Web Application Penetration Testing Report

Robert Barbulescu, BSc (Hons) Cyber Security

Number of words: 6531 (Main Report: 3908; Appendices: 2623)

Contents

I. Executive Summary

We conducted a comprehensive security assessment of BadStore.net in order to determine existing vulnerabilities and establish the current level of security risk associated with the web application and services in use. This assessment used penetration testing techniques to provide BOZBITS PLC management with an understanding of the risks and security posture of their application.

SUMMARY OF RESULTS

After conducting the security review, we present the following main concerns:

- The web application's overall security is compromised
- Administrator access can be obtained through several methods with relatively low complexity
- The application stores unencrypted sensitive user information such as passwords, email addresses and IP addresses
- The application stores unencrypted financial information such as credit cards and expiry dates
- The current version of software and services are outdated, which affects the integrity of the application
- The network traffic is not secure and can result in leakage of confidential information

Due to the critical state of the security, a high risk exists of sensitive information being already compromised. It is our recommendation to notify current users to change passwords and contact their financial institutions regarding the credit cards used on the website. Furthermore, the web application is in non-compliance with the current GDPR legislation which can result in a fine of up to 20 million pounds, or 4% of the company's annual revenue.

REMEDIATION STRATEGY

We recommend the website to be immediately made offline and implement the following measures:

- Contact providers and update all software and services
- Disable outdated/vulnerable protocols and replace with secure versions
- Implement cryptographic methods to secure the transportation and storage of sensitive information
- Validate all user input on the website
- Fix the current broken Password Reset platform

At the current state, the company is at risk for fines and public image damage due to the unsecure platform. A complete analysis of the application's code and functionality is required, with necessary modification of the affected parts we presented in the report. Testing should also be undertaken for each part of the development to ensure all components are fully operational and secure before publishing.

II. Scoping Document [1]

This document serves as an AGREEMENT between the "Consultant" ROBERT BARBULESCU and the "Client" BOZBITS PLC and shall be effective as of 17th of FEBRUARY 2020.

RECITALS

WHEREAS, Consultant is engaged in the business of providing Cyber Security Consulting and

WHEREAS, Client wishes to utilize the services of Consultant in the form of a Black Box Penetration Test

NOW, THEREFORE, Consultant and Client agree as follows:

1. Scope of Services

   Consultant will provide the following work ("Consultant's Work" or the "Work") for Client:

   i. A security review of the Client's web application by
   ii. Analysing the given web application only via port 80 and/or port 443, and
   iii. Producing a report describing and analysing the processes used, the vulnerabilities discovered as well as the exploits used, and
   iv. Producing an executive summary presenting an overview of the findings.

2. Time and Cost Estimates

   This penetration test will cover one web application, with no additional physical or social engineering testing required, only through port 80 and/or port 443. The test will take approximately 4 weeks to complete, with the deadline set for 29th of APRIL 2020.

3. Deliverables

   The deliverable described hereafter (the "Deliverables") will be provided to the Client in final form upon completion of the Work described in the Scope of Services. Such Deliverables include:

   i. Executive summary,
   ii. Technical report.

4. Price and Payment Terms

   Client agrees to cooperate with Consultant's reasonable requests with respect to the scheduling and performance of the work and to pay Consultant for Consultant's Work as follow:

   i. A one-time payment of a fixed amount of £4000, with a breakdown of
   ii. 100 hours over a 4 weeks period, at a price of
   iii. £40/per hour.

   Any material change in the Services or Deliverables described above requires a written change order signed by the parties to the Agreement. Such change order may include an adjustment to the price or delivery dates.

5. Invoices

Services will be invoiced upon completion of the penetration test.

6. Payment

Payment is due fifteen (15) days after date of invoice. Client may not withhold any amounts due hereunder and Consultant reserves the right to pursue legal action if amounts are not paid when due. Any late payment will be subject to any costs of collection (including reasonable legal fees) and will bear interest at the rate of one (1) percent per month or fraction thereof until paid.

7. Project Organization and Personnel Requirements

Any changes to the scope present in this agreement due to any unforeseen circumstances shall be made aware to all parties in due time. The main reporting personnel for the Consultant shall be the CEO of the Client's company.

8. Confidential Information

All information relating to all relevant parties that is known to be confidential or proprietary, or which is clearly marked as such, shall be held in confidence by all parties and shall not be disclosed or used by any party except to the extent that such disclosure or use is reasonably necessary to the performance of the Work.

9. Limitation of Liability

In no event shall Consultant be liable for any loss of profit or revenue by Client, or for any other consequential, incidental, indirect or economic damages incurred or suffered by Client arising as a result of or related to Consultant's Work, whether in contract, tort, or otherwise, even if Client has advised of the possibility of such loss or damages.

IN WITNESS WHEREOF, the parties have executed this Agreement on the date first stated above.

Consultant	Client
Robert Barbulescu	BozBits PLC By: Authorized Designee
Title: Penetration Tester	Title: CEO
Signature_____	Signature_____

III. Technical Report

DISCLAIMERS

The information presented in this document is provided as is and without warranty. Vulnerability assessments are a “point in time” analysis and unpredictable as such it is possible that something in the environment could have changed since the tests in this report were run. Also, it is possible that new vulnerabilities may have been discovered since the tests were run. For this reason, this report should be considered a guide, not a 100% representation of the risk threatening your systems, networks and applications.

1 Introduction

We've been employed to conduct a web penetration test in order to assess the security and exposure of a given operating system hosting a web application, attempts were made at identifying and exploiting security weaknesses that will allow an external attacker to breach the security of the web application and gain unauthorized access to sensitive and confidential information.

1.1 Purpose

For the purpose of this security review, we were provided with a virtual machine containing the operating system. This web platform is complete and active on the public domain and we were given access to a copy in order to perform our security test within a safe environment. The testing efforts took place in March and April 2020, preliminary findings were provided under a separate document, this report is being presented to show our full process and results, as well as recommend remediation strategies where necessary.

1.2 Scope

The scope of this test was limited to a single web application using only URL/PORT 80/PORT 443. We approached this as a Black Box Test, no information on our target was given, the intent was to simulate an adversary with no internal knowledge.

1.3 Summary of Results

By performing a detailed web application penetration testing against BOZBITS PLC, we identified several issues of concern that, if exploited by a malicious party, would have a dramatic effect on the operations and confidentiality of the company. Overall, we found the application to be insecure. Throughout this report we provide description and remediation strategies of each testing category, with more information on negative findings.

The table below shows a summary of the vulnerabilities identified based on their impact on the system:

Issue number	Vulnerability	Impact
Issue 1	*SQL Injection*	HIGH
Issue 2	*Cross-Site Scripting*	HIGH
Issue 3	*Weakly Encrypted Credentials*	HIGH
Issue 4	*Unencrypted Network Traffic*	HIGH
Issue 5	*Web Parameter Tampering*	HIGH
Issue 6	*Broken Password Reset Platform*	HIGH
Issue 7	*Sensitive Data Exposure*	HIGH
Issue 8	*Robot.txt*	MEDIUM
Issue 9	*Directory Exposure*	MEDIUM
Issue 10	*Secret Admin Platform*	LOW

Other vulnerabilities discovered but not attempted such as: Heartbleed, SWEET32, POODLE, Buffer Overflow and Cookie Manipulation remain relevant and should be mitigated to ensure the integrity of the system.

2 Methodology

Throughout this security review, we conducted all activities in accordance with the NIST SP 800-115 [2] recommendations and methodology.

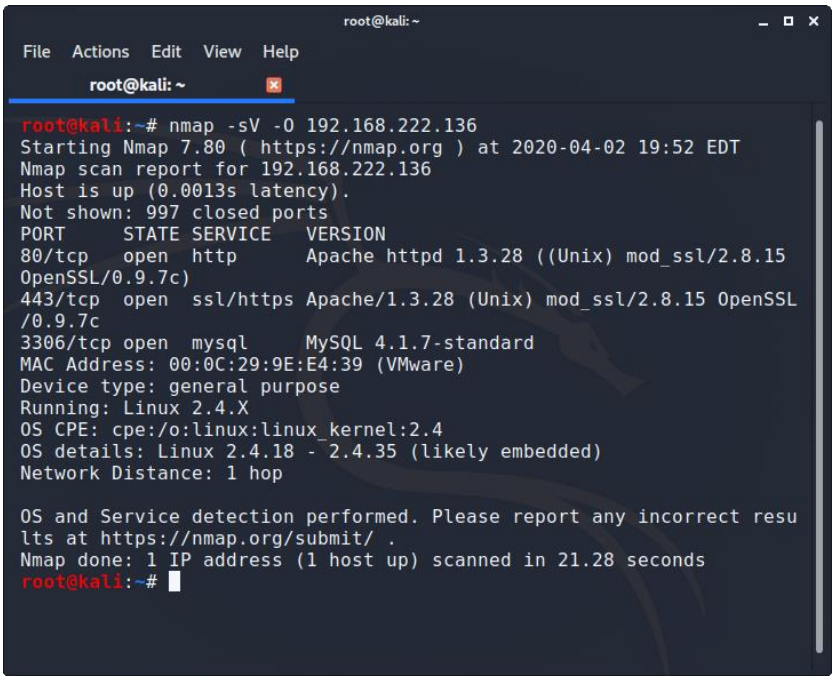
2.1 Rationale

During this examination we have used the NIST methodology as it provides both technical and operational guidelines necessary during our security review. NIST is globally recognised and offers a detailed guidance with the necessary explanations for many major components of security testing.

3 Attack Narrative

In order to identify the attack surface, we used Nmap, a network mapper and scanner utility to perform a service detection and operating system scan on our target (**Figure 1**). For the full process of discovering the IP address, please see **Appendix 6.1**.

Figure 1 - Nmap service and OS detection.



The scan performed (**Figure 1**) returned the following information: our target runs a Linux based OS (Operating System), Apache 1.3.28 on ports 80 and 443, and a MySQL server on port 3306. The MySQL server will not be attempted, as port 3306 is out of scope.

3.1 Website

We continue by looking at the website (**Error! Reference source not found.**) in order to see what type of functionalities our web application has.

Figure 2 - BadStore.net Home page.

After an initial look, we find the website to be a platform for online purchases with two login portals (**Suppliers & Regular Logins**), the regular login form provides also a **Register** form for new users. For registered users we can see a section called '**My Account**', which allows you to reset your password. Another field for any type of user is '**Sign Our Guestbook**' which provides a form in where website visitors can post messages. Other functionalities include selecting products and adding them to the shopping cart, making card payments and placing orders, and viewing previous orders.

3.1.1 SQL Injection

SQL Injection is one of the most popular attacks used to gain sensitive information from a web application (please see **Appendix 6.2.1** for more information) and it can be performed on any parts of the website that accepts user input. Two Boolean conditions that will allow SQL injection were initially detected during an automated scan we performed using OWASP ZAP (please see **Appendix 6.1.3**), a tool designed for testing web applications.

We attempted several other queries for the purpose of gaining administrative privileges on the two login platforms and created the table below showing successful and unsuccessful attempts along with the affected components:

Affected Component	Unsuccessful	Successful
Login to My Account	- admin - admin' - or 1=1 - admin or 1=1	- admin' or 1=1# - admin' or '1'='1
Supplier Login	- admin - admin' - admin or 1=1 - admin or 1'=1 - admin' or '1'='1	- admin' or 1=1#

Further documentation on the successful attempts is available in **Appendix 6.1.5**.

Important note: Any part of a web application that allows user input can result in a security risk, in order to avoid such attacks, we highly recommend validating and escaping all user input. Further recommendations in **Appendix 6.2.1**.

3.1.2 Cross-site Scripting (XSS)

Cross-site scripting (XSS) is a common attack technique used by hackers to distribute malicious code through web applications (for more information, please see **Appendix 6.2.2**). In order to check for XSS, we performed an automated scan using OWASP ZAP (**Appendix 6.1.3**), as well as several manual queries on parts of the websites that accept user input.

We were successful in exploiting several flaws, we used simple alert scripts to determine if the application would be affected by an XSS attack. The table below provides a breakdown of affected parts of the websites we discovered:

Affected Component	Script Used	Target URL
Password Reset	<script>alert(1);</script>	http://192.168.222.136/cgi-bin/badstore.cgi?action=moduser
Supplier Upload Portal	<img src=x onerror=alert(1);>	http://192.168.222.136/cgi-bin/badstore.cgi?action=supupload
Guestbook	<script>alert(1);</script>	http://192.168.222.136/cgi-bin/badstore.cgi?action=doguestbook

Documentation of our successful XSS attacks is available in **Appendix 6.1.6.**

The scripts used during the testing for XSS did not cause any damage or changes to the system as their purpose was only to verify the existence of the vulnerability. An attacker will use complex scripts to gain sensitive information or distribute malicious code through the website, as well as render the application unavailable or even change the HTML code.

Further evidence of XSS vulnerability, obtained during our OWASP ZAP scan, is available in **Appendix 6.1.3, paragraph 1.**

Important Note: In order to reduce/avoid risks of such an attack we recommend validating and filtering all user input (a more detailed remediation strategy available in **Appendix 6.2.2**).

3.1.3 Robots.txt

Another Nmap scan (**Figure 4**) confirms the presence of a **robots.txt** file. We used the command below to check for disallow entries in the **robots.txt** file by specifying the service (**-sC**), the port (**-p 80**) and the target (**192.168.222.136**).

Figure 4 - Nmap reveals robots.txt file.

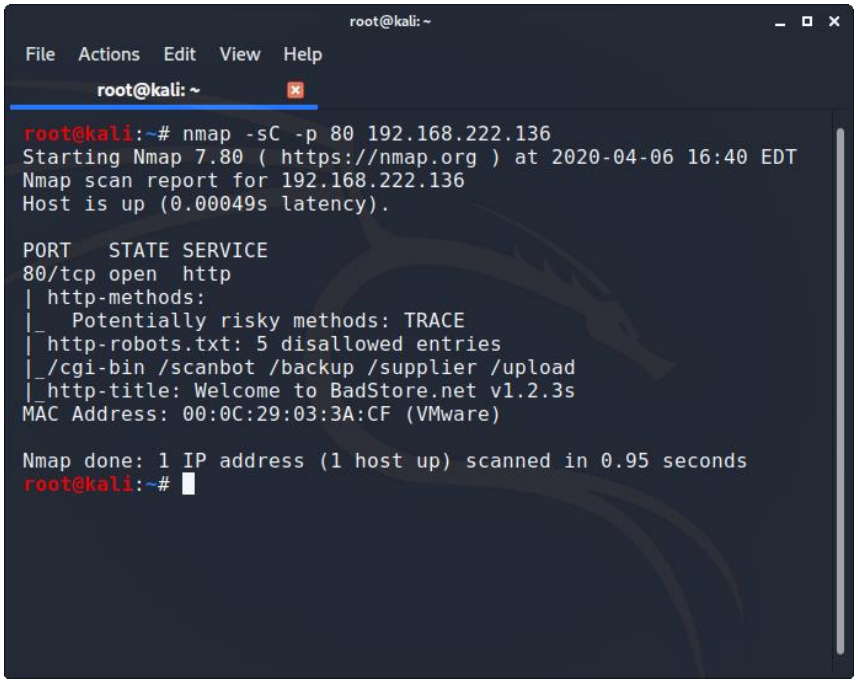
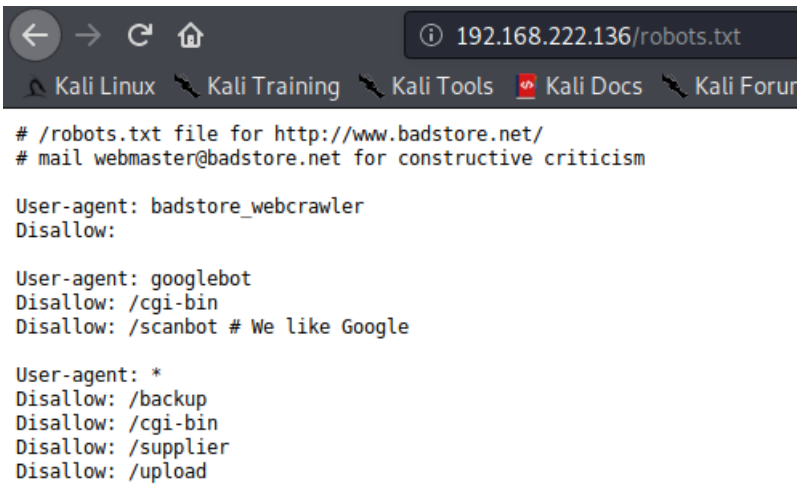


Figure 3 - Accessing robots.txt.



The scan was successful in discovering several entries. We continued by accessing the file through the browser (**Figure 3**) for inspection.

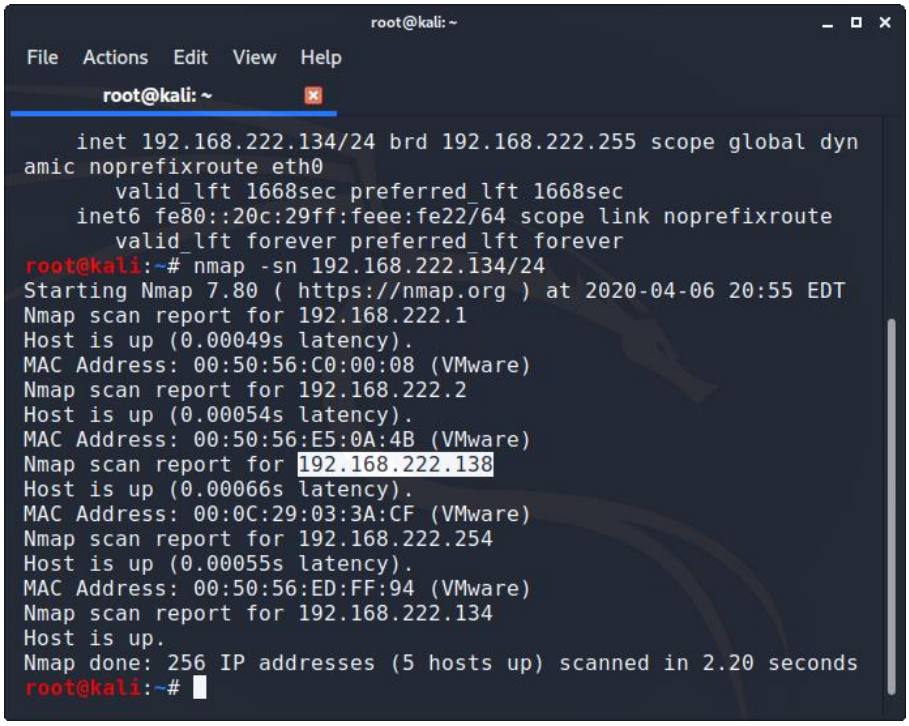
- The **User-agent** is name of the robot the file is providing access for,
- And **Disallow** specifies the parts that are not to be visited.

From **Figure 3** we can see that:

- **badstore_webcrawler** can retrieve all URLs,
- **googlebot** is only allowed **/cgi-bin** and **/scnbot**, and any other robot is denied access.

IMPORTANT: For the remaining of the testing we will be using a different machine, in consequence we had to repeat the process of discovering the target's IP address (Appendix 6.1.1). The new IP address of our target is: 192.168.222.138.

Figure 5 - Change of Environment.



3.1.4 Publicly Available Directories

Following the discovery of the robot.txt file, we were able to access the /supplier directory with no privileges, as well as the files inside the directory. On the future, we recommend disabling the directory listing feature to avoid such information disclosure.

Figure 7 - Access to /supplier directory.

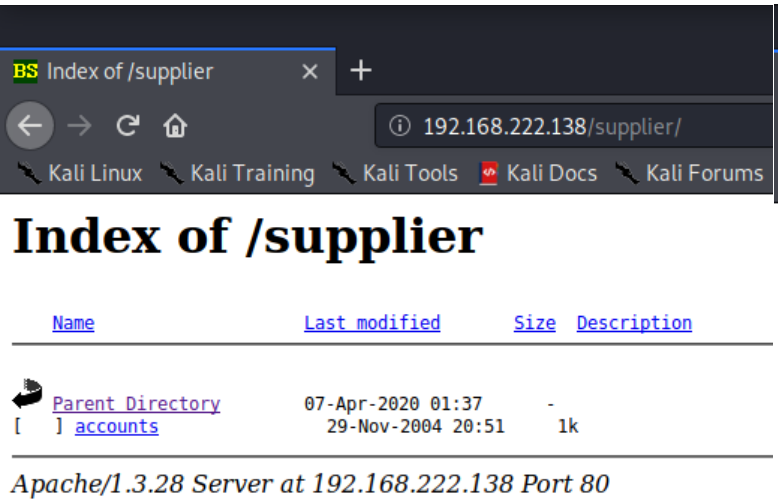
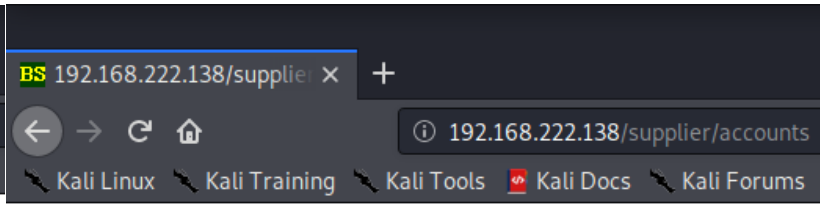


Figure 6 - Access to files inside the /supplier directory.

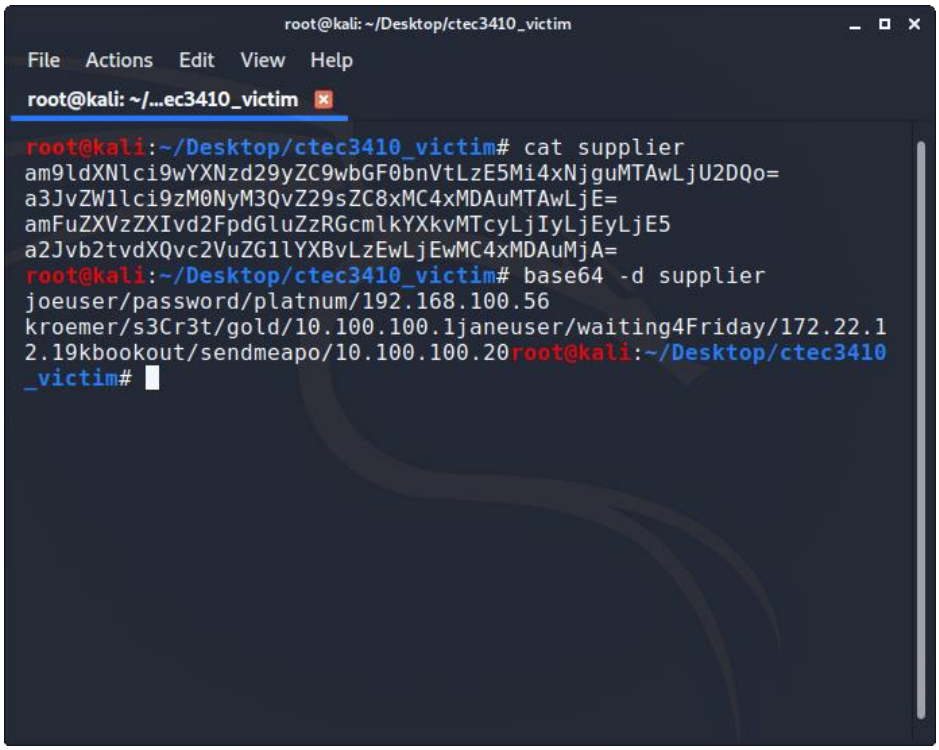


Our attempts of accessing the other directories with no privileges were unsuccessful and can be found in Appendix 6.1.7.

3.1.5 Weak Credentials Encoding

The data present in the **accounts** file (**Figure 6**) is using a weak form of encoding to obfuscate information, and can be easily decoded. After researching the contents of the file, we found it to be encoded using **base64,** usually used by the Linux operating system for storing passwords.

Figure 8 - Decrypting accounts file.



In order to decode the data from the file, we exported it to our own made file (**supplier**), a breakdown of the command used above is: (**base64**) the type of encoding, (**-d**) decode, and the file containing the encoded information (**supplier**). We were successful in decoding the data, which contains all the authentication details of the suppliers such as: **username, password, hint** and **IP address**.

Important note: Sensitive and confidential information should always be secured by using strong cryptographic/hashing methods (for more information, please see **Appendix 6.2.5**).

3.1.6 Broken Password Reset Platform

During our inspection of the website we found an issue with the **Password Reset** function. For a user to reset their password, they are required to provide the **email address** and select their password hint (with 6 available options: **green**, **blue, red, orange, purple, yellow**). Normally, the system will then check if the email and the hint provided by the user is matching the record in the database, if valid it will then reset the password.

We tested the functionality of the platform by making an empty request. We tried resetting the password without providing any user to analyse the application's response. Looking at HTML code of the page with the response (**Figure 9**) we can see that the password was reset to **Welcome** regardless of having no input.

Figure 9 - Broken Password Reset Function.

```
</TBODY></TABLE></TD>
    <TD width=1 bgColor=#ecece0>
      </TD>
    <TD width=615>
      <TABLE cellSpacing=0 cellPadding=0 width=614 border=0>
        <TBODY>
        <TR>
          <TD bgColor=#333333></TD></TR></TBODY></TABLE>
      <TABLE cellSpacing=0 cellPadding=0 width=614 border=0>
        <TBODY>
        <TR bgColor=#ecece0></TR>
        <TR bgColor=#ecece0>
          <TD class=normal width=550 height=20> 
<IFRAME name="bsheader" src="bsheader.cgi" height=27 width=450 marginheight=0 marginwidth=0 scrolling="no" noresize></IFRAME>
          </TD>
          <TD class=normal width=120><IMG SRC="/images/cart.jpg"><A
HREF="/cgi-bin/badstore.cgi?action=cartview">View Cart</A></TD></TR>
        <TR bgColor=#333333>
          </TR></TBODY></TABLE><FONT FACE='Arial'>
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<HTML><HEAD><TITLE>BadStore.net - Reset Password for User</TITLE>
</HEAD><BODY><H2>The password for user:    <P>  ...has been reset to:  Welcome</H2><HR><P><BR><Center><FONT SIZE=2, FACE='Times'>BadStore v1.2.3s - Copyright &#169; 2004-2005</Center>
</BODY></HTML>
```

We continued further testing to see if we could gain access by resetting the password for an administrator account (Figure 11) by inputting **admin** as our email address and using a random password hint.

Figure 11 - Resetting Password for admin.

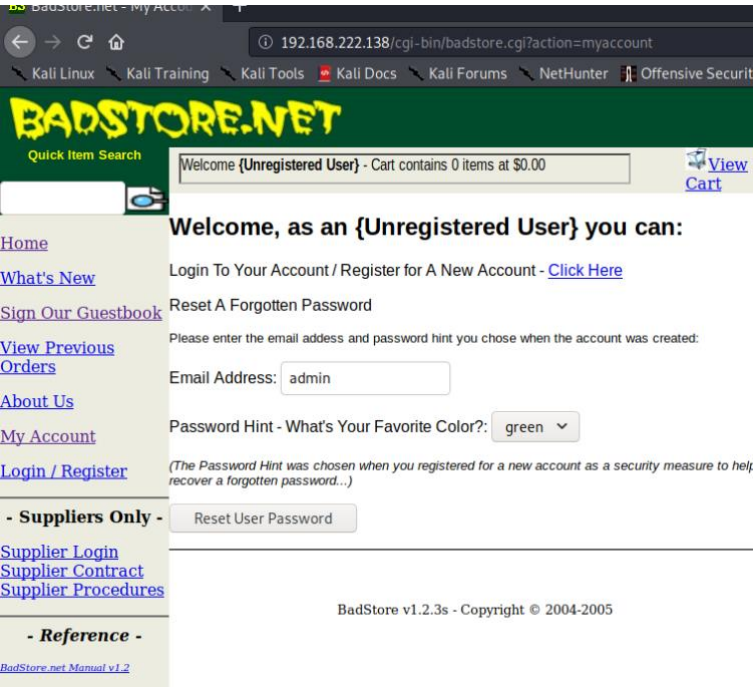
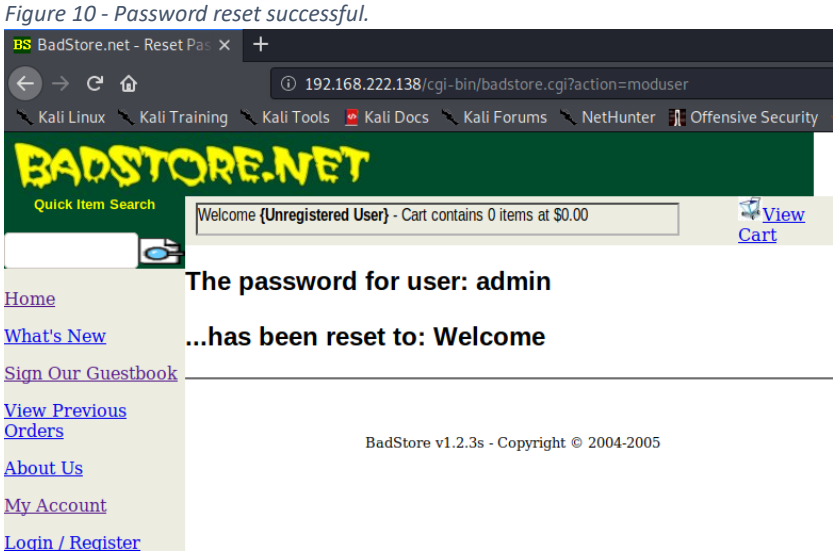


Figure 10 - Password reset successful.



As we can see from Figure 10, we were successful in resetting the password for **admin**. Next, we will be attempting to log in with our newly found credentials to check if they are valid.

Important Note: Publishing an incomplete/not tested web application can result in loss of sensitive information and/or attackers exploiting the application for malicious purposes. We recommend undertaking testing in early stages of the development of each component before publishing.

3.1.7 Unencrypted Network Traffic

Using the credentials discovered previously, we attempt to log in as an administrator while our network capturing software (Wireshark) is active on the background capturing network packets for later analysis. When using Wireshark, we used a filter to only capture traffic on PORT 80/TCP/UDP.

Figure 13 - Login Attempt.

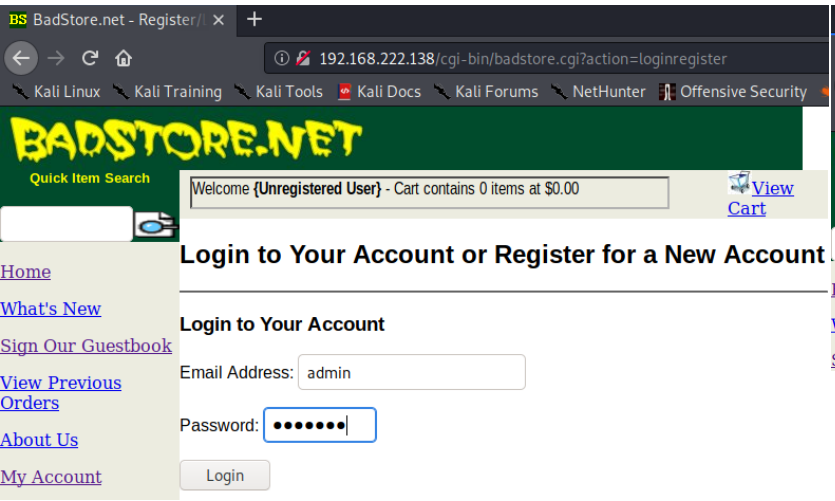
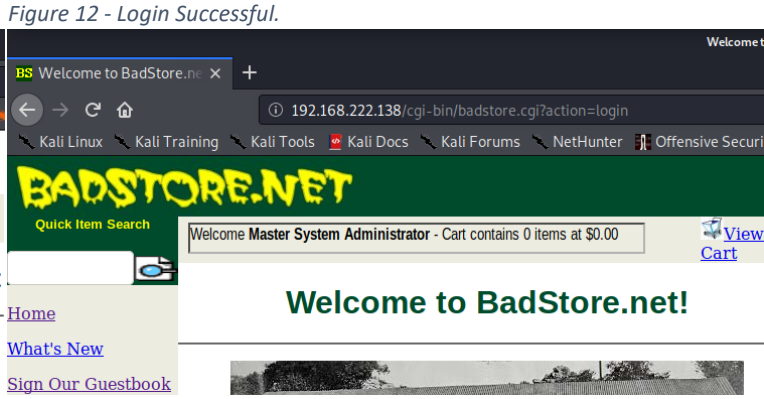


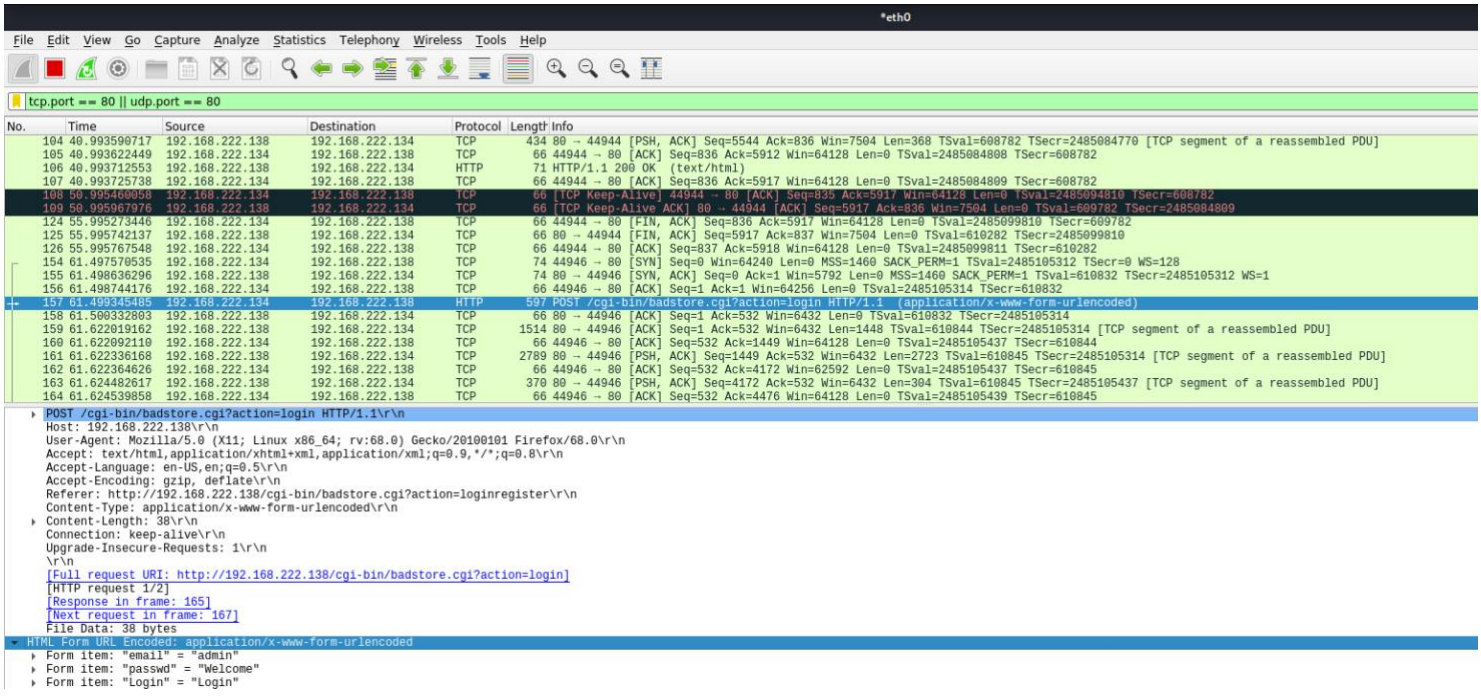
Figure 12 - Login Successful.



The login was successful, and we now have administrator privileges.

Furthermore, the network packages captured with Wireshark were not encrypted and we were able see our login information in clear plain text (**Figure 14**).

Figure 14 - Unencrypted network traffic.



Important Note: All sensitive data should always be encrypted when distributed over insecure channels. As shown above anyone can listen on the network and steal confidential data if not properly secured (more information in **Appendix 6.2.6**).

3.1.8 Web Parameter Tampering

Considering the results of our previous attacks, we wanted to check if the application allows us to manipulate the parameters exchanged between client and server. By exporting the HTML code of the **Login/Register** page (**Figure 15**) we were able to modify parameters and register our own account with administrator privileges.

Figure 15 - HTML code of Login/Register page.

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<HTML><HEAD><TITLE>BadStore.net - Register/Login</TITLE>
</HEAD><BODY><H2>Login to Your Account or Register for a New Account</H2><HR><P><H3>Login to Your Account</
H3><FORM METHOD="POST" ACTION="/cgi-bin/badstore.cgi?action=login" ENCTYPE="application/x-www-form-urlencoded">
Email Address:  <INPUT TYPE="text" NAME="email"  SIZE=20 MAXLENGTH=40><P>Password:  <INPUT TYPE="password"
NAME="passwd"  SIZE=8 8><P><INPUT TYPE="submit" NAME="Login" VALUE="Login"></FORM><HR><P><H3>Register for a
New Account</H3><FORM METHOD="POST" ACTION="/cgi-bin/badstore.cgi?action=register" ENCTYPE="application/x-www-
form-urlencoded">
Full Name:  <INPUT TYPE="text" NAME="fullname"  SIZE=25 MAXLENGTH=40><P>Email Address:  <INPUT TYPE="text"
NAME="email"  SIZE=20 MAXLENGTH=40><P>Password:  <INPUT TYPE="password" NAME="passwd"  SIZE=8 8><P>Password
Hint - What's Your Favorite Color?:  <SELECT NAME="pwdhint">
<OPTION  VALUE="green">green
<OPTION  VALUE="blue">blue
<OPTION  VALUE="red">red
<OPTION  VALUE="orange">orange
<OPTION  VALUE="purple">purple
<OPTION  VALUE="yellow">yellow
</SELECT>
<P><font face=Arial size=2><i>(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.)</i></font><P><INPUT TYPE="hidden" NAME="role" VALUE="U"><INPUT TYPE="submit"
NAME="Register" VALUE="Register"></FORM><P><HR><P><BR><Center><FONT SIZE=2, FACE='Times'>BadStore v1.2.3s -
Copyright &#169; 2004-2005</Center>
</BODY></HTML>
```

HTML ▾ Tab Width: 8 ▾ Ln 92, Col 276 ▾ INS

For us to be able to register our own admin user we had make changes to the following lines:

<INPUT TYPE="hidden" NAME="role" VALUE="U"><INPUT TYPE="submit"

- As we can see above, the application uses hidden fields to assign the roles to new users, by changing **U** (user) to **A** (admin) we are giving ourselves administrator privileges.

METHOD="POST" ACTION="*http://192.168.222.138*/cgi-bin/badstore.cgi?action=register"

- As the application accepts the POST method, we will attempt to load our version of the page. Finally, we added the URL address to redirect the register action.

Figure 17 - Create admin account.

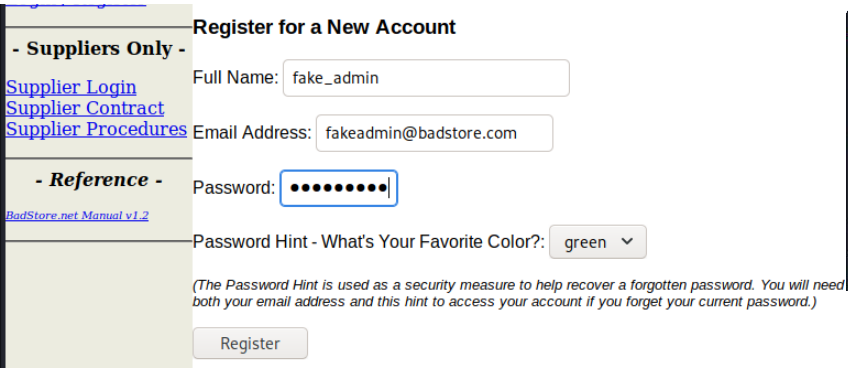
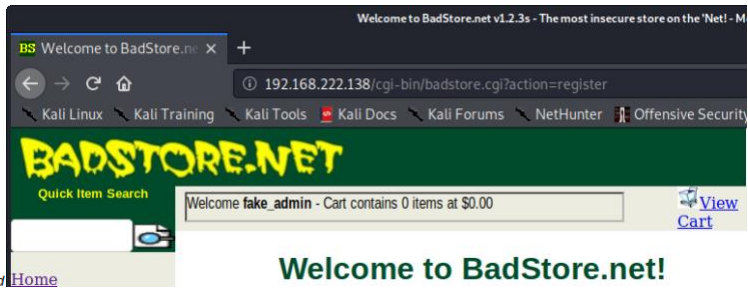


Figure 16 - Login with admin account successful.



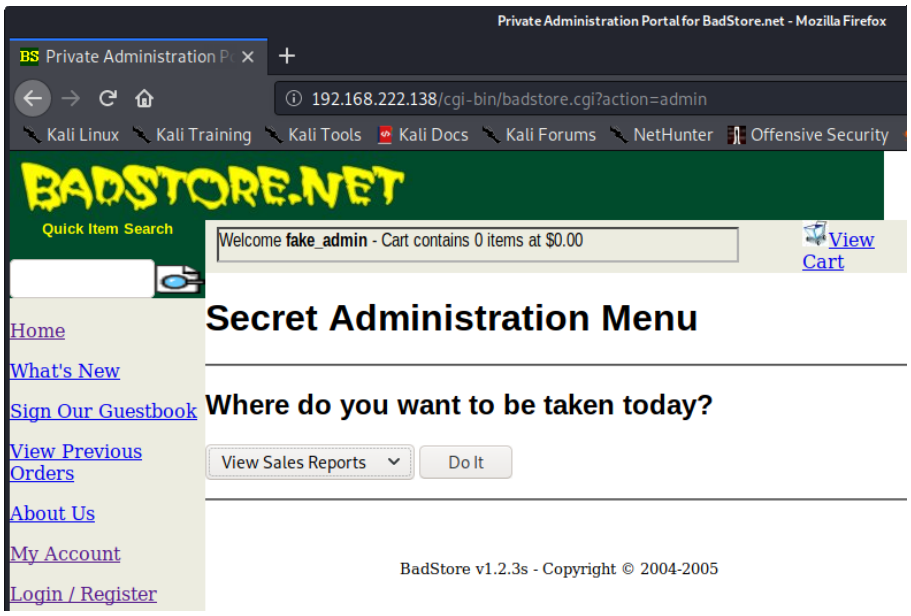
We then created our own account (**Figure 17**) and after registering, we were successfully logged in (**Figure 16**). For more information and remediation strategies on this attack please see **Appendix 6.2.7**.

3.1.9 Secret Admin Portal

After our successful login (Figure 16) we looked at the URL address and took note of this part:

- ?action=register, we continued by changing the action to admin to see the response

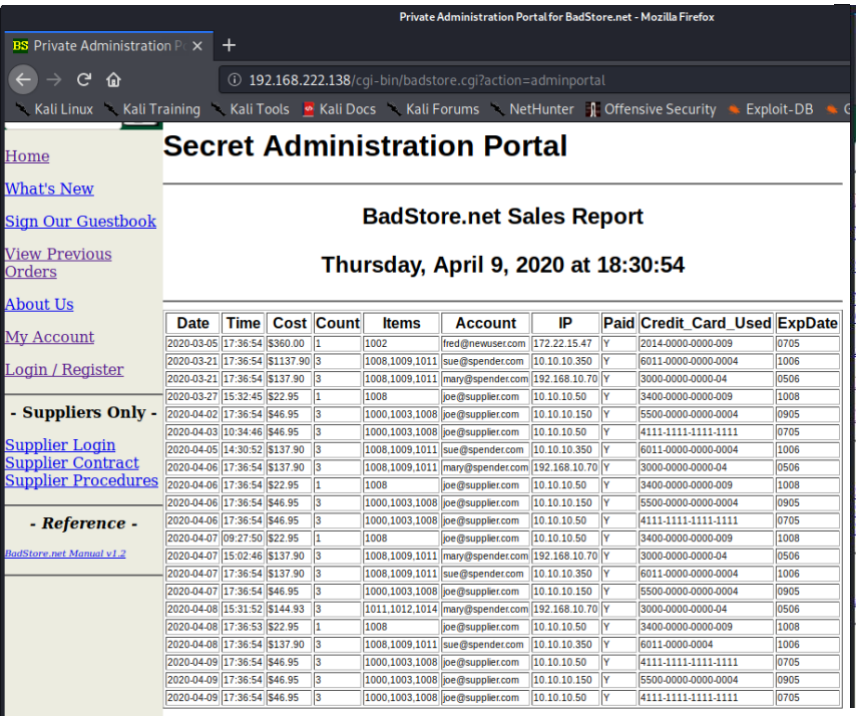
Figure 18 - Changing URL action to admin.



By changing the action to admin we were taken to a Secret Administration Menu (Figure 18), from there we had options such as: Sales Reports, Users, Add/Delete Users, etc..

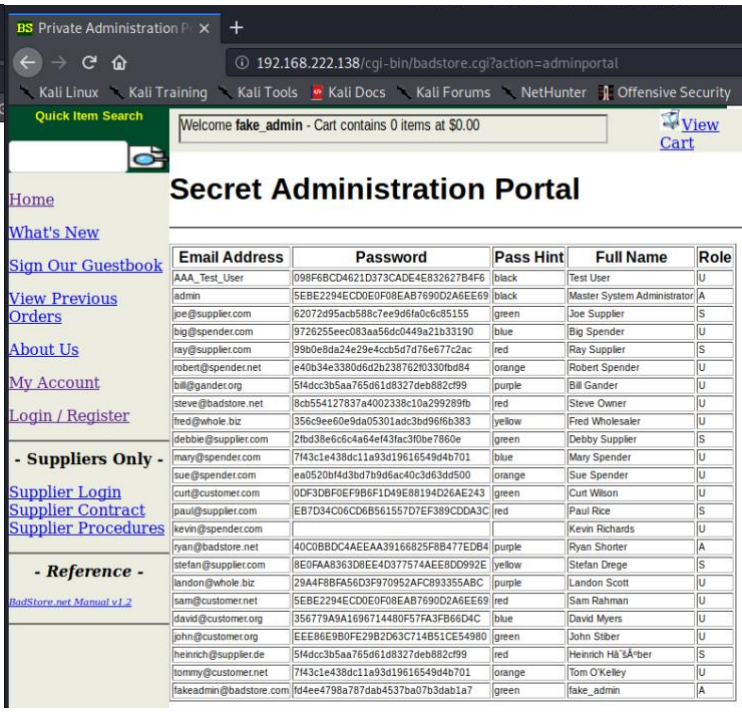
In order to fully test our admin account we attempted to see if we can access any of the options from the portal (we have previously attempted at accessing without any account, please see Appendix 6.1.8).

Figure 20 - Accessing Sales Reports.



Date	Time	Cost	Count	Items	Account	IP	Paid	Credit_Card_Used	ExpDate
2020-03-05	17:36:54	$360.00	1	1002	fred@newuser.com	172.22.15.47	Y	2014-0000-0000-009	0705
2020-03-21	17:36:54	$1137.90	3	1008,1009,1011	sue@spender.com	10.10.10.350	Y	6011-0000-0000-0004	1006
2020-03-21	17:36:54	$137.90	3	1008,1009,1011	mary@spender.com	192.168.10.70	Y	3000-0000-0000-04	0506
2020-03-27	15:32:45	$22.95	1	1008	joe@supplier.com	10.10.10.50	Y	3400-0000-0000-009	1008
2020-04-02	17:36:54	$46.95	3	1000,1003,1008	joe@supplier.com	10.10.10.150	Y	5500-0000-0000-0004	0905
2020-04-03	10:34:46	$46.95	3	1000,1003,1008	joe@supplier.com	10.10.10.50	Y	4111-1111-1111-1111	0705
2020-04-05	14:30:52	$137.90	3	1008,1009,1011	sue@spender.com	10.10.10.350	Y	6011-0000-0000-0004	1006
2020-04-06	17:36:54	$137.90	3	1008,1009,1011	mary@spender.com	192.168.10.70	Y	3000-0000-0000-04	0506
2020-04-06	17:36:54	$22.95	1	1008	joe@supplier.com	10.10.10.50	Y	3400-0000-0000-009	1008
2020-04-06	17:36:54	$46.95	3	1000,1003,1008	joe@supplier.com	10.10.10.150	Y	5500-0000-0000-0004	0905
2020-04-06	17:36:54	$46.95	3	1000,1003,1008	joe@supplier.com	10.10.10.50	Y	4111-1111-1111-1111	0705
2020-04-07	09:27:50	$22.95	1	1008	joe@supplier.com	10.10.10.50	Y	3400-0000-0000-009	1008
2020-04-07	15:02:46	$137.90	3	1008,1009,1011	mary@spender.com	192.168.10.70	Y	3000-0000-0000-04	0506
2020-04-07	17:36:54	$137.90	3	1008,1009,1011	sue@spender.com	10.10.10.350	Y	6011-0000-0000-0004	1006
2020-04-07	17:36:54	$46.95	3	1000,1003,1008	joe@supplier.com	10.10.10.150	Y	5500-0000-0000-0004	0905
2020-04-08	15:31:52	$144.93	3	1011,1012,1014	mary@spender.com	192.168.10.70	Y	3000-0000-0000-04	0506
2020-04-08	17:36:53	$22.95	1	1008	joe@supplier.com	10.10.10.50	Y	3400-0000-0000-009	1008
2020-04-08	17:36:54	$137.90	3	1008,1009,1011	sue@spender.com	10.10.10.350	Y	6011-0000-0004	1006
2020-04-09	17:36:54	$46.95	3	1000,1003,1008	joe@supplier.com	10.10.10.50	Y	4111-1111-1111-1111	0705
2020-04-09	17:36:54	$46.95	3	1000,1003,1008	joe@supplier.com	10.10.10.150	Y	5500-0000-0000-0004	0905
2020-04-09	17:36:54	$46.95	3	1000,1003,1008	joe@supplier.com	10.10.10.50	Y	4111-1111-1111-1111	0705

Figure 19 - Accessing Users.



Email Address	Password	Pass Hint	Full Name	Role
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 Hã¨ß¨ber	S
tommy@customer.net	7f43c1e438dc11a93d19616549d4b701	orange	Tom O'Kelley	U
fakeadmin@badstore.com	fd4ee4798a787dab4537ba07b3dab1a7	green	fake_admin	A

3.1.10 Sensitive data exposure

Following the discovery of the **Secret Administration Menu** (Figure 18) we accessed the **Current Users** (Figure 19) and **Sales Reports** (Figure 20) and found several issues:

1. Clear-Text storage of credit card information
2. Weakly encrypted/encoded passwords
3. Non-encrypted user information

In **Weak Credentials Encoding 3.1.5**, we found the supplier's passwords to be encoded with **base64**. As we already discovered that, we continued by looking at the password for **Test User.**

Figure 22 - Hash Identifier.

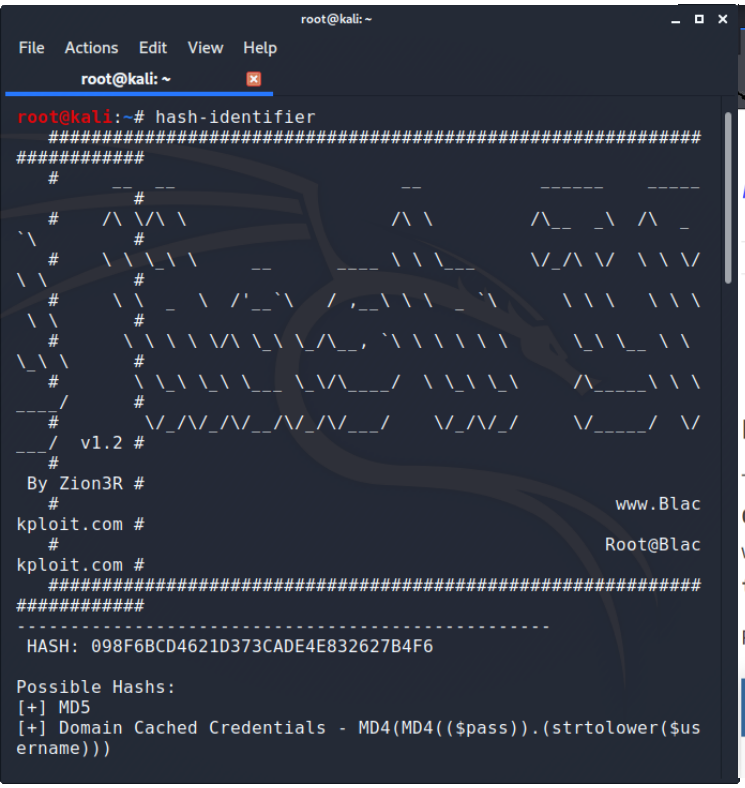
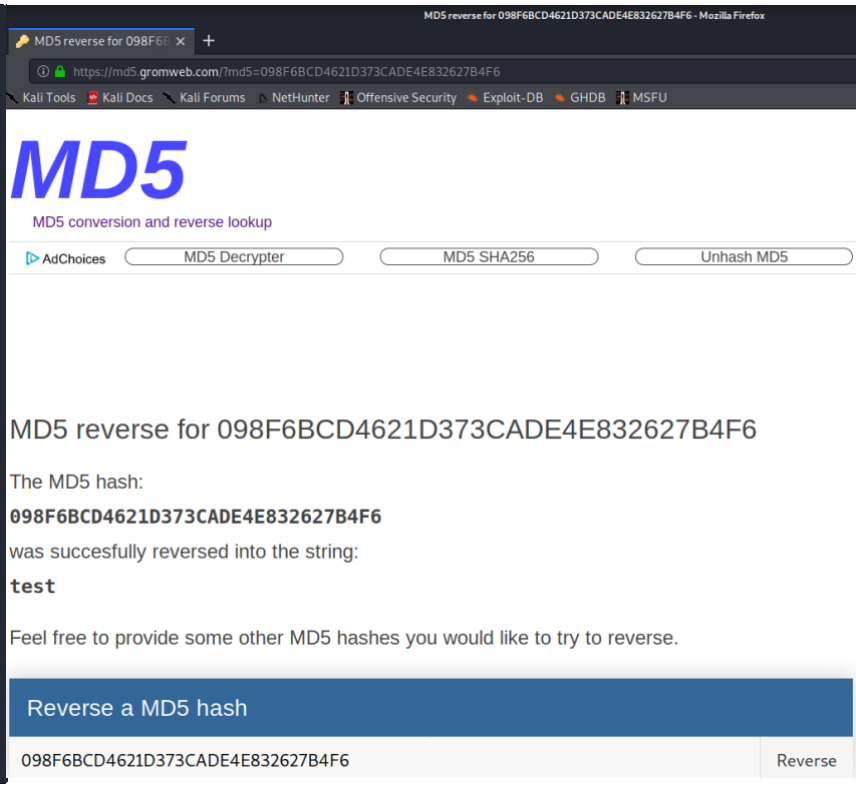


Figure 21 - Reversing Hash.



By using hash-identifier to see if the password matches any known hash, we found a possible hash to be **MD5** (Figure 22). We later used an online **MD5** reverse platform to discover the password, which was successful, and further confirmed that passwords were hashed using **MD5**.

Important note: MD5 is broken and obsolete, we recommend using stronger methods of encryption/hashing in order to ensure the protection of sensitive data. Sensitive information such as passwords, credit card numbers and expiry dates should **NOT** be stored in plain text.

3.2 Operating System

By analysing the software and services running on the server, we found it to be outdated and exposed to several vulnerabilities. We used Nessus to compile a list of vulnerabilities (**Appendix 6.1.4**), which we later attempted to exploit. All attempts and information regarding the Operating System can be found in **Appendix 6.1.10**.

4 Conclusion

After conducting several tests and scans we found that BadStore.net (the application) suffered a series of control failures that led to a complete compromise of critical and privileged information of its users. Current flaws within the application allow malicious parties to intercept and steal confidential information, modify parts of the application and distribute malicious code. We recommend that appropriate efforts should be undertaken to mitigate the security issues presented throughout the report.

4.1 Remediation Strategy

Details and information on all vulnerabilities discovered, as well as their remediation strategies can be found in Appendix 6.2.

5 References

[1] [Online]. Available: http://www.cengage.com/resource_uploads/downloads/143548360X_429597.pdf.

[2] [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-115.pdf.

[3] [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-30r1.pdf.

[4] [Online]. Available: https://owasp.org/www-community/attacks/SQL_Injection.

[5] [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html.

[6] [Online]. Available: https://owasp.org/www-community/attacks/xss/.

[7] [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html.

[8] [Online]. Available: https://www.tenable.com/plugins/nessus/10302.

[9] [Online]. Available: https://www.netsparker.com/blog/web-security/disable-directory-listing-web-servers/.

[10] [Online]. Available: https://owasp.org/www-community/vulnerabilities/Password_Plaintext_Storage.

[11] [Online]. Available: https://paragonie.com/blog/2015/08/you-wouldnt-base64-a-password-cryptography-decoded.

[12] [Online]. Available: https://www.guru99.com/web-security-vulnerabilities.html.

[13 [Online]. Available: https://owasp.org/www-community/attacks/Web_Parameter_Tampering.
]

[14 [Online]. Available: https://cwe.mitre.org/data/definitions/472.html.
]

[15 [Online]. Available: https://heartbleed.com/.
]

[16 [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2016-2183.
]

[17 [Online]. Available: https://sweet32.info/.
]

[18 [Online]. Available: https://www.us-cert.gov/ncas/alerts/TA14-290A.
]

[19 [Online]. Available: https://www.tenable.com/plugins/nessus/78479.
]

6 Appendices

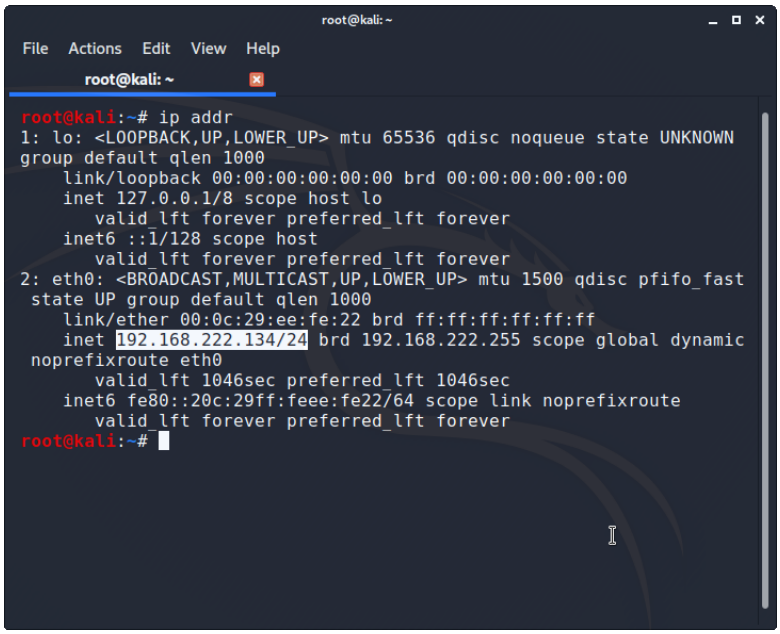
6.1 Appendix 1 – Full Details of Scans, Exploits & Processes Used

BOZBITS PLC provided us with a virtual machine as our environment for performing our tests, in order to avoid targeting other systems on our network, both our host and our target were set to bridged network.

6.1.1 IP address discovery

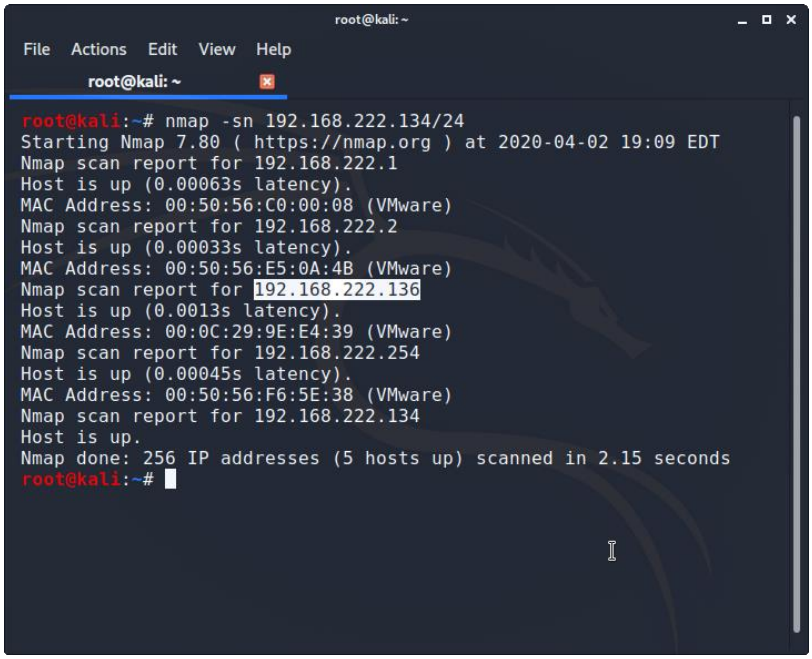
As shown in the figure below, we have used a basic IP address command to display our network adaptor (**eth0**) and our IP range (**192.168.222.134/24**).

Figure 23 - IP range discovery.



With our IP range identified, we conducted a service network scan using Nmap to discover the systems on our network.

Figure 24 – Target IP address found.



The scan returned several addresses, after analysing them, we found our target to be 192.168.222.136. The first two addresses (192.168.222.1 & 192.168.222.2) are default addresses used by VMware, the last address belongs to our Kali Linux machine, while the fourth address (192.168.222.254) is a reserved address by the DHCP server.

6.1.2 Nmap intensive scan

With our target discovered, we started to gather information, such as open ports and available services by performing a full intensive TCP scan with Nmap. A breakdown of the command used below is as follow: (-p) port range, (1-65535) number of ports available, (-A) all features, (-v) verbose, extra details.

Figure 25 - Nmap full TCP scan - part 1.

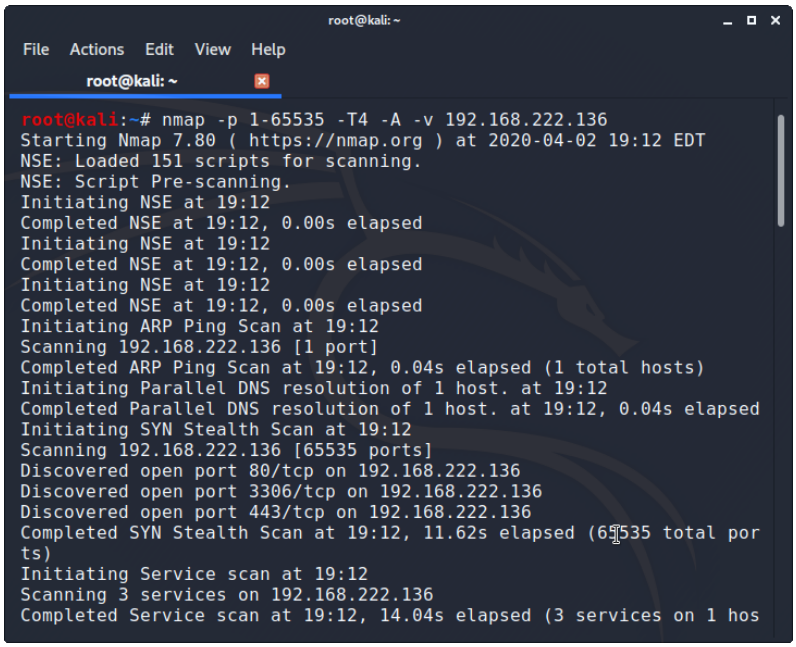


Figure 26 - Nmap full TCP scan - part 2.

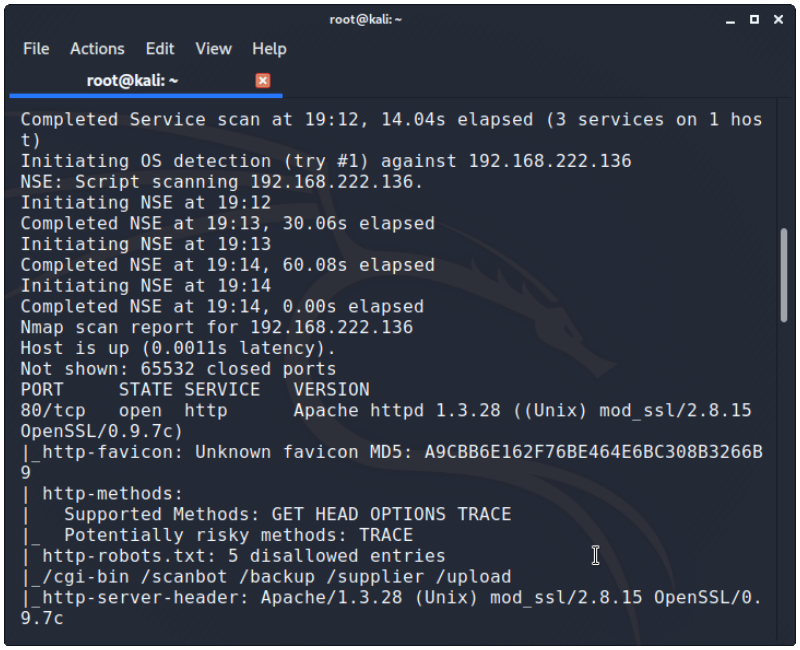


Figure 28 - Nmap full TCP scan - part 3.

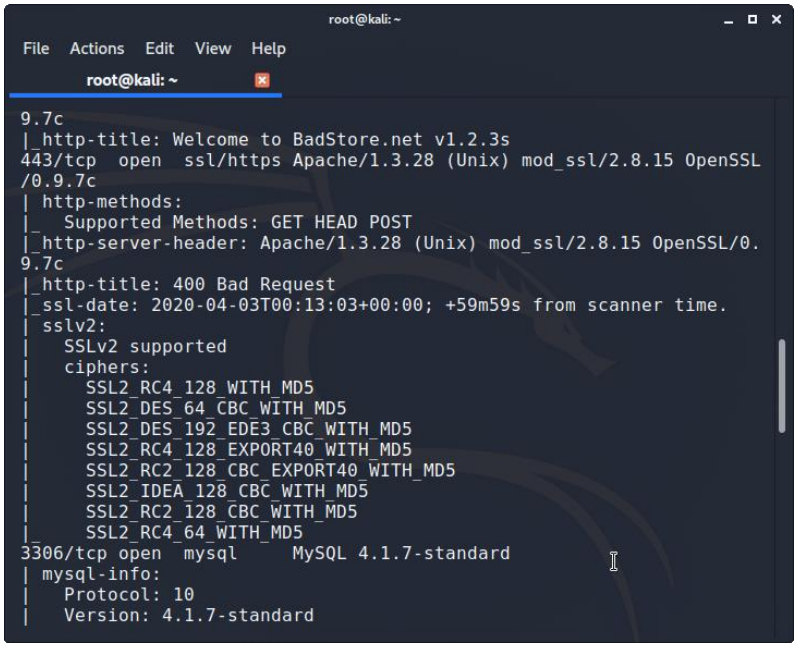
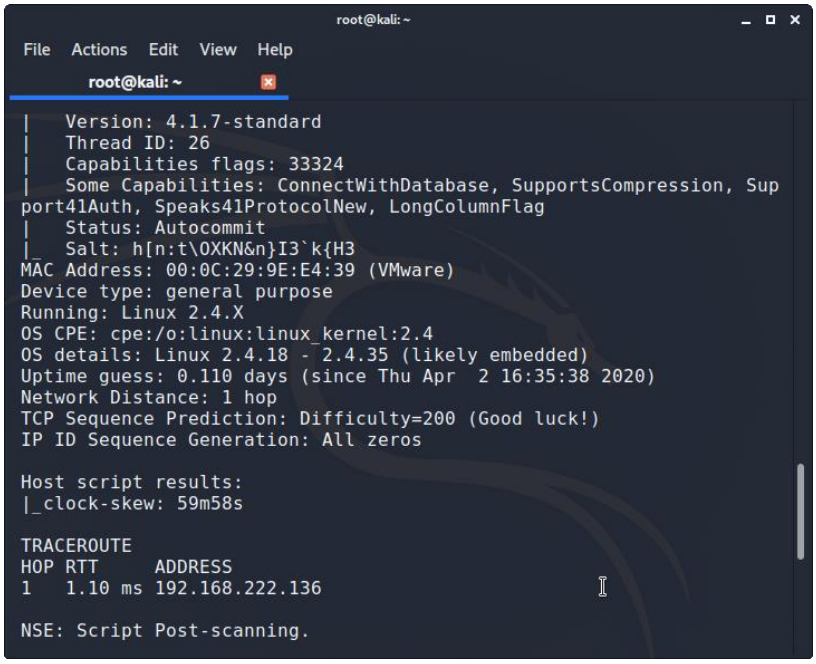


Figure 27 - Nmap full TCP scan - part 4.



The scan was successful, and several open ports were discovered, as well as available services on each port such as:

- PORT 80/TCP HTTP (Apache 1.3.28/OpenSSL 0.9.7c) – methods (GET/HEAD/OPTIONS/TRACE)
- PORT 443/TCP SSL/HTTPS (Apache 1.3.28/OpenSSL 0.9.7c) – methods (GET/HEAD/POST), SSLv2 ciphers
- PORT 3306/TCP MySQL (MySQL 4.1.7) – informative (out of scope)

6.1.3 OWASP ZAP automated scan

Looking at the information obtained from the Nmap intensive scan, we can see http-methods available, (GET/HEAD/TRACE) and (POST) for port 443, so we continue by performing web spidering using OWASP ZAP. We will perform an automated scan, no extra parameters set, which will provide a starting point at discovering any flaws within the application.

Figure 29 - ZAP automated scan.

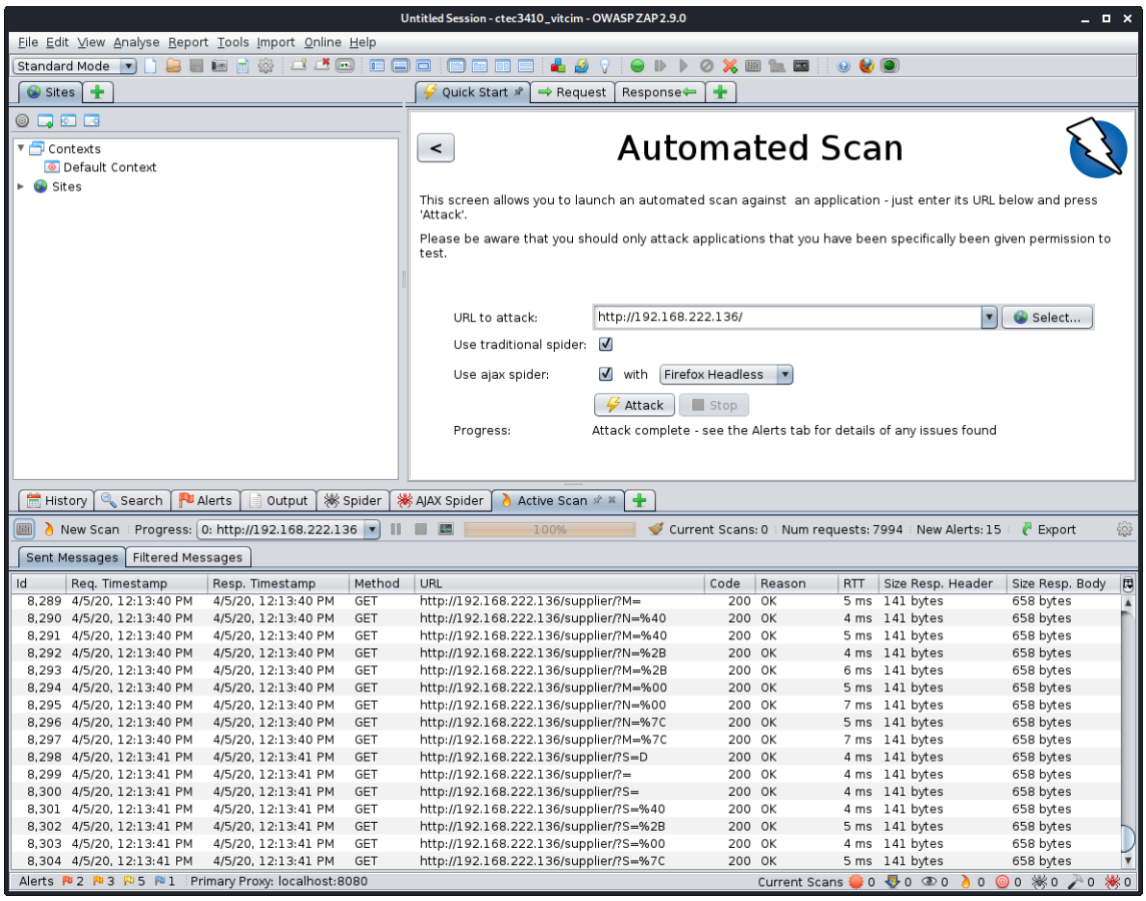
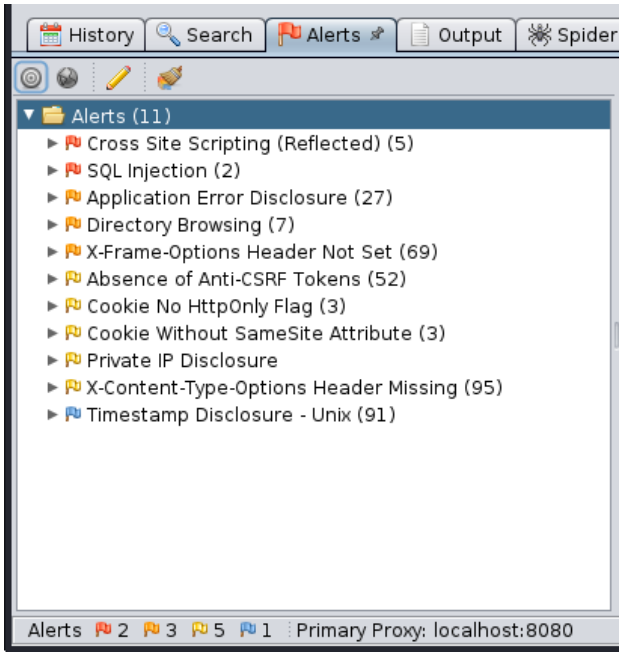


Figure 30 - Alerts returned by the ZAP scan.



Several issues were discovered, which provide an overall picture of the vulnerabilities present on our target, such as:

- XSS
- SQL Injection
- Directory Exposure
- Application/IP Disclosure
- Unsecured Cookies

With the current results, we looked at each alert individually and assessed their relevance and impact on the platform.

1. Cross-site Scripting (XSS)

Figure 31 - XSS Alert 1.

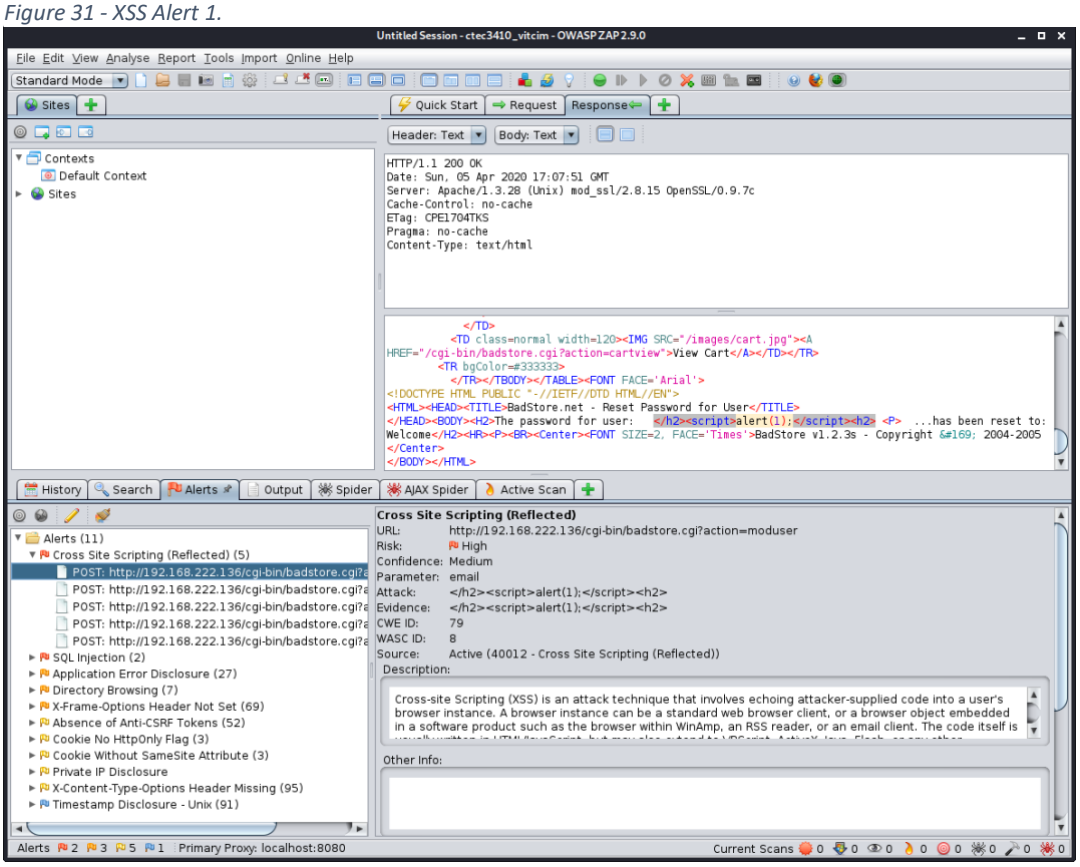


Figure 32 - XSS Alert 2.

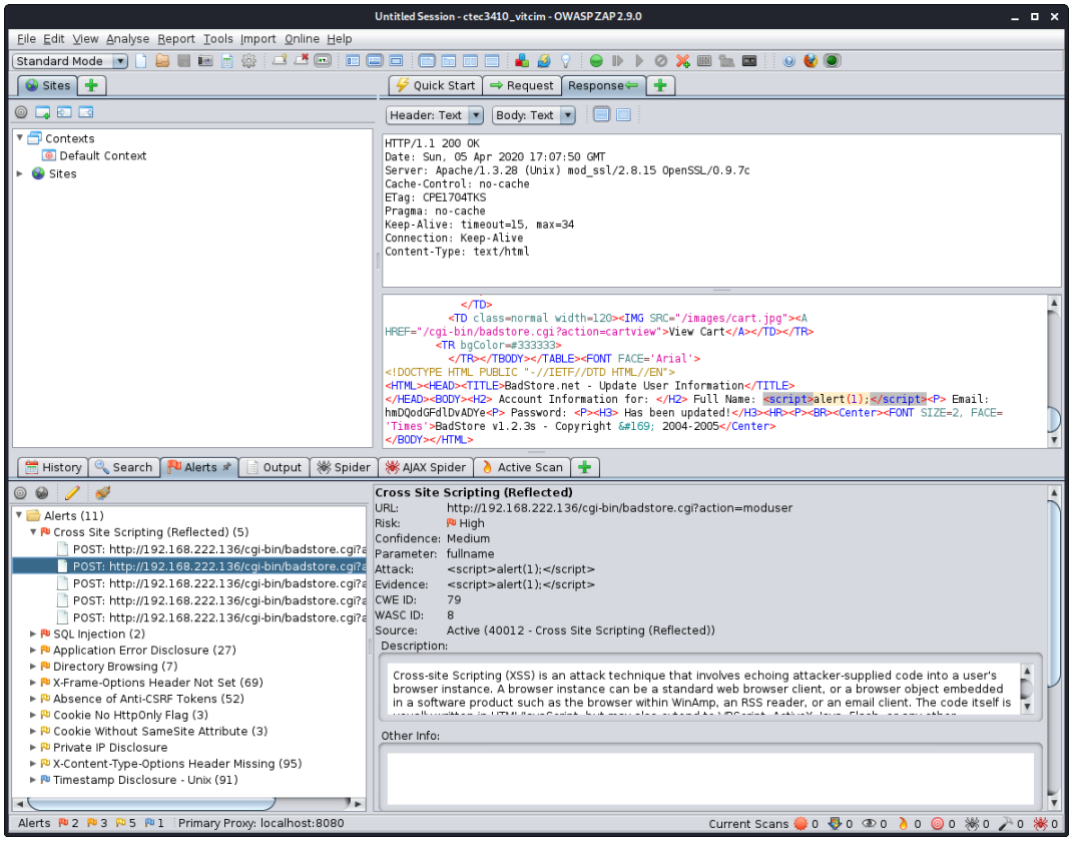


Figure 34 - XSS Alert 3.

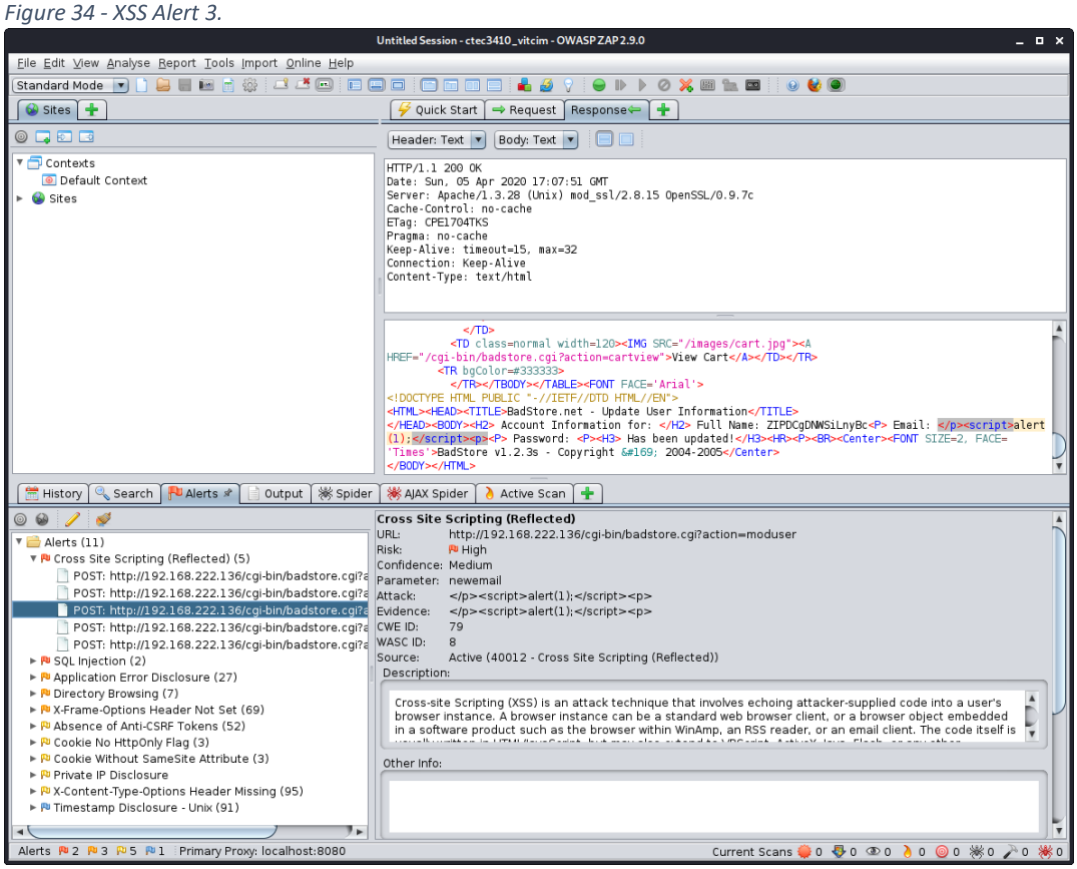


Figure 33 - XSS Alert 4.

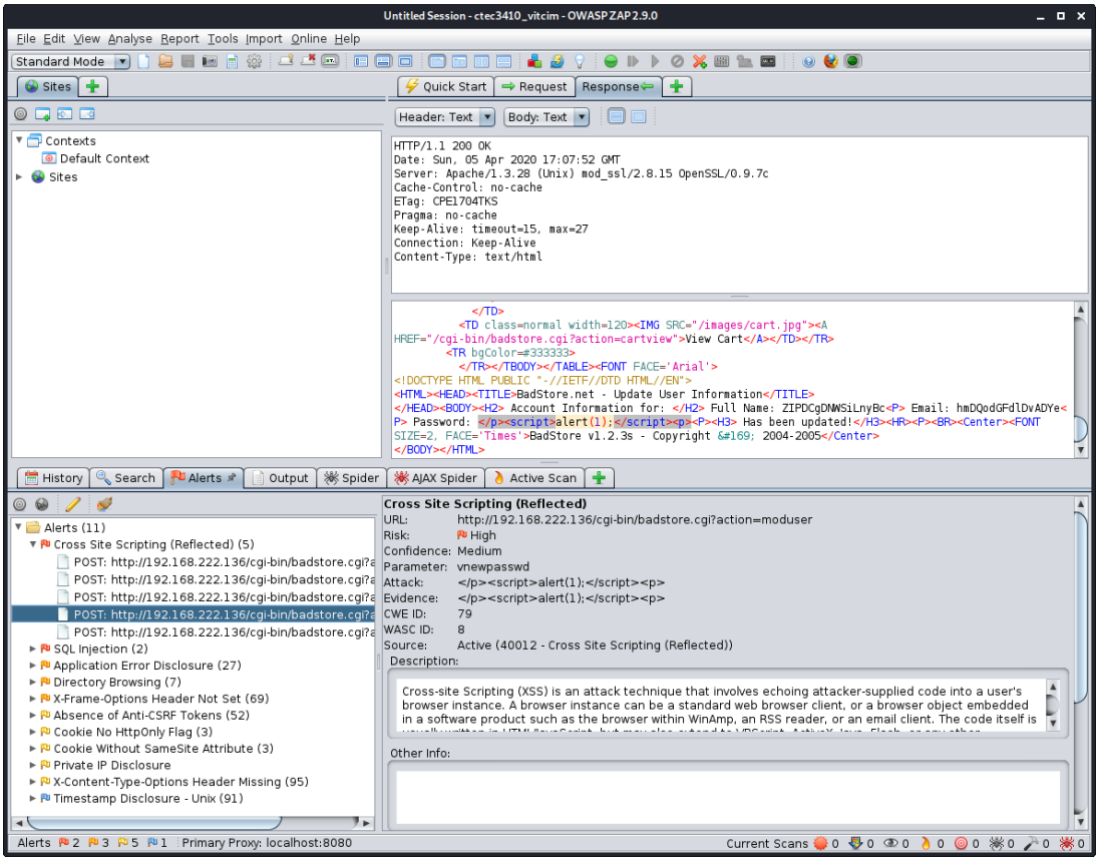
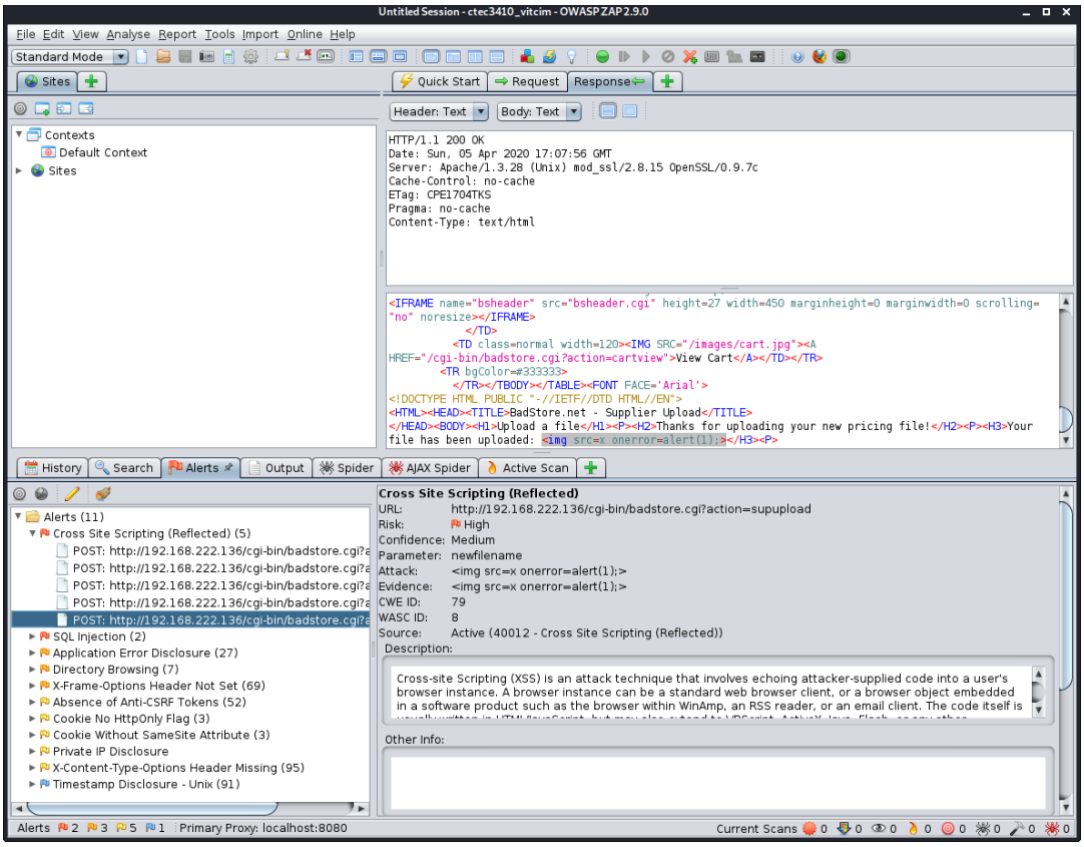


Figure 35 - XSS Alert 5.



2. SQL Injection

Figure 36 - SQL Alert 1.

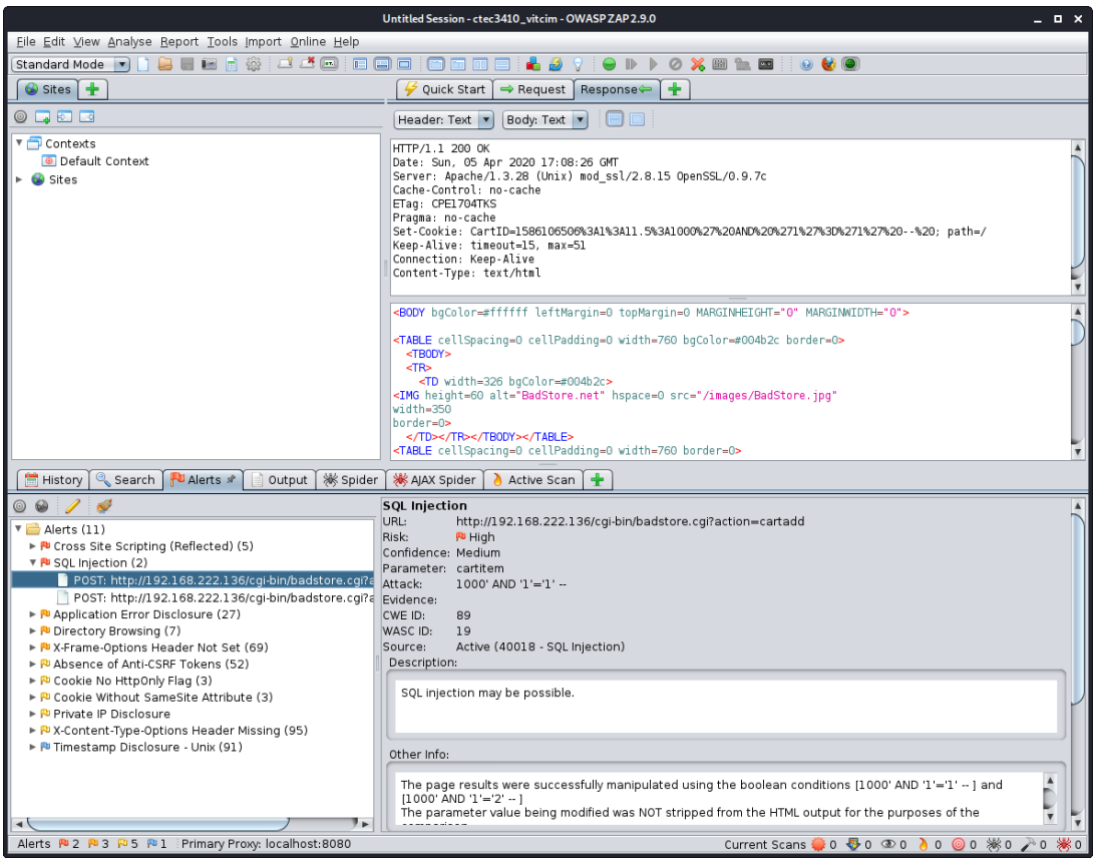
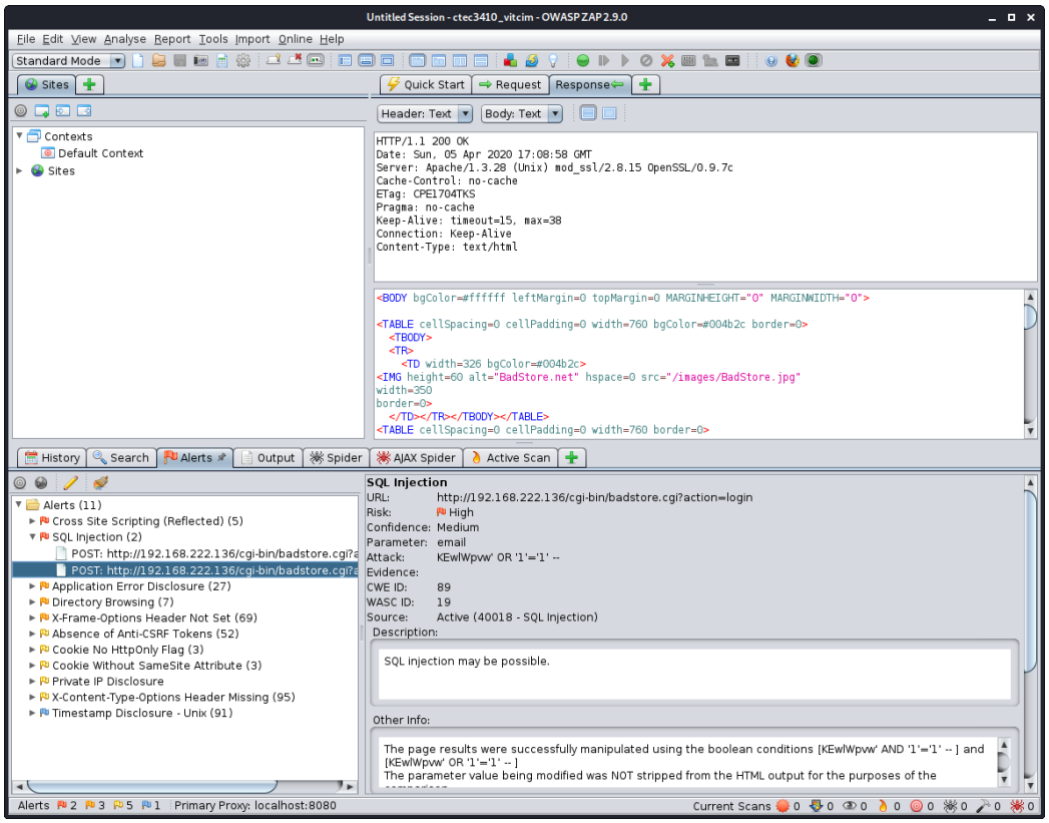
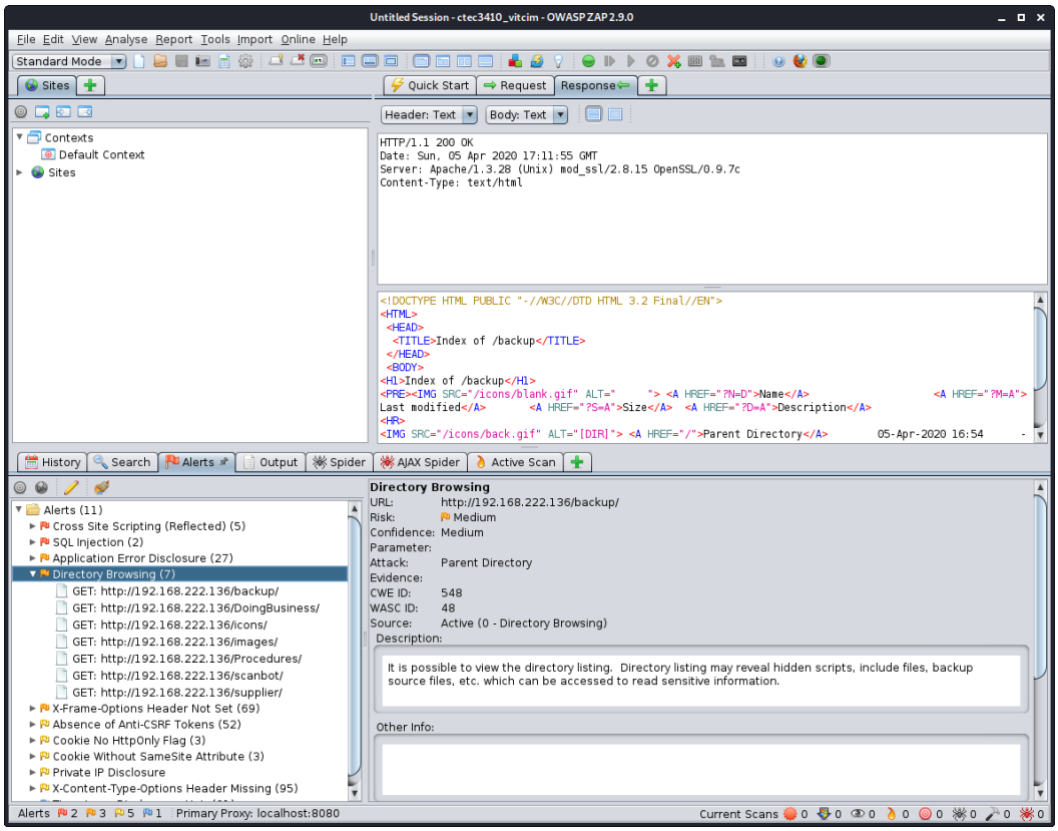


Figure 37 - SQL Alert 2.



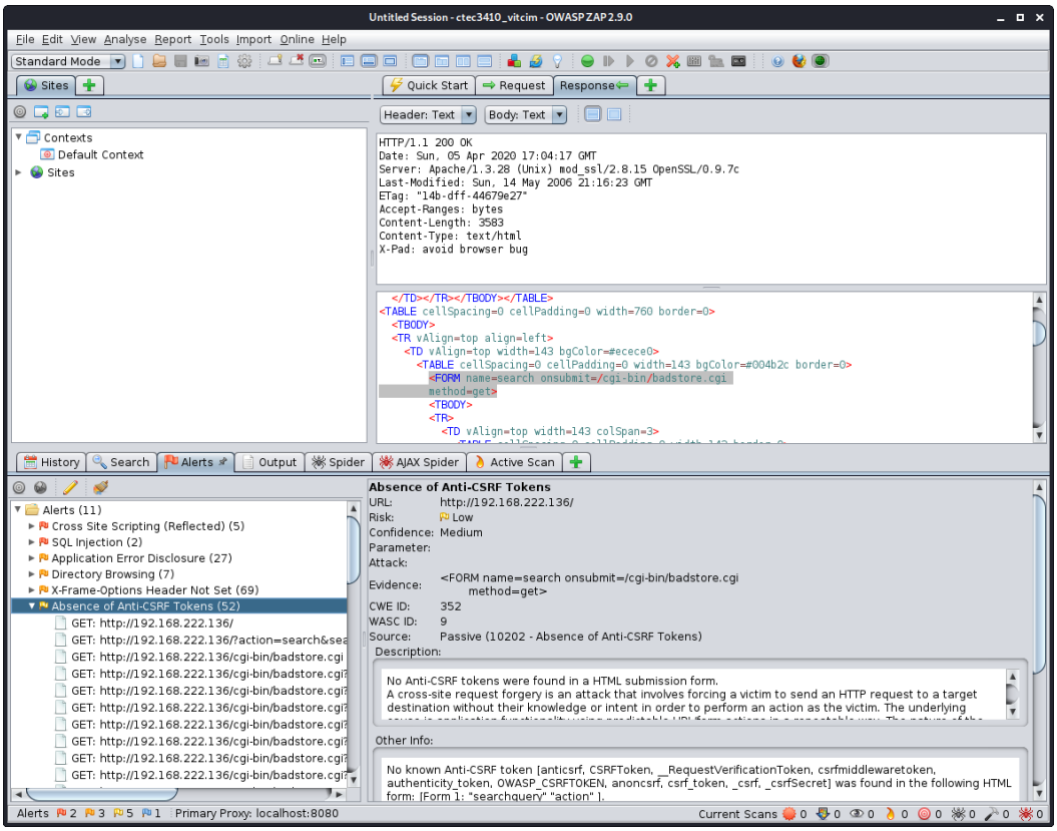
3. Private Directory Exposure

Figure 38 - Directory Exposure Alert.



4. Cross-site Request Forgery

Figure 39 - CSRF Alert.



5. Cookie Manipulation

Figure 40 - No HTTP Only Flag Alert 1

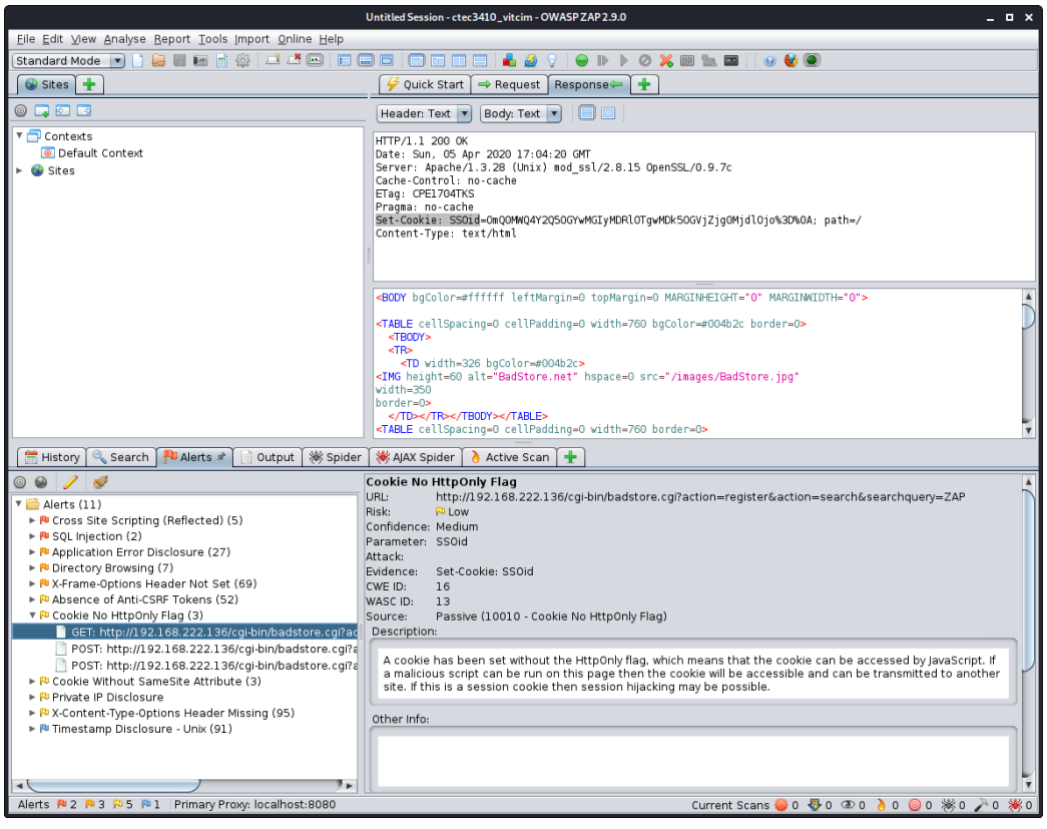


Figure 42 - No HTTP Only flag Alert 2.

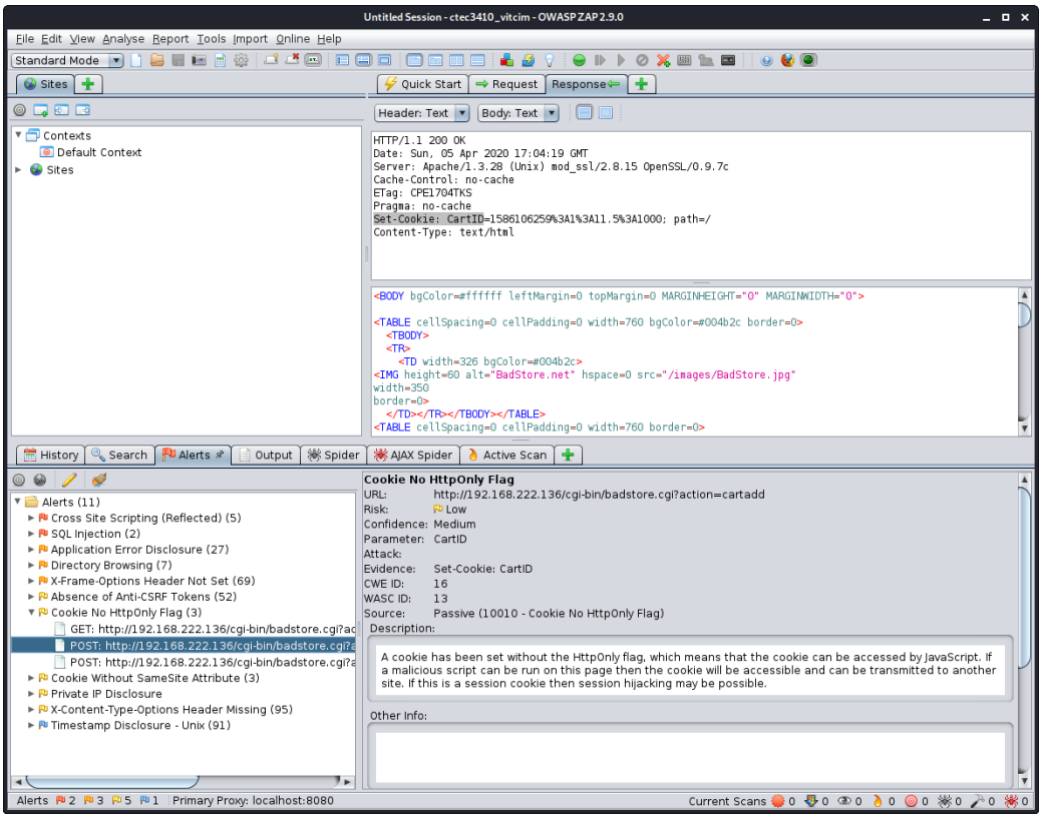


Figure 41 - No HTTP Only flag Alert 3.

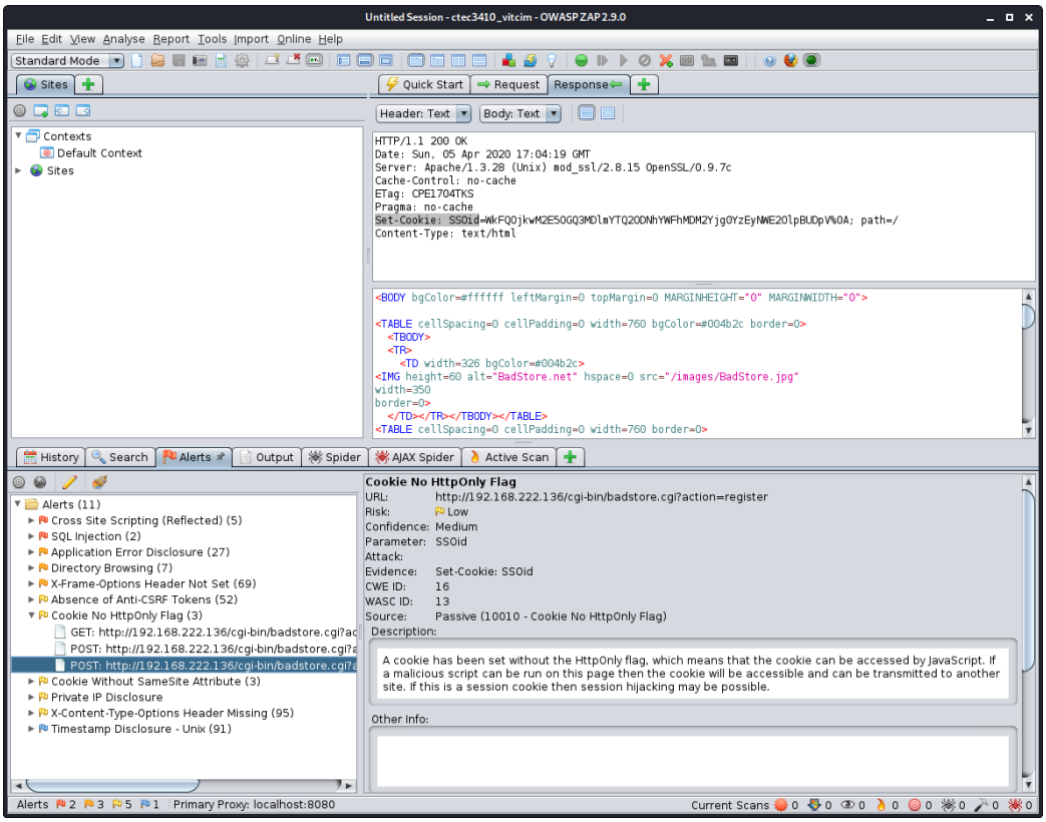


Figure 44 - No Same Site Attribute Alert 1.

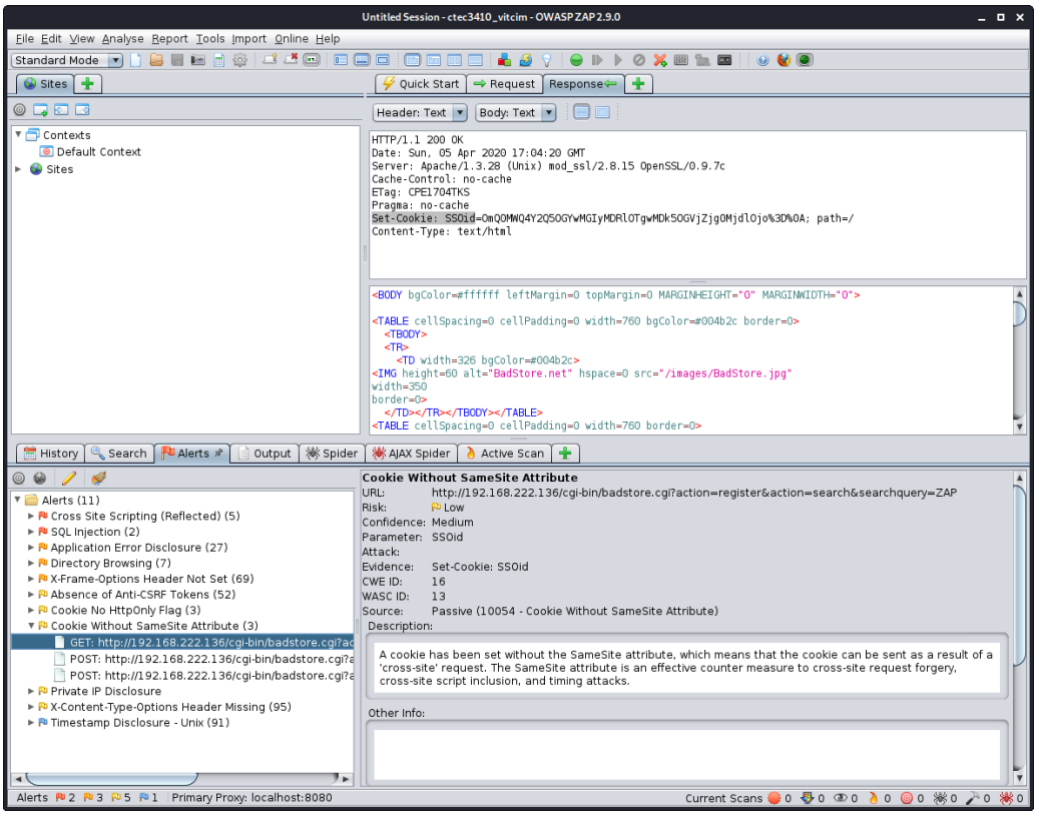


Figure 43 - No Same Site Attribute Alert 2.

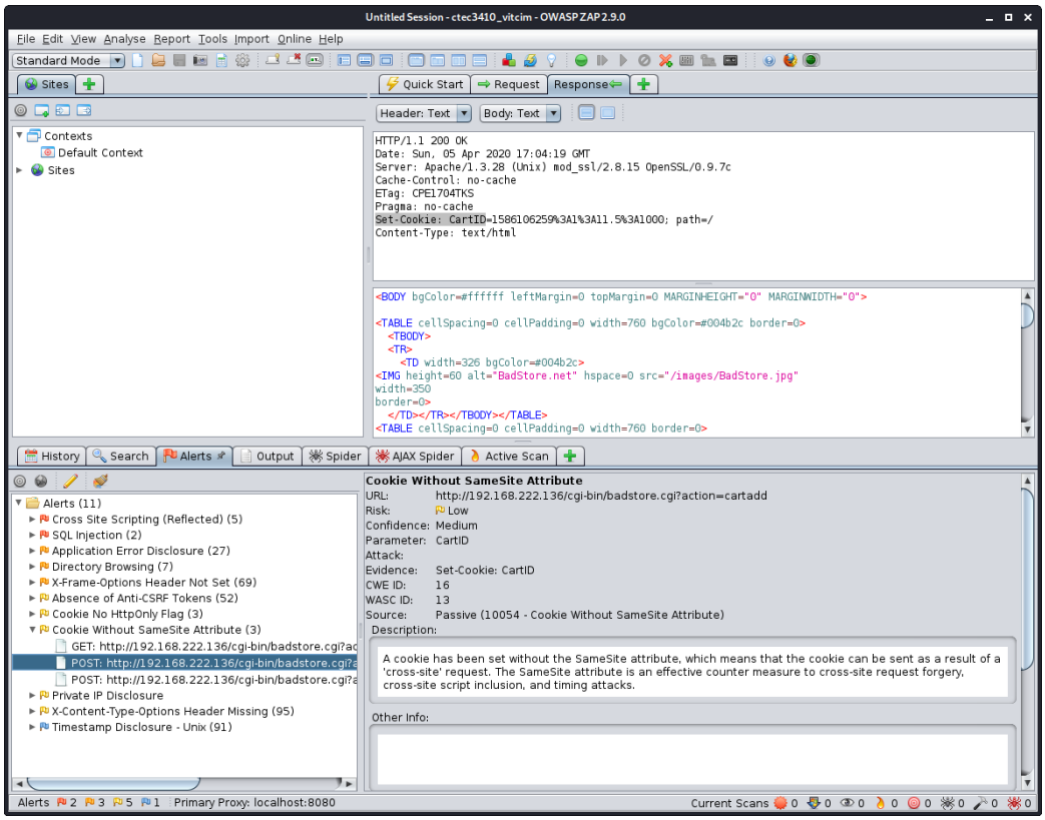
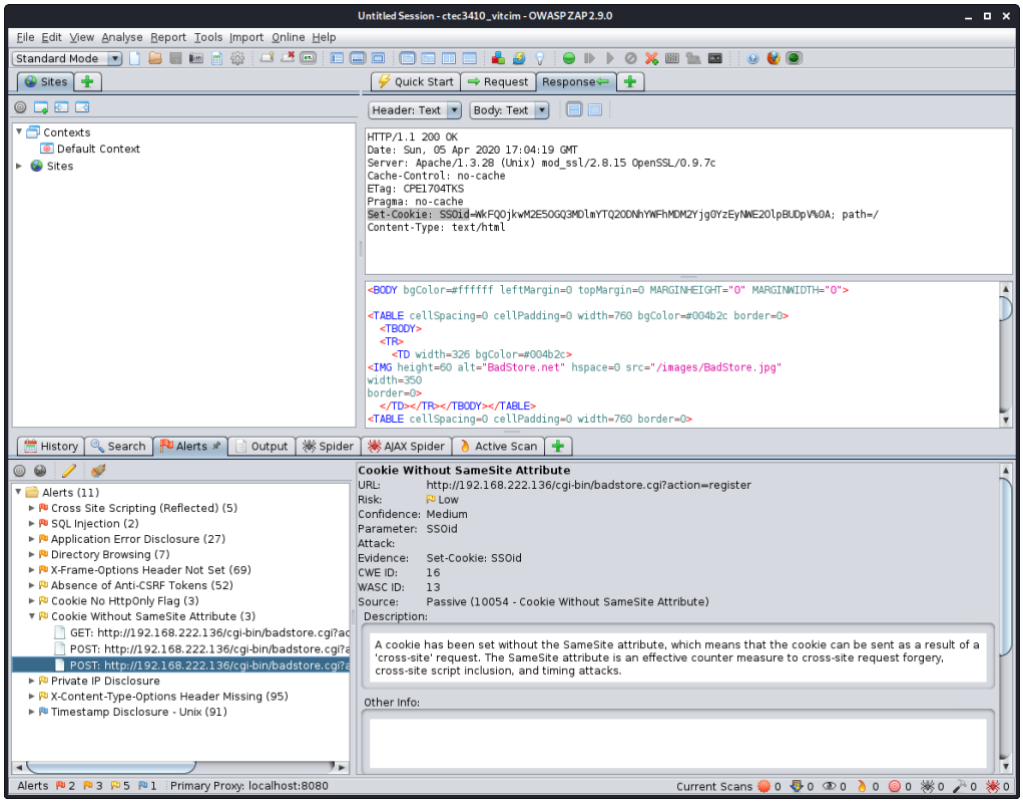


Figure 45 - No Same Site Attribute Alert 3.



6.1.4 Nessus Basic Network Scan

In order to get a more detailed and comprehensive examination of our target we further performed a network scan using Nessus, a vulnerability scanner.

Figure 46 – Overall Vulnerabilities found by Nessus.

Figure 47 - Breakdown of Vulnerabilities found by Nessus.

Severity	CVSS	Plugin	Name
CRITICAL	10.0	15555	Apache mod_proxy Content-Length Overflow
CRITICAL	10.0	17757	OpenSSL < 0.9.7l / 0.9.8d Multiple Vulnerabilities
CRITICAL	10.0	78555	OpenSSL Unsupported
HIGH	9.3	17760	OpenSSL < 0.9.8f Multiple Vulnerabilities
HIGH	9.3	57459	OpenSSL < 0.9.8s Multiple Vulnerabilities
HIGH	7.5	31654	Apache < 1.3.37 mod_rewrite LDAP Protocol URL Handling Overflow
HIGH	7.5	13651	Apache mod_ssl ssl_engine_log.c mod_proxy Hook Function Remote Format String
HIGH	7.5	58799	OpenSSL < 0.9.8w ASN.1 asn1_d2i_read_bio Memory Corruption
HIGH	7.5	34460	Unsupported Web Server Detection
HIGH	7.5	12255	mod_ssl ssl_util_uuencode_binary Remote Overflow
HIGH	7.2	11915	Apache < 1.3.29 Multiple Modules Local Overflow
HIGH	7.1	20007	SSL Version 2 and 3 Protocol Detection
MEDIUM	6.4	51192	SSL Certificate Cannot Be Trusted
MEDIUM	5.8	17762	OpenSSL < 0.9.8j Signature Spoofing
MEDIUM	5.8	42880	SSL / TLS Renegotiation Handshakes MiTM Plaintext Data Injection

Figure 48 - High to Medium Vulnerabilities.

MEDIUM	5.8	17762	OpenSSL < 0.9.8j Signature Spoofing
MEDIUM	5.8	42880	SSL / TLS Renegotiation Handshakes MiTM Plaintext Data Injection
MEDIUM	5.1	17765	OpenSSL < 0.9.8l Multiple Vulnerabilities
MEDIUM	5.0	11213	HTTP TRACE / TRACK Methods Allowed
MEDIUM	5.0	59076	OpenSSL 0.9.8 < 0.9.8x DTLS CBC Denial of Service
MEDIUM	5.0	17750	OpenSSL < 0.9.6m / 0.9.7d Denial of Service
MEDIUM	5.0	12110	OpenSSL < 0.9.6m / 0.9.7d Multiple Remote DoS
MEDIUM	5.0	17755	OpenSSL < 0.9.7h / 0.9.8a Protocol Version Rollback
MEDIUM	5.0	17759	OpenSSL < 0.9.8 Weak Default Configuration
MEDIUM	5.0	17761	OpenSSL < 0.9.8i Denial of Service
MEDIUM	5.0	17763	OpenSSL < 0.9.8k Multiple Vulnerabilities
MEDIUM	5.0	58564	OpenSSL < 0.9.8u Multiple Vulnerabilities
MEDIUM	5.0	15901	SSL Certificate Expiry
MEDIUM	5.0	35291	SSL Certificate Signed Using Weak Hashing Algorithm
MEDIUM	5.0	42873	SSL Medium Strength Cipher Suites Supported (SWEET32)
MEDIUM	4.3	17696	Apache HTTP Server 403 Error Page UTF-7 Encoded XSS

Figure 49 - Medium to Low Vulnerabilities.

MEDIUM	4.3	17696	Apache HTTP Server 403 Error Page UTF-7 Encoded XSS
MEDIUM	4.3	88098	Apache Server ETag Header Information Disclosure
MEDIUM	4.3	17756	OpenSSL < 0.9.7k / 0.9.8c PKCS Padding RSA Signature Forgery Vulnerability
MEDIUM	4.3	56996	OpenSSL < 0.9.8h Multiple Vulnerabilities
MEDIUM	4.3	89058	SSL DROWN Attack Vulnerability (Decrypting RSA with Obsolete and Weakened eNcryption)
MEDIUM	4.3	65821	SSL RC4 Cipher Suites Supported (Bar Mitzvah)
MEDIUM	4.3	26928	SSL Weak Cipher Suites Supported
MEDIUM	4.3	81606	SSL/TLS EXPORT_RSA <= 512-bit Cipher Suites Supported (FREAK)
MEDIUM	4.3	78479	SSLv3 Padding Oracle On Downgraded Legacy Encryption Vulnerability (POODLE)
LOW	2.6	64532	OpenSSL < 0.9.8y Multiple Vulnerabilities
LOW	2.6	83875	SSL/TLS Diffie-Hellman Modulus <= 1024 Bits (Logjam)
LOW	2.6	83738	SSL/TLS EXPORT_DHE <= 512-bit Export Cipher Suites Supported (Logjam)
LOW	2.1	17754	OpenSSL < 0.9.7f Insecure Temporary File Creation
LOW	N/A	69551	SSL Certificate Chain Contains RSA Keys Less Than 2048 bits
INFO	N/A	10114	ICMP Timestamp Request Remote Date Disclosure
INFO	N/A	48204	Apache HTTP Server Version

6.1.5 Blind SQL Injection

While we only attempted a few queries to test the application and stopped once successful, other commands may be available for this type of attack. Our goal was to verify the existence of the security flaw and escalate privileges, the successful attempts are presented below:

Figure 51 - SQL Injection Attempt 1.

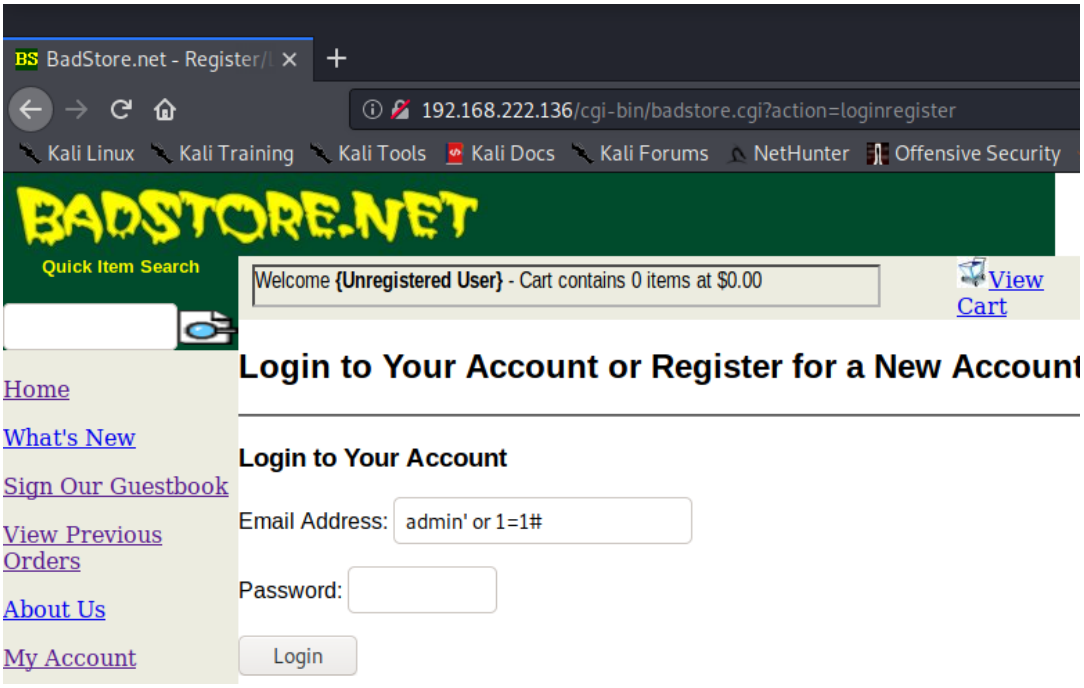
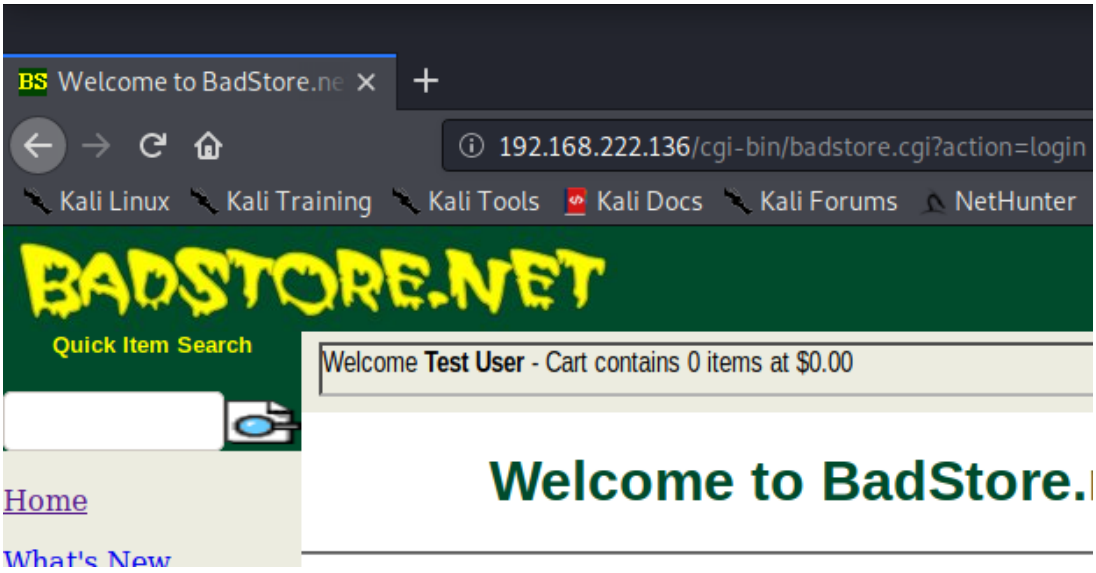


Figure 50 - SQL Injection Attempt 1 Successful.



admin' or 1=1# was used to query the database and execute a predefined SQL command. By adding **1=1,** which is a true statement, we are selecting the password for the user **admin,** since **1=1** is always true, each row will be verified until it meets the condition and allow us access. This is only a general concept of how SQL Injection works, we used several different combinations and variations of queries in order to achieve access. The command was successful, and we were logged in as a **Test User**.

Figure 53 - SQL Injection Attempt 2.

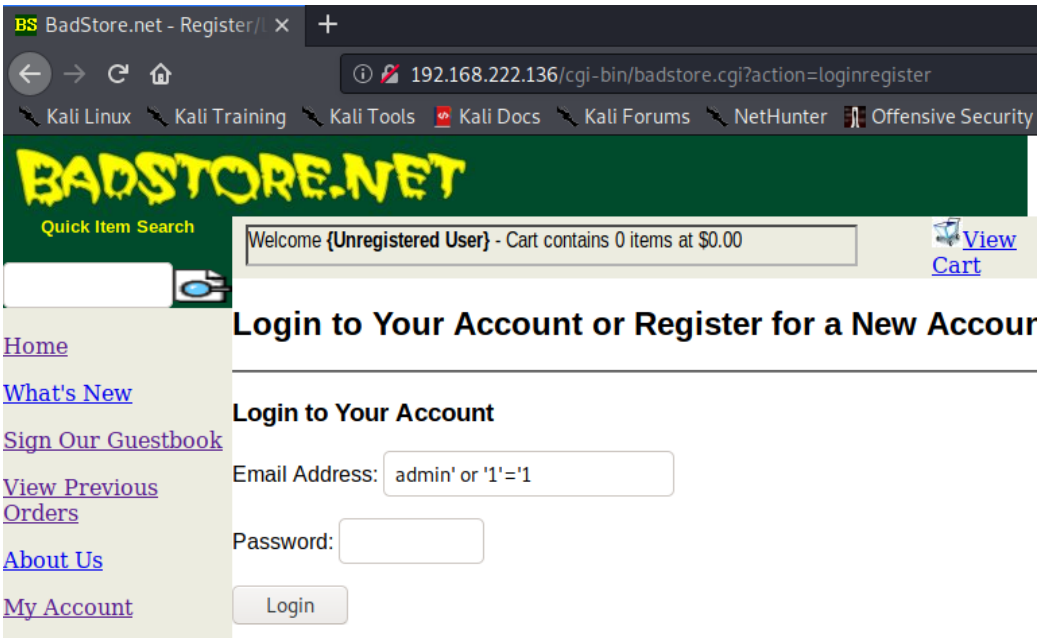
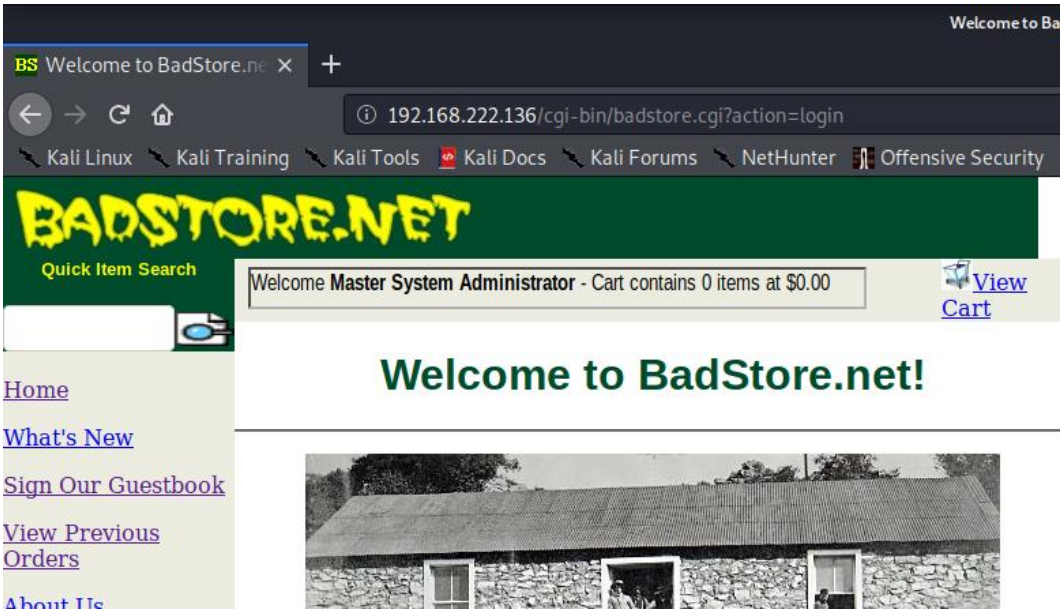


Figure 52 - SQL Injection Attempt 2 Successful.



The first attempt provided us with access to the Test User account, we have further attempted at escalating privileges. With modifications to the initial command and several unsuccessful attempts, we managed to gain Administrator access by using the following command: **admin' or '1'='1**.

The process was repeated for the Supplier Login, after a number of unsuccessful attempts we managed to override the login form using the previous command that provided us access to the Test User (**admin' or 1=1#**). Unlike the previous case, we remain an Unregistered User, but we were allowed access to the Supplier Upload Platform.

Figure 55 - SQL Injection Attempt 3.

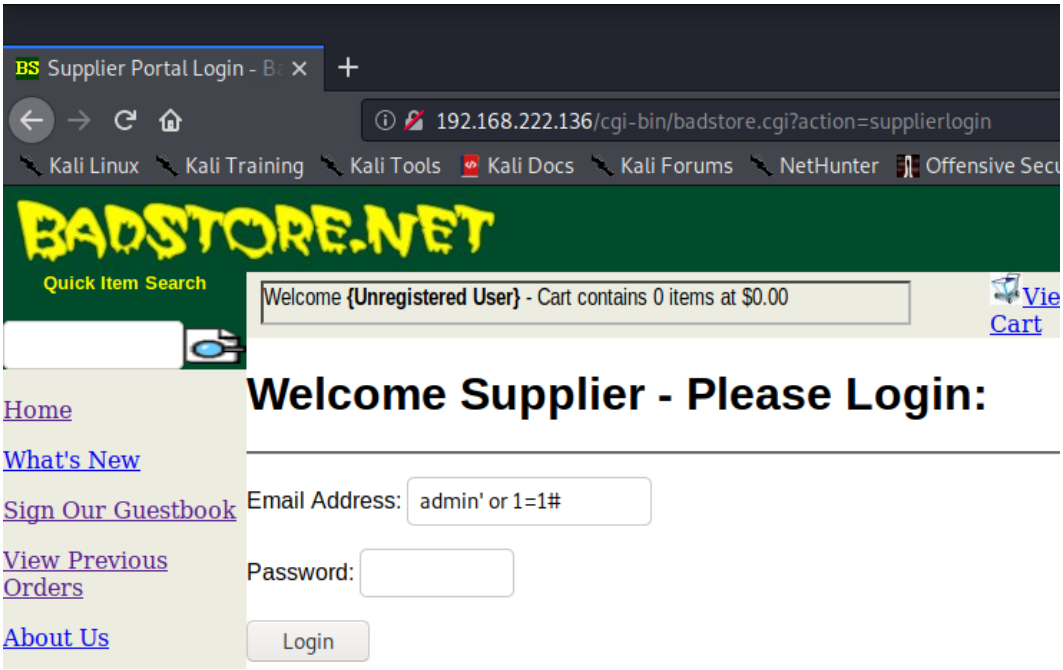
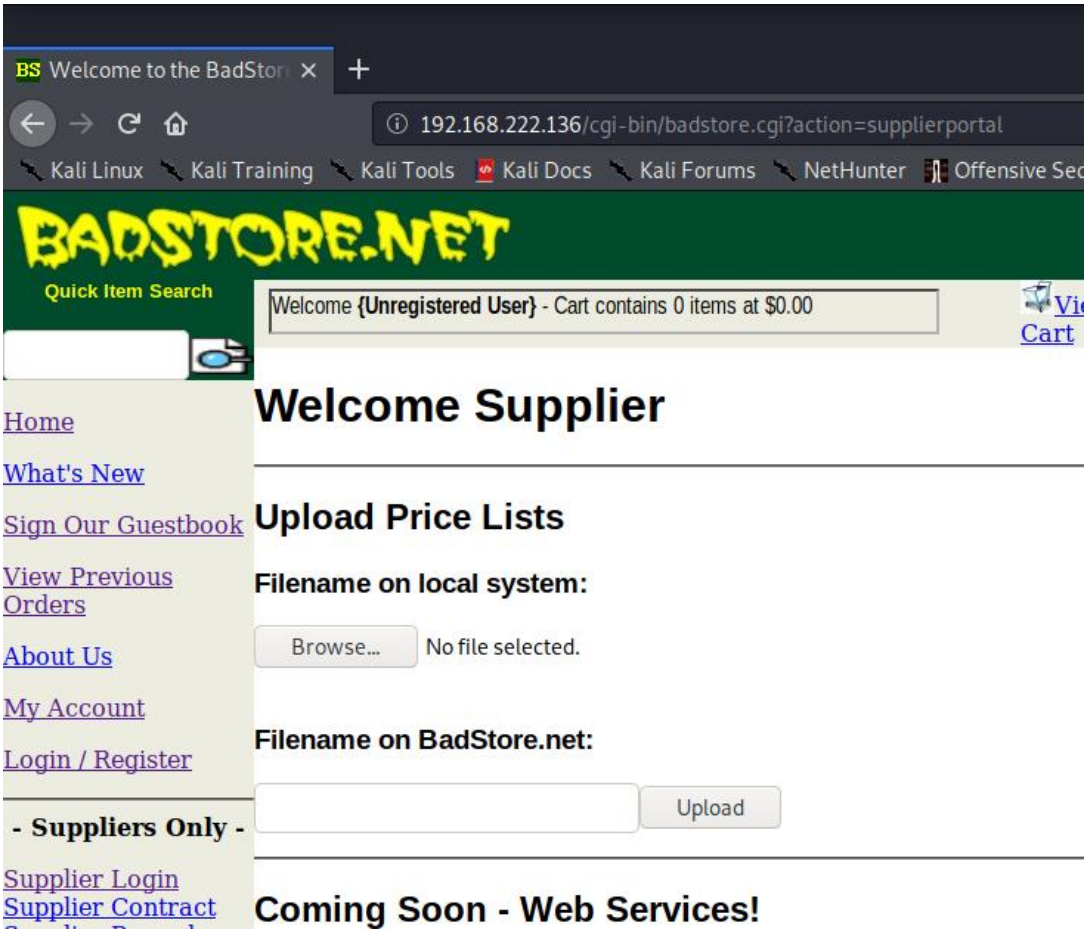


Figure 54 - SQL Injection Attempt 3 Successful.



6.1.6 Cross-site Scripting (XSS) Attack

In order to check for any vulnerable parts of the website, we used the following scripts:

- **</h2><script>alert(1);</script><h2>**
- **<img src=x onerror=alert(1);>**

If vulnerable, the alert scripts will simply be inserted into the HTML code. To be noted that the scripts are harmless, and their only purpose is verifying the existence of the vulnerability.

Figure 56 - XSS Attack on Password Reset Portal.

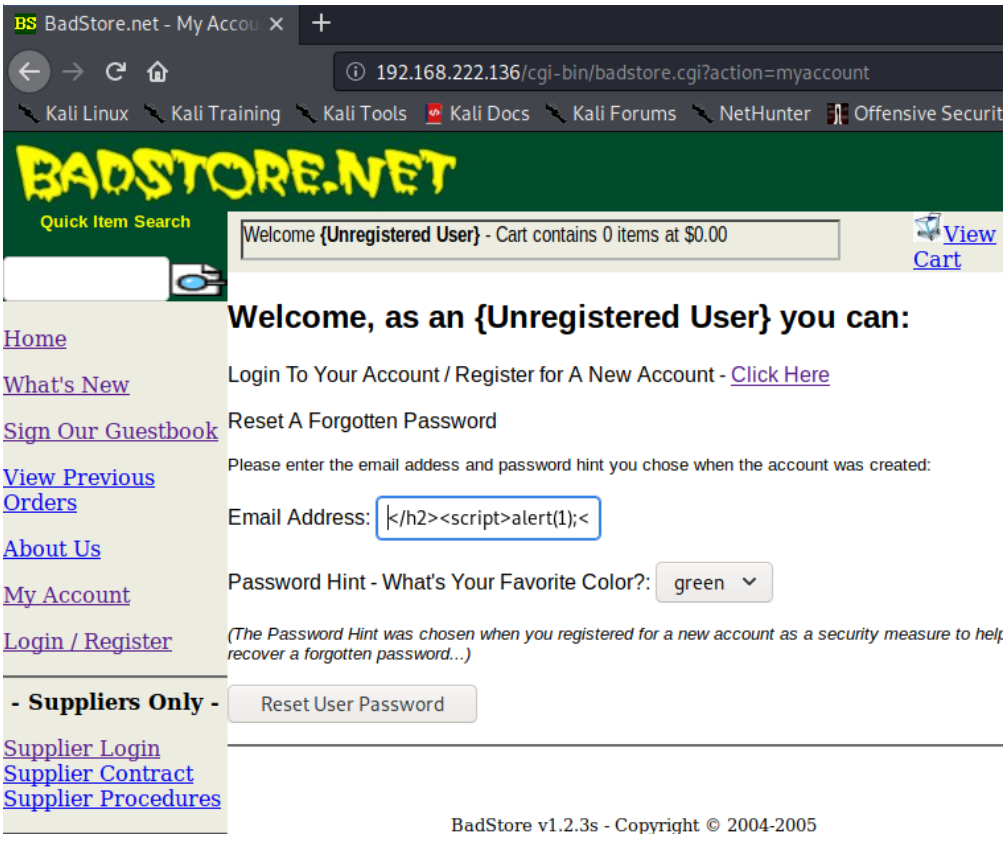


Figure 57 - Evidence of XSS Attack on Password Reset Portal

```
</TBODY></TABLE></TD>
    <TD width=1 bgColor=#ecece0>
      </TD>
    <TD width=615>
      <TABLE cellSpacing=0 cellPadding=0 width=614 border=0>
        <TBODY>
        <TR>
          <TD bgColor=#333333></TD></TR></TBODY></TABLE>
      <TABLE cellSpacing=0 cellPadding=0 width=614 border=0>
        <TBODY>
        <TR bgColor=#ecece0></TR>
        <TR bgColor=#ecece0>
          <TD class=normal width=550 height=20> 
<IFRAME name="bsheader" src="bsheader.cgi" height=27 width=450 marginheight=0 marginwidth=0 scrolling="no" noresize></IFRAME>
          </TD>
          <TD class=normal width=120><IMG SRC="/images/cart.jpg"><A
HREF="/cgi-bin/badstore.cgi?action=cartview">View Cart</A></TD></TR>
        <TR bgColor=#333333>
          </TR></TBODY></TABLE><FONT FACE='Arial'>
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<HTML><HEAD><TITLE>BadStore.net - Reset Password for User</TITLE>
</HEAD><BODY><H2>The password for user:   </h2><script>alert(1);</script><h2> <P>  ...has been reset to:  Welcome</H2><HR><P><
</BODY></HTML>
```

Figure 58 - XSS Attack on Supplier Upload Platform.

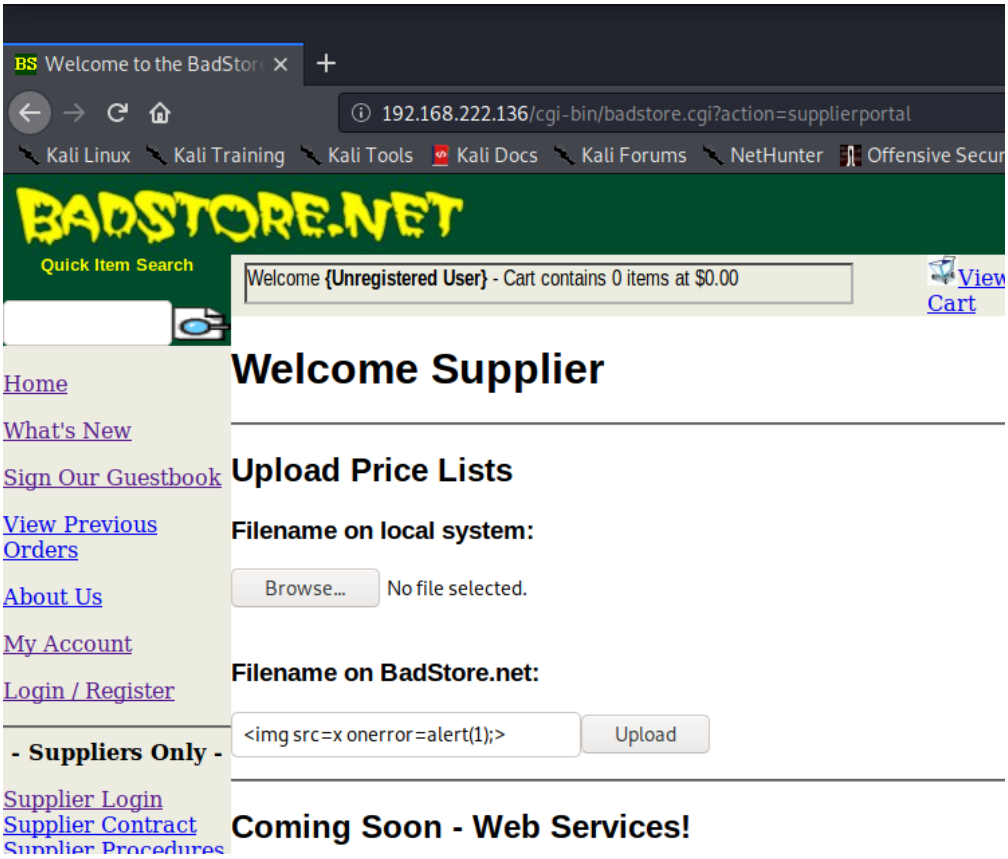


Figure 59 - Evidence of XSS Attack on Supplier Upload Platform.

```
</TBODY></TABLE></TD>
    <TD width=1 bgColor=#ecece0>
      </TD>
    <TD width=615>
      <TABLE cellSpacing=0 cellPadding=0 width=614 border=0>
        <TBODY>
        <TR>
          <TD bgColor=#333333></TD></TR></TBODY></TABLE>
      <TABLE cellSpacing=0 cellPadding=0 width=614 border=0>
        <TBODY>
        <TR bgColor=#ecece0></TR>
        <TR bgColor=#ecece0>
          <TD class=normal width=550 height=20> 
<IFRAME name="bsheader" src="bsheader.cgi" height=27 width=450 marginheight=0 marginwidth=0 scrolling="no" noresize></IFRAME>
          </TD>
          <TD class=normal width=120><IMG SRC="/images/cart.jpg"><A
HREF="/cgi-bin/badstore.cgi?action=cartview">View Cart</A></TD></TR>
        <TR bgColor=#333333>
          </TR></TBODY></TABLE><FONT FACE='Arial'>
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<HTML><HEAD><TITLE>BadStore.net - Supplier Upload</TITLE>
</HEAD><BODY><H1>Upload a file</H1><P><H2>Thanks for uploading your new pricing file!</H2><P><H3>Your file has been uploaded: <img src=x onerror=alert(1);></H3><P>
```

Figure 61 - XSS Attack on Guestbook.

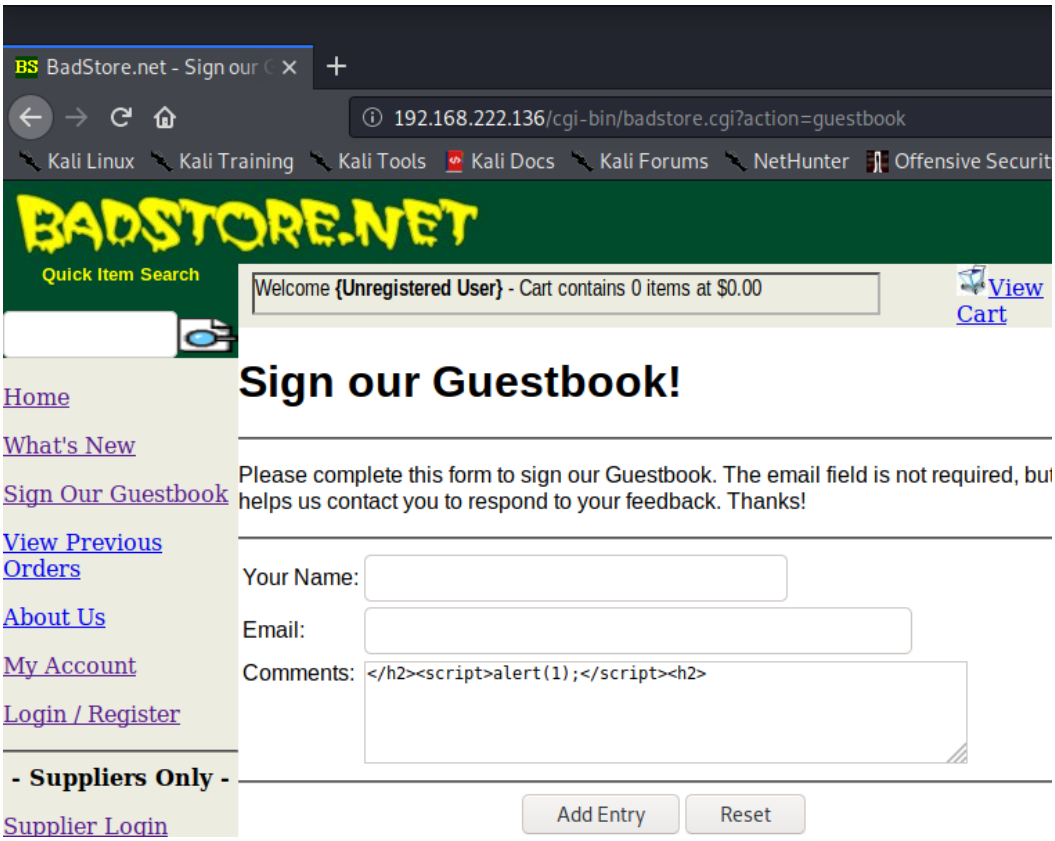


Figure 60 - Evidence of XSS Attack on Guestbook.

```
94 Wednesday, February 18, 2004 at 11:41:07: <B>John Q. Public</B> <A HREF=mailto:jqp@whitehouse.gov>jqp@whitehouse.gov</A
95 <OL><I>Let me know when the summer items are in.
96 </I></OL>
97 <HR>
98 Friday, February 20, 2004 at 14:05:22: <B>Big Spender</B> <A HREF=mailto:billg@microsoft.com>billg@microsoft.com</A>
99 <OL><I>Where's the big ticket items?
100 </I></OL>
101 <HR>
102 Sunday, February 22, 2004 at 06:16:05: <B>Evil Hacker</B> <A HREF=mailto:s8n@haxor.com>s8n@haxor.com</A>
103 <OL><I>You have no security!  I can own your site in less than 2 minutes.  Pay me $100,000 US currency by the end of da
104 </I></OL>
105 <HR>
106 Monday, April 6, 2020 at 19:57:06: <B></B> <A HREF=mailto:></A>
107 <OL><I></h2><script>alert(1);</script><h2>
108 </I></OL>
109 <HR>
110 <HR><P><BR><Center><FONT SIZE=2, FACE='Times'>BadStore v1.2.3s - Copyright &#169; 2004-2005</Center>
111 </BODY></HTML>
```

The attacks were successful, and the alert scripts were inserted into the HTML code.

6.1.7 Unsuccessful attempts of accessing directories from robot.txt

We have attempted at accessing the following directories found in the robot.txt file:

- /backup
- /cgi-bin
- /upload

Figure 64 - Attempt at accessing /backup directory.

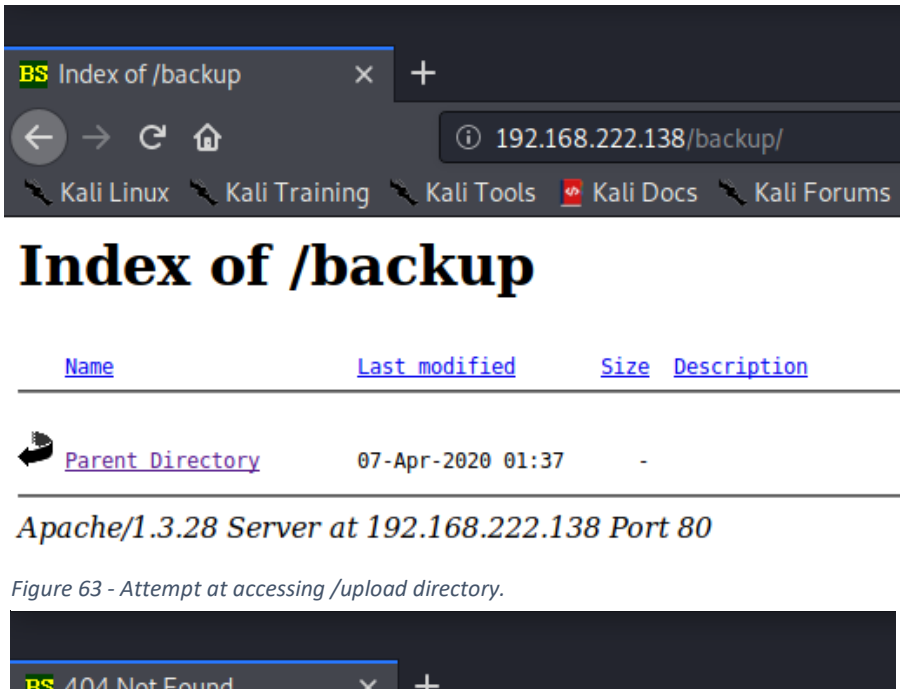


Figure 63 - Attempt at accessing /upload directory.

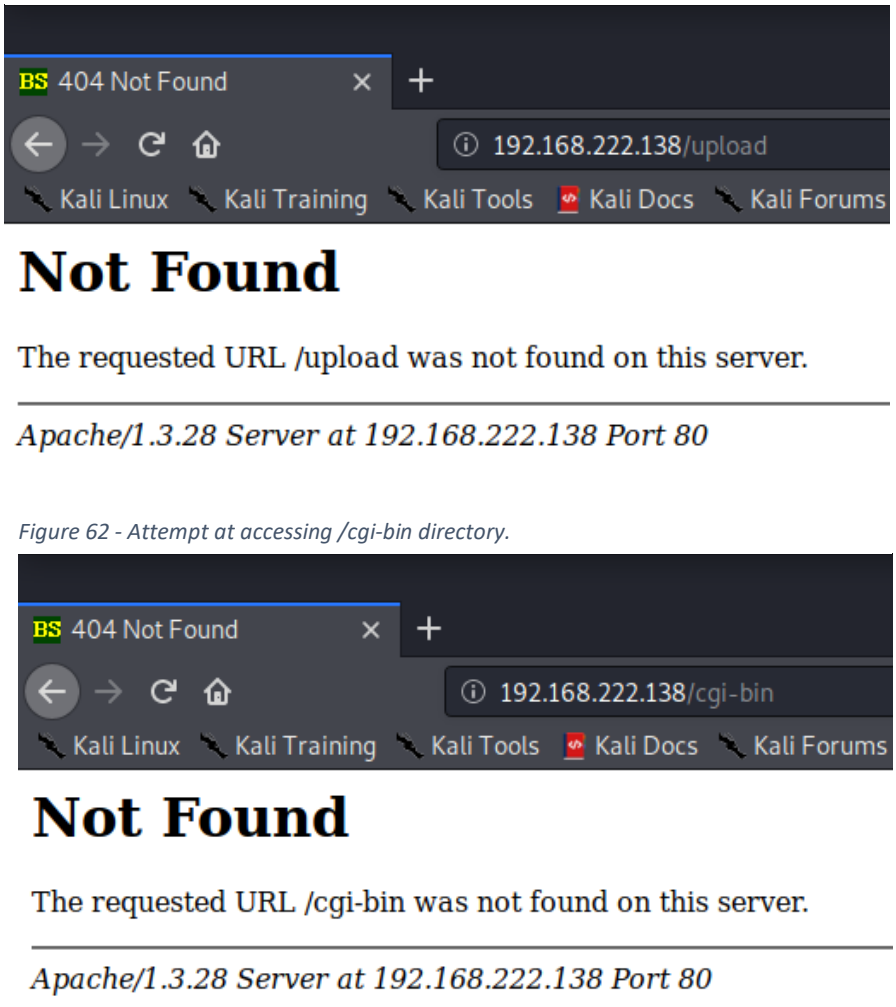
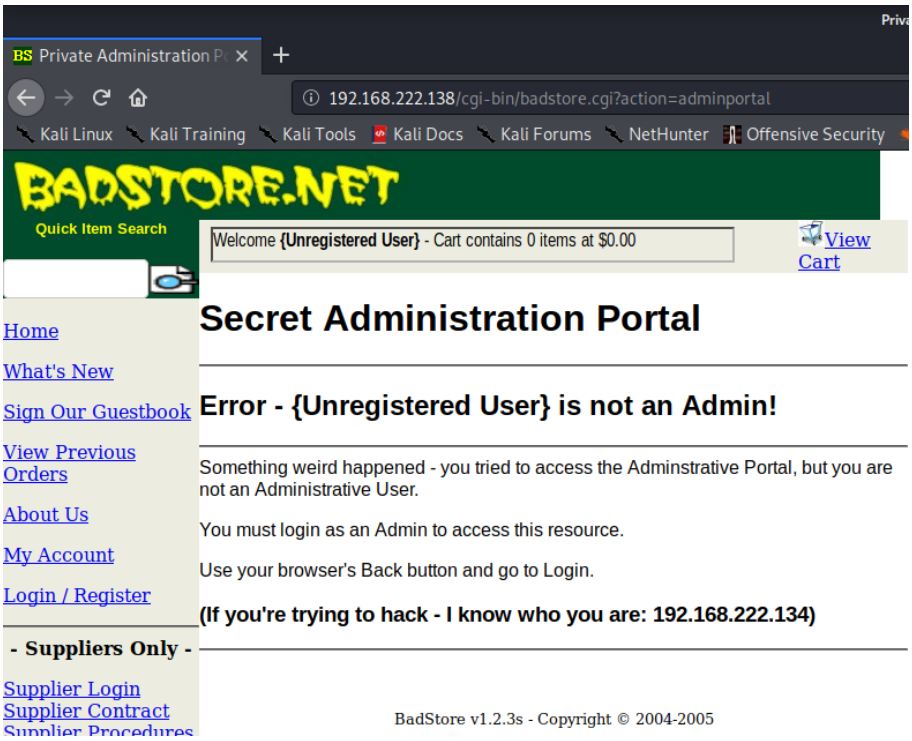


Figure 62 - Attempt at accessing /cgi-bin directory.

6.1.8 Secret Admin Portal

After discovering the secret admin portal, we attempted to access the sales report but without an admin account we were unsuccessful.

Figure 65 - Trying to access Sales Report.



6.1.9 More Exposed Directories

After the early attempts to access the directories found in the robot.txt file, we created a regular user and re-attempted to access the /backup directory.

Figure 66 - Create user account.

As we can see from the figures below, we were provided access to both, directory and files, using a regular user account created earlier. The backups are stored in plain text with no extra security which creates a high security risk and exposure.

Figure 69 - Access to /backup directory.

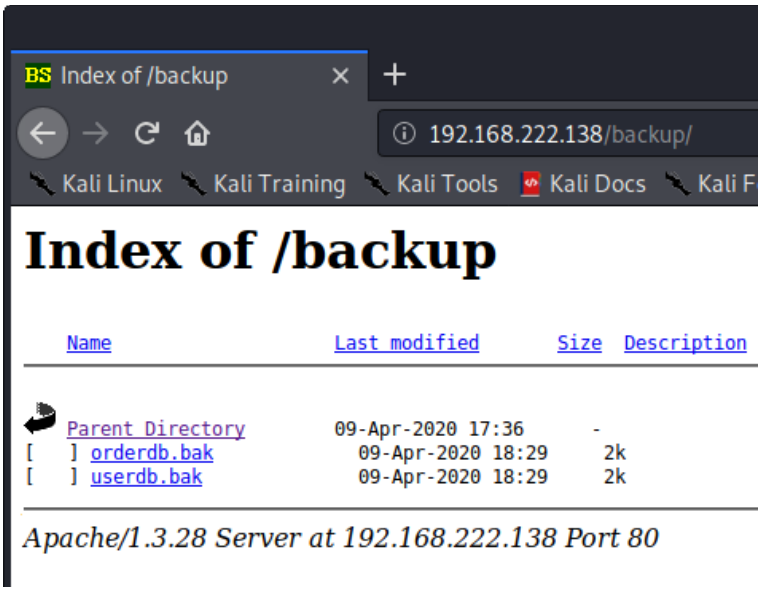


Figure 68 - Access to orderdb.bak.

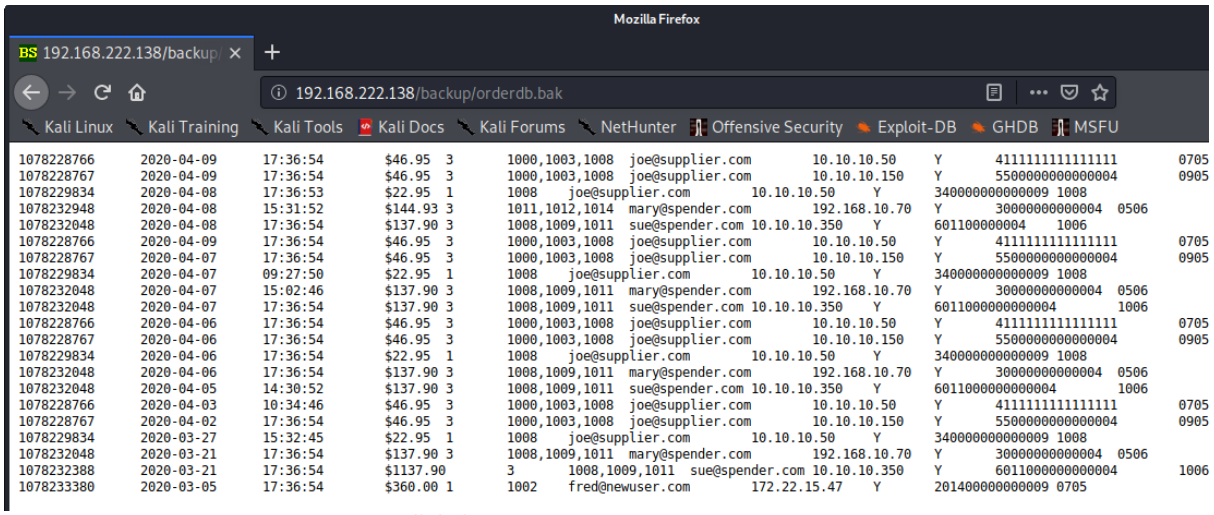
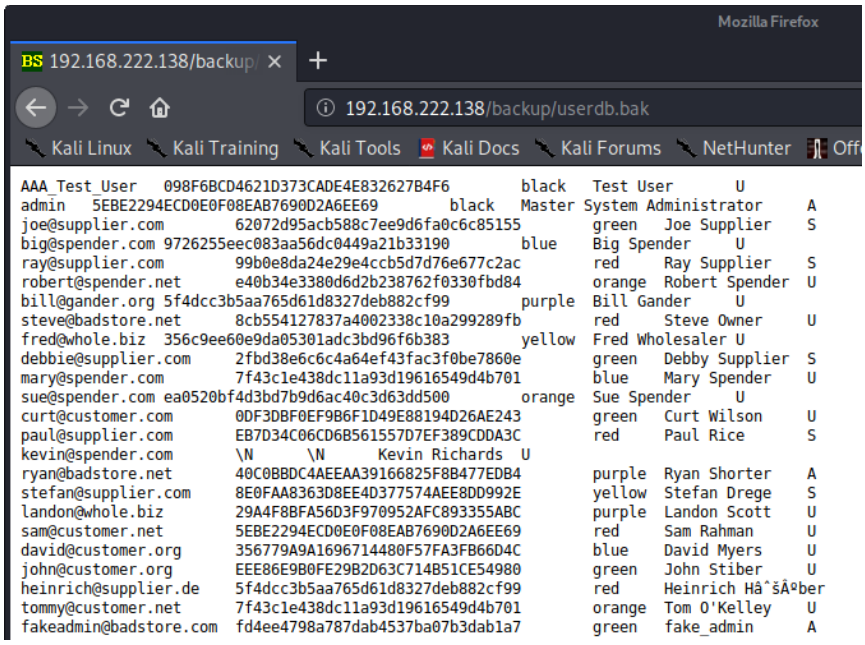


Figure 67 - Access to userdb.bak



6.1.10 Operating System Exploitation

1. Heartbeat Extension 'Heartbleed attack'

From previous scans (Nessus) we found the application to be potentially vulnerable to a 'Heartbleed Attack'. In order to make a full assessment we performed the attack using Metasploit (a framework used for vulnerability exploitation).

Figure 71 - Metasploit Configuration for Heartbleed attack.

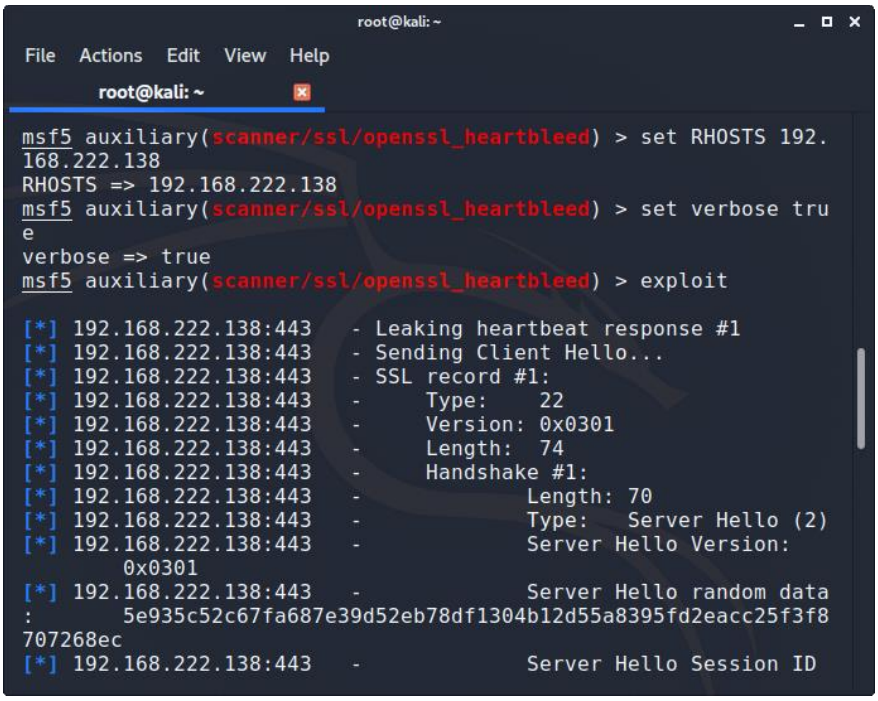


Figure 70 - Attempting to exploit Heartbleed.

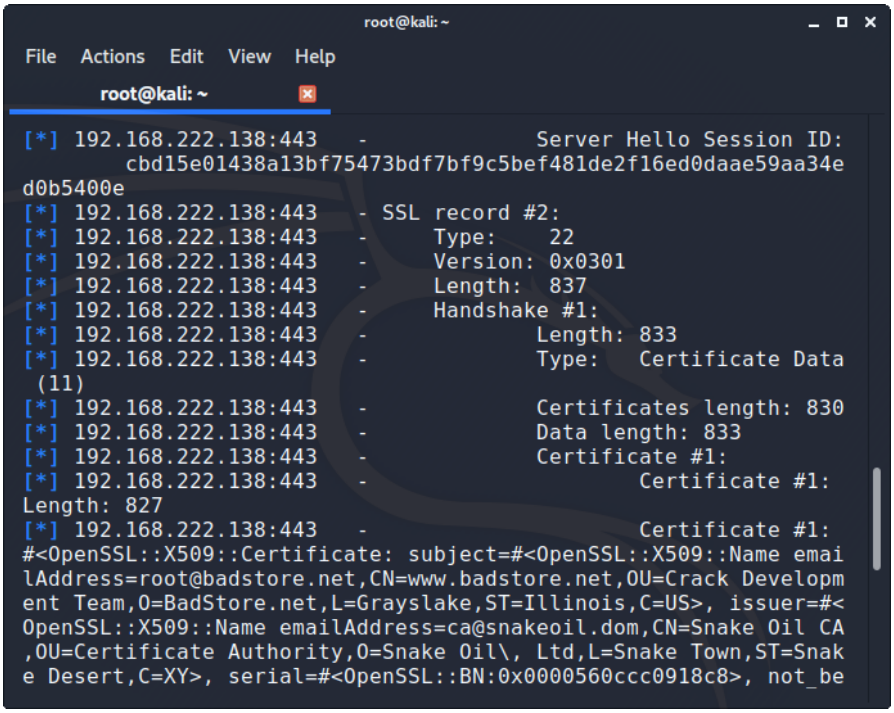
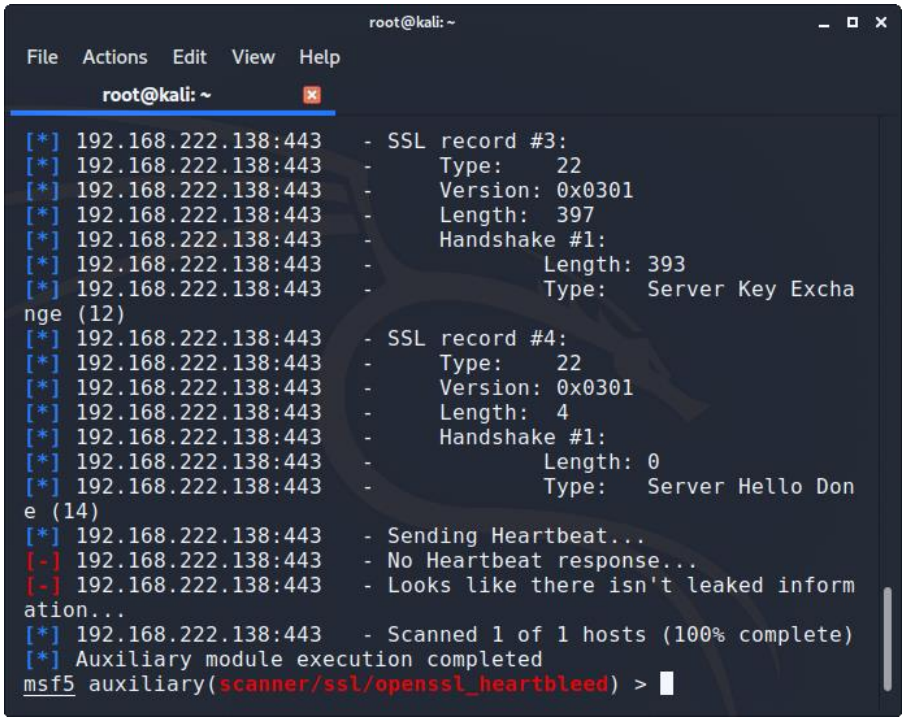


Figure 72 - Heartbleed Response.



As we can see above, the attack was unsuccessful, and no information is leaked. We configured Metasploit framework by setting (RHOSTS) remote hosts to our target and selecting (verbose) to receive details about the attack.

More information and mitigation available in Appendix 6.2.8.

2. SWEET32: Birthday Attack

Due to the vulnerable OpenSSL, an attacker could perform a 'birthday attack' by taking advantage of the SSL/TLS protocol. We used Nmap to check for any vulnerable ciphers.

Figure 73 - Nmap script for listing vulnerable ciphers.

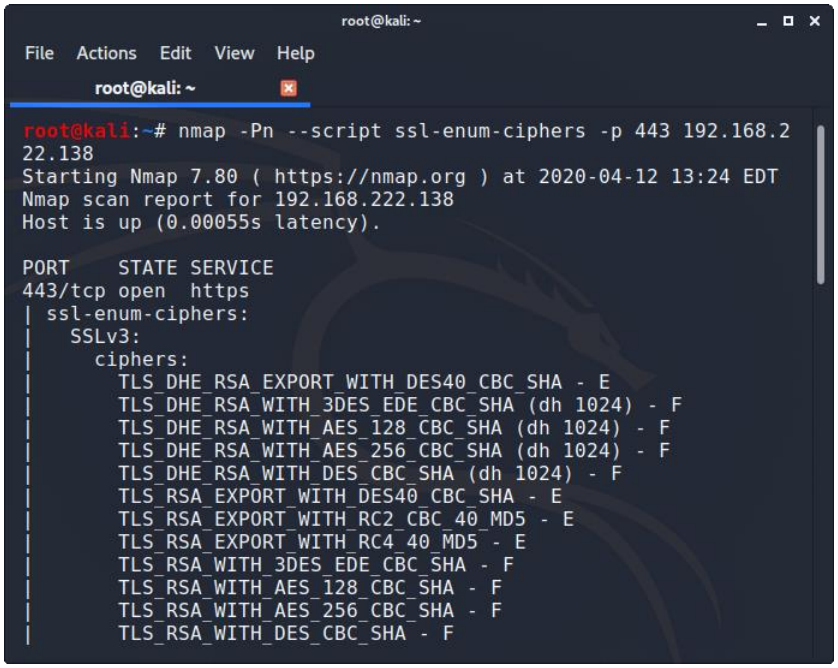


Figure 75 - Nmap returns vulnerable ciphers.

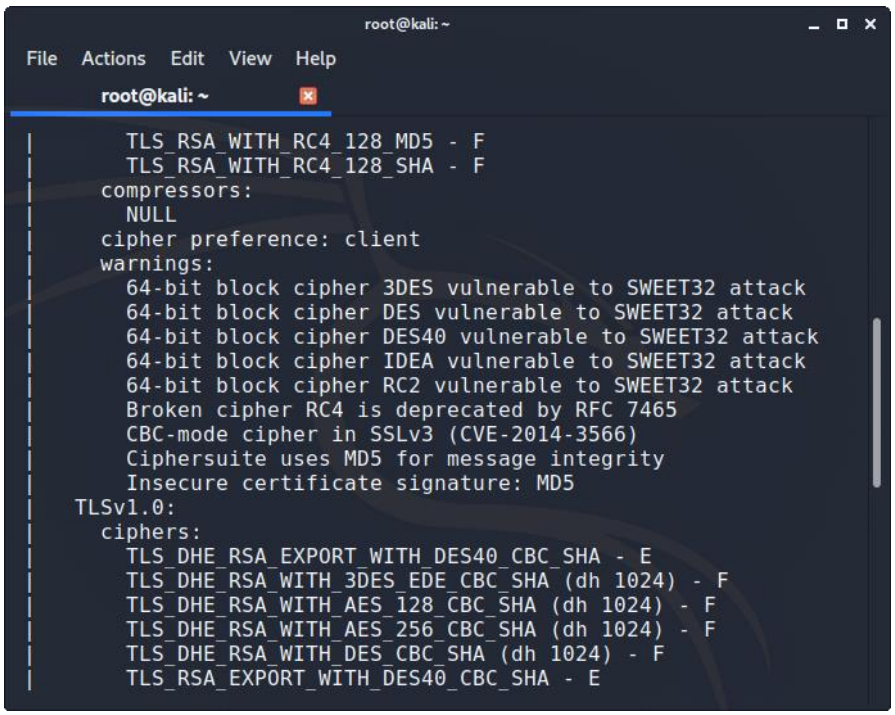
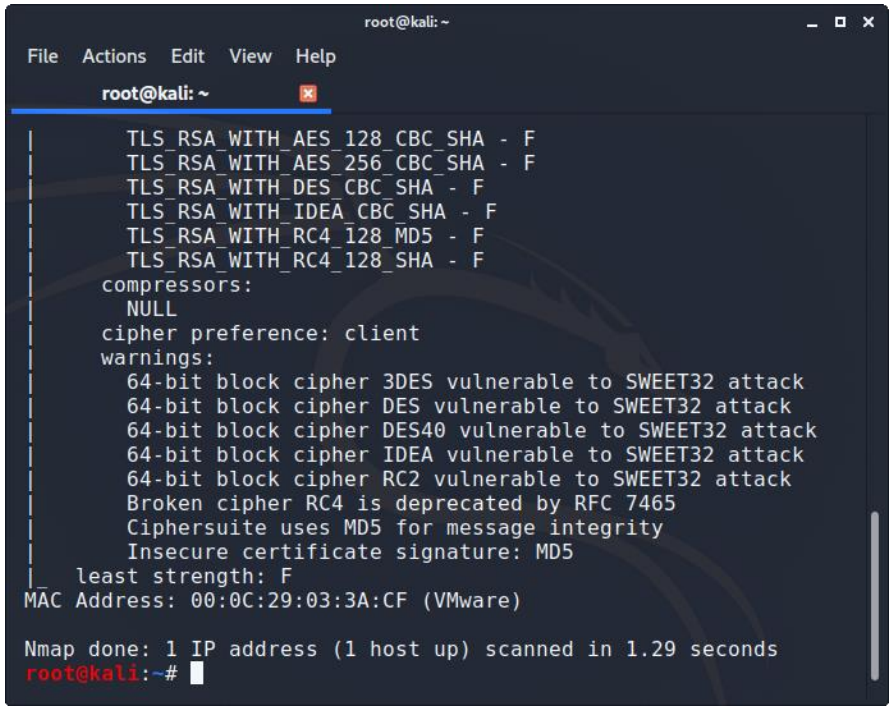


Figure 74 - Vulnerable ciphers discovered.



The scan returned several vulnerable ciphers. While the application is vulnerable to such an attack, a full-scale attack proved to be unpractical and has not been attempted.

More information and remediation strategy available in Appendix 6.2.9.

3. POODLE (‘padding-oracle attack’’) Attack

The outdated SSL protocol also makes it possible for a poodle attack due to the use of nondeterministic CBC (cipher block chaining) padding, which makes it easier for man-in-the-middle attackers to obtain cleartext data.

We used a script provided by Nmap to check for the vulnerability.

Figure 76 - Use Nmap Script to check for POODLE.

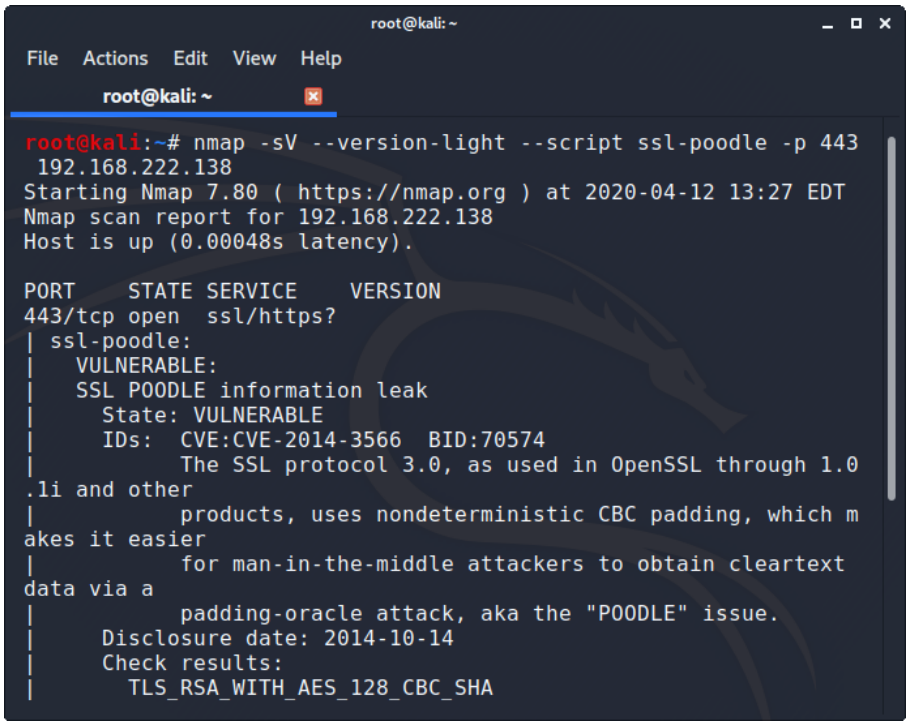
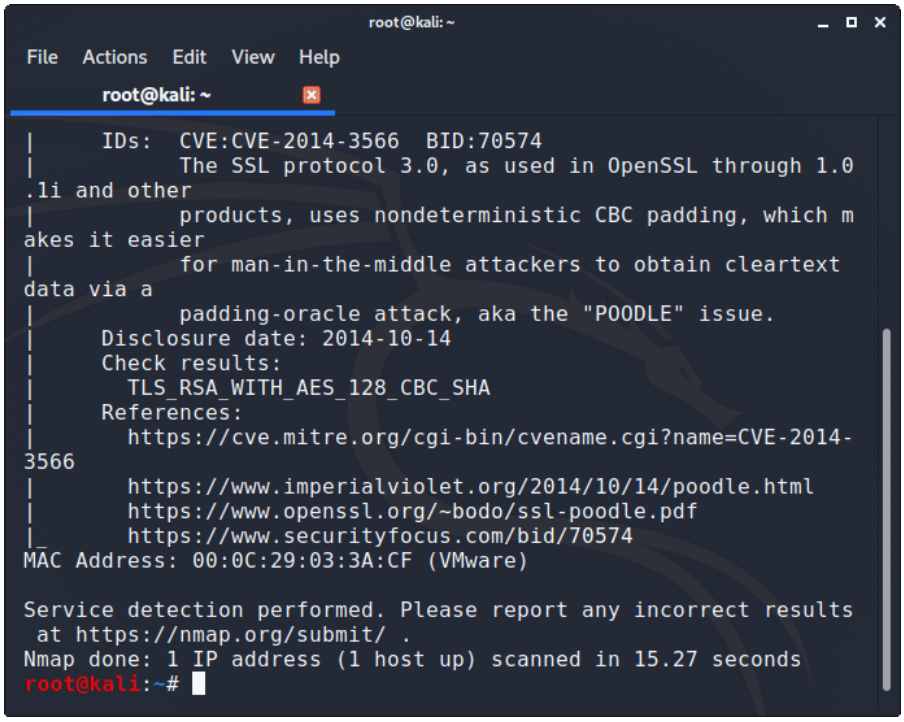


Figure 77 - Application vulnerable to POODLE.



As in previous cases, the system is vulnerable to such an attack but is unpractical and unlikely to be exploited. That as it may, we highly recommend disabling this version of the protocol or replacing with a more secure version.

For more information please see Appendix 6.2.10.

6.2 Appendix 2 – Vulnerability Details & Mitigation

The Risk Assessment performed is in accordance with the NIST SP 800-30r1 [3], the vulnerabilities exploited were based upon likelihood in the current state of the system and impact to determine overall risk.

6.2.1 SQL Injection

IMPACT: HIGH

DESCRIPTION:

SQL Injection is the use of special queries (in order to raise the execution of predefined SQL commands) and their insertion into a web application via input data from clients. If successful, an SQL Injection attack can read/modify data on a database and execute administration operations, including shutting down the database. In some cases, commands can be issued to the operating system [4].

MITIGATION:

- The use of Prepared Statements (with Parameterized Queries),
- The use of Stored Procedures,
- Whitelist Input Validation,
- Escaping All User Supplied Input [5].

For more information and further details on remediation strategies, please see the **OWASP SQL Injection Prevention Sheet** [5].

6.2.2 Cross-site Scripting (XSS)

IMPACT: HIGH

DESCRIPTION:

XSS occurs when a web application uses input from a user within the output it generates without validation or encoding. XSS is often used by attackers to send malicious code to end users, an end user's browser will not check the script as it's considered trusted and execute it. The malicious script can access several sensitive information such as cookies, session tokens and can rewrite the content for the HTML page in some cases [6].

MITIGATION:

- Perform appropriate validation and escaping on the server-side,
- Use HTML encoding entity method along with escape syntax for the part of the HTML document that will input untrusted data,
- Use a security encoding library [7].

The rules above are a brief example of mitigation techniques against this type of attack, for a more comprehensive and detailed remediation strategy, we recommend the **Cross-site Prevention Cheat Sheet from OWASP** [7].

6.2.3 Robots.txt

IMPACT: MEDIUM

DESCRIPTION:

Robots.txt is a file intended to prevent web robots from visiting certain directories in a website for maintenance or indexing purposes. An attacker may be able to use the contents of this file to learn sensitive documents or directories on the affected site and either retrieve them directly or target them for other attacks [8].

MITIGATION:

To avoid the file being used for malicious purposes, we recommend the following:

- Review contents of file,
- Use of META tags rather than entries,
- Adjust the web server's access controls to limit access to sensitive material [8].

6.2.4 Directory Listing

IMPACT: MEDIUM

DESCRIPTION:

Directory listing is a feature on the web servers that can list the content of a directory when there is no index file (e.g. index.php/index.html) present. If a request is made to a directory on which directory listing is enabled, and there is no index file present such as index.php, even if there are files from a web application, the web server will send a directory listing as a response. This creates an information leakage issue, attackers can use such information to perform other attacks, including direct impact vulnerabilities such as XSS [9].

MITIGATION:

We recommend disabling directory listing at web server level as a secure solution.

6.2.5 Weak Credential Encoding

IMPACT: HIGH

DESCRIPTION:

Insecure storage of credentials, such as passwords, occurs when the password is stored in plaintext or protected using weak forms of obscuring it in an application’s properties or configuration file [10].

MITIGATION:

To protect sensitive information, we recommend the following:

- The use of a high-level cryptography library reviewed by experts,
- The use of password hashing algorithms, rather than encryption,

- Hashing the password and encrypting the hash,
- The use of digital signatures to provide authenticity for downloads [11].

6.2.6 Insufficient Transport Layer Protection

IMPACT: HIGH

DESCRIPTION:

Applications frequently transmit sensitive information like authentication details, credit card information, and session tokens over a network. When using weak algorithms or using expired or invalid certificates or not using SSL, the communication can be exposed to untrusted users, which may compromise a web application and or steal sensitive information [12].

MITIGATION:

- Enable secure HTTP and enforce credential transfer over HTTPS only,
- Ensure your certificate is valid and not expired,
- Encrypt the data using a reliable encryption scheme before transmitting [12].

6.2.7 Web Parameter Tampering

IMPACT: HIGH

DESCRIPTION:

An attack based on the manipulation of parameters exchanged between client and server in order to modify application data, such as user credentials and permissions, etc. Usually, this information is stored in cookies, hidden form fields, or URL Query Strings, and is used to increase application functionality and control [13].

This attack can be performed by malicious users with the intent to exploit the application for personal benefit or who attempt to attack a third person using a man-in-the-middle-attack. The attack success depends on integrity and logic validation mechanism errors, and its exploitation can result in other consequences including XSS, SQL Injection, file inclusion, and path disclosure attacks [13].

MITIGATION:

- Input Validation (“Assume all input is malicious”),
- Use a list of acceptable inputs that strictly conform to specifications,
- Reject any input that does not strictly conform to specifications [14].

6.2.8 Heartbeat Extension 'Heartbleed Attack'

IMPACT: LOW

DESCRIPTION:

The Heartbleed Bug is a vulnerability in the OpenSSL cryptographic software library. This allows stealing the information protected, under normal conditions, by the SSL/TLS encryption used to secure the Internet. This allows attackers to eavesdrop on communications, steal data directly from the services and users and to impersonate services and users [15].

MITIGATION:

We recommend updating the vulnerable OpenSSL or contacting your service providers for obtaining a fix.

6.2.9 SWEET32: 'Birthday attack'

IMPACT: MEDIUM

DESCRIPTION:

Protocols such as TLS, SSH, and IPSec use DES and Triple DES ciphers have a birthday bound of approximately four billion blocks, that makes it easier for remote attackers to obtain cleartext data via a birthday attack against a long-duration encrypted session [16].

A network attacker who can monitor a long-lived Triple-DES HTTPS connection between a web browser and a website can recover secure HTTP cookies by capturing traffic [16].

MITIGATION:

To avoid such an attack:

- stop using legacy 64-bit block ciphers,
- rekeying the session frequently [17].

6.2.10 POODLE Attack

IMPACT: LOW

DESCRIPTION:

The POODLE attack takes advantage of the protocol version negotiation feature built into SSL/TLS to force the use of SSL 3.0 and then leverages this new vulnerability to decrypt select content within the SSL session. The decryption is done byte by byte and will generate a large number of connections between the client and server [18].

MITIGATION:

Solution is to:

- disable SSLv3,
- additionally, services that must support SSLv3 should enable the TLS Fallback SCSV mechanism [19].