Penetration Test

Badstore.net

Report Ref: [B4D5t0r3]

6th March 2024

Huzaifah Machher

3234 total words, 2181 contained in findings

Contents

Disclosures and contact details 3

Summaries and overviews 4

- Overview 4
- Key findings table 4
- Executive summary 5
- Technical summary 6
- Risk key 8
- Tools used 9

Introduction 9

- Purpose 9
- In scope 9
- Out of scope 9

Findings 10

- SQL injection (a) 10
- Cross site scripting 12
- Source code manipulation 14
- DoS attack susceptibility 18
- Discovery of hidden directories 21
- SQL injection (b) 24
- Improper password resetting functionality 26
- Cookie manipulation 28

Appendix 31

- In scope vulnerabilities 31
- Out of scope vulnerabilities 31

Disclosure

The contents of this report are strictly confidential and only for the use of Badstore.net and those involved in the penetration testing.

Sharing

Permission for distribution of this information is granted to Badstore.net for the purpose of collaboration within the business or for the attention of regulation authorities.

Customer contact

Company: Badstore.net
URL: 172.16.119.130

Penetration tester contact

Company name: Pentesters
Name: Huzaifah Machher
Title: Cyber security student
Email: P2507563@my365.dmu.ac.uk
Business address: DMU
URL: DMU.ac.uk

1 Summaries and overviews

1.1 Overview

Pentesters were commissioned to perform a penetration test on the web application hosting Badstore.net. The tests were conducted over the course of a month and findings were submitted by 18th of April.

Findings of which have been collated in this report with the aim to effectively convey the results of the testing including remediation techniques both at technical and executive level. The report also aims to outline in depth the procedures used, findings from the testing, remediation for any issues found whilst quantifying impact and risk to the business.

1.2 Key findings

Finding
Out of date software
SQL injection (a, b)
Cross site scripting (XSS)
Source code manipulation
Discoverable hidden directories
HTTP request manipulation
Low encryption
Susceptibility to DoS attacks
Improper password resetting functionality
Lack of input validation and privilege segregation
Susceptibility to dictionary attacks

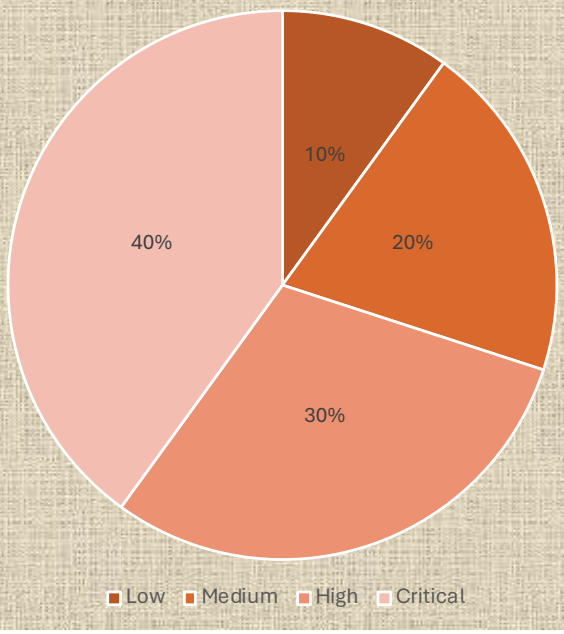
1.3 Executive summary

Vulnerabiltiies present within the site impact the safety and longevity of Badstore.net.

Vulnerabilities range from having the ability to freeze the webstie, to stealing all user information at both customer, supplier, and administrator level.

Severity	Number present
Low	1
Medium	2
High	3
Critical	4

VULNERABILITY COUNT



VULNERABILITIES FOUND

All accounts compromised

3 pages vulnerable through user input

It is possible to manipulate the web applications features

1 vulnerability found to freeze site

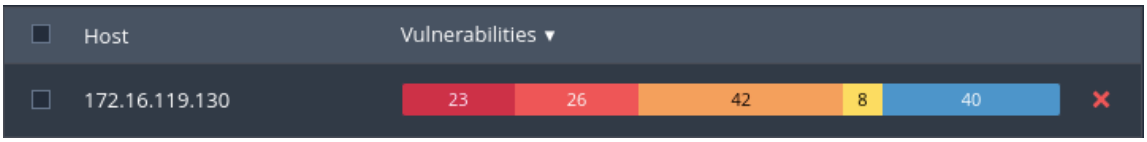
LOADING

It is possible to stop other users from accessing the site

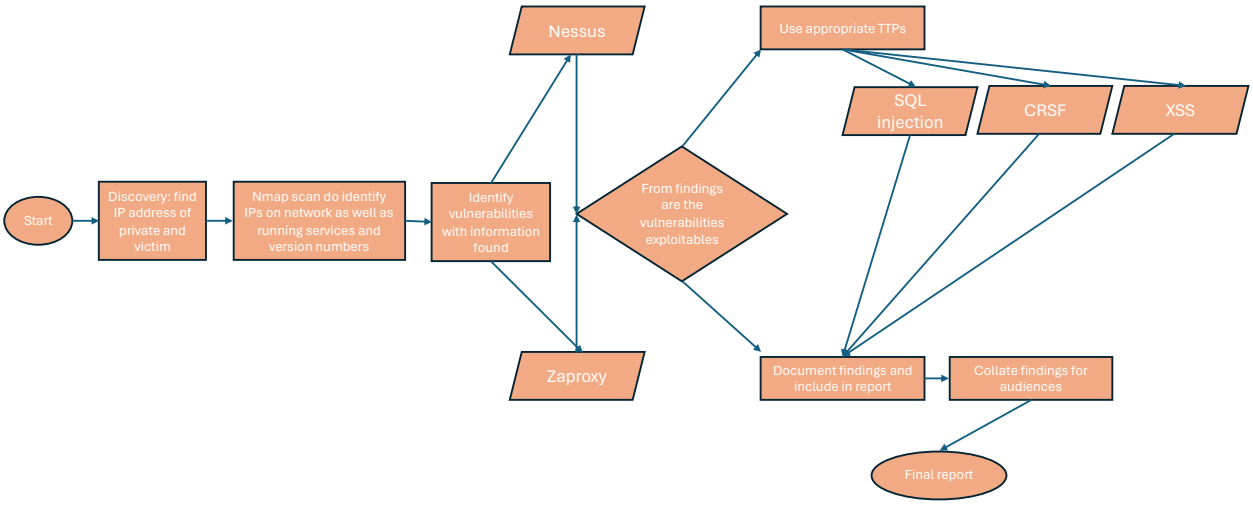
25 OWASP Identified vulnerabilities found

1.4 Technical summary overview

Whilst there are a range of vulnerabilities present within the web application many of the findings that are within scope trace back to a lack of input validation and sanitisation. Implementing these fixes will promote the continued operation of Badstore.net. The tool 'Nessus' outlines 23 critical, 26 high, 42 medium, 8 low risks when completing a basic network scan, whilst also making a further remark on informational data. Key findings, with their associated risk ratings, loss to business, likelihood, and general remediation techniques address some of the more prominent issues.



1.5 Attack tree



1.6 Technical summary findings

Description of risk	Threat rating	Corporate loss	Likelihood	Recommendations
SQL injection Bad actors can manipulate the MySQL server beneath the site to gain access to sensitive information. With this the bad actor can also obtain information on the server itself e.g., version, name, databases.		As a result of successful injection all user accounts are compromised, leading to a huge leak of personal information against GDPR guidelines.	Once the version of SQL is identified SQL injection requires little effort and minimal technical skill, meaning it is very likely.	Parametrised queries, ignoring special characters used by SQL. Proper sanitization of user input. i.e., Removing ' or – .
Cross site scripting (XSS) Bad actors can inject malicious script into web pages which allow for the theft of session IDs, sensitive information and control of victim hardware.		With the required security misconfigurations many personal details may be stolen, as well as facilitating for further attacks on user hardware.	Script to carry out such attacks are readily available on the internet; it does not require much skill and is highly probable.	Input sanitisation is a key deterrent, unsafe HTML tags should be removed, and special characters should be escaped before processing.
DoS attack susceptibility An anomalous number of packets can be forward to the server causing the site to freeze, as connections remain open the server has no free resources for new request.		As users will not be able to navigate or even reach the site, trading will come to a halt causing financial loss.	DoS and DDoS attacks can be carried out with one command of the terminal, making them relatively easy to do.	IDS can detect and filter abnormal traffic through the network whilst Web Application Firewalls (WAF) can block traffic before it reaches the server. Both can achieve this by detecting DoS patterns.
Lack of input validation Users and bad actors can both log in to the site without proper credentials, meaning unauthorised entry to privileged pages.		Personal details may be swiped after entry. Bad actors may also pretend to be legitimate users causing confusion and mistrust between Badstore and user.	Ideally a user will enter a username and password, but the bad actors looking for entry points will use this as an easy way to exploit the site, making it likely.	The site must check for valid input, e.g. asking users to meet a certain criteria of password before allowing the user to proceed. i.e. minimum length and special characters.
Source code manipulation and low encryption Information within the source code of the web application can be manipulated for the gain of a bad actor, which includes price alteration and privilege escalations.		With administrator access all accounts are compromised, prices can be changed which will result in a loss of revenue.	A bad actor with the right skill set can take advantage of this, being so easy to do it his very possible but not common.	The source code must check for that the role is set to U before accepting the request to the database server, cookies must be better encrypted.
Discoverable hidden directories and HTTP request manipulation The robots.txt file reveals information which includes weakly encrypted passwords, as well as other hidden directories.		Once discovered a bad actor can gain access to a few different accounts, which will ultimately lead to a loss in revenue and trust.	Being a common vulnerability it is highly likely this can be discovered with little hacking knowledge.	Indexing of the robots.txt and other hidden directories must be removed, making the discovery of these difficult unless directly targeted.
Improper password resetting functionality and susceptibility to brute force attacks The password reset function does not require any valid details and the new password is displayed in plain text, once the default password "Welcome" is discovered it is easier for a bad actor to brute force after attempting login with known email address.		There is little loss to the business with an issue like this unless the bad actor can gain entry to an admin account.	With little to no knowledge of hacking a procedure like this is in multiple parts, therefore it can be assumed that this would be more difficult to achieve.	The source code must check for valid input such as a valid email address and new passwords should be encrypted, with aims to be sent directly to the one requesting the reset as opposed to displaying it publicly.

1.7 Risk key

The metrics used are decided based on the impact to the business and factor in multiple avenues of determent, the score system is inclusive of information gathered from the Common Vulnerability Scoring System

		Impact				
		Ignorable	Low	Medium	High	Catastrophic
Chance	Very high chance					
	High chance					
	Medium chance					
	Little chance					
	Little to no chance					

Colour	Risk	Description	Example
	Critical	The vulnerability grants total control of the web application to bad actors, this includes crippling functionality and catastrophic repercussions.	SQL injection in the email field on the login page will grant administrator privilege.
	High	A vulnerability of this nature will have a significant impact on the functionalities of the web application but not cause catastrophic and irreversible damage.	Source code manipulation allows for a bad actor to gain session IDs and steal personable information, including financial details.
	Medium	The vulnerability will grant the bad actor authority over some parts of the web application however it is will not be in administrator capacity.	The password reset function displays the default password in plain text allowing a bad actor to brute face entry of an account.
	Low	Vulnerabilities as such cannot cause much damage on their own however, used in conjunction with other techniques they may be challenging to navigate.	When monitoring network traffic, details of network activity are in plain text and provide insight to the web applications inner workings.
	Informational	Although not considered dangerous, it is important to note and monitor such occurrences as they make cause issues.	Software versions may provide insight into present vulnerabilities.

1.8 Tools used

Tool	Description
BurpSuite	Used as a proxy for the purpose of interception traffic and manipulating requests made to the server before being sent.
Wireshark	For the purpose of monitoring network traffic between source and destination IPs in order to gather information.
Zaproxy	To identify vulnerabilities across the web application.
hping	For forwarding packets to the destination IP from various spoof IPs.
Nessus	To identify vulnerabilities across the web application.
Dirbuster	To identify hidden directories within the web application.

2. Introduction

Pentesters performed a penetration test against the Badstore.net web application as per the request of Badstore.net and found that there were many issues, details of which are to be shared.

2.1 Purpose

Badstore.net has experienced issues in the past where the finance and security teams have reported mishaps. The penetration test was conducted to find any reasons for why that has been the case.

2.2 In scope

Testing outlined by Badstore.net was limited to the following areas and vectors:

- Port 80
- Port 443
- Any scripts
- All tools excluding SQLmap (see 2.3)

2.3 Out of scope

Badstore.net requests that the tests conducted are not inclusive of:

- Testing on any other ports
- Testing using SQLmap
- Interaction with the files on the web application
- Offline attacks

3. Findings

3.1a SQL injection (a)

Users can manipulate the MySQL server beneath the web application using common SQL injection techniques.

Source	Destination	Tool
172.16.119.128	http://172.16.119.130/cgi-bin/badstore.cgi?action=loginregister	

3.1b Instructions for Replication:

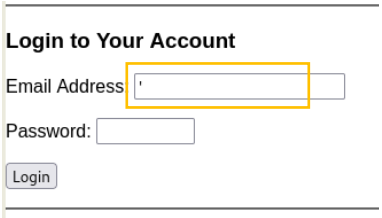
1. **Input Single Quote**: Enter a single quote character (') into the email field on the login page.
2. **Observe Error Message**: Check the response from the web application. Note the syntax error message containing an apparent MySQL query.
3. **Identify Database Server**: Use the information from the error message to determine that the web application is using a MySQL server.
4. **Construct Manipulated Request**: Based on the MySQL server information, create a request that exploits the apparent SQL injection vulnerability.
5. **Submit Manipulated Request**: Send the crafted request to the server.
6. **Access Test Account**: If the manipulation is successful, you should gain access to a test account within the web application.

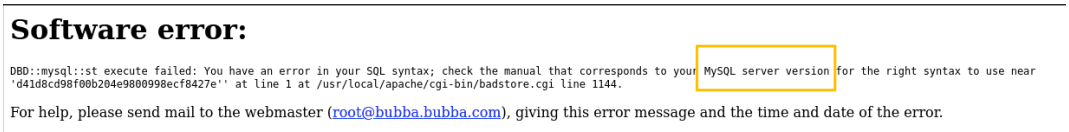
3.1c Code used

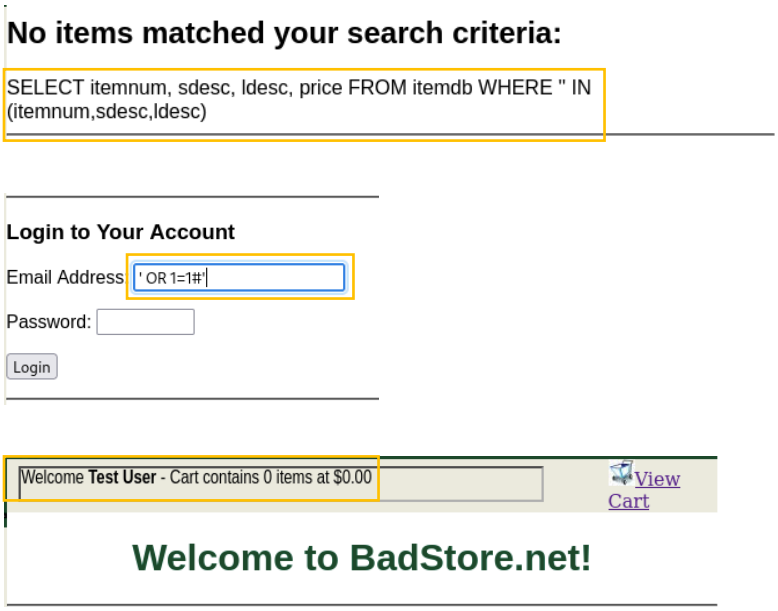
‘ OR 1=1#’

1=1 will always equate to true, by inserting the single quotation the SQL query is being manipulated, # will comment the rest of the request out.

3.1d Screenshots:







3.1e Remediation

Prepared statements can prevent the input of unsafe and malicious SQL injection code, as referenced by OWASP parameterised queries can be built into the source code.

3.1f References

https://www.google.com/url?sa=t&source=web&rct=j&opi=89978449&url=https://community.microfocus.com/cyberres/b/off-by-on-software-security-blog/posts/how-do-prepared-statements-protect-against-sql-injection&ved=2ahUKEwj1_YPygsyFAxVLWkEAHYlMA7IQh-wKegQIUBAC&usg=AOvVaw2NU3NkOwVClz-lehlHrAtp

https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

https://owasp.org/www-project-top-ten/2017/A1_2017-Injection

3.2a Cross Site Scripting (XSS)

According to OWASP 'cross site scripting attacks are another type of injection where malicious scripts are injected into websites that allow for user input', this type of attack is common and can be demonstrated within the guestbook page of the Badstore.net web application. The script is executed on behalf of users unsuspectingly to them which can cause several issues. Theft of personable information, sensitive data and session cookies are amongst some of the dangers of XSS. If there is no input sanitisation the request is carried out as a regular HTML request.

Source	Destination	Tool
172.16.119.128	http://172.16.119.130/cgi-bin/badstore.cgi?action=guestbook	

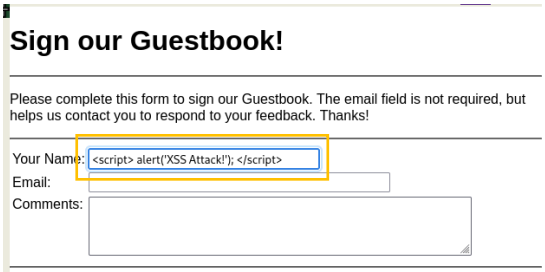
3.2b Instructions for Replication:

1. **Add Malicious XSS Script:** In the guestbook page, under the name field, enter a malicious XSS script.
2. **Submit the Request:** After entering the script in the name field, submit the request as you normally would.
3. **Observe Script Execution:** Once the request is submitted, observe the output on the screen.
4. **Verify Alert Message:** Check for the execution of the malicious script, which should result in a pop-up alert with the message "XSS Attack!".
5. **Demonstrate Exploitability:** This simple example demonstrates how XSS vulnerabilities can be exploited by allowing a script to be executed immediately after submitting the request.

3.2c Code used

```
<script>alert('XSS Attack!');</script>
```

3.2d Screenshots:



3.2e Remediation

OWASP mentions the key to 'perfect injection resistance' is sanitising and validating all user input before a request is submitted, by doing so no code can be injected other than what is allowed by the source code. Special characters and other HTML specific words should be escaped. There are tools which automatically clean up HTML text however it is important that it is checked over before implementation.

3.2f References

https://owasp.org/www-community/attacks/xss/

https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

https://snyk.io/advisor/npm-package/sanitize-html/example

3.3a Source code manipulation

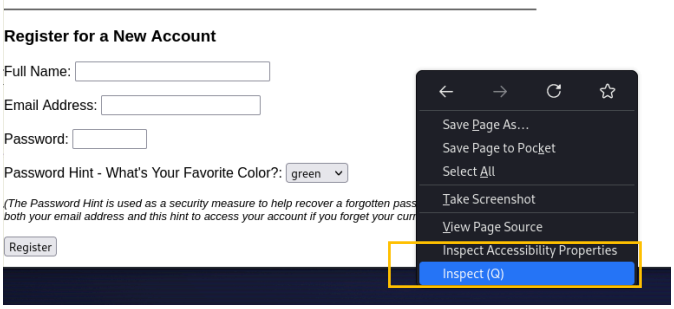
Badstore.net's web application features a critical flaw; privileges can be escalated which then grants users administrator level capabilities. The MITRE ATT&CK framework says, 'Exploitation of a software vulnerability occurs when an adversary takes advantage of a programming error in a program, service, or within the operating system software or kernel itself to execute adversary-controlled code.'

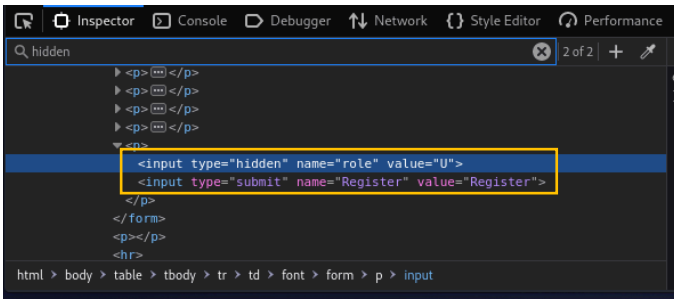
Source	Destination	Tool
172.16.119.128	http://172.16.119.130/cgi-bin/badstore.cgi?action=loginregister	

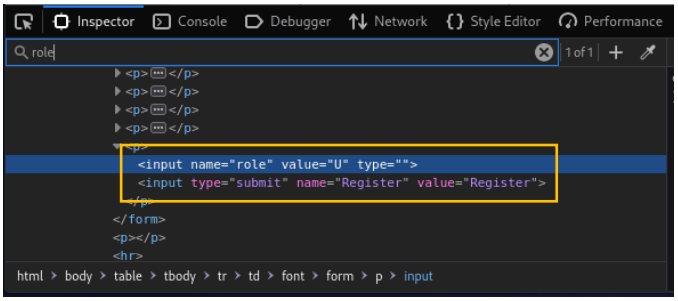
3.3b Instructions for Replication:

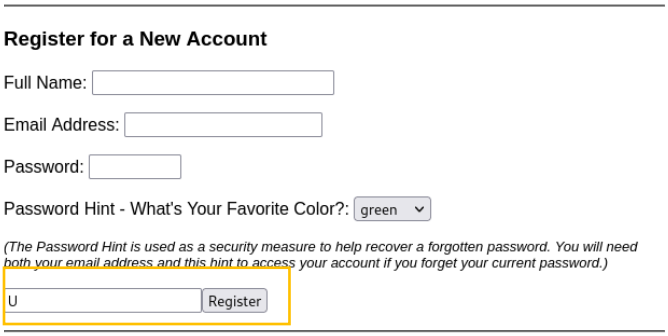
1. **Inspect Registration Page Code**: Open the registration page and inspect the source code behind the interface.
2. **Locate Hidden Role Field**: Within the code, locate the hidden field named **role** that determines user privileges. The field value is set to 'U' for user and 'A' for Administrator.
3. **Expose Hidden Field**: Modify the code to remove the hidden attribute from the **role** field, making it visible on the registration page.
4. **Change User Role**: Once the **role** field is visible, change its value from 'U' (user) to 'A' (Administrator) to grant the new user administrative privileges.
5. **Register New Admin Account**: Complete the registration process with the modified **role** field to create a new administrator account.
6. **Navigate to Secret Admin Menu**: After registering the new account, replace the URL segment **loginregister** with **admin** and navigate to the modified URL.
7. **Access Admin Menu**: Upon navigating to the modified URL, you will be taken to a secret administrator menu.
8. **View Sensitive User Information**: In the admin menu, you can see various details, including other directories and sensitive user information, all of which may be in plain text.
9. **Find Created Accounts**: The accounts created during the penetration test, with administrator access, can be found at the bottom of the user list.
10. **Direct Manipulation of 'U'**: By directly manipulating the 'U' value within the source code, you can achieve the same result of granting administrator privileges.

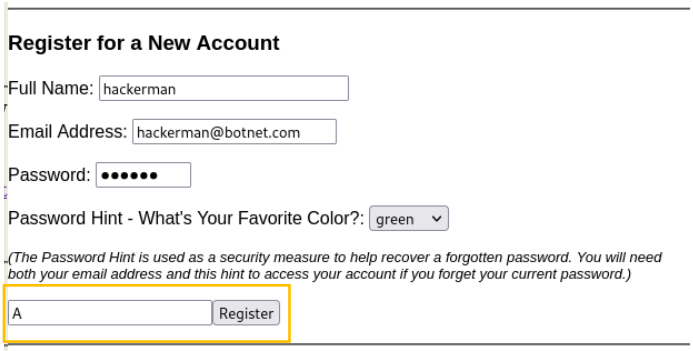
3.3c Screenshots:

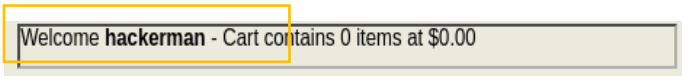












Secret Administration Menu

Where do you want to be taken today?

View Sales Reports | Do It

AAA_Test_User	098F6BCD4621D373CADE4E832627B4F6	black	Test User	U
admin	5EBE2294ECD0E0F08EAB7690D2A6EE69	black	Master System Administrator	A
joe@supplier.com	62072d95acb588c7ee9d6fa0c6c85155	green	Joe Supplier	S
big@spender.com	9726255eec083aa56dc0449a21b33190	blue	Big Spender	U
ray@supplier.com	99b0e8da24e29e4ccb5d7d76e677c2ac	red	Ray Supplier	S
robert@spender.net	e40b34e3380d6d2b238762f0330fbd84	orange	Robert Spender	U
bill@gander.org	5f4dcc3b5aa765d61d8327deb882cf99	purple	Bill Gander	U
steve@badstore.net	8cb554127837a4002338c10a299289fb	red	Steve Owner	U
fred@whole.biz	356c9ee60e9da05301adc3bd96f6b383	yellow	Fred Wholesaler	U
debbie@supplier.com	2fbd38e6c6c4a64ef43fac3f0be7860e	green	Debby Supplier	S
mary@spender.com	7f43c1e438dc11a93d19616549d4b701	blue	Mary Spender	U
sue@spender.com	ea0520bf4d3bd7b9d6ac40c3d63dd500	orange	Sue Spender	U
curt@customer.com	0DF3DBF0EF9B6F1D49E88194D26AE243	green	Curt Wilson	U
paul@supplier.com	EB7D34C06CD6B561557D7EF389CDDA3C	red	Paul Rice	S
kevin@spender.com			Kevin Richards	U
ryan@badstore.net	40C0BBDC4AEEAA39166825F8B477EDB4	purple	Ryan Shorter	A
stefan@supplier.com	8E0FAA8363D8EE4D377574AEE8DD992E	yellow	Stefan Drege	S
landon@whole.biz	29A4F8BFA56D3F970952AFC893355ABC	purple	Landon Scott	U
sam@customer.net	5EBE2294ECD0E0F08EAB7690D2A6EE69	red	Sam Rahman	U
david@customer.org	356779A9A1696714480F57FA3FB66D4C	blue	David Myers	U
john@customer.org	EEE86E9B0FE29B2D63C714B51CE54980	green	John Stiber	U
heinrich@supplier.de	5f4dcc3b5aa765d61d8327deb882cf99	red	Heinrich Hâ¨sÃ¶ber	S
tommy@customer.net	7f43c1e438dc11a93d19616549d4b701	orange	Tom O'Kelley	U
	d41d8cd98f00b204e9800998ecf8427e	purple	fMOlOwZY	U
	d41d8cd98f00b204e9800998ecf8427e	green	viQdVJqK	U
	d41d8cd98f00b204e9800998ecf8427e	green	viQdVJqK	U
	d41d8cd98f00b204e9800998ecf8427e	green		A
hackerman@botnet.com	4d4098d64e163d2726959455d046fd7c	green	hackerman	A
hackerman@botnet.com	4d4098d64e163d2726959455d046fd7c	green	hackerman	A

3.3d Remediation

When a request is made for a new registration, the role should be set to 'U' before it can be submitted, input validation will prevent the escalation of privileges. It must be written as a rule within the source code for this to be effective against such an attack.

3.3e References

https://attack.mitre.org/techniques/T1499/

3.4a DoS attack susceptibility

A denial-of-service attack renders the site completely redundant, by flooding the server with TCP requests from spoof IPs through port 80, all the server's resources are used to respond to the requests, because of this it is unable to do anything else. An attack like this without the right mitigation techniques will cause the site to become inaccessible, driving down traffic and potential sales in turn.

Source	Destination	Tool
172.16.119.128	http://172.16.119.130	hping3, wireshark

3.4b Instructions for Replication:

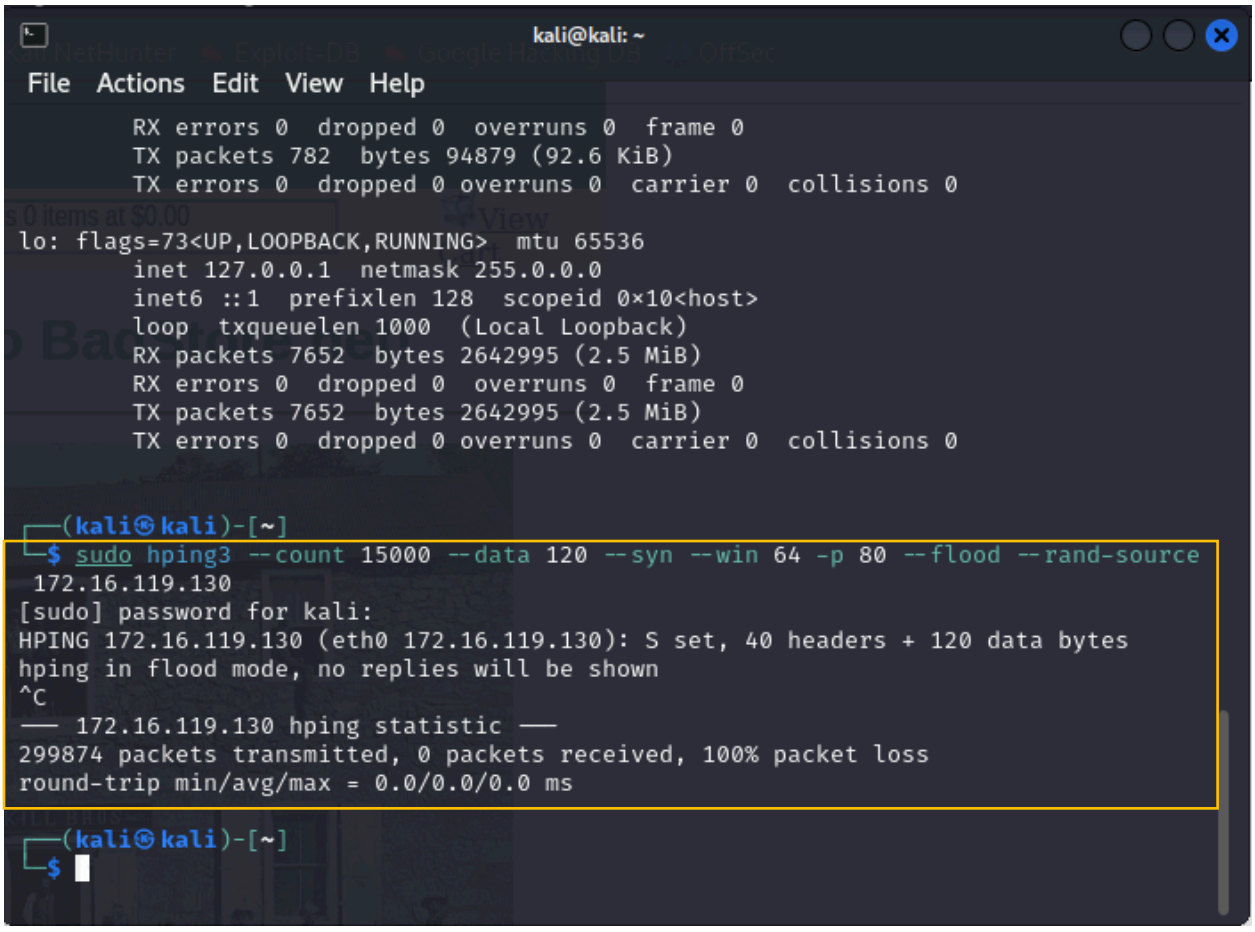
1. **Open terminal:** Use terminal to use the hping command against the badstore server
2. **Set flags:** use appropriate flags to construct the DoS attack
3. **Observe effects:** Using a network monitoring tool observe all outbound packets to the server
4. **Test web application:** Attempt navigation of the web application

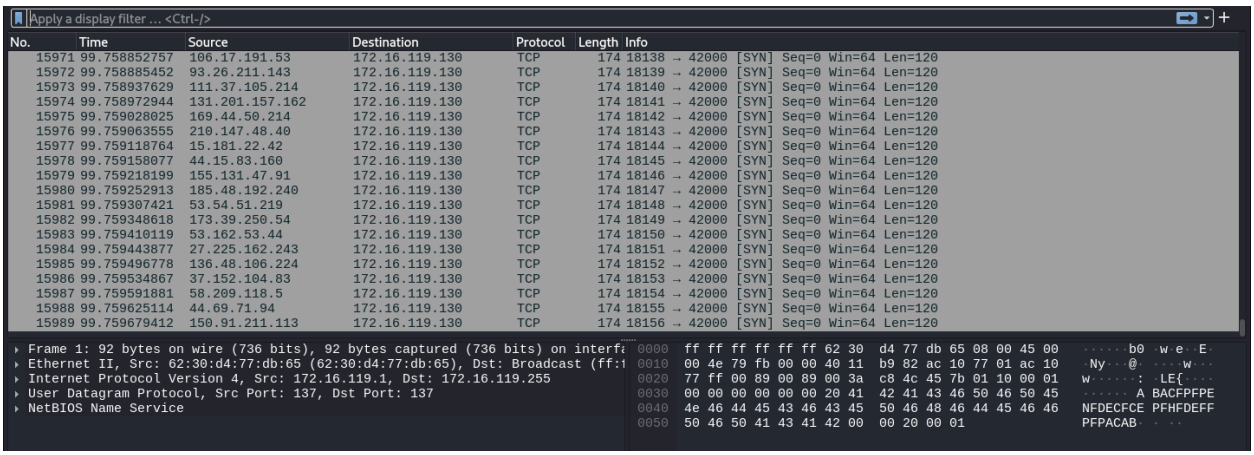
3.4c Code used

sudo hping3 --count 15000 --data 120 --syn --win 64 -p 80 --flood --rand-source 172.16.119.130

In summary, this command uses hping3 to flood the target IP address 172.16.119.130 with 15,000 SYN packets, each containing 120 bytes of data, with a window size of 64 and destination port 80. The source IP addresses are randomized to make the attack harder to trace.

3.4d Screenshots:





3.4e Remediation

MITRE ATT&CK frameworks suggest that 'Content Delivery Networks and services providers who specialise in DoS mitigation should filter traffic upstream and block source addresses from where the attacks may be originating. Also to enable SYN cookies to defend against SYN floods.'

3.4f References

https://owasp.org/www-community/attacks/Denial_of_Service

https://cheatsheetseries.owasp.org/cheatsheets/Denial_of_Service_Cheat_Sheet.html

https://attack.mitre.org/techniques/T1499/

3.5a Discoverable hidden directories

A robots.txt file directory is widely known to provide crucial information, specifically on what directories can be indexed, but more importantly exposes those that shouldn't be. If the robots.txt file is made available, it can provide intel and expose potential sensitive data which may be used against the victim when launching an attack. Badstore.net's robots.txt can be accessed and contains much sensitive data including weakly encrypted passwords of users.

Source	Destination	Tool
172.16.119.128	http://172.16.119.130/robots.txt	Hashcat, John

3.5b Instructions for Replication:

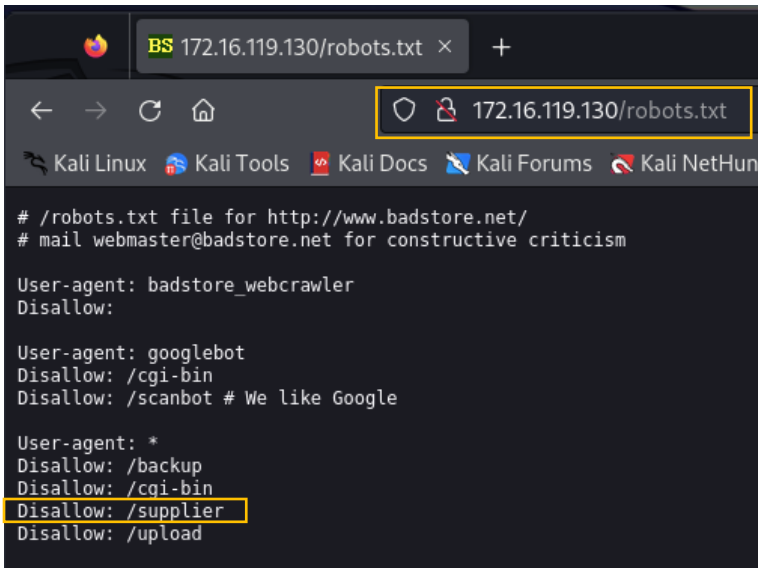
Instructions for Replication:

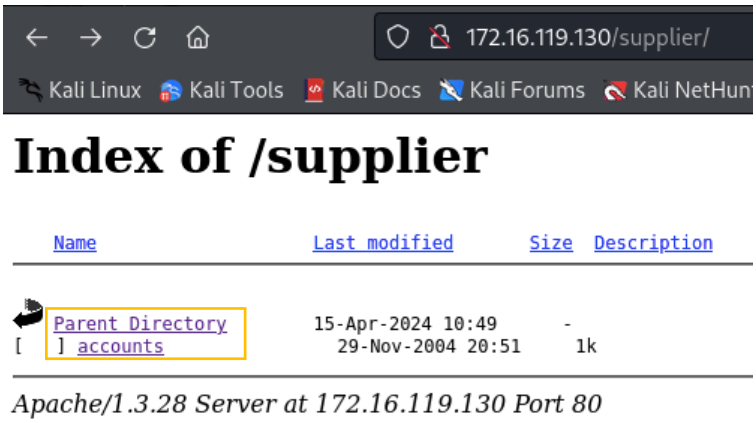
1. **Navigate to robots.txt**: In your web browser, enter the server IP address followed by /robots.txt. In this case enter http://172.16.119.130/robots.txt.
2. **Observe Disallowed Directories**: Upon arrival at the robots.txt file, review the contents for a list of disallowed directories that are not indexed by search engines.
3. **Append Hidden Directory**: Identify a hidden directory from the list in robots.txt and append it to the server IP address in the browser's URL bar. In this case enter http://172.16.119.130/supplier.
4. **Locate Accounts Directory**: Navigate to the supplier directory and look for an accounts directory within it.
5. **Examine Encrypted Data**: In the accounts directory, locate files containing text that appears to be weakly encrypted using base64 encoding.
6. **Decrypt Base64 Text**: You can decrypt base64 encoded text directly in the terminal
7. **Use Decryption Programs**: Alternatively, you can use specialized programs to identify the encryption type and carry out the decryption. Look for tools that support base64 decoding.

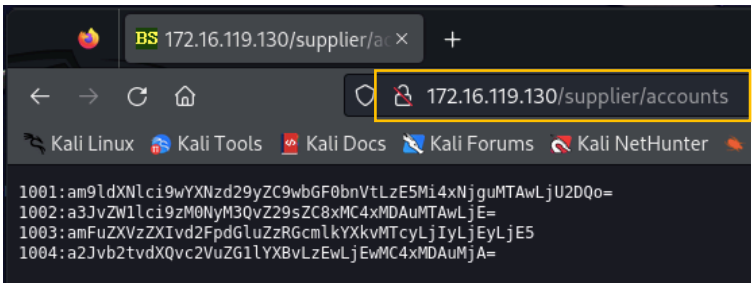
3.5c Code used

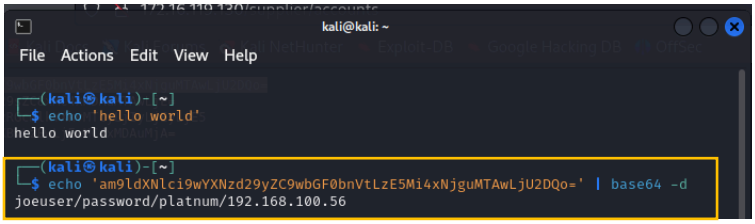
echo [encoded_text] | base64 -d
Replace [encoded_text] with the base64 encoded text you want to decrypt.

3.5d Screenshots:









3.5e Remediation

Indexing can be disabled so that the robots.txt file is non discoverable. This can be achieved by finding the .htaccess file and adding the command 'Options – Indexes'.

3.5f References

https://owasp.org/Top10/A05_2021-Security_Misconfiguration/

3.6a SQL injection (b)

Similarly to the previous instance, it is possible to log in to an administrator account by simple SQL injection in the email field on the login page of the site. Here instead of being logged into any account the administrator account can be targeted with the insertion of 'admin' when malforming the SQL query.

Source	Destination	Tool
172.16.119.128	http://172.16.119.130/cgi-bin/badstore.cgi?action=loginregister	

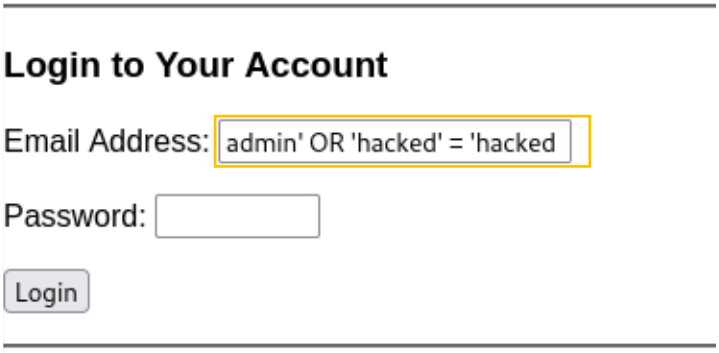
3.6b Instructions for Replication:

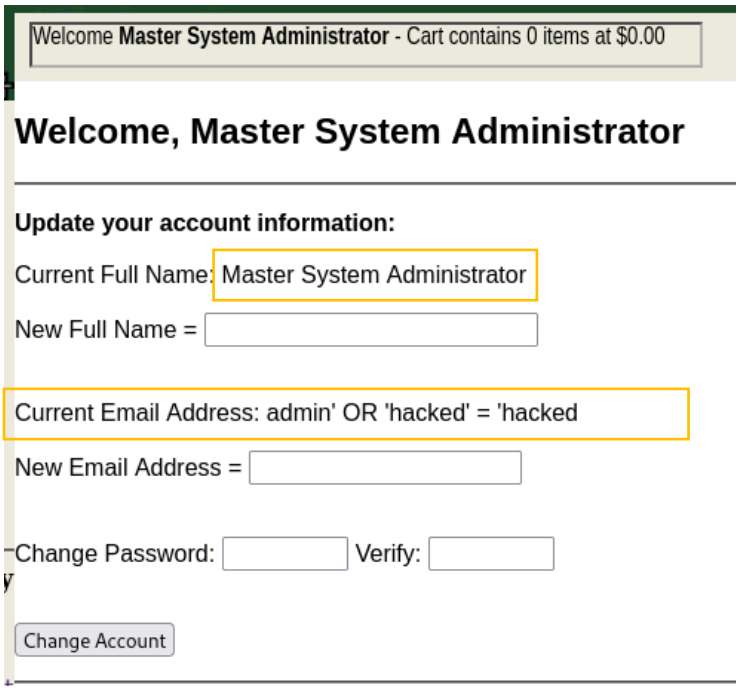
1. **Input Single Quote**: Enter manipulated SQL query into the email field on the login page.
2. **Submit Manipulated Request**: Send the crafted request to the server.
3. **Access Test Account**: If the manipulation is successful, you should gain access to an admin account within the web application.

3.6c Code used

admin' OR 'hacked' = 'hacked'
hacked=hacked will always equate to true, by inserting the single quotations the SQL query is being manipulated, admin gives us access to the master account since there is no input sanitisation or validation.

3.6d Screenshots:





3.6e Remediation

OWASP makes mention of implementing two factor authentication under authentication. By doing so an additional layer of security is added to user accounts if login is not enough of a deterrent. 2FA would ensure only authorised logins are made into the accounts. Another way in which this can be prevented is to implement the function 'mysqli_real_escape_string()' within the code which sanitise user input, removing special characters specific to MySQL.

3.6f References

https://attack.mitre.org/mitigations/M0932/

3.7a Improper password resetting functionality

Badstore.net has minimal input validation to the extent that fields can be left and still produce a result, in this case when resetting a password, no valid account or details are required to acquire a new password, of which it is pasted in plain text to the screen. Issues which arise from this include no encryption of new password which can be used against the login page in conjunction with an SQL injection since a password is now known. It also makes the use of a security question redundant as it is not required.

Source	Destination	Tool
172.16.119.128	http://172.16.119.130/cgi-bin/badstore.cgi?action=myaccount, http://172.16.119.130/cgi-bin/badstore.cgi?action=loginregister	

3.7b Instructions for Replication:

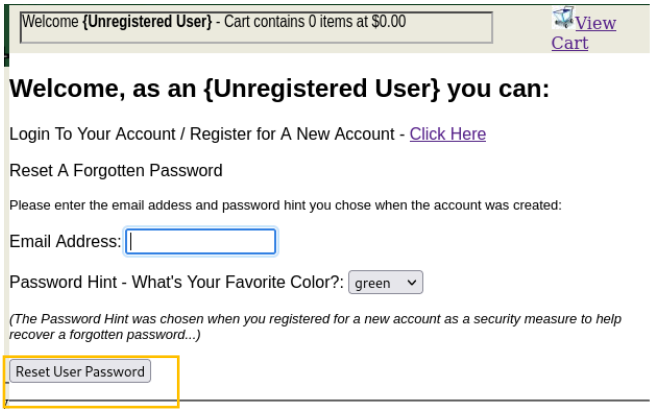
1. **Navigate to 'myaccount':** When on homepage of the site navigate to 'myaccount' as an unregistered user.
2. **Press enter:** Without entering any details press enter.
3. **Observe result:** The site will take you to another page containing the new password 'password'.
4. **Navigate to the login page:** From the current page move to the login page
5. **Manipulate the email field:** Using an SQL query style exploit, enter an injection into the email field.
6. **Use known password:** In the password field use 'Welcome' that was provided
7. **Observe result:** Log in to the user account.

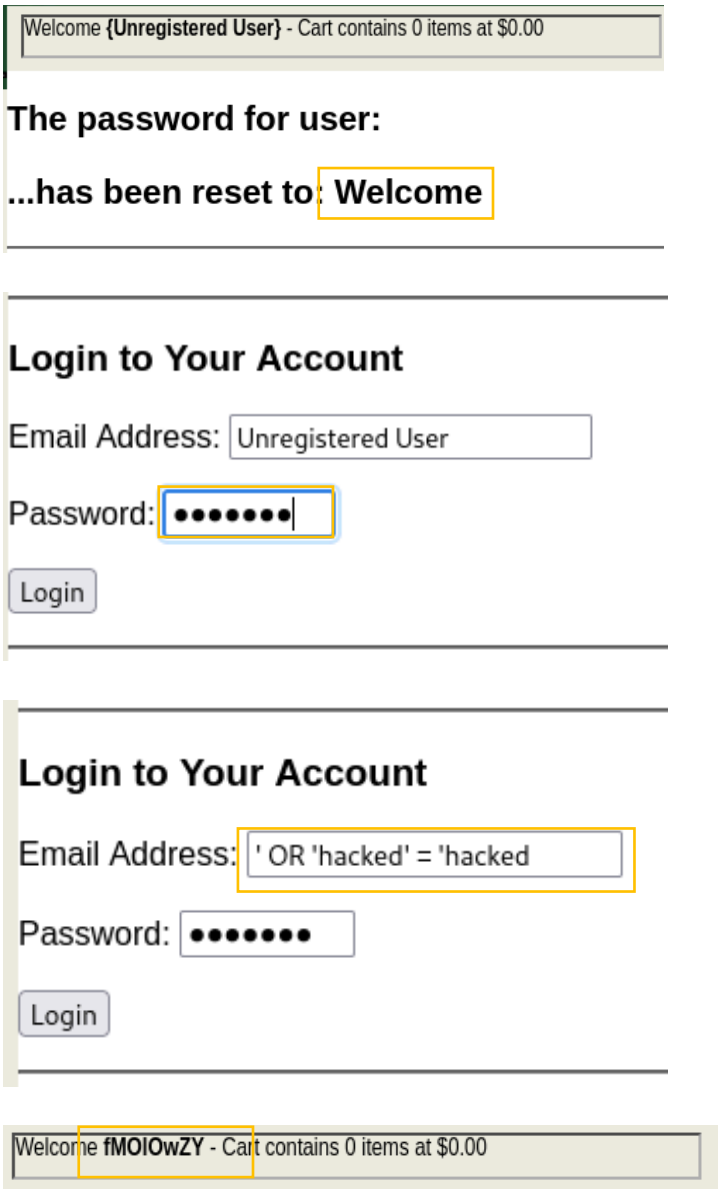
3.7c Code used

' OR 1=1#'

1=1 will always equate to true, by inserting the single quotation the SQL query is being manipulated, # will comment the rest of the request out. Use 'Welcome' in password field

3.7d Screenshots:





3.7e Remediation

Input validation, the site should validate the email address and security question before resetting a password. The password should then be sent via a secure channel to the user associated with that email.

3.7f References

https://attack.mitre.org/mitigations/M0932/

https://attack.mitre.org/techniques/T0822/

https://attack.mitre.org/techniques/T0859/

3.8a Cookie manipulation

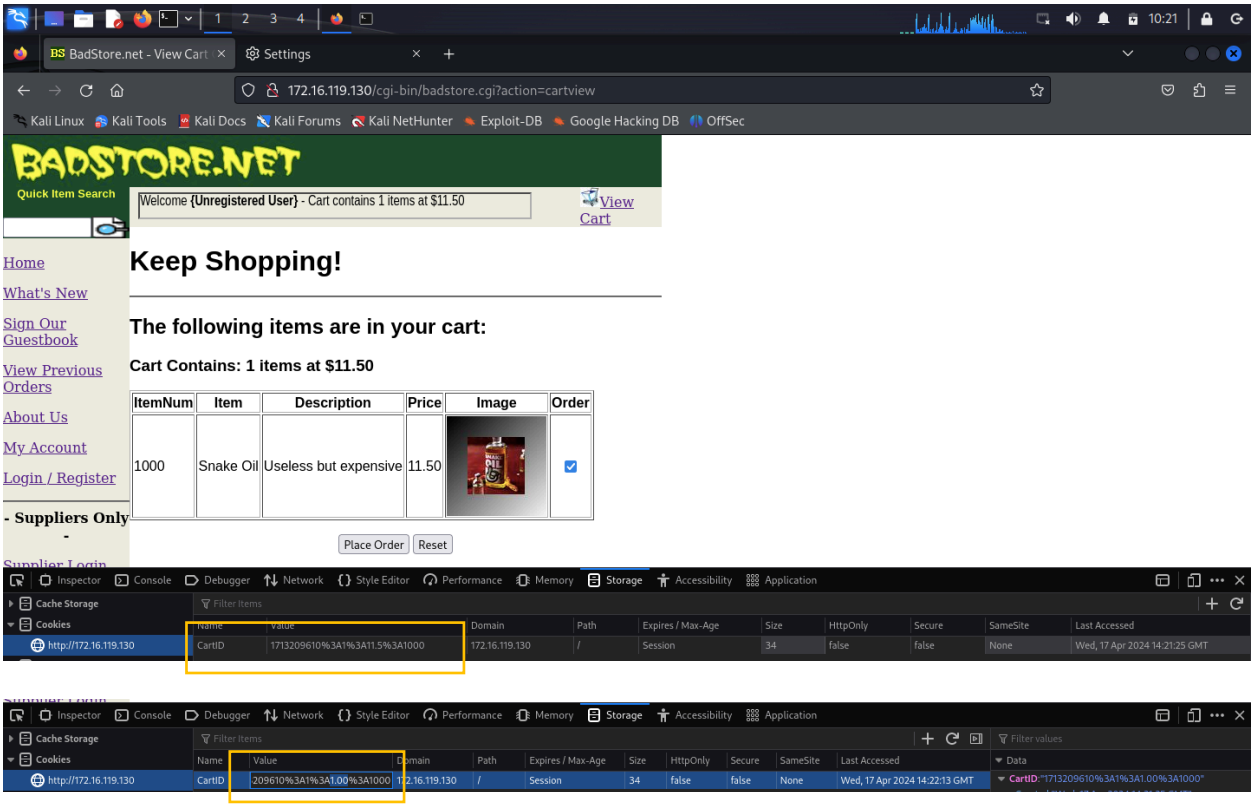
When cookies are created and stored in plain text the web application becomes vulnerable to exploits like cross site request forgery and session hijacking. The cookies on Badstore.net are stored without encryption and can be identified at many different points, if a user is able to identify their own cookies by inspecting the page, then a bad actor sniffing traffic can do the same. When looking at the 'CartID' cookie it is possible to infer where the price information is by cross referencing the item in cart, allowing for the manipulation of the price before checkout.

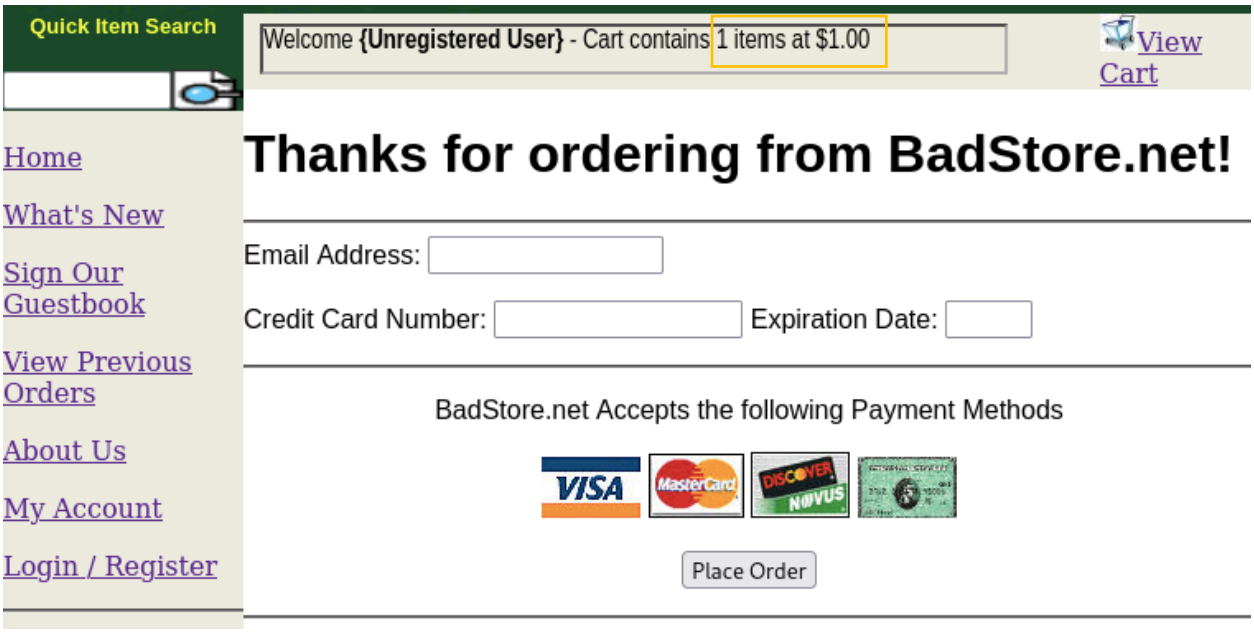
Source	Destination	Tool
172.16.119.128	http://172.16.119.130/cgi-bin/badstore.cgi?action=whatsnew	Wireshark, Burpsuite

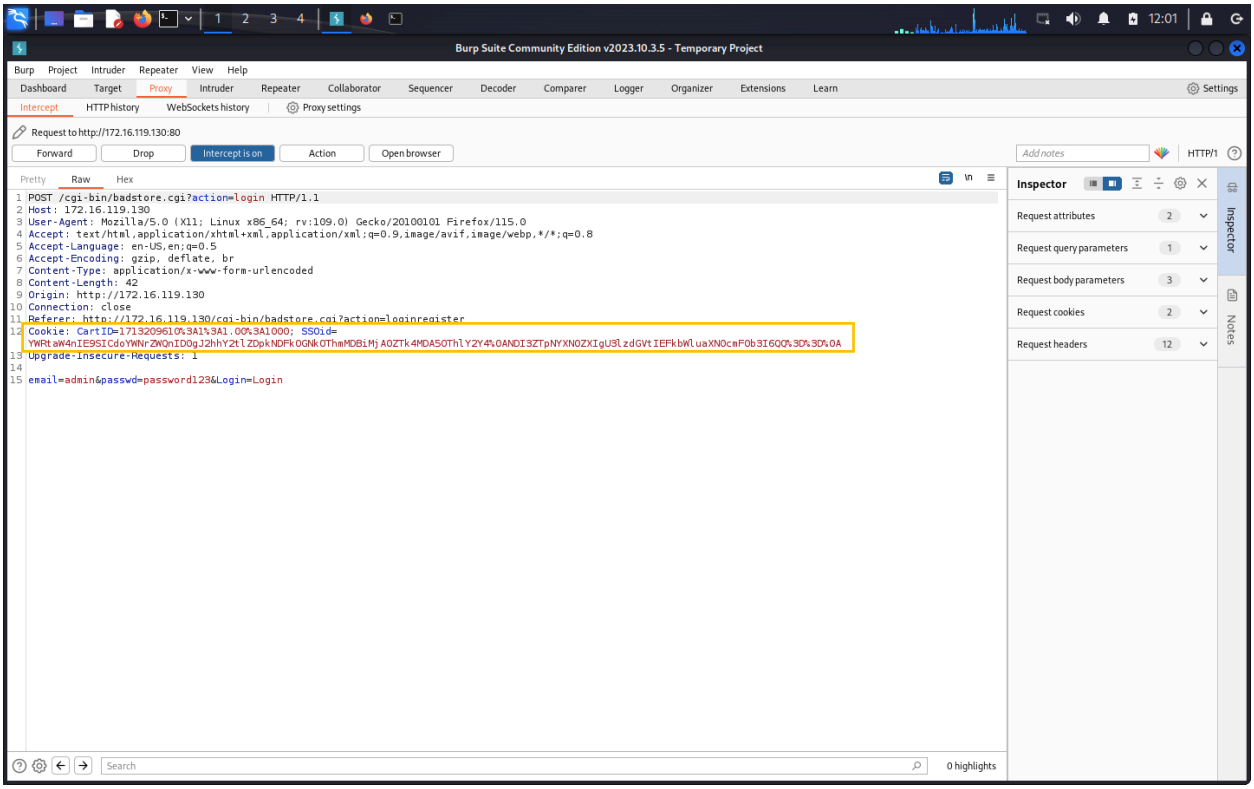
3.8b Instructions for Replication:

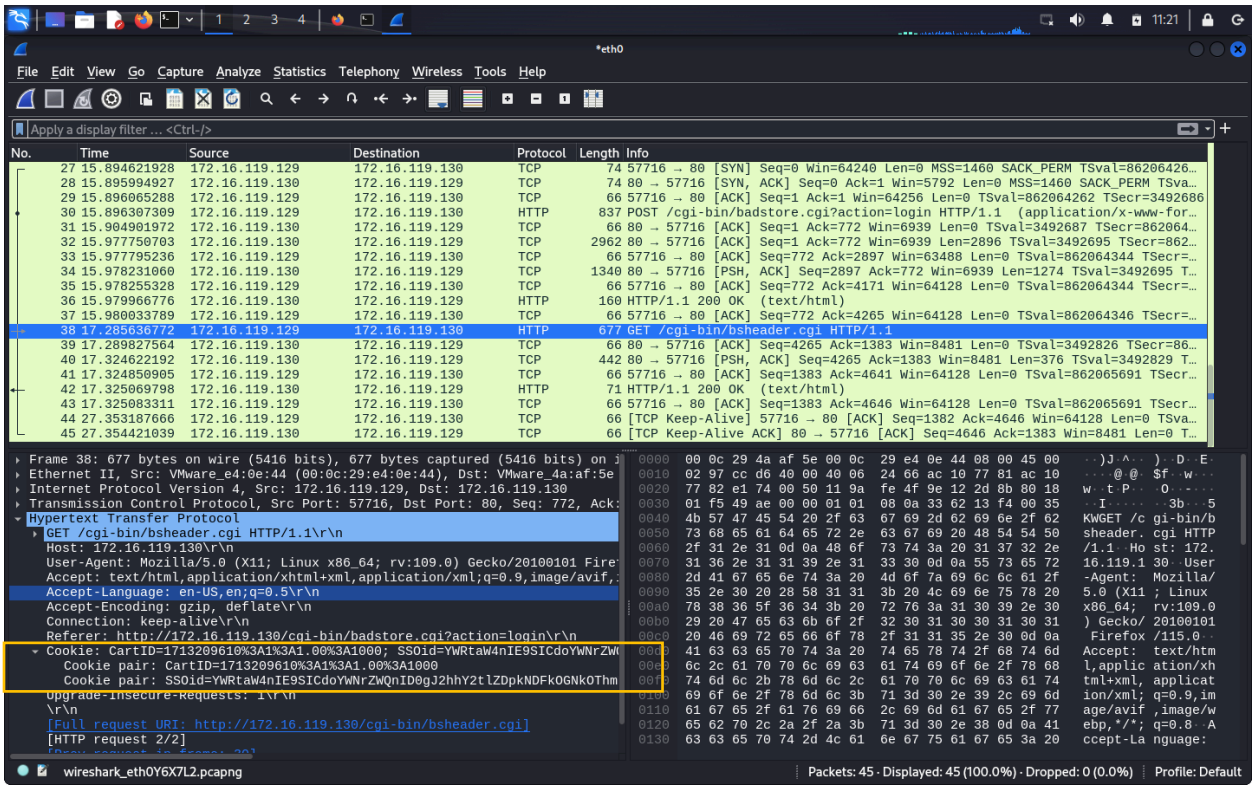
1. **Add item to cart:** Navigate to what's new' and add an item to basket.
2. **Inspect page:** Using inspect element find the cookie containing 'CartID'.
3. **Manipulate price:** Change the price where the number corresponds with the item price.
4. **Analyse packet information:** Use a network sniffer to analyse the packets being sent to the server, find the session ID and CartID.

3.8c Screenshots:




Quick Item Search
Welcome {Unregistered User} - Cart contains 1 items at $1.00
View Cart
Home
What's New
Sign Our Guestbook
View Previous Orders
About Us
My Account
Login / Register
Thanks for ordering from BadStore.net!
Email Address:
Credit Card Number:
Expiration Date:
BadStore.net Accepts the following Payment Methods
Place Order


Burp Suite Community Edition v2023.10.3.5 - Temporary Project
Request to http://172.16.119.130:80
Forward
Drop
Intercept is on
Action
Open browser
POST /cgi-bin/badstore.cgi?action=login HTTP/1.1
Host: 172.16.119.130
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 42
Origin: http://172.16.119.130
Connection: close
Referer: http://172.16.119.130/cgi-bin/badstore.cgi?action=loginregister
Cookie: CartID=1713209610%3A1%3A1.00%3A1000; SSOid=
YWRtaW4nIE9SICdoYWNrZWQnID0gJ2hhY2tlZDpkNDFkOGNkOThmMDBiMjA0ZTk4MDA5OThlY2Y4%0ANDI3ZTpNYXN0ZXIgU3lzdGVtIEFkbWluaXN0cmF0b3I6QQ%3D%3D%0A
Upgrade-Insecure-Requests: 1
email=admin&passwd=password123&Login=Login
Inspector
Request attributes
Request query parameters
Request body parameters
Request cookies
Request headers
Search
0 highlights



3.8d Remediation

Setting the 'HttpOnly' flag will prevent cross origin cookie sharing and CSRF attacks, encrypting and signing cookies can also ensure the integrity as well as the confidentiality of cookie data, making it less accessible to bad actors. Another fix would be to use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid, for example, use anti-CSRF packages such as the OWASP CSRFGuard.

3.8e References

http://projects.webappsec.org/Cross-Site-Request-Forgery

https://cwe.mitre.org/data/definitions/352.html

4. Appendix

4.1 Other in scope vulnerabilities

Vulnerability	Description	Risk
Supplier portal user validation	Supplier portal does not check for privileges when allowing log in to users. Could allow for malicious file upload.	
Brute force attack	Login is susceptible to brute forcing to gain access to accounts.	
Content Security Policy (CSP) header not set	When set, a CSP will help detect and mitigate issues like cross site scripting and data injection attacks.	
Private IP disclosure	When using network sniffers, private IPs can be identified, and potential used for further attacks.	

4.2 Out of scope vulnerabilities

Vulnerability	Description	Risk
MySQL database does not require authentication	Users can directly access the MySQL database server, by direct interaction information can be gathered, deleted, and skewed.	