Pentesting Assessment Report

Badstore Web App

Contents

Badstore Web App
- Contents
- Executive Summary
- Technical Summary
- Assessment Summary
- Assessment Methodology
- Scope of Work
- Assessment Findings
  - Target Discovery
  - Nmap Scan
  - Web Server Enumeration and Findings
  - Broken Access Control
  - Directory Listing
  - Cryptographic Failure
  - Insecure Password Recovery
  - SQL Injection
  - Cross Site Scripting
  - Insecure Communications
  - Business Logic Vulnerability
  - Unrestricted File Upload
- Recommendations

Executive Summary

The Badstore web application was subjected to a comprehensive penetration testing exercise aimed at identifying potential vulnerabilities and risks that could be exploited by malicious actors. The testing methodology involved a combination of automated and manual testing techniques, including black-box testing.

The testing uncovered multiple vulnerabilities, including SQL injection, cross-site scripting (XSS), and broken access control vulnerabilities. These vulnerabilities could allow attackers to gain unauthorized access to sensitive data, modify application functionality, and execute arbitrary code on the application server.

The vulnerabilities discovered were classified according to their severity level, risk rating, and potential impact on the application and its users. Proof-of-concept exploits were developed to demonstrate how these vulnerabilities could be exploited by attackers.

Based on the findings of the testing, we recommend that the Badstore application be immediately patched to address the identified vulnerabilities. Additionally, it is recommended that security controls such as access controls, input validation, and output encoding be implemented to prevent future vulnerabilities from arising.

Overall, the penetration testing exercise provided valuable insights into the security posture of the Badstore web application and highlighted the need for ongoing vigilance and proactive measures to maintain the application's security.

Technical Summary

The Penetration Testing project on the Badstore vulnerable web application utilized the OWASP methodology and various tools including netdiscover, nmap, burpsuite, wireshark, gobuster, hash-identifier, cyberchef, and crackstation. The techniques used during the Penetration Testing project involved a combination of automated and manual testing techniques, including black-box testing.

The Penetration Testing project identified several vulnerabilities within the Badstore web application, each of which was rated based on the severity of the vulnerability and its potential impact on the system. The Broken Access Control allowed an attacker to access restricted content and perform unauthorized actions. The Directory Listing exposed sensitive files and directories to unauthorized users. The Cryptographic Failure allowed an attacker to decrypt sensitive data. The Insecure Password Recovery allowed an attacker to reset passwords and gain unauthorized access. The SQL Injection vulnerabilities allowed an attacker to execute SQL code on the server. The Cross Site Scripting vulnerabilities allowed an attacker to execute javascript code. The Insecure Communications allowed an attacker to intercept and manipulate sensitive data in transit. The Unrestricted File Upload allowed an attacker to upload and execute malicious files on the server. The Business Logic vulnerabilities allowed an attacker to manipulate the application's workflows to gain unauthorized access to sensitive information or perform unauthorized actions.

Mitigation recommendations were provided for each vulnerability identified during the Penetration Testing project. These recommendations include implementing strong password policies, limiting user access to only necessary resources, implementing proper input validation and sanitization, encrypting sensitive data, implementing secure communication protocols, and improving the business logic of the application. It is highly recommended that the organization implements the remediation measures as soon as possible to reduce the risk of a successful attack. Additionally, regular Penetration Testing should be conducted to identify any new vulnerabilities and ensure that the remediation measures are effective.

The Penetration Testing project identified several vulnerabilities within the Badstore web application, each of which was rated based on the severity of the vulnerability and its potential impact on the system. The table below provides a summary of the vulnerabilities identified, their risk level, risk matrix, description, potential impact, and recommendations to mitigate the vulnerability.

Vulnerability	Risk Level	Risk Matrix	Description	Potential Impact	Recommendations

Vulnerability	Risk Level	Risk Matrix	Description	Potential Impact	Recommendations
Directory Listing	Medium	6.0	Exposing sensitive files and directories to unauthorized users	Disclosure of sensitive information	Disable directory listing in web server configurations
Broken Access Control	Critical	9.5	Accessing restricted content and performing unauthorized actions	Unauthorized access to sensitive information and system resources	Implement proper access control and input validation to prevent Broken Access Control attacks
Cryptographic Failure	Critical	9.0	Decrypting sensitive data	Disclosure of sensitive information	Implement proper encryption and key management practices to prevent Cryptographic Failure attacks
Insecure Password Recovery	Critical	10.0	Resetting passwords and gaining unauthorized access	Unauthorized access to user accounts	Implement proper authentication and input validation to prevent Insecure Password Recovery attacks
SQL Injection	Critical	10.0	Executing SQL code on the server	Unauthorized access to sensitive information and system resources	Implement proper input validation and sanitization to prevent SQL Injection attacks
Cross Site Scripting	Critical	8.0	Executing arbitrary code in the user's browser	Unauthorized access to sensitive information and system resources	Implement proper input validation and output encoding to prevent Cross Site Scripting attacks
Insecure Communications	Critical	9.5	Intercepting and manipulating sensitive data in transit	Disclosure of sensitive information	Implement secure communication protocols, such as SSL/TLS, to encrypt communication channels

Vulnerability	Risk Level	Risk Matrix	Description	Potential Impact	Recommendations
Unrestricted File Upload	High	8.0	Uploading and executing malicious files on the server	Unauthorized access to sensitive information and system resources	Implement proper input validation and file type verification to prevent Unrestricted File Upload attacks
Business Logic Vulnerability	Critical	9.0	Inconsistent application of business rules	Unauthorized access to sensitive data/functions	Implement proper business logic validation

Assessment Summary

The Badstore web application penetration test was conducted using both automated and manual testing techniques including black box testing. The testing focused on identifying vulnerabilities in the web application that could potentially be exploited by an attacker.

The tests identified several vulnerabilities in the application including:

1. Broken Access Control: The application was found to have several broken access control vulnerabilities, which could allow an attacker to gain unauthorized access to sensitive information.
2. Directory Listing: The application was found to have directory listing enabled, which could potentially allow an attacker to gain unauthorized access to sensitive information.
3. Cryptographic Failure: The application was found to have several cryptographic vulnerabilities, including weak encryption algorithms and insecure storage of encryption keys.
4. Insecure Password Recovery: The application was found to have insecure password recovery mechanisms, which could allow an attacker to gain unauthorized access to user accounts.
5. SQL Injection: The application was found to be vulnerable to SQL injection attacks, which could potentially allow an attacker to execute arbitrary SQL code on the server.
6. Cross Site Scripting: The application was found to be vulnerable to cross site scripting attacks, which could potentially allow an attacker to execute malicious javascript code on a victim's browser.
7. Insecure Communications: The application was found to have several insecure communication channels, which could potentially allow an attacker to intercept sensitive information.
8. Unrestricted File Upload: The application was found to be vulnerable to unrestricted file upload, which could potentially allow an attacker to upload malicious files to the server.
9. Business Logic Vulnerability: The application was found to be vulnerable to business logic vulnerabilities, which could potentially allow an attacker to bypass the application's business logic and perform unauthorized actions.

Overall, the results of the penetration testing demonstrate the importance of conducting regular vulnerability assessments of web applications to identify and mitigate potential security risks. It is recommended that the identified vulnerabilities be addressed promptly to ensure the security and integrity of the application and its users.

Assessment Methodology

The methodology used in this penetration testing project was based on the Open Web Application Security Project (OWASP) testing guide. The OWASP methodology is a comprehensive framework for testing web applications and provides a structured approach to identifying potential vulnerabilities and risks.

Initially, we used tools such as netdiscover and nmap to identify the target system's IP address and determine the target's open ports.

We used Burp Suite to perform a manual test of the Badstore web application, identifying vulnerabilities such as Broken Access Control, Directory Listing, Cryptographic Failure, Insecure Password Recovery, SQL Injection, Cross Site Scripting, Insecure Communications, Unrestricted File Upload and Business Logic Vulnerabilities. Wireshark was used to analyze network traffic and identify potential security weaknesses, such as weak encryption or improper handling of sensitive data. Tools such as Gobuster was used to identify hidden directories and files on the web server. We also used Hash-identifier and CyberChef to crack password hashes and gain access to the application. Additionally, we used CrackStation to crack weak passwords and gain unauthorized access to the application.

Tools:

The following tools were used during the course of this penetration testing project:

- Netdiscover and Nmap: used for network discovery and port scanning
- Burp Suite: used for web application scanning and vulnerability identification
- Wireshark: used for network traffic analysis and identification of potential security weaknesses
- Gobuster: used for directory and file enumeration on the web server
- Hash-identifier and CyberChef: used for password hash identification and cracking
- CrackStation: used for cracking weak passwords and gaining unauthorized access to the application.

The use of these tools enabled us to conduct a comprehensive and detailed penetration testing project, identifying multiple vulnerabilities and recommending appropriate remediation measures to improve the security posture of the Badstore web application.

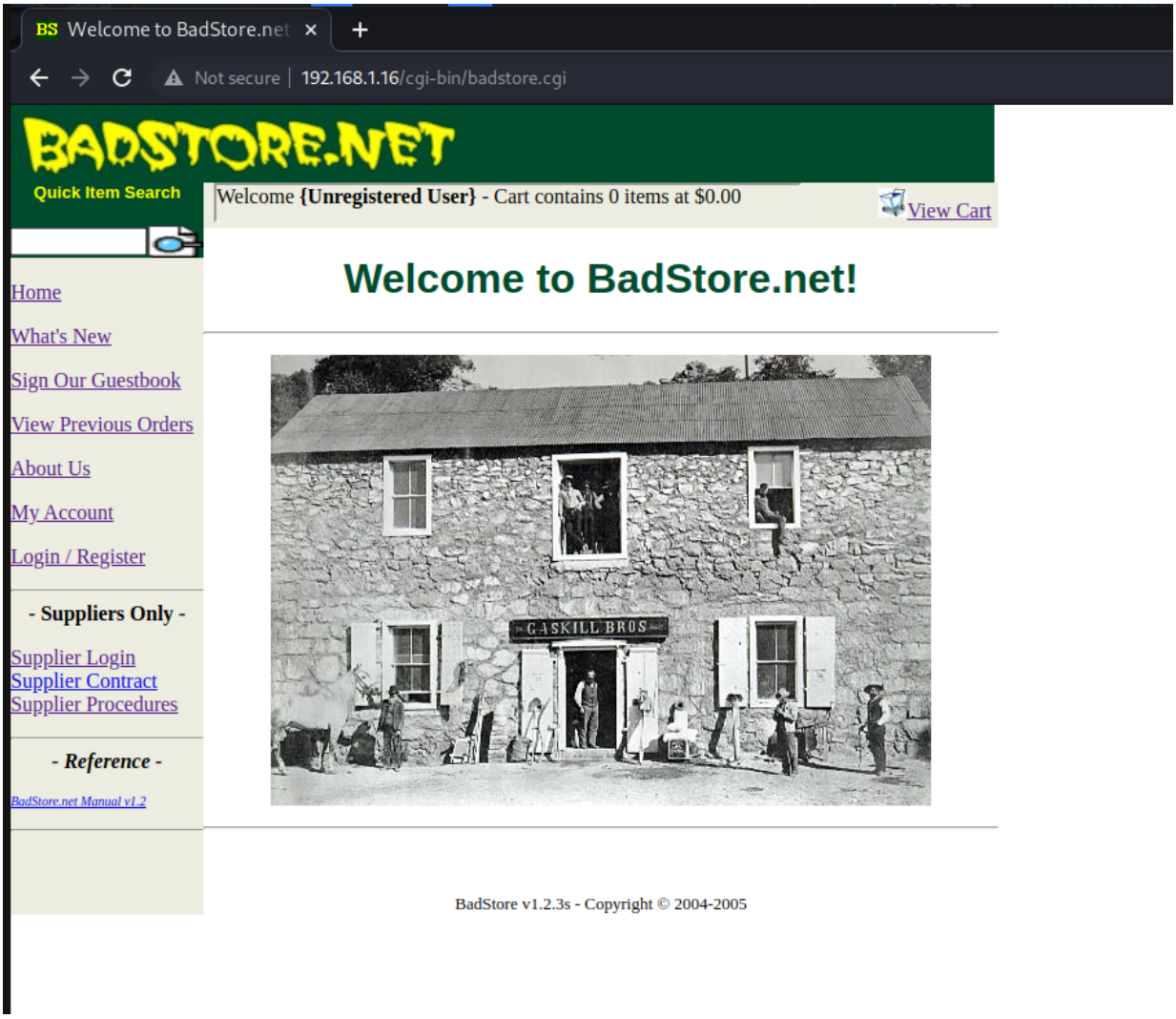
Scope of Work

The scope of this penetration testing project is limited to the Badstore web application and its underlying infrastructure. Testing will be conducted on ports 80 and 443, with the aim of identifying potential vulnerabilities and risks that could be exploited by attackers.

The Rules of Engagement allow for active scanning of the web application for OSINT, and the use of any TTP, including existing exploits and bespoke scripts. However, the use of SQLmap is explicitly excluded from the scope of this project. The testing will not involve any offline attacks on the victim Virtual Hard Disk or interaction with the GRUB loader on the coursework VM. Any interaction with the target must occur through the network, and files should not be accessed directly on the coursework VM.

The exploitation and post-exploitation processes will be documented in detail, and will be replicable. The testing will be conducted with the utmost care and professionalism, with no intentional damage to the web application or underlying infrastructure.

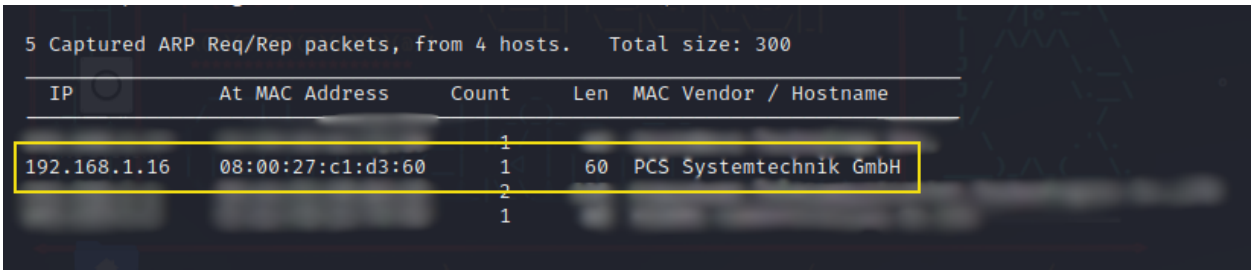
Assessment Findings



Target Discovery

We used “netdiscover” tool to discover the “target IP” and found the “target” to be at the address “192.168.1.16”

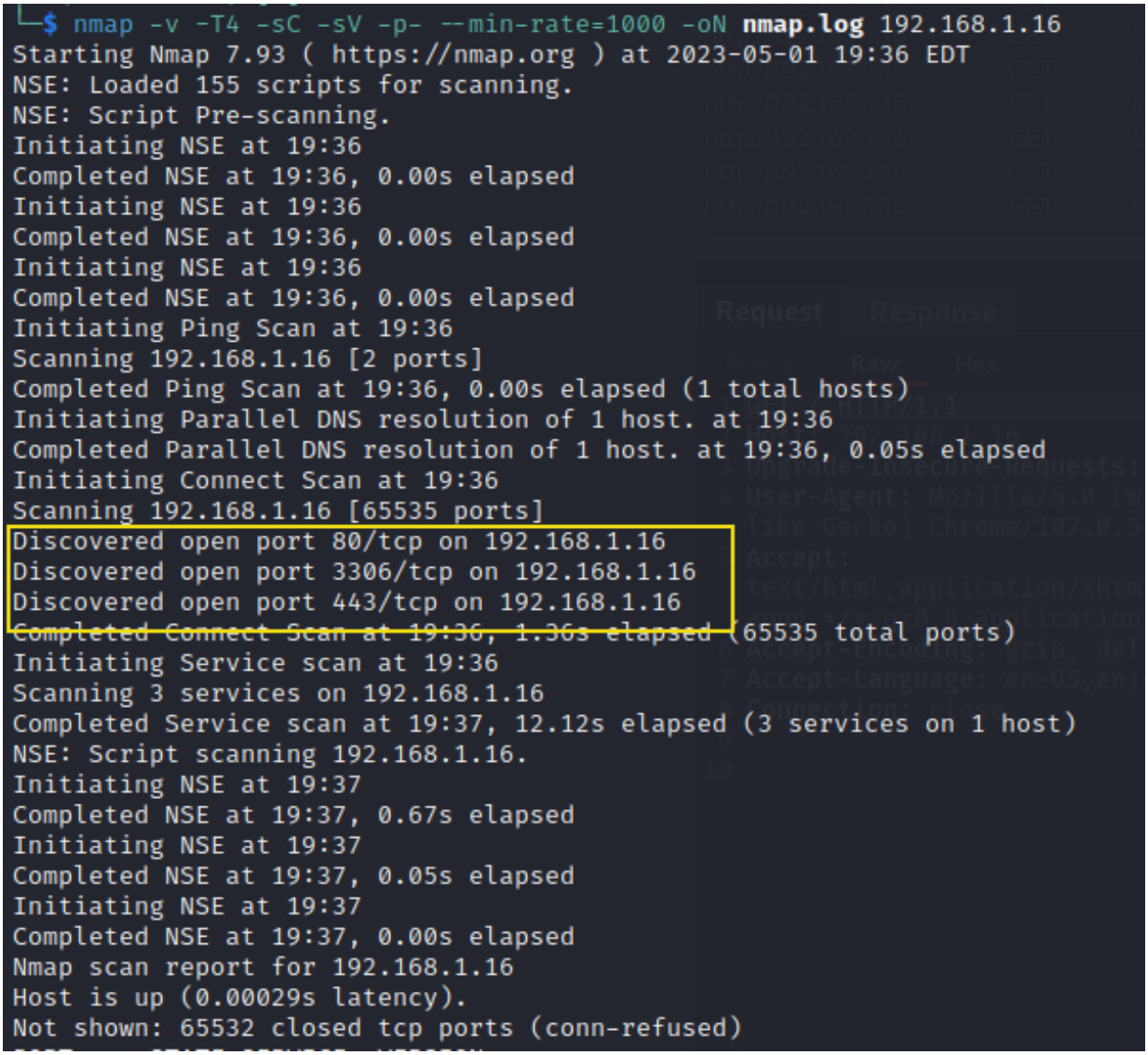
```
sudo netdiscover -i eth0 -r 192.168.1.0/24
```

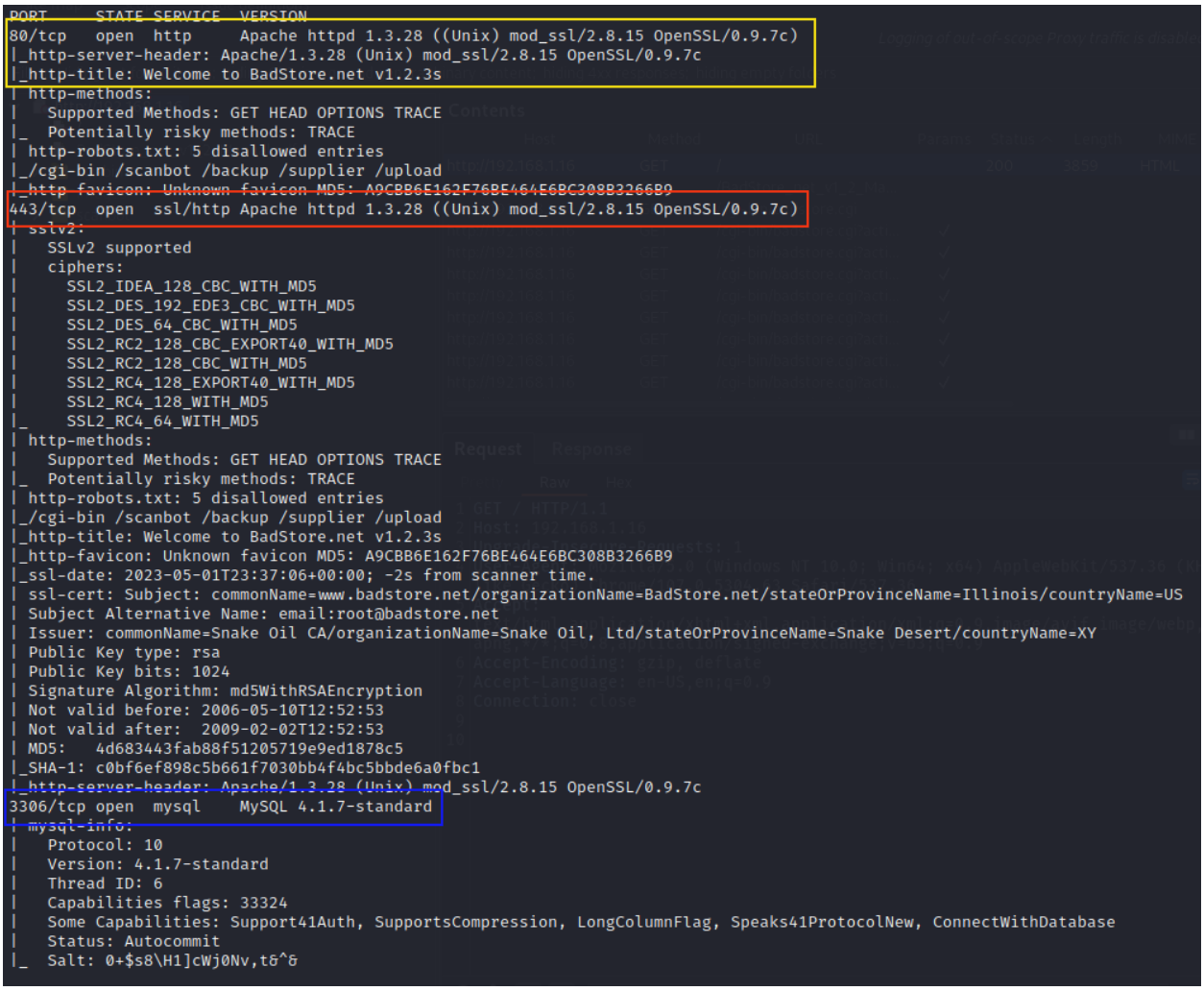


Nmap Scan

After we found out the IP address, we scanned the target for open ports using “Nmap”.

```
nmap -v -T4 -sC -sV -p- --min-rate=1000 -oN nmap.log 192.168.1.16
```


└─$ nmap -v -T4 -sC -sV -p- --min-rate=1000 -oN nmap.log 192.168.1.16
Starting Nmap 7.93 (https://nmap.org) at 2023-05-01 19:36 EDT
NSE: Loaded 155 scripts for scanning.
NSE: Script Pre-scanning.
Initiating NSE at 19:36
Completed NSE at 19:36, 0.00s elapsed
Initiating NSE at 19:36
Completed NSE at 19:36, 0.00s elapsed
Initiating NSE at 19:36
Completed NSE at 19:36, 0.00s elapsed
Initiating Ping Scan at 19:36
Scanning 192.168.1.16 [2 ports]
Completed Ping Scan at 19:36, 0.00s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 19:36
Completed Parallel DNS resolution of 1 host. at 19:36, 0.05s elapsed
Initiating Connect Scan at 19:36
Scanning 192.168.1.16 [65535 ports]
Discovered open port 80/tcp on 192.168.1.16
Discovered open port 3306/tcp on 192.168.1.16
Discovered open port 443/tcp on 192.168.1.16
Completed Connect Scan at 19:36, 1.36s elapsed (65535 total ports)
Initiating Service scan at 19:36
Scanning 3 services on 192.168.1.16
Completed Service scan at 19:37, 12.12s elapsed (3 services on 1 host)
NSE: Script scanning 192.168.1.16.
Initiating NSE at 19:37
Completed NSE at 19:37, 0.67s elapsed
Initiating NSE at 19:37
Completed NSE at 19:37, 0.05s elapsed
Initiating NSE at 19:37
Completed NSE at 19:37, 0.00s elapsed
Nmap scan report for 192.168.1.16
Host is up (0.00029s latency).
Not shown: 65532 closed tcp ports (conn-refused)

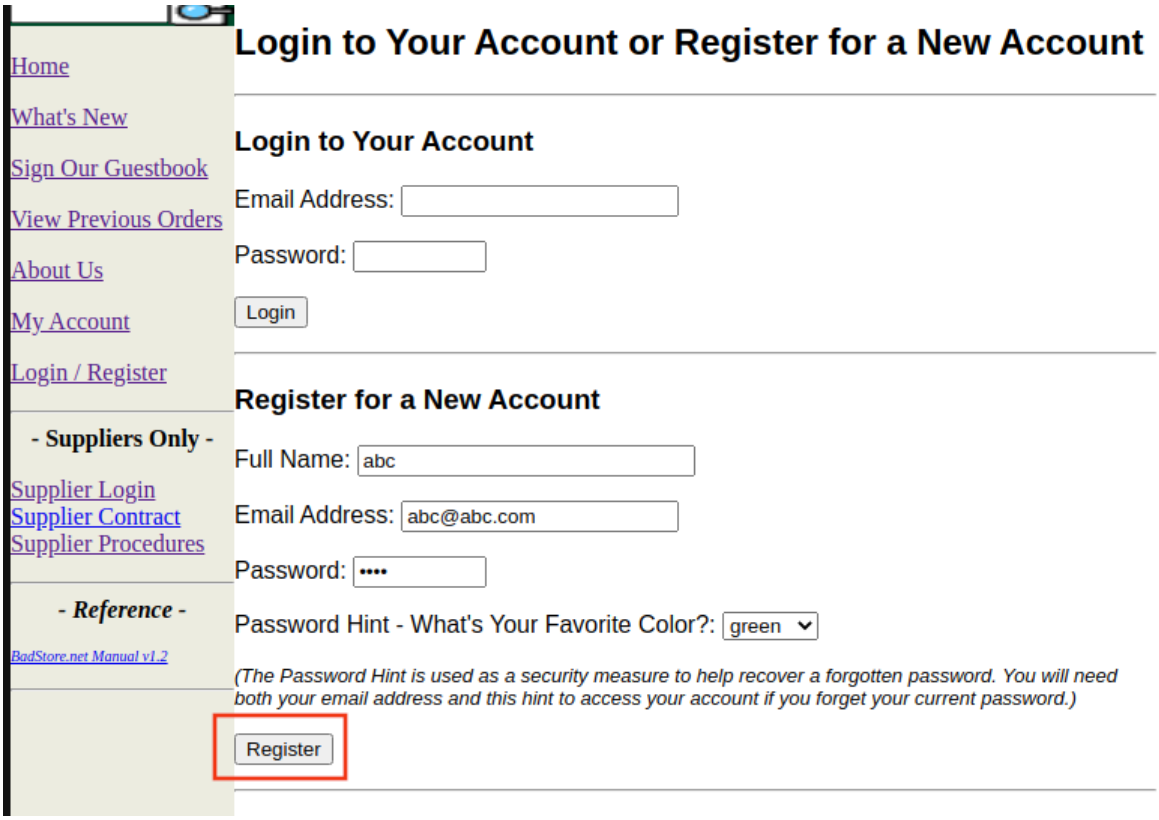


We found out that there were 3 ports open on the “target” machine. Port 80, 443 and 3306, containing a “web server” and a “mysql” instance.

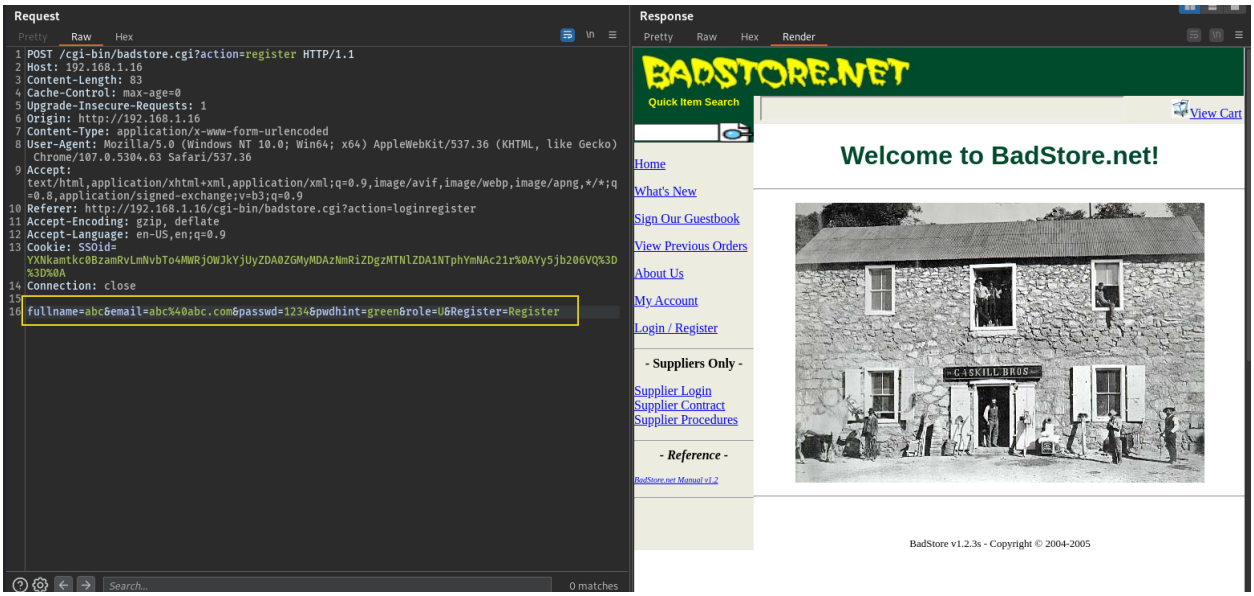
Web Server Enumeration and Findings

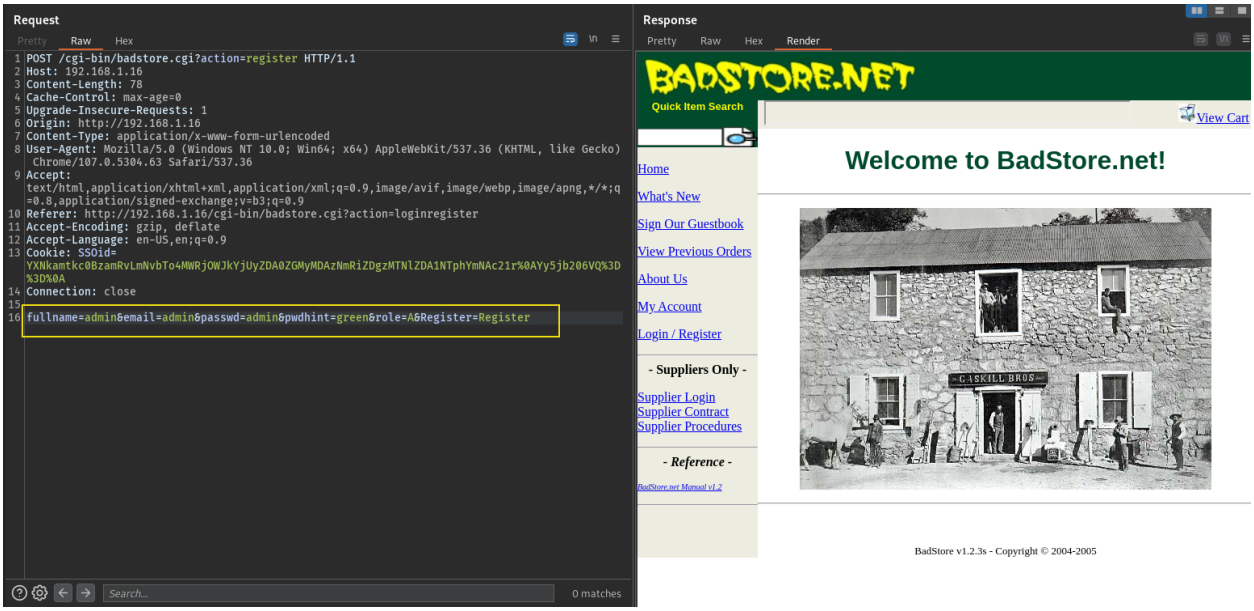
We clicked around the application and found a “login” and “registration” page.

Broken Access Control

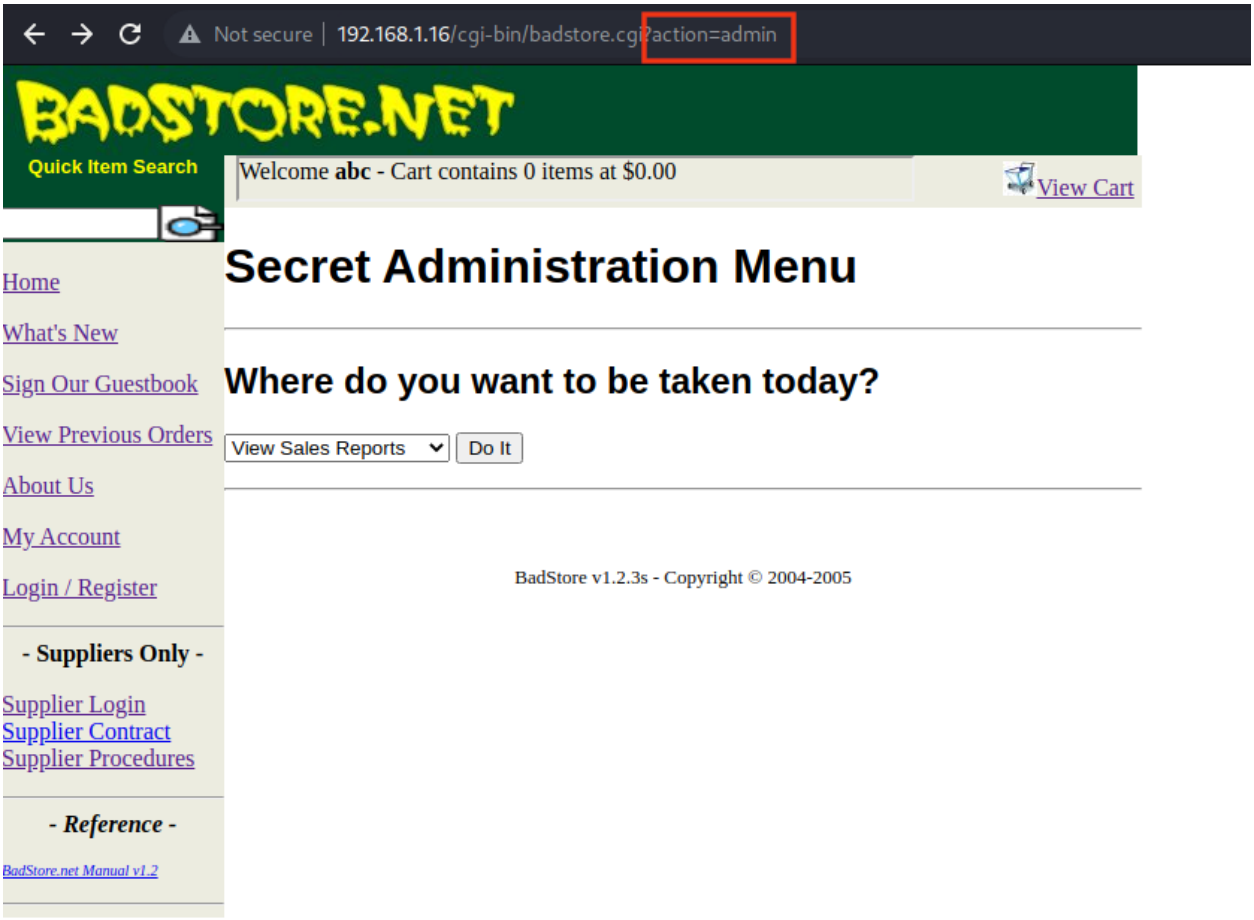


We tried registering a user account and intercepted the request using “Burpsuite”. There was some interesting info. First, the traffic was unencrypted and then there was a “roles” field, which was exploitable. We changed the “role=U” to “role=A” and we had an “admin”account created.

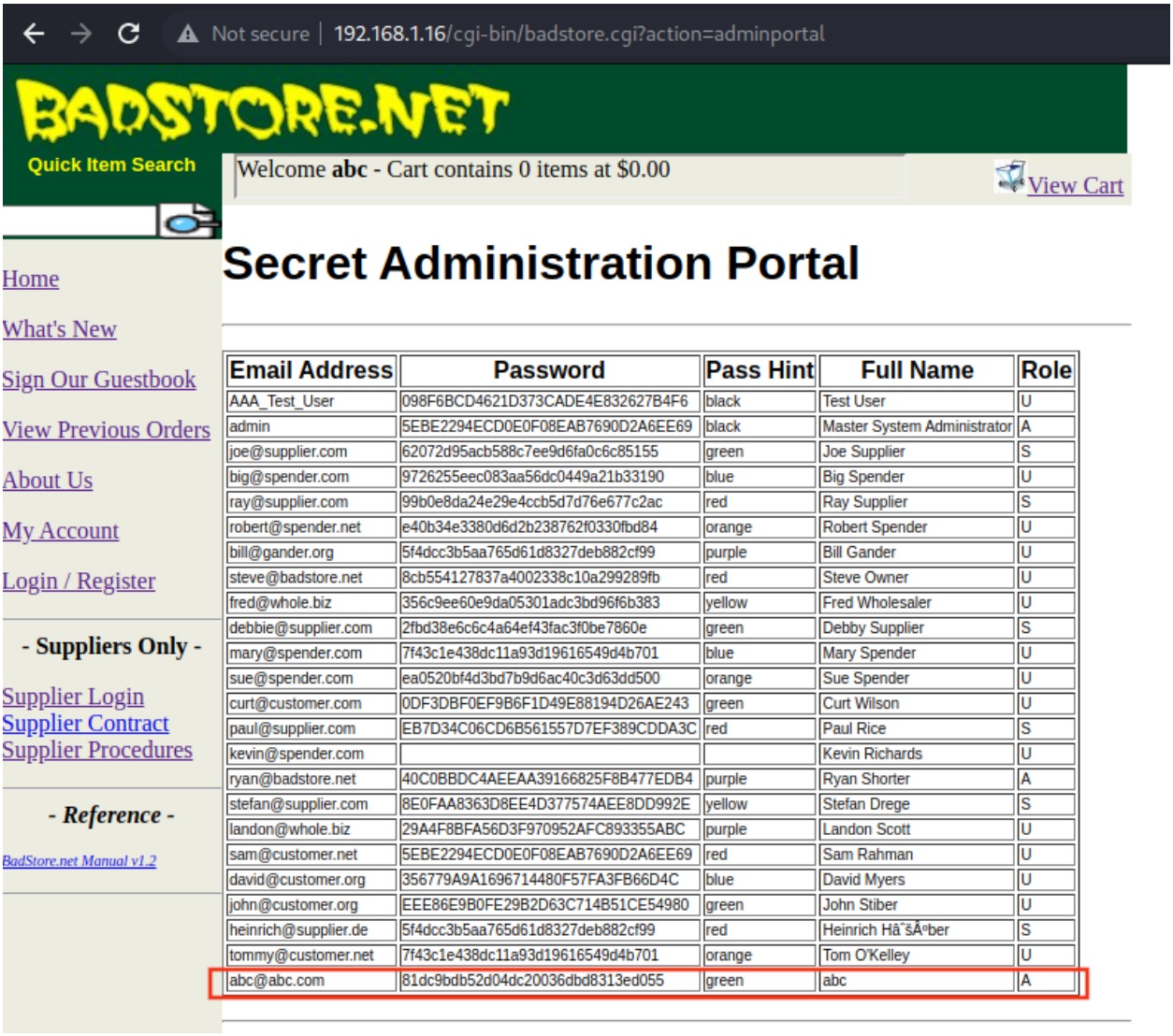




The request was successful and we were registered as “Admin” role, which would also be confirmed in the next step. We saw that there was an “action” field which we could manipulate. We tried action=admin and got the secret admin page.



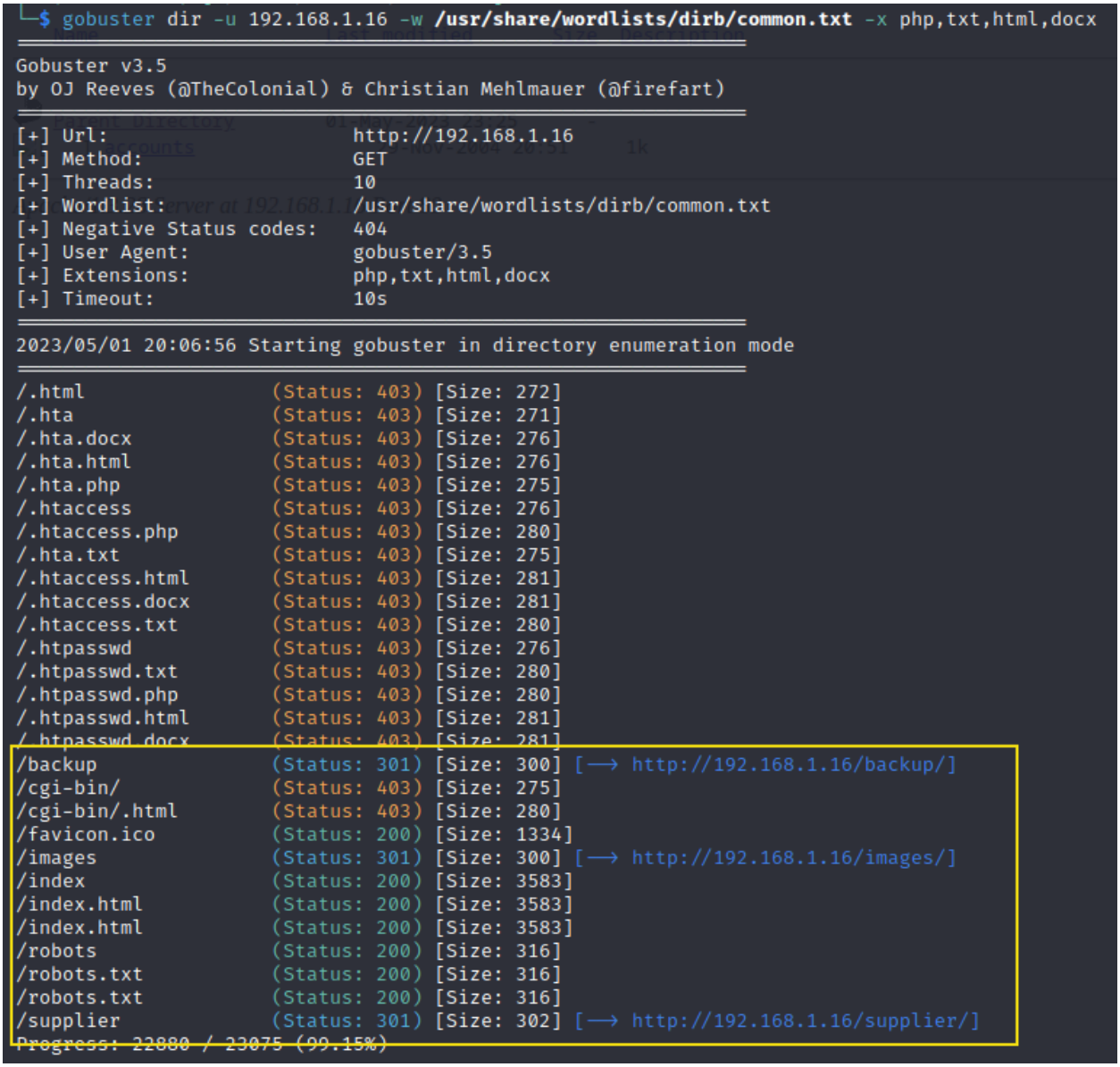
We clicked the drop down menu and found a “show current users” option. Pressed “Do It’ and our newly created user had the “admin” role as we expected.



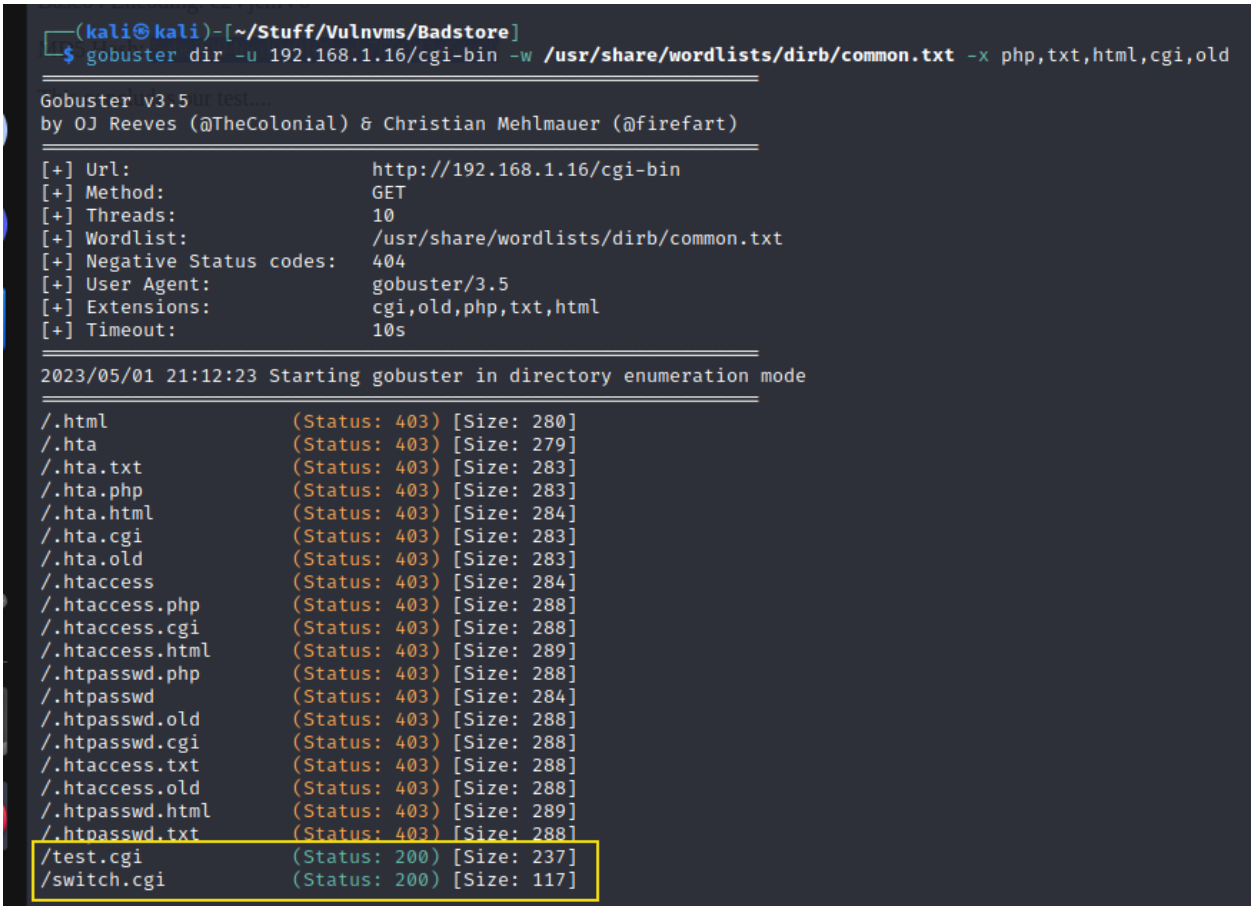
Directory Listing

We used “gobuster” tool to list out common directories and found some interesting ones.

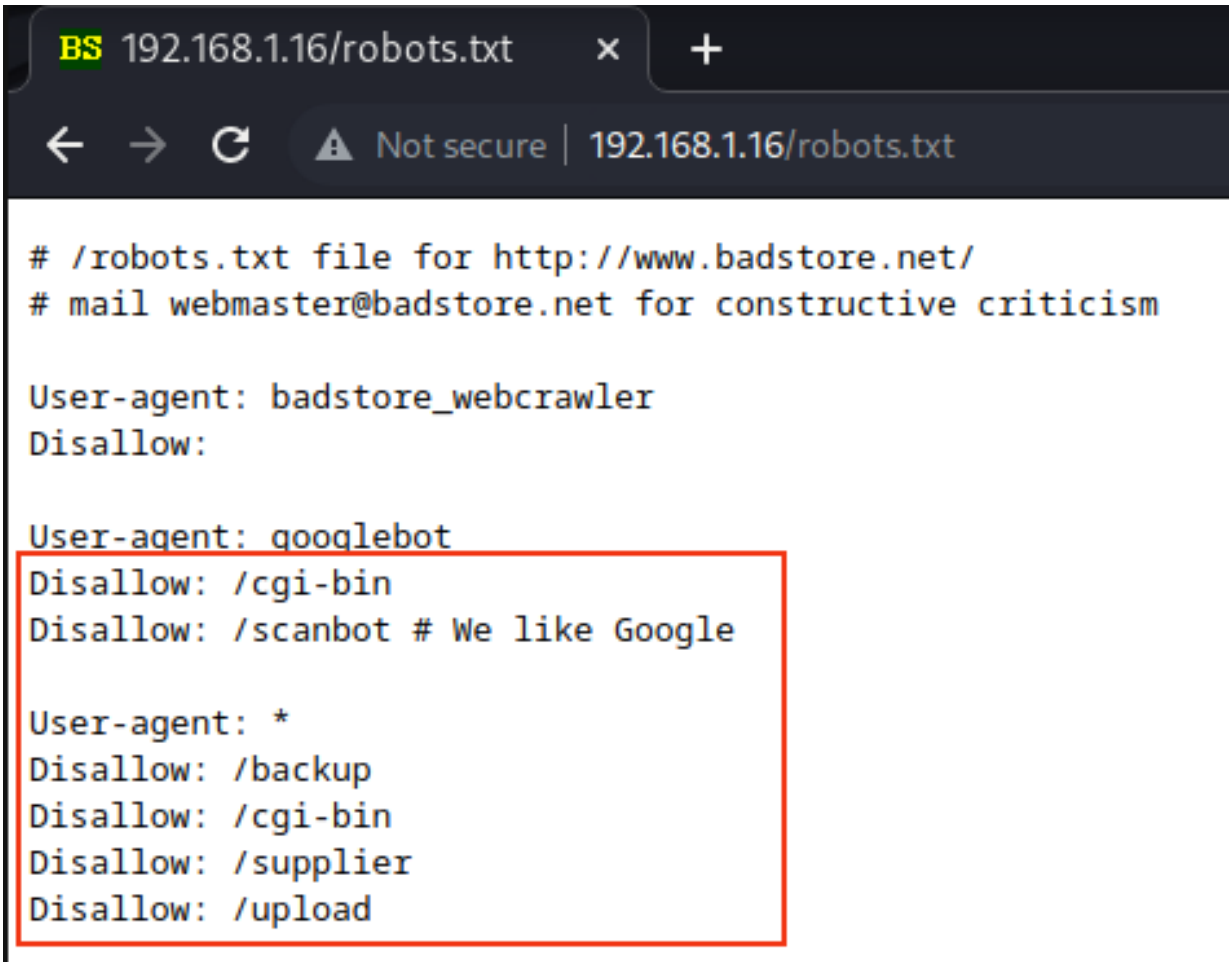
```
gobuster dir -u 192.168.1.16 -w /usr/share/wordlists/dirb/common.txt -x php,txt,html,docx
```



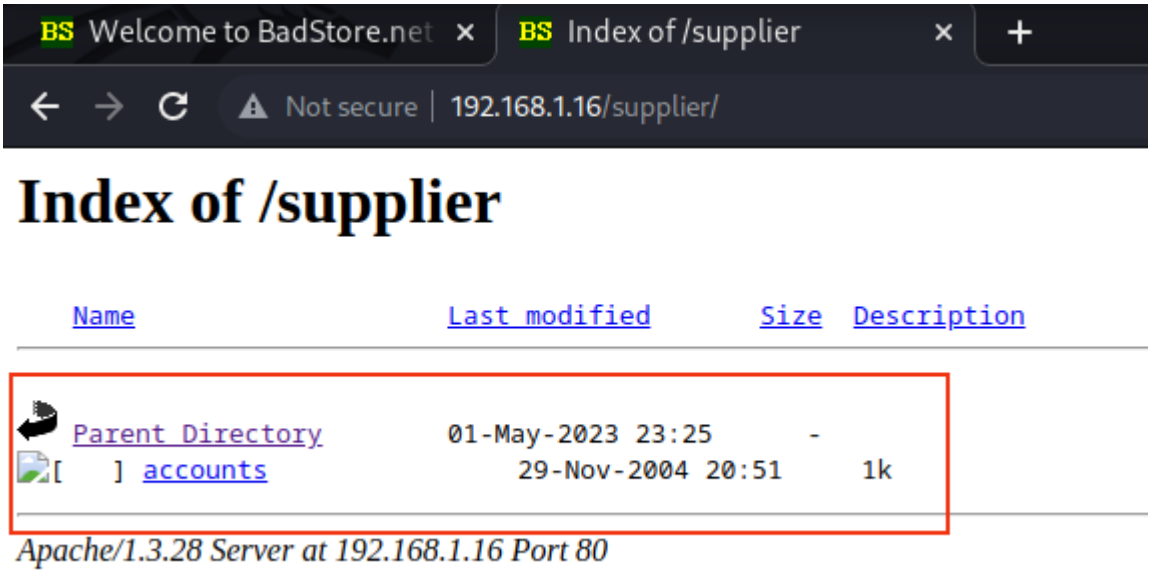

```
gobuster dir -u 192.168.1.16/cgi-bin -w /usr/share/wordlists/dirb/common.txt -x php,txt,html,cgi,old
```



The robots.txt had some interesting info, so we looked into it.

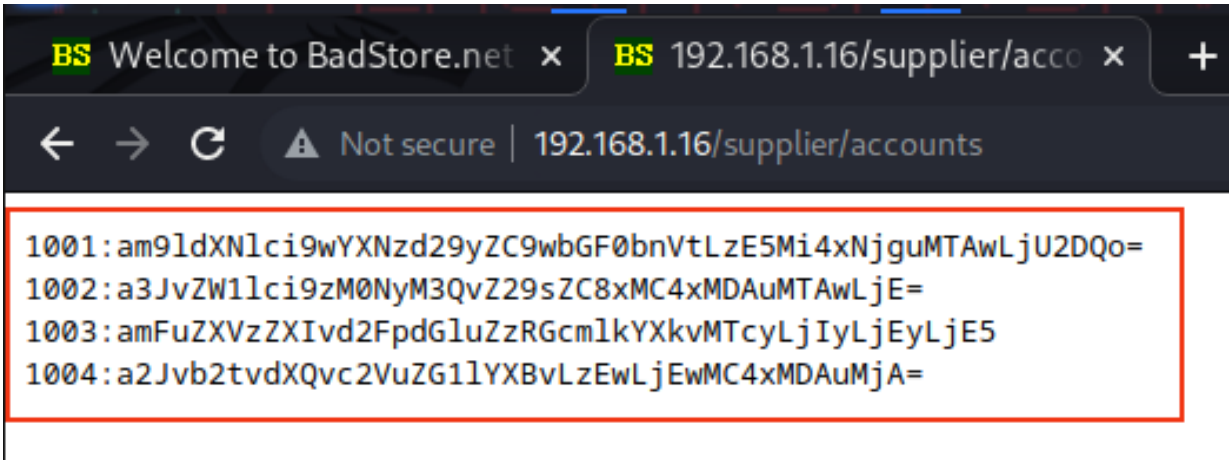


The “supplier” directory had some interesting information that led us to our next step in the testing.

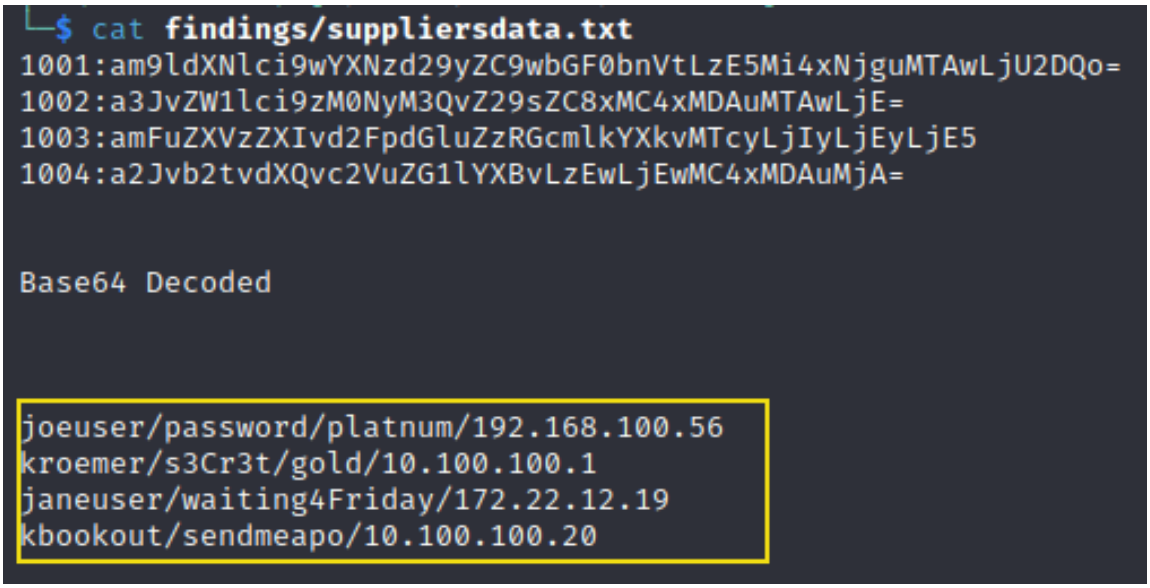


Cryptographic Failure

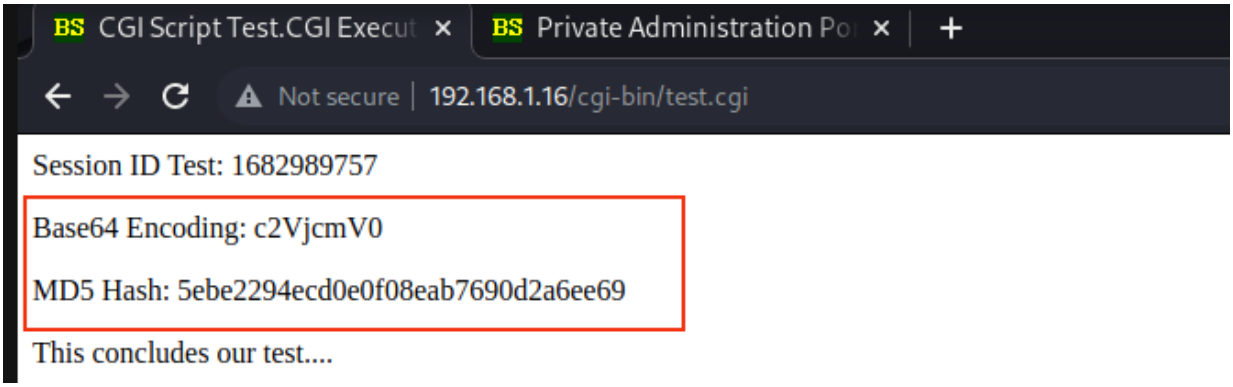
The “accounts” file within the “supplier” directory had Base64 encoded strings, which we decoded using the “Cyberchef” tool.



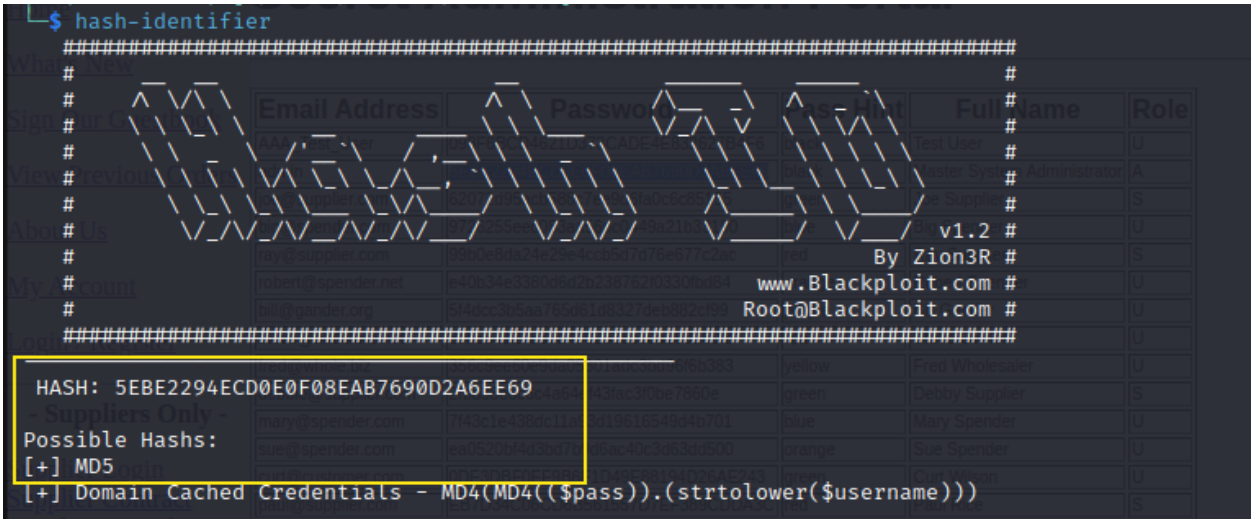
These contained the username, passwords, IP addresses of various “suppliers”.



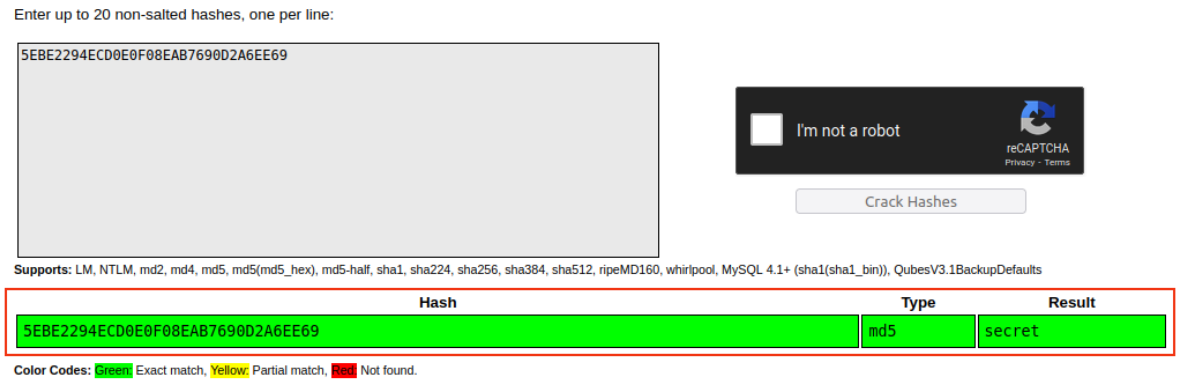
The “test.cgi” directory found using “gobuster” also had a Base64 encoded string along with a MD5 Hash string. Both when decoded had the string “secret”, which is the default password for the “admin” account.

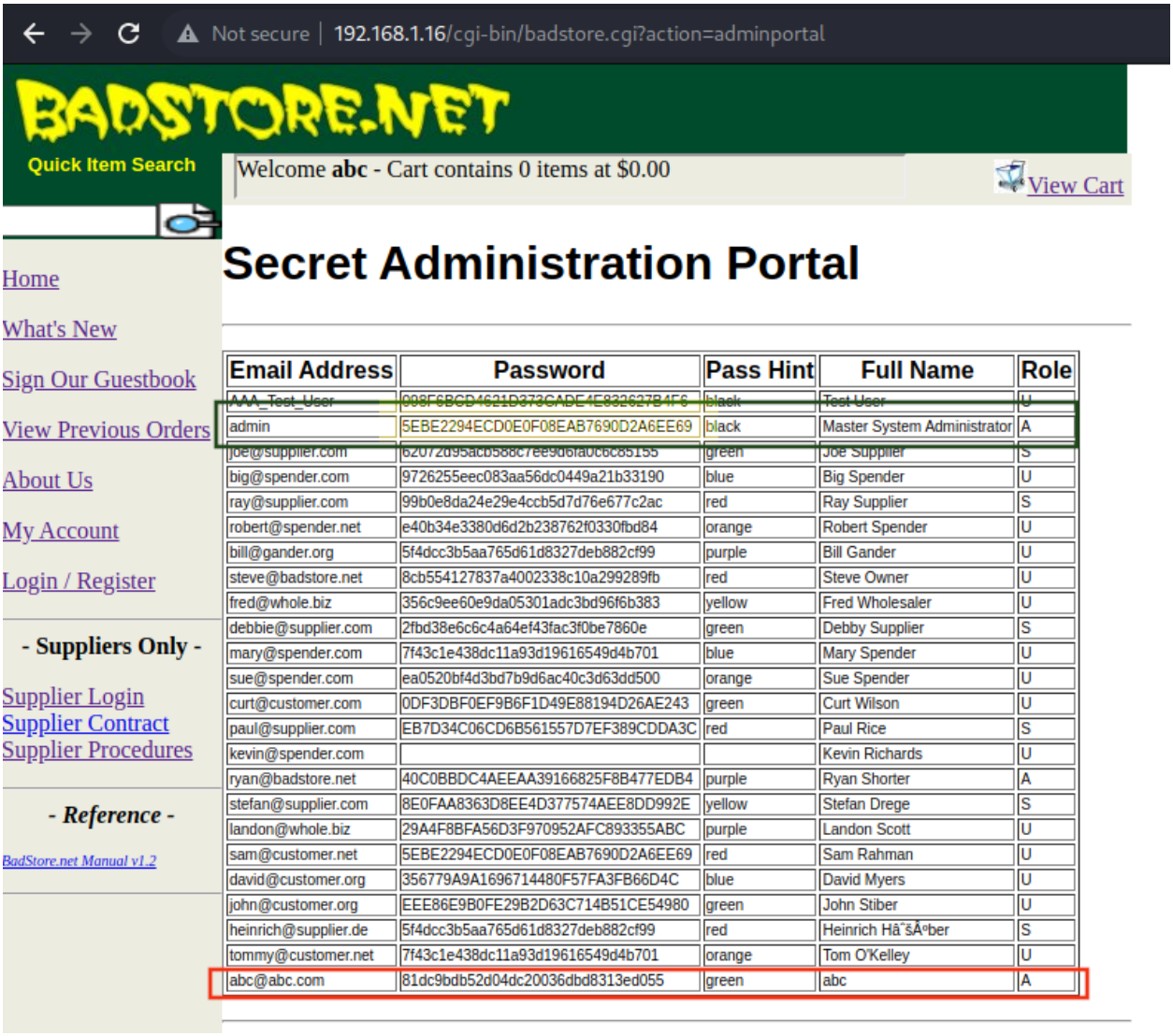


We verified that the “Hash” is actually MD5 by using the “hash-identifier” tool.

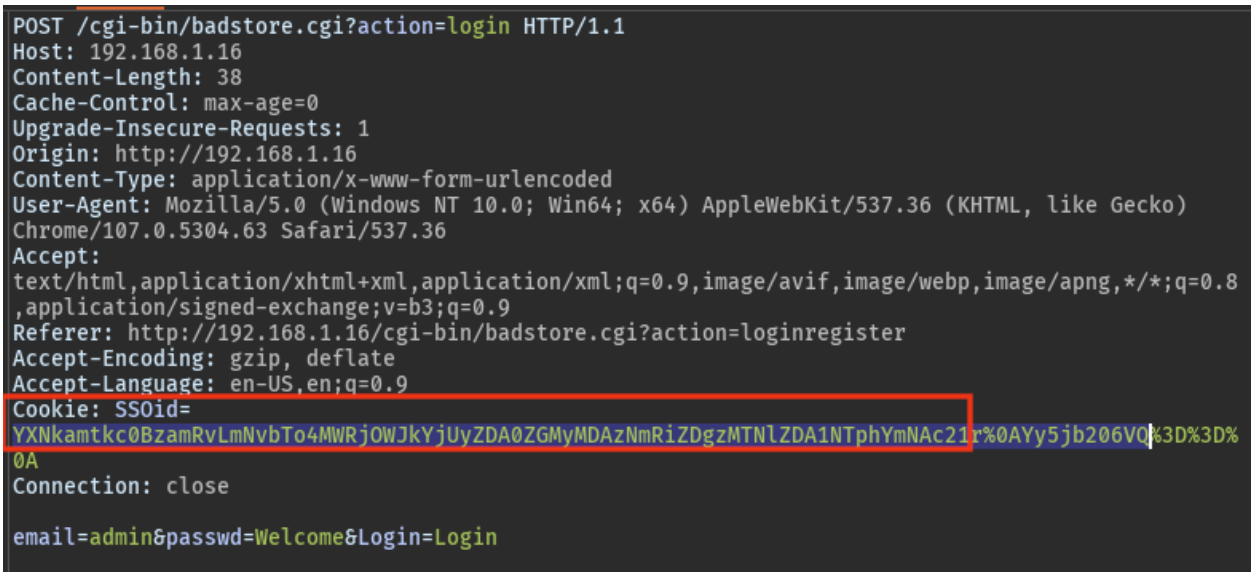


We got the Admin’s password by cracking the “Hash” with “crackstation”.

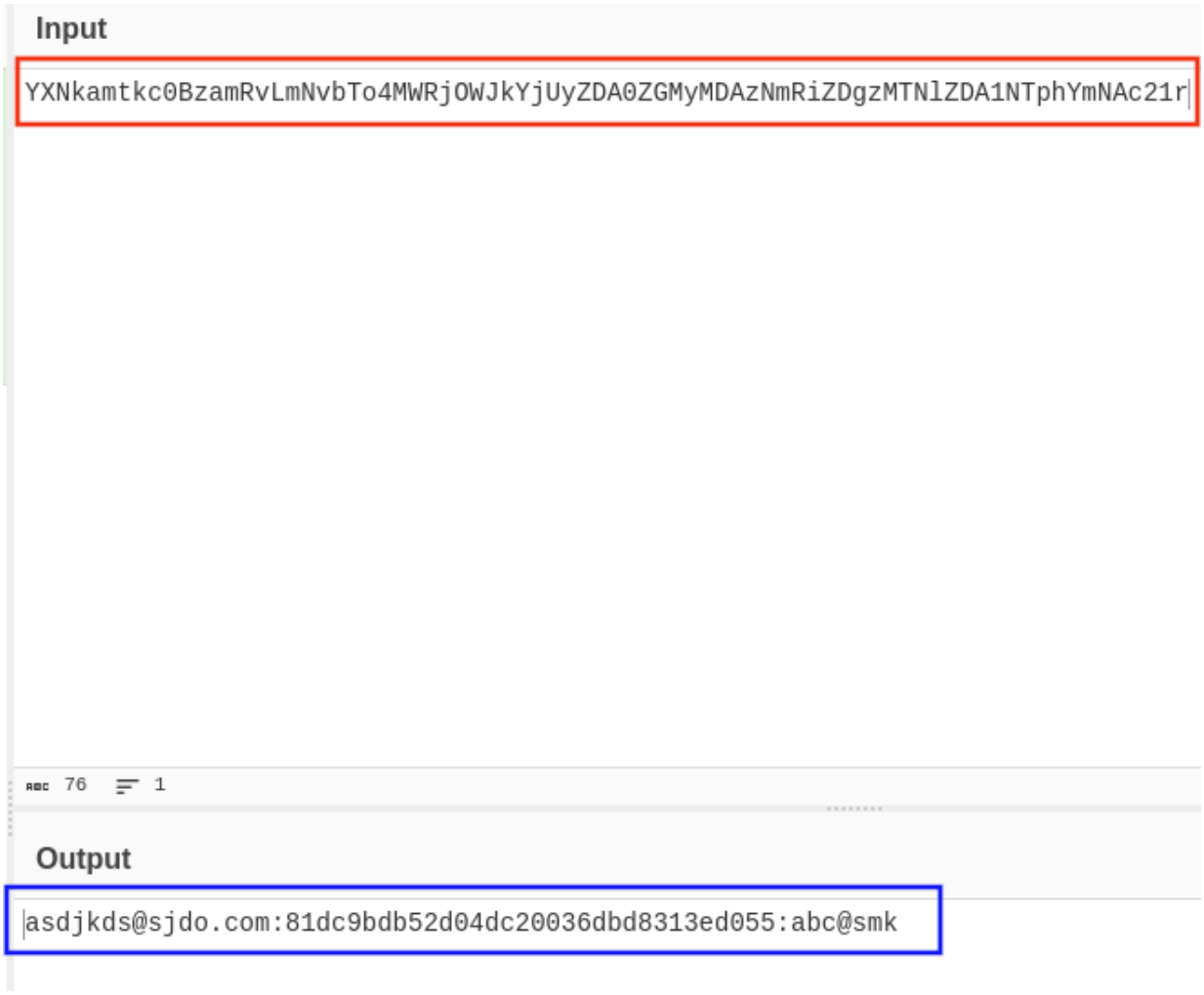




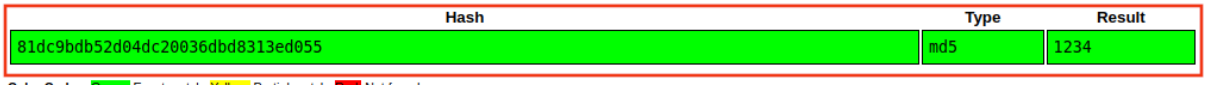
We also found that the “Cookies” being used for the session are Base64 encoded by using “Cyberchef” tool.



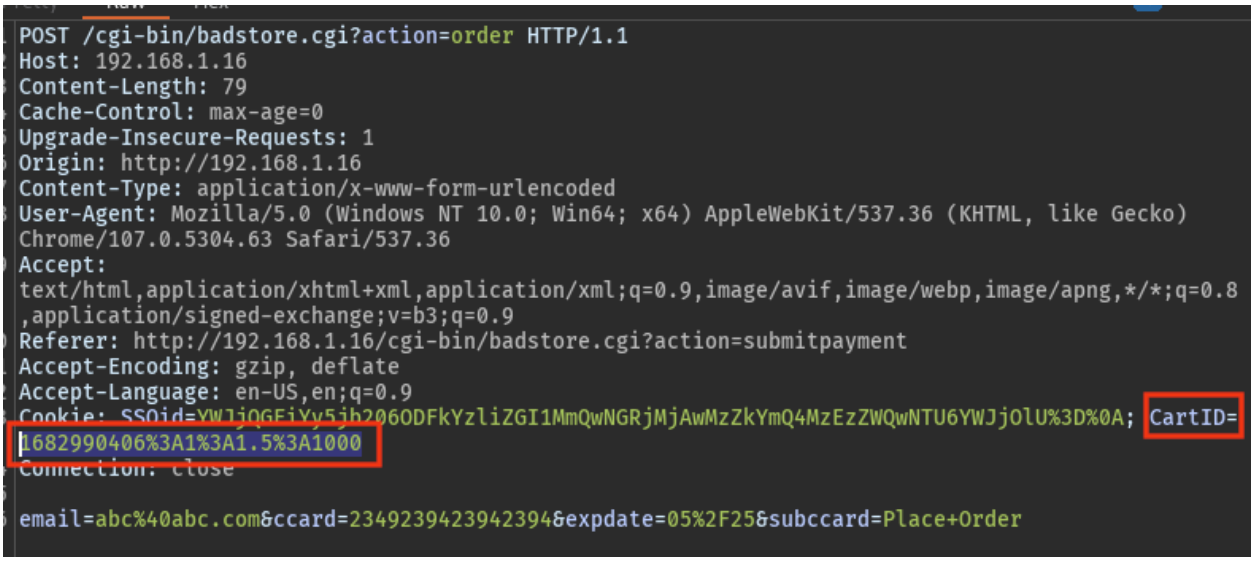
By decoding the Base64 string we found out that the session “Cookies” consisted of the name, password hash and email of the logged in user.



We decoded the password “Hash” using “crackstation” and got the plain text “password” of the user.

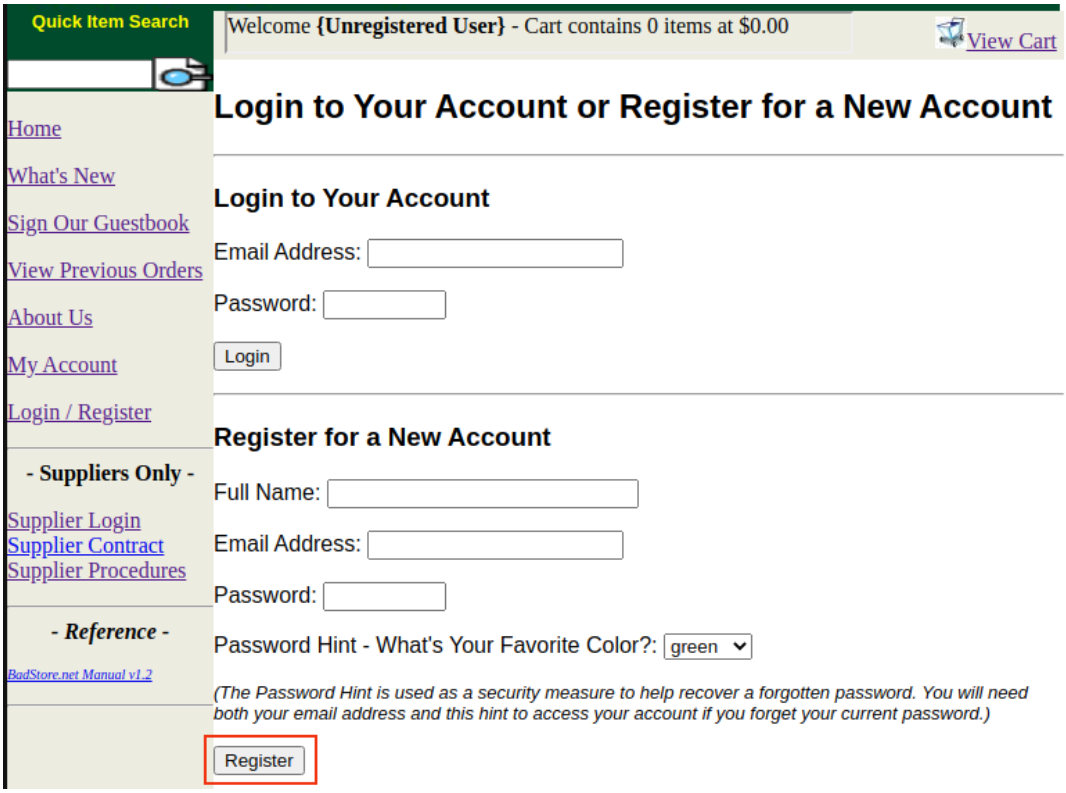


We also were able to identify the information within the encoded “CartID” string.

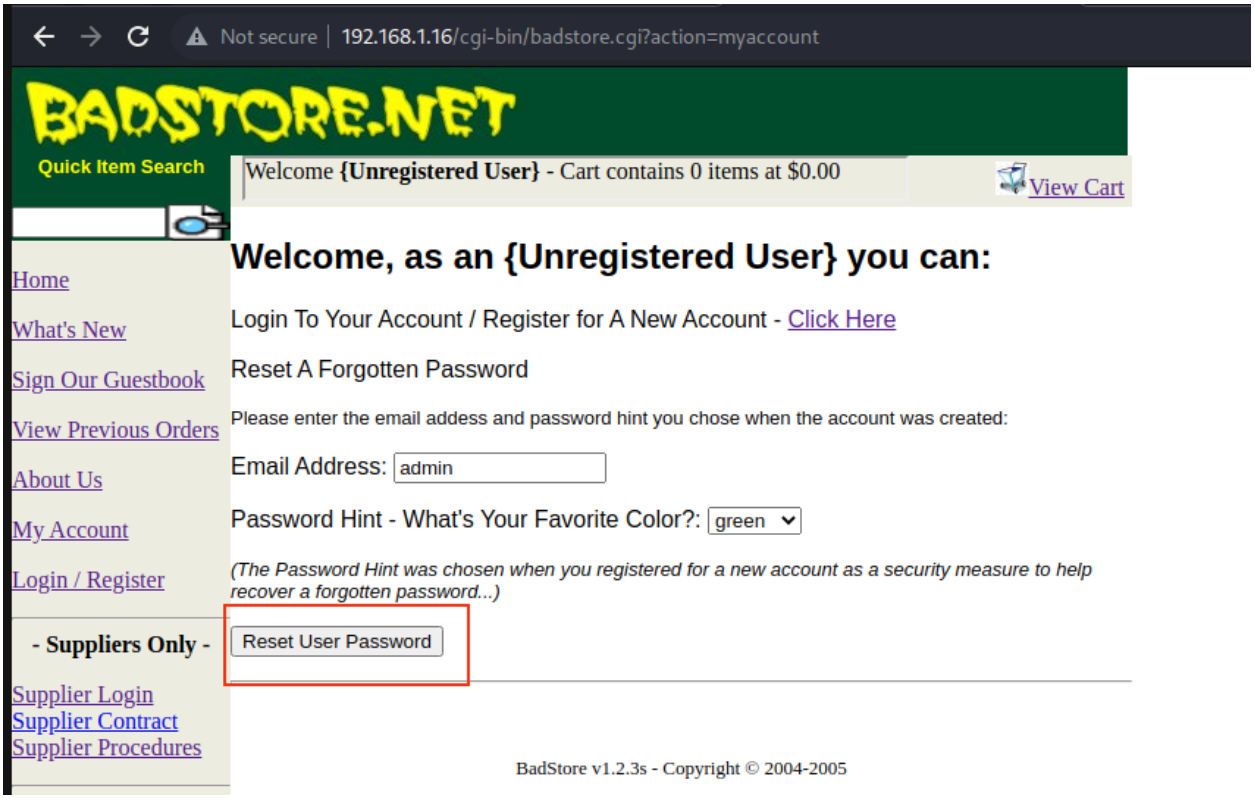

POST /cgi-bin/badstore.cgi?action=order HTTP/1.1
Host: 192.168.1.16
Content-Length: 79
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://192.168.1.16
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/107.0.5304.63 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8
,application/signed-exchange;v=b3;q=0.9
Referer: http://192.168.1.16/cgi-bin/badstore.cgi?action=submitpayment
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: SSOid=YWJjOGFiYy5jb206ODFkYzliZGI1MmQwNGRjMjAwMzZkYmQ4MzEzZWQwNTU6YWJjOlU%3D%0A; CartID=
1682990406%3A1%3A1.5%3A1000
Connection: close
email=abc%40abc.com&ccard=2349239423942394&expdate=05%2F25&subccard=Place+Order

Insecure Password Recovery

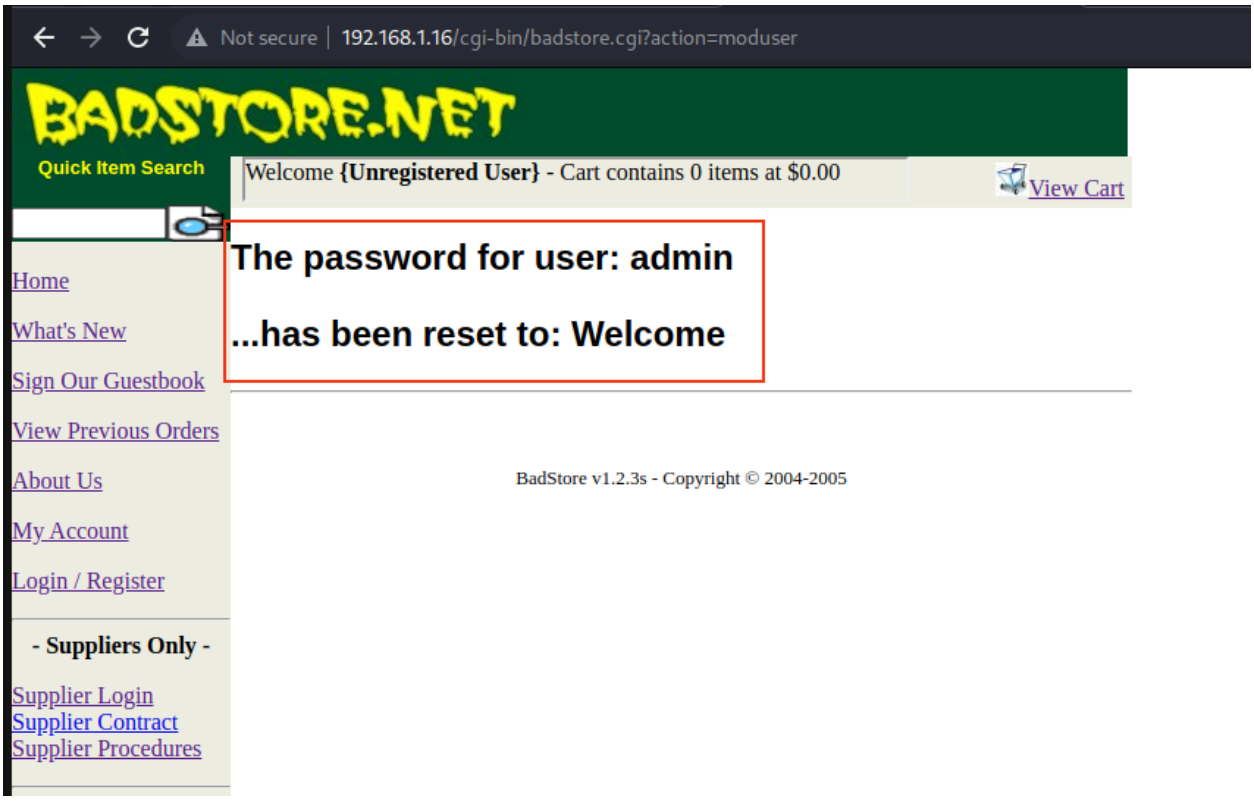
We also found out that by pressing the “Register” button, we would be led to a “Password Reset” page.



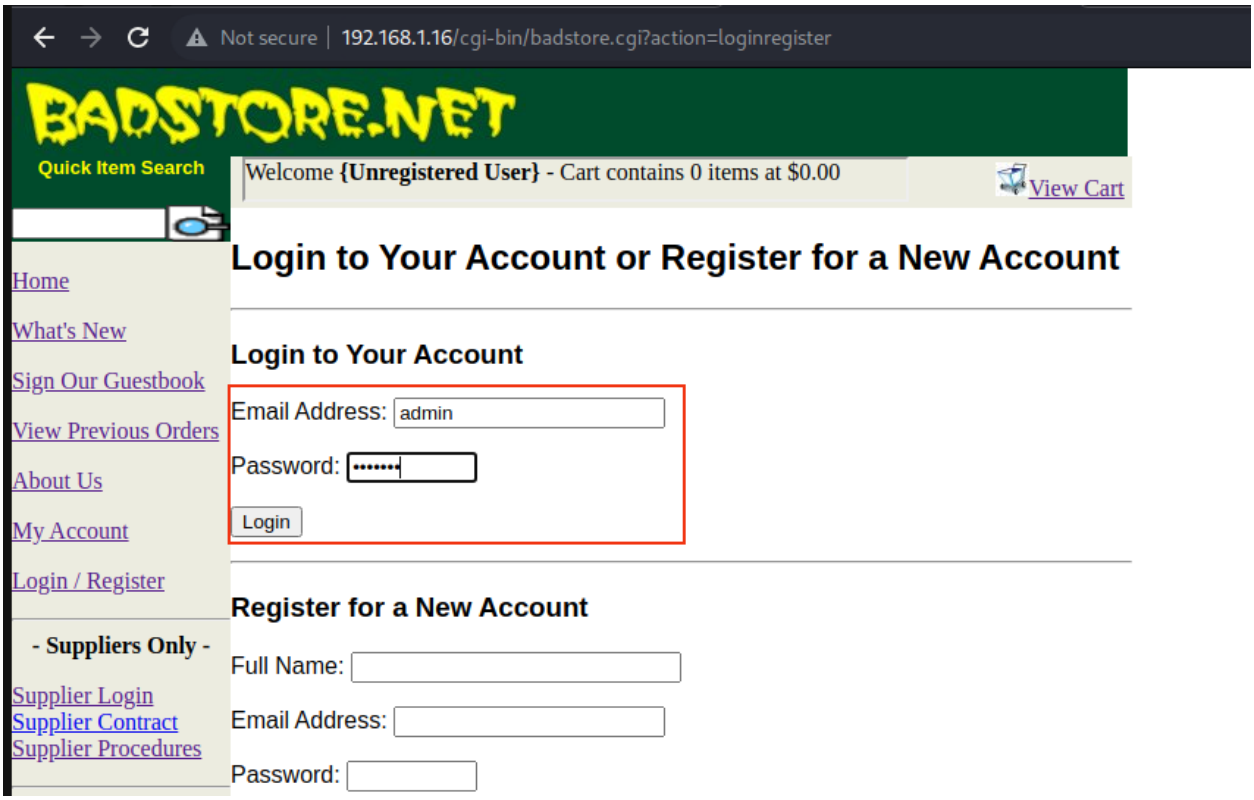
This allowed us to reset the “admin” user’s password by providing just the email.



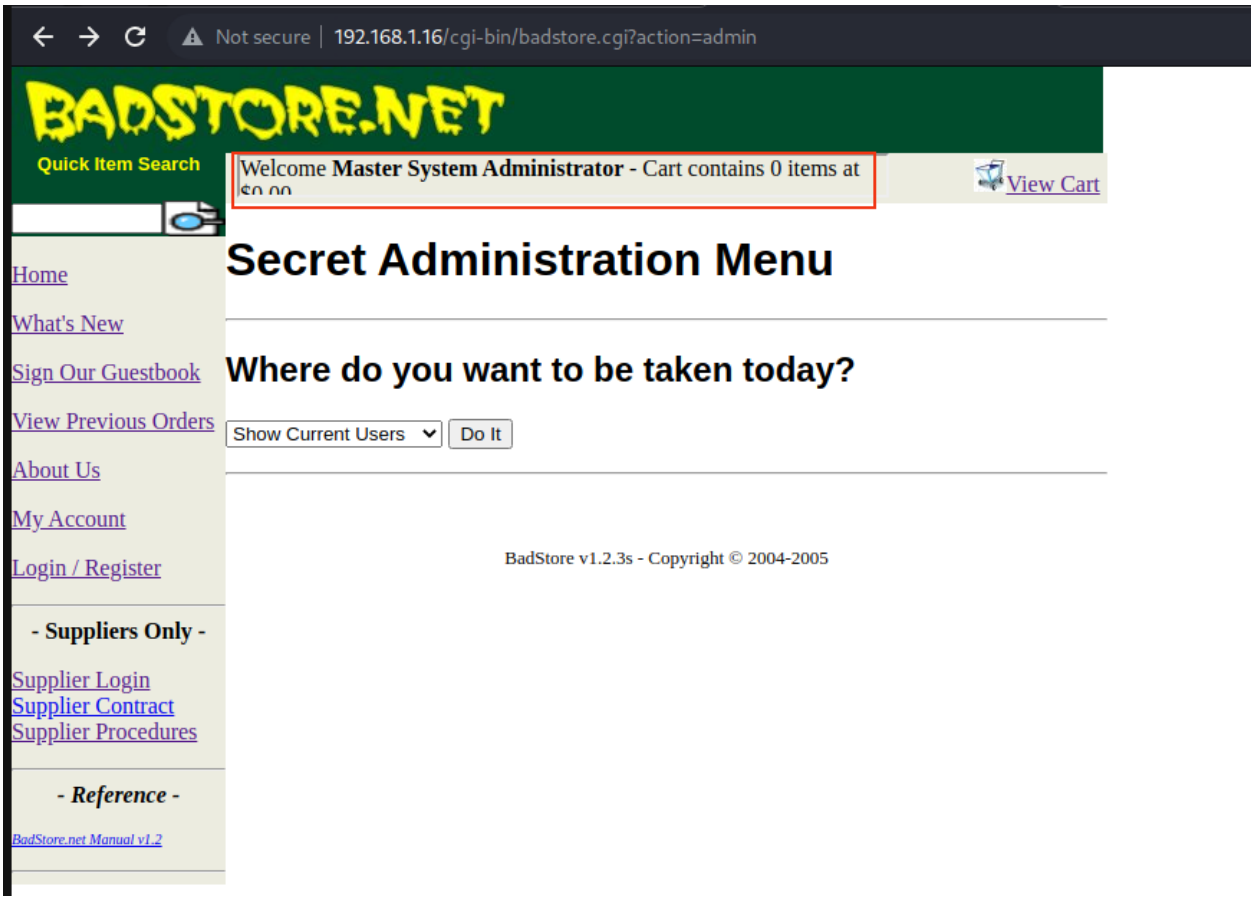
The reset password is always the same string “Welcome”.



We tried the “admin” email and password “Welcome”.

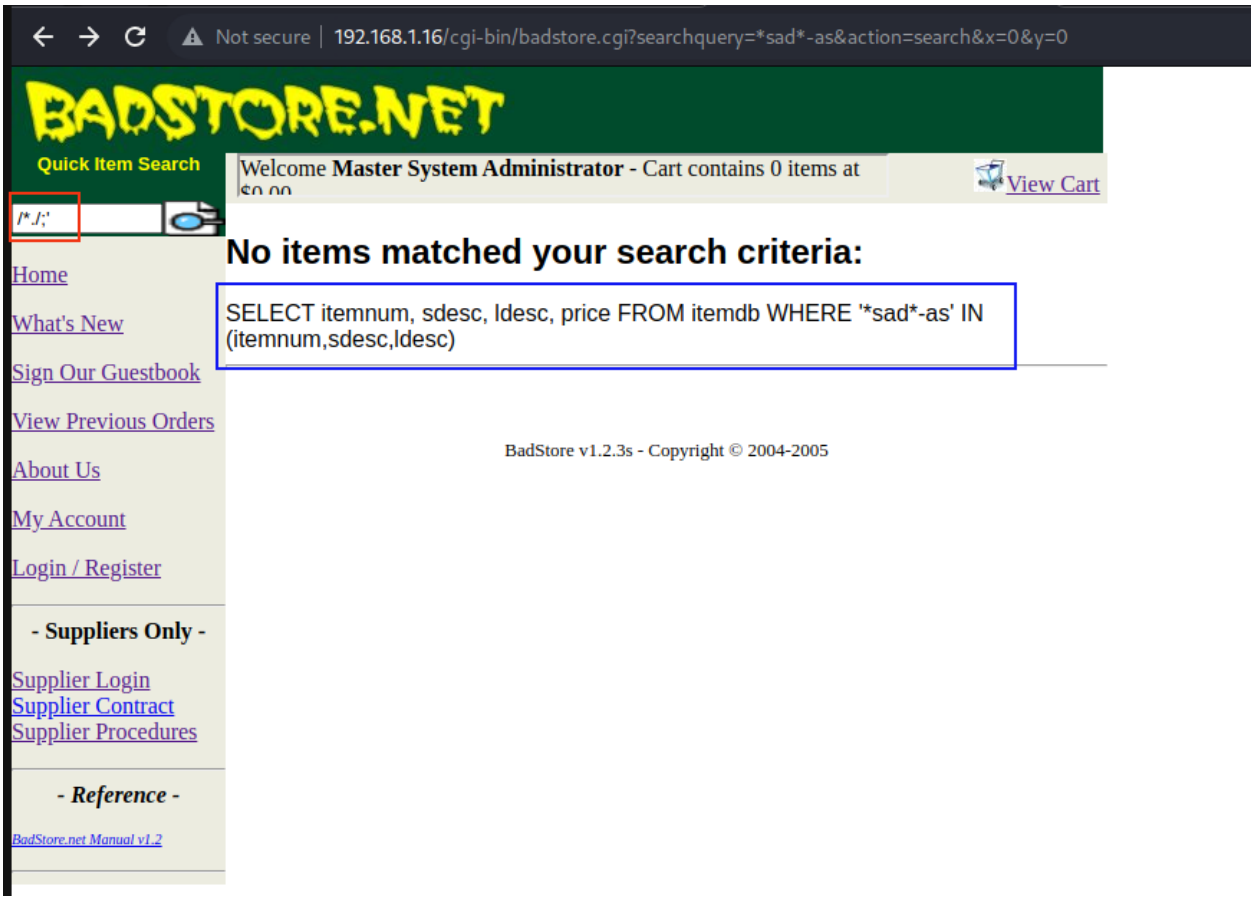


We got access to the “Admin” account.



SQL Injection

The application also contained SQL Injection vulnerabilities. We confirmed it through the “search” field by inputting some symbols.



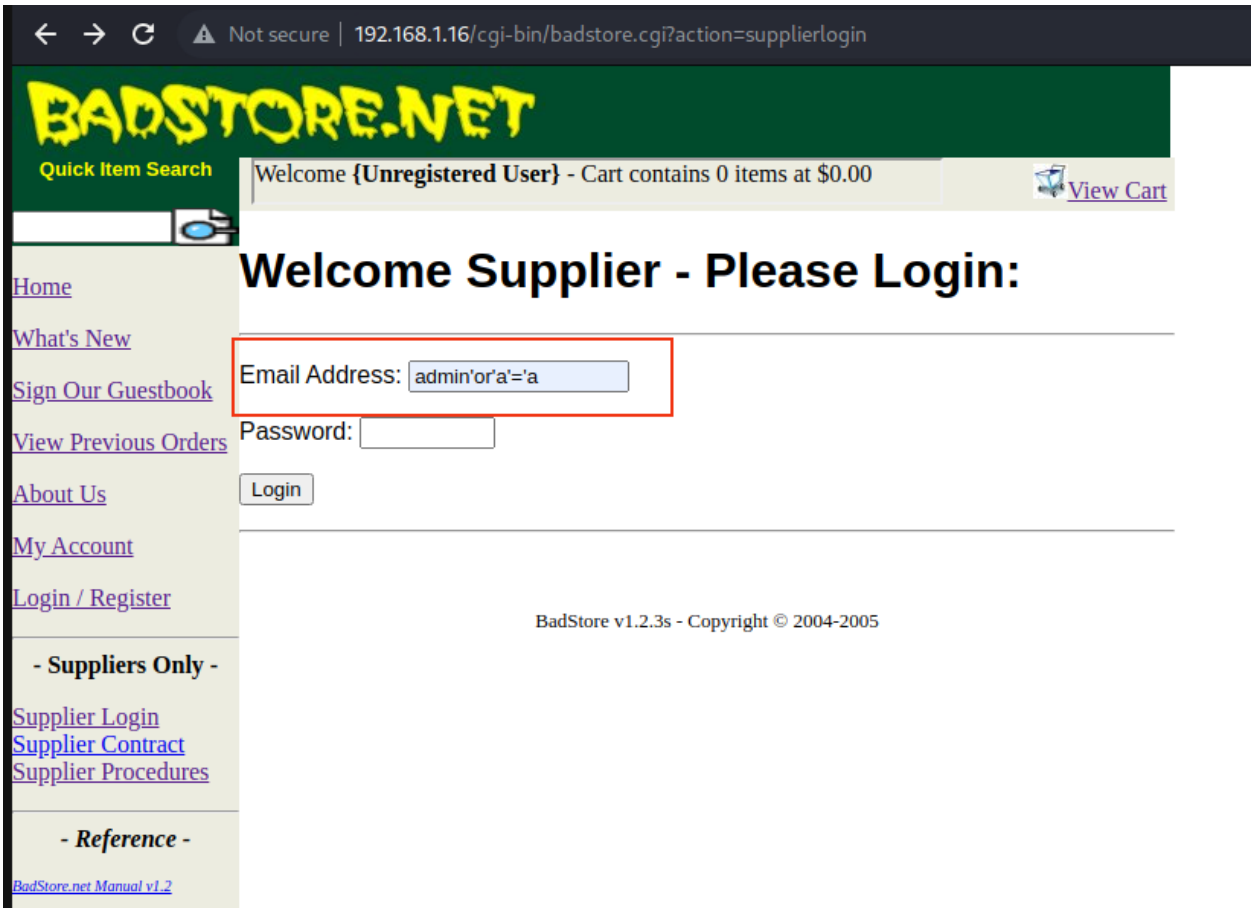
We tried bypassing the admin login using SQLi and got the access.

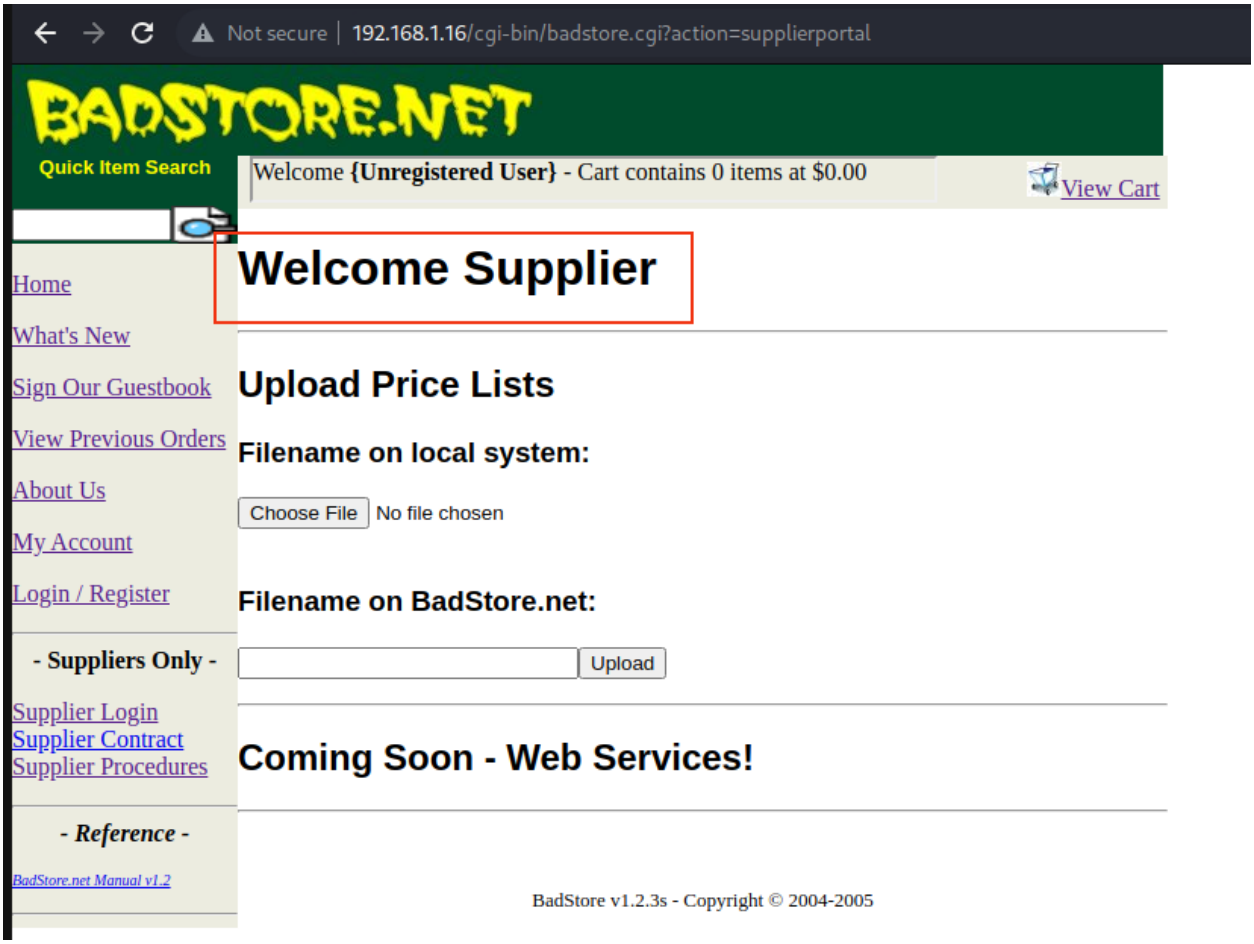
```
admin'or'a'='a
```


Not secure | 192.168.1.16/cgi-bin/badstore.cgi?action=loginregister
BADSTORE.NET
Quick Item Search
Welcome {Unregistered User} - Cart contains 0 items at $0.00
View Cart
Home
What's New
Sign Our Guestbook
View Previous Orders
About Us
My Account
Login / Register
- Suppliers Only -
Supplier Login
Supplier Contract
Supplier Procedures
- Reference -
BadStore.net Manual v1.2
Login to Your Account or Register for a New Account
Login to Your Account
Email Address: admin'or'a'='a
Password:
Login
Register for a New Account
Full Name:
Email Address:
Password:
Password Hint - What's Your Favorite Color?: green
(The Password Hint is used as a security measure to help recover a forgotten password. You will need both your email address and this hint to access your account if you forget your current password.)
Register

We also checked for SQLi at the “Suppliers” Login page and got access.

```
admin'or'a'='a
```

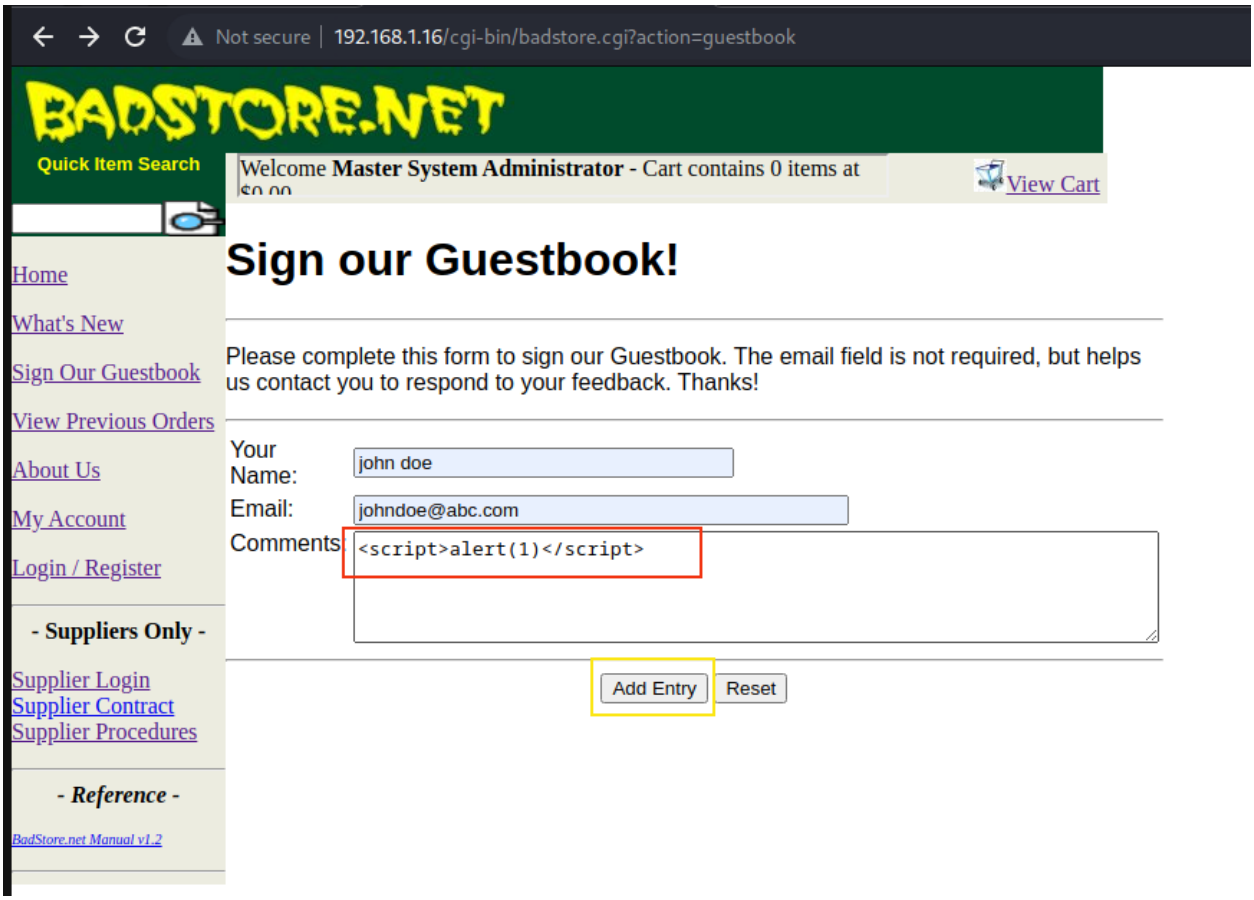

Not secure | 192.168.1.16/cgi-bin/badstore.cgi?action=supplierlogin
BADSTORE.NET
Quick Item Search
Welcome {Unregistered User} - Cart contains 0 items at $0.00
View Cart
Home
What's New
Sign Our Guestbook
View Previous Orders
About Us
My Account
Login / Register
- Suppliers Only -
Supplier Login
Supplier Contract
Supplier Procedures
- Reference -
BadStore.net Manual v1.2
Welcome Supplier - Please Login:
Email Address: admin'or'a'='a
Password:
Login
BadStore v1.2.3s - Copyright © 2004-2005



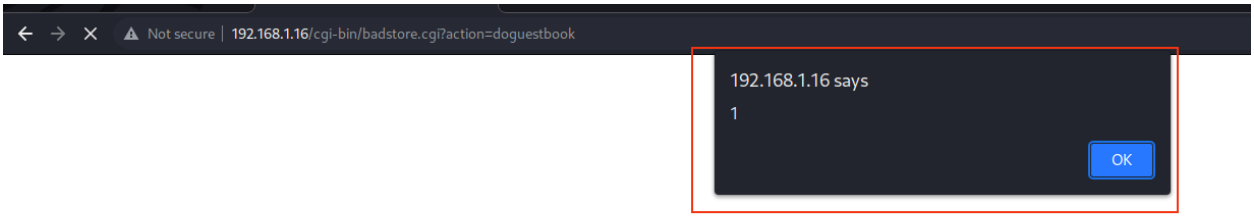
Cross Site Scripting

We checked the site for XSS by inputting an XSS payload in the guestbook form.

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

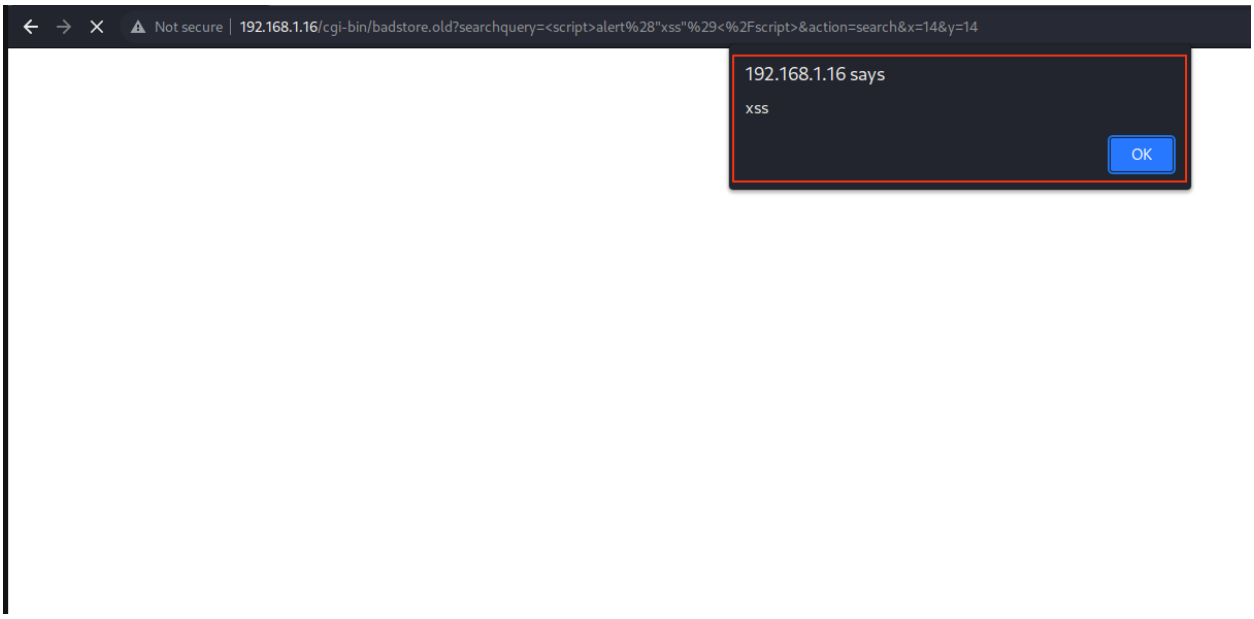


Pwned!



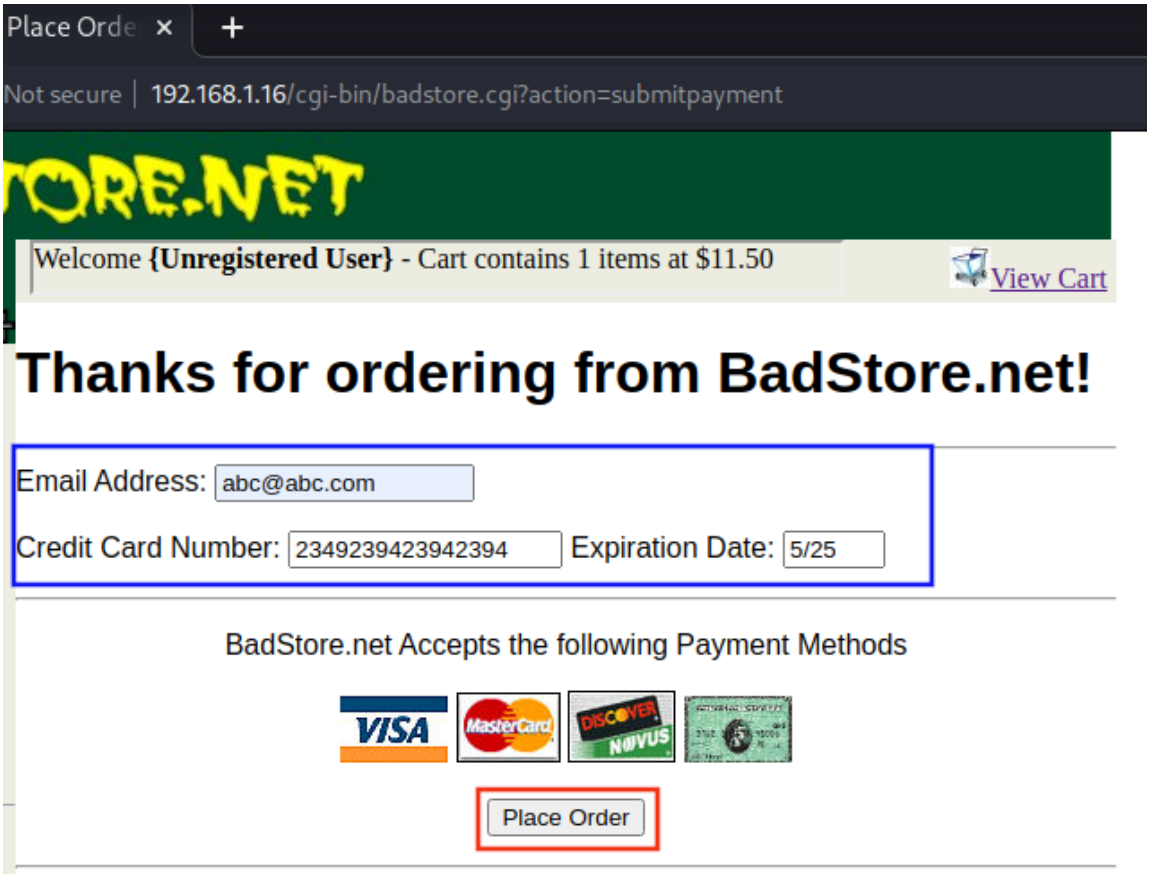
Same with the “search” field and an XSS vulnerability was present.

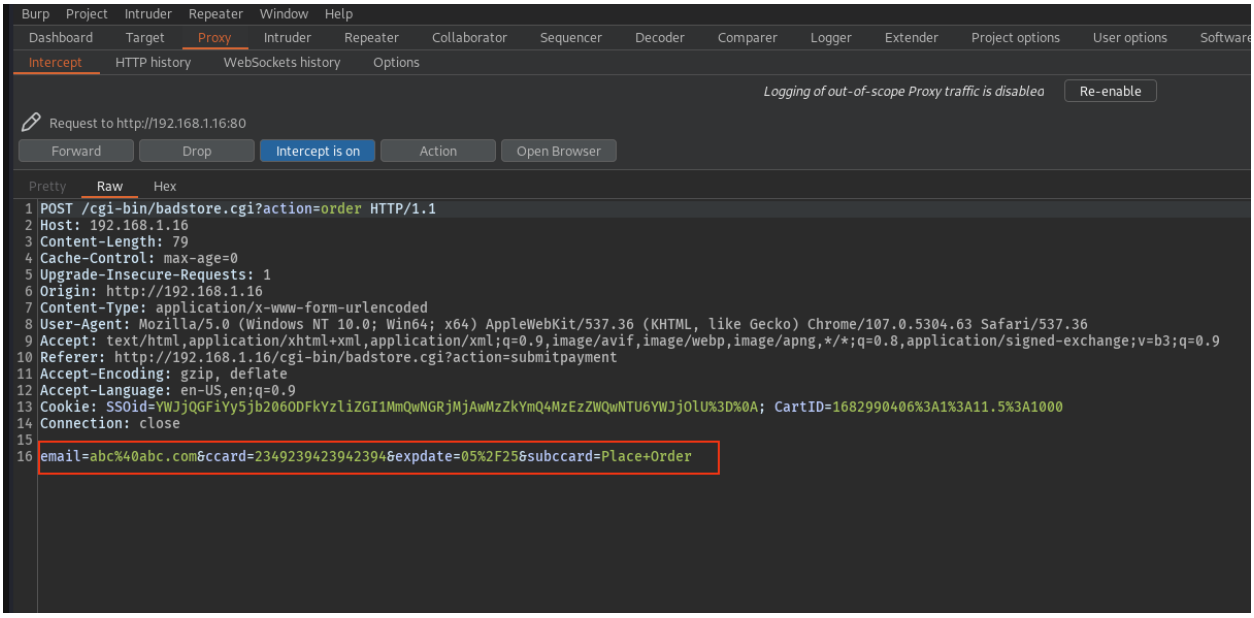
```
<script>alert("xss")</script>
```



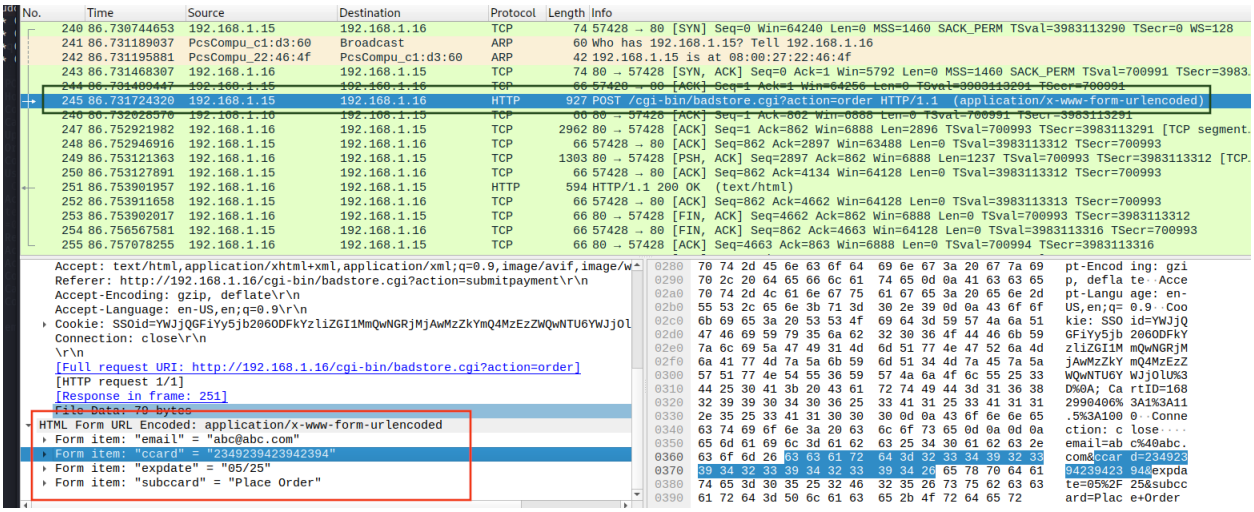
Insecure Communications

We also found out earlier that the traffic sent to and received from the application is “Unencrypted”. Hence, We confirmed it using “Burpsuite” and “Wireshark”.



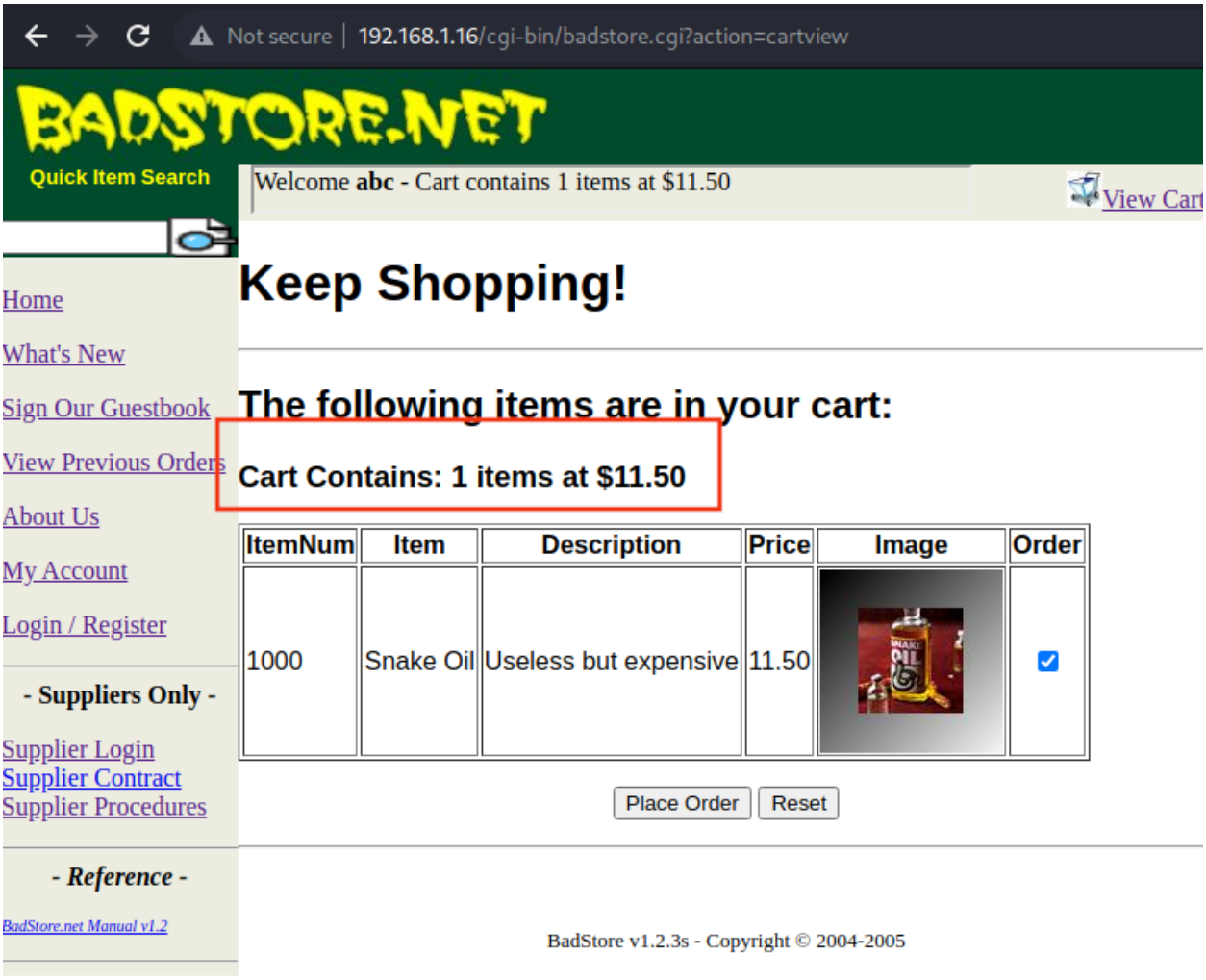


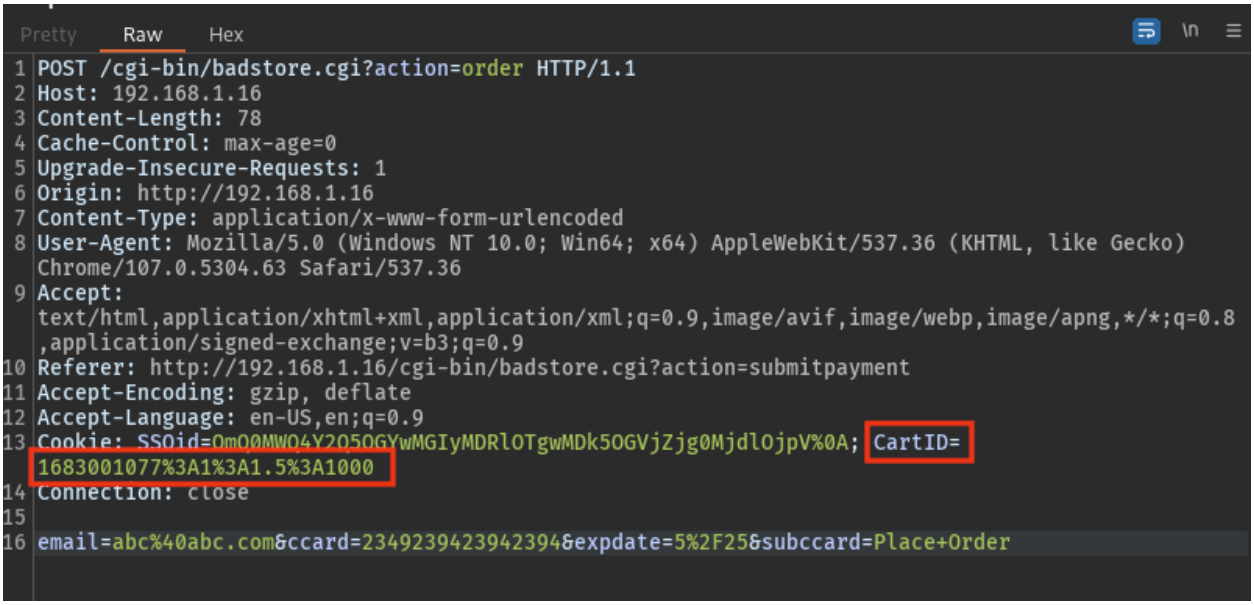
We can easily see the Personally Identifiable Information and Credit Card number of the user.

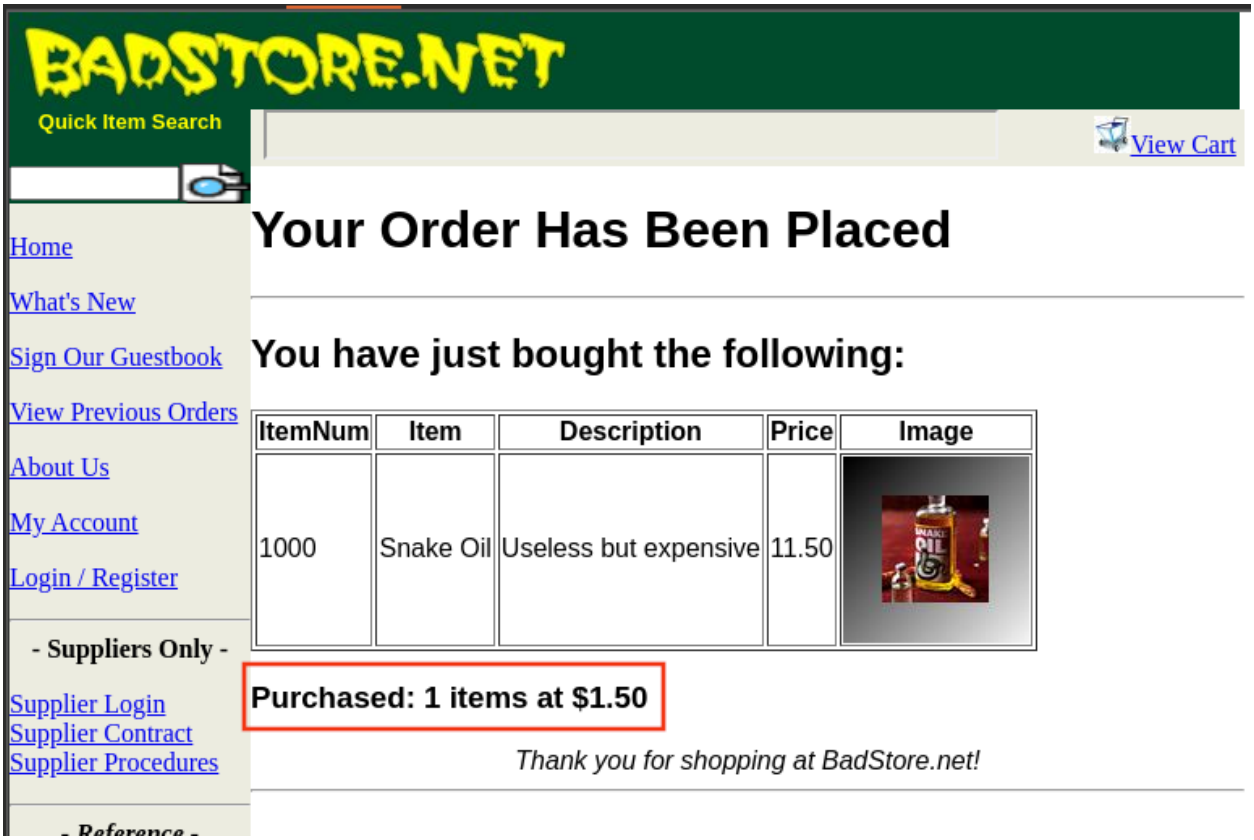


Business Logic Vulnerability

A Business Logic Vulnerability was also present in the Web app which allowed us to modify the price of items in the “cart”, to get discounted price on order.

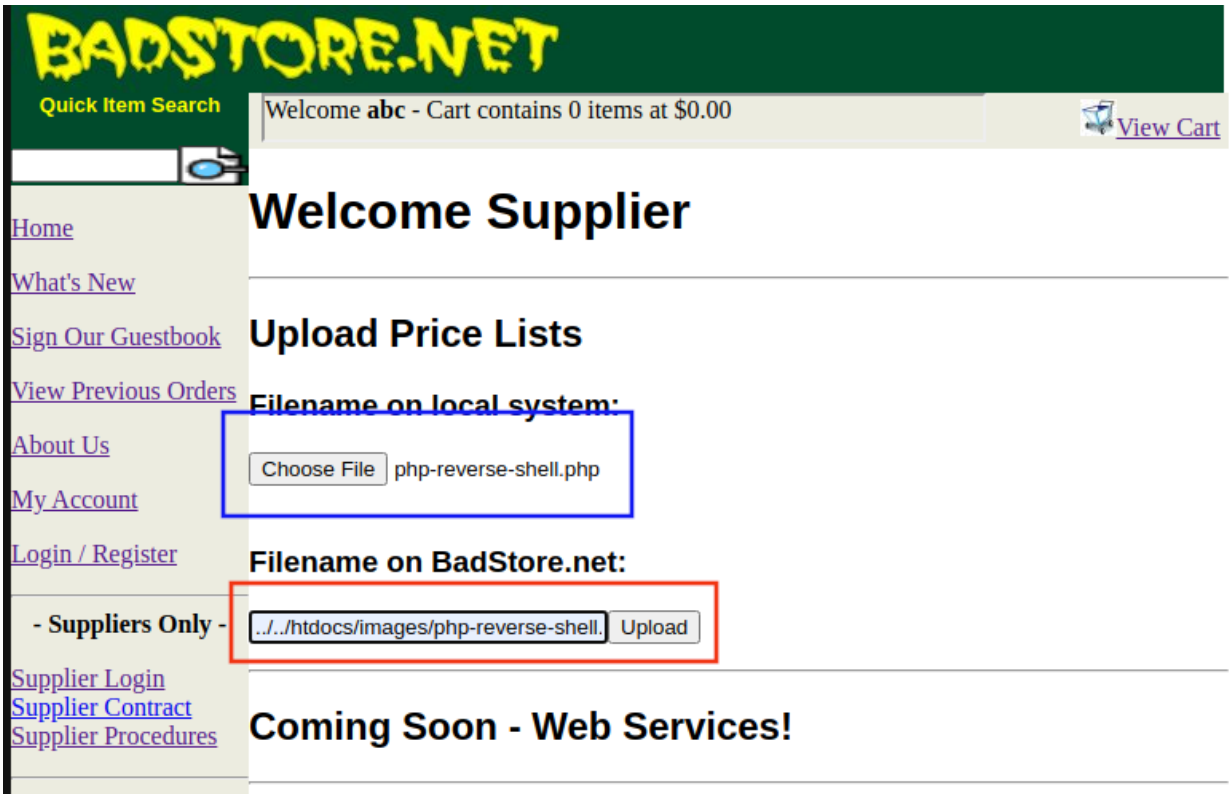

Not secure | 192.168.1.16/cgi-bin/badstore.cgi?action=cartview
BADSTORE.NET
Quick Item Search
Welcome abc - Cart contains 1 items at $11.50
View Cart
Home
What's New
Sign Our Guestbook
View Previous Orders
About Us
My Account
Login / Register
- Suppliers Only -
Supplier Login
Supplier Contract
Supplier Procedures
- Reference -
BadStore.net Manual v1.2
Keep Shopping!
The following items are in your cart:
Cart Contains: 1 items at $11.50
ItemNum
Item
Description
Price
Image
Order
1000
Snake Oil
Useless but expensive
11.50
Place Order
Reset
BadStore v1.2.3s - Copyright © 2004-2005

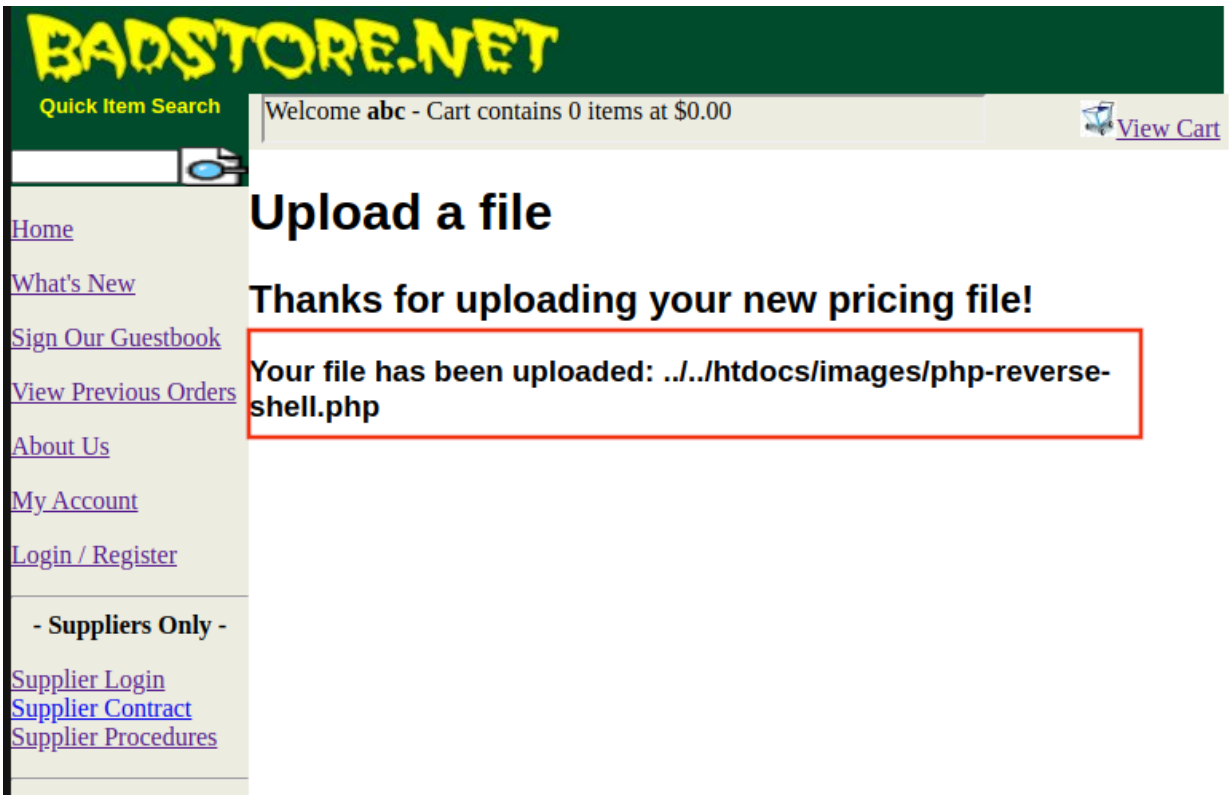

Pretty
Raw
Hex
\n
POST /cgi-bin/badstore.cgi?action=order HTTP/1.1
Host: 192.168.1.16
Content-Length: 78
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://192.168.1.16
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/107.0.5304.63 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Referer: http://192.168.1.16/cgi-bin/badstore.cgi?action=submitpayment
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: SSOid=OmQ0MWQ4Y2Q5OGYwMGIyMDRlOTgwMDk5OGVjZjg0MjdlOjpV%0A; CartID=1683001077%3A1%3A1.5%3A1000
Connection: close
email=abc%40abc.com&ccard=2349239423942394&expdate=5%2F25&subccard=Place+Order



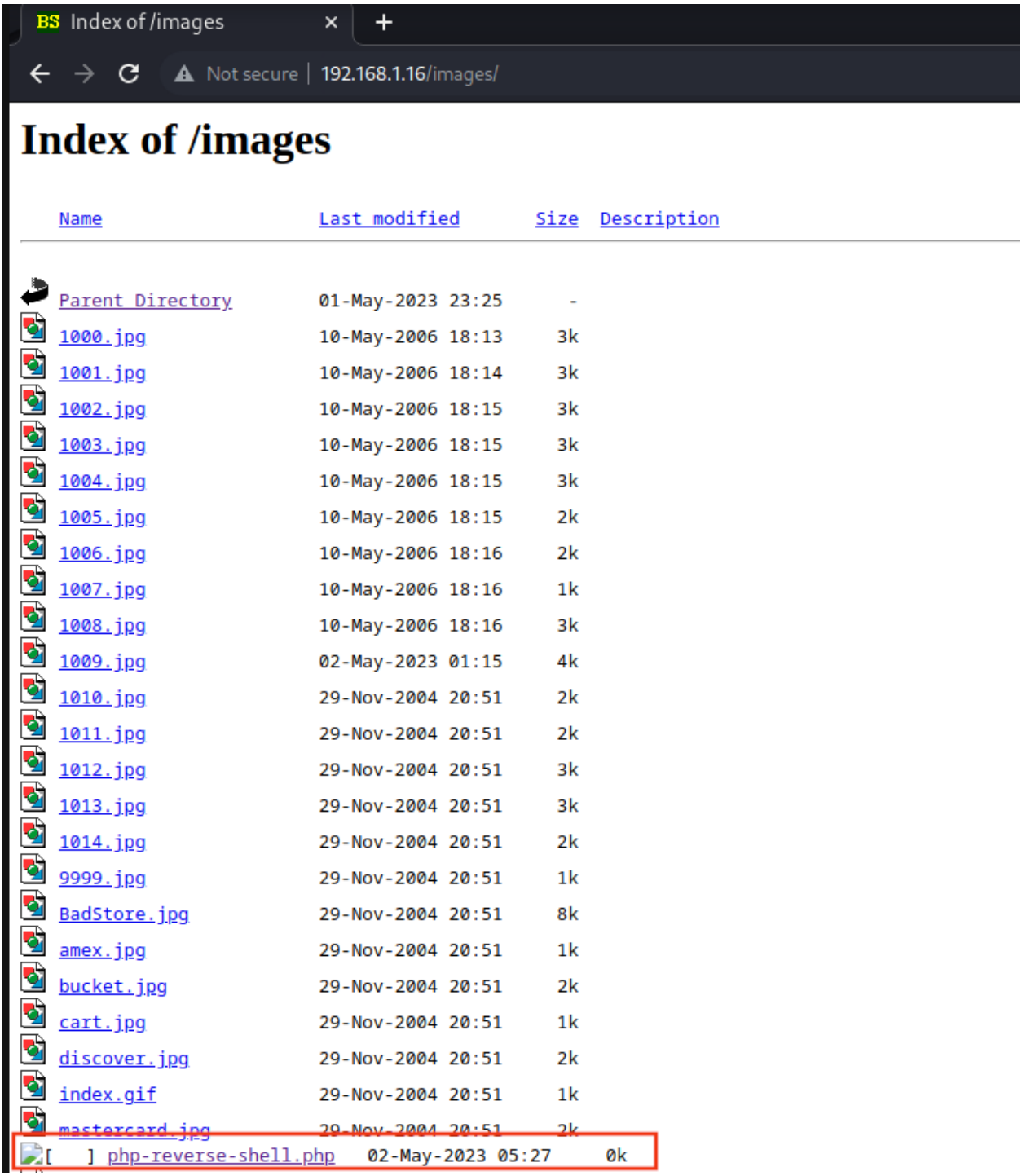
Unrestricted File Upload

The “Supplier” page also had the functionality for file upload. We uploaded a malicious php shell on it and it was successful, as there were no restrictions on which “file types” you can upload on the web server.





We could also list down our file by providing the “path” for the file.



Recommendations

1. Broken Access Control:
    - Implement strong access controls and limit the privileges of users and applications.

- Use role-based access control (RBAC) to enforce access policies based on user roles.
- Enforce strong password policies and multi-factor authentication to prevent unauthorized access.
- Regularly review access logs and audit trails to detect any suspicious activity.

2. Directory Listing:
    - Disable directory listing by default.
    - Implement proper access controls to restrict access to sensitive files and directories.
    - Use web server configuration settings to disable directory listing and enable directory access only for authorized users.
    - Regularly monitor access logs and audit trails to detect any unauthorized access attempts.
3. Cryptographic Failure:
    - Use strong cryptographic algorithms and protocols to protect data in transit and at rest.
    - Implement proper key management and storage practices to ensure the security of cryptographic keys.
    - Regularly update cryptographic libraries and protocols to prevent vulnerabilities and weaknesses.
    - Use secure randomness sources for generating cryptographic keys and nonces.
4. Insecure Password Recovery:
    - Implement secure password recovery mechanisms, such as security questions, email verification, or phone verification.
    - Use multi-factor authentication to verify user identity before resetting passwords.
    - Enforce strong password policies, such as requiring complex passwords and implementing password expiration policies, to reduce the likelihood of successful password guessing or cracking attacks.
    - Monitor password recovery logs and audit trails to detect any suspicious activity.
5. SQL Injection:
    - Implement proper input validation and sanitization techniques to prevent SQL injection attacks.
    - Use parameterized queries, input validation libraries, and output encoding to ensure that user input is properly validated and sanitized before being processed.
    - Regularly update application frameworks and libraries to prevent SQL injection vulnerabilities and exploits.
    - Use secure coding practices to prevent SQL injection vulnerabilities from being introduced during development.

6. Cross Site Scripting (XSS):
    - Implement proper input validation and sanitization techniques to prevent XSS attacks.
    - Use output encoding to prevent untrusted data from being interpreted as code.
    - Implement Content Security Policy (CSP) to restrict the execution of untrusted code.
    - Regularly update application frameworks and libraries to prevent XSS vulnerabilities and exploits.
7. Insecure Communications:
    - Use encryption to protect data transmitted over the network.
    - Implement secure communication protocols, such as HTTPS, SSL, or TLS, which encrypt data before transmission.
    - Use secure encryption ciphers to ensure that transmitted data is properly protected.
    - Avoid transmitting sensitive data via GET requests, and use secure cookies to prevent session hijacking attacks.
8. Unrestricted File Upload:
    - Implement proper input validation and sanitization techniques to ensure that only authorized files are uploaded.
    - Check file types, file size, and implement file upload restrictions based on user roles and privileges.
    - Use secure file storage and access controls to ensure that uploaded files are properly protected from unauthorized access or modification.
    - Regularly scan uploaded files for malware and viruses.
9. Business Logic Vulnerabilities:
    - Implement proper input validation and sanitization techniques to prevent business logic vulnerabilities.
    - Use secure coding practices and regularly review code to detect any business logic flaws.
    - Implement access controls and privilege escalation policies to prevent unauthorized access to sensitive data or features.
    - Regularly perform threat modeling and security assessments to identify and address potential business logic vulnerabilities.