indirectly (via third-party services).	Tested by Sepp Eyckmans
Fingerprint Web Server (WSTG-INFO-02)	▪ Determine the version and type of a running web server to enable further discovery of any known vulnerabilities.	Tested by Sepp Eyckmans
Enumerate Applications on Webserver (WSTG-INFO-04)	▪ Enumerate the applications within the scope that exist on a web server.	Tested by Sepp Eyckmans
Review Web Page Content for Information Leakage (WSTG-INFO-05)	▪ Review web page comments, metadata, and redirect bodies to find any information leakage. ▪ Gather JavaScript files and review the JS code to better understand the application and to find any information leakage. ▪ Identify if source map files or other frontend debug files exist.	Tested by Sepp Eyckmans
Identify Application Entry Points (WSTG-INFO-06)	▪ Identify possible entry and injection points through request and response analysis.	Tested by Sepp Eyckmans
Test Role Definitions (WSTG-IDNT-01)	▪ Identify and document roles used by the application. ▪ Attempt to switch, change, or access another role. ▪ Review the granularity of the roles and the needs behind the permissions given.	Tested by Sepp Eyckmans
Testing for Account Enumeration and Guessable User Account (WSTG-IDNT-04)	▪ Review processes that pertain to user identification (*e.g.* registration, login, etc.). ▪ Enumerate users where possible through response analysis.	Not applicable (placeholder accounts)
Testing for Credentials Transported over an Encrypted Channel (WSTG-ATHN-01)	▪ Review HTTP request for unencrypted user account credentials	Out-Of-Scope (HTTP)
Testing for Default Credentials (WSTG-ATHN-02)	▪ Determine whether the application has any user accounts with default passwords. ▪ Review whether new user accounts are created with weak or predictable passwords.	Not applicable (placeholder accounts)

Testing for Weak Lock Out Mechanism (WSTG-ATHN-03)	▪ Evaluate the account lockout mechanism's ability to mitigate brute force password guessing. ▪ Evaluate the unlock mechanism's resistance to unauthorized account unlocking.	Tested by Sepp Eyckmans
Testing for Bypassing Authentication Schema (WSTG-ATHN-04)	▪ Ensure that authentication is applied across all services that require it.	Tested by Sepp Eyckmans
Testing for Vulnerable Remember Password (WSTG-ATHN-05)	▪ Validate that the generated session is managed securely and do not put the user's credentials in danger.	Tested by Sepp Eyckmans
Testing for Browser Cache Weakness (WSTG-ATHN-06)	▪ Review if the application stores sensitive information on the client-side. ▪ Review if access can occur without authorization.	Tested by Sepp Eyckmans
Testing for Weak Authentication Methods (WSTG-ATHN-07)	▪ Determine the resistance of the application against brute force password guessing using available password dictionaries by evaluating the length, complexity, reuse, and aging requirements of passwords.	Tested by Sepp Eyckmans
Testing for Weak Password Change or Reset Functionalities (WSTG-ATHN-09)	▪ Determine whether the password change and reset functionality allows accounts to be compromised.	Not applicable
Testing for Weaker Authentication in Alternative Channel (WSTG-ATHN-10)	▪ Identify alternative authentication channels. ▪ Assess the security measures used and if any bypasses exists on the alternative channels.	Not applicable
Testing Multi-Factor Authentication (MFA) (WSTG-ATHN-11)	▪ Identify the type of MFA used by the application. ▪ Determine whether the MFA implementation is robust and secure. ▪ Attempt to bypass the MFA.	Not applicable
Testing Directory Traversal File Include (WSTG-AUTHZ-01)	▪ Identify injection points that pertain to path traversal. ▪ Assess bypassing techniques and identify the extent of path traversal.	Tested by Sepp Eyckmans
Testing for Bypassing Authorization Schema (WSTG-ATHZ-02)	▪ Assess if horizontal or vertical access is possible.	Tested by Sepp Eyckmans
Testing for Privilege Escalation (WSTG-ATHZ-03)	▪ Identify injection points related to privilege manipulation. ▪ Fuzz or otherwise attempt to bypass security measures.	Tested by Sepp Eyckmans
Testing for Insecure Direct Object References (WSTG-ATHZ-04)	▪ Identify points where object references may occur. ▪ Assess the access control measures and if they're vulnerable to IDOR.	Tested by Sepp Eyckmans
Testing for OAuth Weaknesses (WSTG-ATHZ-05)	▪ Determine if OAuth2 implementation is vulnerable or using a deprecated or custom implementation.	Not applicable

Testing for Session Management Schema (WSTG-SESS-01)	▪ Gather session tokens, for the same user and for different users where possible. ▪ Analyze and ensure that enough randomness exists to stop session forging attacks. ▪ Modify cookies that are not signed and contain information that can be manipulated.	Out-Of-Scope (HTTP)
Testing for Cookies Attributes (WSTG-SESS-02)	▪ Ensure that the proper security configuration is set for cookies.	Out-Of-Scope (HTTP)
Testing for Session Fixation (WSTG-SESS-03)	▪ Analyze the authentication mechanism and its flow. ▪ Force cookies and assess the impact.	Out-Of-Scope (HTTP)
Testing for Session Hijacking (WSTG-SESS-09)	▪ Identify vulnerable session cookies. ▪ Hijack vulnerable cookies and assess the risk level.	Out-Of-Scope (HTTP)
Testing for Concurrent Sessions (WSTG-SESS-11)	▪ Evaluate the application's session management by assessing the handling of multiple active sessions for a single user account.	Not applicable
Testing for Reflected Cross Site Scripting (WSTG-INPV-01)	▪ Identify variables that are reflected in responses. ▪ Assess the input they accept and the encoding that gets applied on return (if any).	Tested by Sepp Eyckmans
Testing for Stored Cross Site Scripting (WSTG-INPV-02)	▪ Identify stored input that is reflected on the client-side. ▪ Assess the input they accept and the encoding that gets applied on return (if any).	Tested by Sepp Eyckmans
Testing for SQL Injection (WSTG-INPV-05)	▪ Identify SQL injection points. ▪ Assess the severity of the injection and the level of access that can be achieved through it.	Tested by Sepp Eyckmans
Testing for XML Injection (WSTG-INPV-07)	▪ Identify XML injection points. ▪ Assess the types of exploits that can be attained and their severities.	Tested by Sepp Eyckmans
Testing for Code Injection (WSTG-INPV-11)	▪ Identify injection points where you can inject code into the application. ▪ Assess the injection severity.	Tested by Sepp Eyckmans
Testing for Command Injection (WSTG-INPV-12)	▪ Identify and assess the command injection points.	Tested by Sepp Eyckmans
Testing for HTTP Incoming Requests (WSTG-INPV-16)	▪ Monitor all incoming and outgoing HTTP requests to the Web Server to inspect any suspicious requests. ▪ Monitor HTTP traffic without changes of end user Browser proxy or client-side application.	Out-Of-Scope (HTTP)
Testing for Host Header Injection (WSTG-INPV-17)	▪ Assess if the Host header is being parsed dynamically in the application. ▪ Bypass security controls that rely on the header.	Out-Of-Scope (HTTP)

Testing for Server-Side Request Forgery (WSTG-INPV-19)	▪ Identify SSRF injection points. ▪ Test if the injection points are exploitable. ▪ Asses the severity of the vulnerability.	Tested by Sepp Eyckmans
Testing for Improper Error Handling (WSTG-ERRH-01)	▪ Identify existing error output. ▪ Analyze the different output returned.	Tested by Sepp Eyckmans
Testing for Weak Transport Layer Security (WSTG-CRYP-01)	▪ Validate the service configuration. ▪ Review the digital certificate's cryptographic strength and validity. ▪ Ensure that the TLS security is not bypassable and is properly implemented across the application.	Out-Of-Scope (HTTP)
Testing for Sensitive Information Sent via Unencrypted Channels (WSTG-CRYP-03)	▪ Identify sensitive information transmitted through the various channels. ▪ Assess the privacy and security of the channels used.	Out-Of-Scope (HTTP)
Testing for Weak Encryption (WSTG-CRYP-04)	▪ Provide a guideline for the identification weak encryption or hashing uses and implementations.	Tested by Sepp Eyckmans
Test Business Logic Data Validation (WSTG-BUSL-01)	▪ Identify data injection points. ▪ Validate that all checks are occurring on the backend and can't be bypassed. ▪ Attempt to break the format of the expected data and analyze how the application is handling it.	Not applicable
Test Ability to Forge Requests (WSTG-BUSL-02)	▪ Review the project documentation looking for guessable, predictable, or hidden functionality of fields. ▪ Insert logically valid data in order to bypass normal business logic workflow.	Not applicable

Test Integrity Checks (WSTG-BUSL-03)	▪ Review the project documentation for components of the system that move, store, or handle data. ▪ Determine what type of data is logically acceptable by the component and what types the system should guard against. ▪ Determine who should be allowed to modify or read that data in each component. ▪ Attempt to insert, update, or delete data values used by each component that should not be allowed per the business logic workflow.	Not applicable
Test Number of Times a Function Can Be Used Limits (WSTG-BUSL-05)	▪ Identify functions that must set limits to the times they can be called. ▪ Assess if there is a logical limit set on the functions and if it is properly validated.	Not applicable
Testing for the Circumvention of Work Flows (WSTG-BUSL-06)	▪ Review the project documentation for methods to skip or go through steps in the application process in a different order from the intended business logic flow. ▪ Develop a misuse case and try to circumvent every logic flow identified.	Not applicable
Test Defenses Against Application Misuse (WSTG-BUSL-07)	▪ Generate notes from all tests conducted against the system. ▪ Review which tests had a different functionality based on aggressive input. ▪ Understand the defenses in place and verify if they are enough to protect the system against bypassing techniques.	Not applicable
Test Upload of Unexpected File Types (WSTG-BUSL-08)	▪ Review the project documentation for file types that are rejected by the system. ▪ Verify that the unwelcomed file types are rejected and handled safely. ▪ Verify that file batch uploads are secure and do not allow any bypass against the set security measures.	Not applicable
Test Upload of Malicious Files (WSTG-BUSL-09)	▪ Identify the file upload functionality. ▪ Review the project documentation to identify what file types are considered acceptable, and what types would be considered dangerous or malicious. ▪ If documentation is not available then consider what would be appropriate based on the purpose of the application. ▪ Determine how the uploaded files are processed. ▪ Obtain or create a set of malicious files for testing. ▪ Try to upload the malicious files to the application and determine whether it is accepted and processed.	Not applicable

Test Payment Functionality (WSTG-BUSL-10)	▪ Determine whether the business logic for the e-commerce functionality is robust. ▪ Understand how the payment functionality works. ▪ Determine whether the payment functionality is secure.	Not applicable
Testing for DOM-Based Cross Site Scripting (WSTG-CLNT-01)	▪ Identify DOM sinks. ▪ Build payloads that pertain to every sink type.	Tested by Sepp Eyckmans
Testing for JavaScript Execution (WSTG-CLNT-02)	▪ Identify sinks and possible JavaScript injection points.	Not applicable
Testing for HTML Injection (WSTG-CLNT-03)	▪ Identify HTML injection points and assess the severity of the injected content.	Tested by Sepp Eyckmans
Testing for Client-side URL Redirect (WSTG-CLNT-04)	▪ Identify injection points that handle URLs or paths. ▪ Assess the locations that the system could redirect to.	Tested by Sepp Eyckmans
Testing for CSS Injection (WSTG-CLNT-05)	▪ Identify CSS injection points. ▪ Assess the impact of the injection.	Tested by Sepp Eyckmans
Testing for Client-side Resource Manipulation (WSTG-CLNT-06)	▪ Identify sinks with weak input validation. ▪ Assess the impact of the resource manipulation.	Tested by Sepp Eyckmans
Testing Cross Origin Resource Sharing (WSTG-CLNT-07)	▪ Identify endpoints that implement CORS. ▪ Ensure that the CORS configuration is secure or harmless.	Not applicable
Testing for Clickjacking (WSTG-CLNT-09)	▪ Assess application vulnerability to clickjacking attacks.	Out-Of-Scope
Testing WebSockets (WSTG-CLNT-10)	▪ Identify the usage of WebSockets. ▪ Assess its implementation by using the same tests on normal HTTP channels.	Not applicable
Testing Web Messaging (WSTG-CLNT-11)	▪ Assess the security of the message's origin. ▪ Validate that it's using safe methods and validating its input.	Not applicable
Testing Browser Storage (WSTG-CLNT-12)	▪ Determine whether the website is storing sensitive data in client-side storage. ▪ The code handling of the storage objects should be examined for possibilities of injection attacks, such as utilizing unvalidated input or vulnerable libraries.	Out-Of-Scope (HTTP)
Testing for Cross Site Script Inclusion (WSTG-CLNT-13)	▪ Locate sensitive data across the system. ▪ Assess the leakage of sensitive data through various techniques.	Not applicable
Test Network Infrastructure Configuration (WSTG-CONF-01)	▪ Review the applications' configurations set across the network and validate that they are not vulnerable. ▪ Validate that used frameworks and systems are secure and not susceptible to known vulnerabilities due to unmaintained software or default settings and credentials.	Out-Of-Scope (No Direct Access)

Test Application Platform Configuration (WSTG-CONF-02)	▪ Ensure that default and known files have been removed. ▪ Validate that no debugging code or extensions are left in the production environments. ▪ Review the logging mechanisms set in place for the application.	Out-Of-Scope (No Direct Access)
Enumerate Infrastructure and Application Admin Interfaces (WSTG-CONF-05)	▪ Identify hidden administrator interfaces and functionality.	Tested by Sepp Eyckmans
Test HTTP Methods (WSTG-CONF-06)	▪ Enumerate supported HTTP methods. ▪ Test for access control bypass. ▪ Test HTTP method overriding techniques.	Out-Of-Scope (HTTP)

5 APPENDIX B : METHODOLOGY

Our penetration testing methodology is meticulously designed to evaluate the security of systems, applications, and networks. This comprehensive approach ensures the thorough identification of potential vulnerabilities, enabling organizations to effectively mitigate risks and fortify their overall security framework.

While our testing strives to uncover significant vulnerabilities, the inherent complexity of modern IT systems and the constraints of time make it impractical—and unnecessary—to identify every potential weakness. Instead, our focus is on identifying and addressing the most critical risks that pose a realistic threat to the organization's security.

We apply the following approach to each security assessment :

Each penetration test is conducted in clearly defined stages, ensuring a structured and methodical approach to uncovering vulnerabilities. By breaking the process into distinct phases, we can maintain focus, adapt strategies as needed, and deliver comprehensive results tailored to the unique security needs of the client.

Kick-off & Preparations

During the initial Kick-off meeting, attended by all stakeholders to discuss and confirm the following :

- Scope and access credentials
- Objective
- Rules of engagement
- Reporting structure
- Time Scope
- Pen testing environment (recommended : non-production environments)

By validating both technical and non-technical specifications, we ensure that all stakeholders, including technical and business teams, have a clear and accurate understanding of the objectives and scope of the project.

Testing

Testing begins based on the agreed upon scope established during the kick-off meeting.

Our approach combines automated and manual testing methods to ensure comprehensive coverage. Each automated test is carefully reviewed and validated manually to maintain the highest standards of quality and accuracy.

Reporting

Upon completing the testing phase, all findings and observations are compiled into a detailed report, which consistently follows this structure:

- Overview & Scope
- Executive Summary
- List of Findings

Any additions to an report can be discussed during the Kick-off meeting and implemented to the final report.

Review

At the conclusion of our security assessment, we conduct a review meeting with the client and all relevant stakeholders to discuss the findings and recommendations in detail. This provides an opportunity to address any questions or concerns the client may have before ending the project.

6 APPENDIX C : SEVERITY CLASSIFICATION

To evaluate the severity of a vulnerability during this penetration test, we will use the Common Vulnerability Scoring System (CVSS) framework. The CVSS framework is a standardized and widely recognized method for evaluating the potential impact and risk associated with security vulnerabilities. By utilizing CVSS, we can assign a numerical score to each vulnerability based on a comprehensive range of factors, including exploitability, potential impact on confidentiality, integrity, and availability.

The weighting of a vulnerability's score may vary depending on the specific business context of our client. This context is reviewed and discussed during the Kick-off and review meeting to determine the risk impact on the client and whether the severity should be weighted more heavily or less significantly based on that result. This contextual severity will be reflected accordingly in the final scoring and report.

Critical

- Low to no knowledge needed to trigger vulnerability
- High likelihood of exploitation
- Minimal preconditions or user interaction required
- No victim required for exploitation
- Any type of user can trigger the exploit
- Significant impact on system confidentiality, integrity and/or availability
- Mission critical to the business

High

- Low extent of knowledge needed to trigger vulnerability
- Moderate to high likelihood of exploitation
- Minimal preconditions or user interaction required
- No victim required for exploitation
- Any type of user can trigger the exploit
- High impact on the business

Medium

- Moderate to low extent of knowledge needed to trigger vulnerability
- Moderate likelihood of exploitation due to skill level needed
- Moderate preconditions or user interaction required
- Little or no impact to business

Low

- Moderate to high extent of knowledge needed to trigger vulnerability
- Low to moderate likelihood of exploitation due to skill level needed
- Little or no impact to business
- Needs to be used in combination with other vulnerabilities

Info

When an finding is in the info category, it is either :

- Extremely unlikely to occur.
- Unable to assign a score due to insufficient information during testing to provide an accurate assessment.
- No impact on confidentiality, integrity, or availability.

7 CONFIDENTIALITY & LIABILITY

The information in this penetration test report is confidential and intended exclusively for the client's use. It contains sensitive and proprietary details about the security state of the client's web application, network infrastructure, IT environment, and associated systems or services. Due to the sensitive nature of this data, the contents of this report must not be shared, disclosed, or disseminated to any unauthorized parties without the client's explicit written consent. Unauthorized disclosure or distribution of this information could lead to significant security risks and legal consequences.

By receiving and reviewing this report, the client acknowledges that the information provided is intended to aid in improving the client's security measures and does not constitute an endorsement of any particular technology or solution. The client is responsible for safeguarding the contents of this report and ensuring it is only accessible to those with a legitimate need to know.

Additionally, while every effort has been made to provide an accurate and thorough assessment, the testing performed in this report is based on the specific scope defined by the client and may not cover all potential vulnerabilities within the environment. Möbius Security does not assume any liability for any damages, losses, or breaches that may occur as a result of the vulnerabilities identified or not identified in this report, or for the client's failure to implement recommended security measures.

REFERENCES

More, T. (n.d.). *Reporting Vulnerabilities* . Retrieved from Reporting Vulnerabilities Document: https://docs.google.com/document/d/1zvcZteRfxKOom6QhhxaDrY_alT_age6_F2zt6iMc3Do/edit?usp=sharing

Toreon. (n.d.). *Penetration Test Report Example*. Retrieved from Toreon Penetration Test Report Example: https://drive.google.com/file/d/1BCPiVkC7qe2nLgaXiDzAyvwfLLDjIUxQ/view?usp=sharing