

# Penetration Test Report

BADSTORE.NET

April 25<sup>th</sup>, 2018

## Table of Contents

OWASP PARAMETER VULNERABILITY REPORT	8
<b>Executive Summary</b>	<b>9</b>
<i>Summary of Results</i>	10
<b>Attack Narrative</b>	<b>11</b>
<i>Enumerate System Find</i>	11
<i>Admin Webserver Interface Compromise</i>	12
<i>Directory Listing Suppliers</i>	14
<i>Cross Site Scripting</i>	20
<i>SQL Injection</i>	27
<i>Cross Site Request Forgery</i>	32
<i>Reset Password Functionality Vulnerability</i>	36
<b>Conclusion</b>	<b>37</b>
<i>Recommendations</i>	38
<i>Risk Rating</i>	44
<b>Appendix A: Vulnerability Detail and Mitigation</b>	<b>45</b>
<i>Risk Rating Scale</i>	45
<i>Directory Path Traversal/Enumerate System Find</i>	46
<i>Admin WebServer Interface Compromise</i>	46
<i>Cross Site Scripting</i>	47
<i>SQL Injection</i>	47
<i>Cross Site Request Forgery</i>	48
<i>Reset Password Functionality Vulnerability</i>	48

## OWASP PARAMETER VULNERABILITY REPORT:

	<p><b>Test Parameters:</b></p> <p><b>Project Objective:</b> Objective is to uncover the vulnerabilities and propose the security solution for it.</p> <p><b>Project Scope:</b> This section is related to securing the online customers who make payment in ecommerce website.</p> <p><b>Project Schedule:</b> 4 Days.</p> <p><b>Targets:</b> Targeted System is entry and exit criteria of Application.</p> <p><b>Limitations:</b> Project is limited to 80 and 443 findings.</p>		
	<p><b>Vulnerabilities</b></p> <p><b>Category/Severity - P0 - Critical, P1 - Major, P2 - Medium, P3 - Low.</b></p>		
<b>Test ID</b>	<b>Test Description</b>	<b>Findings</b>	<b>Severity</b>
<b>Information Gathering</b>			
OTG-INFO-001	Conduct Search Engine Discovery and Reconnaissance for Information Leakage	Y	P0
OTG-INFO-002	Fingerprint Web Server	Y	P0
OTG-INFO-003	Review Webserver Metafiles for Information Leakage	Y	P0

OTG-INFO-004	Enumerate Applications on Webserver	Y	P0
OTG-INFO-005	Review Webpage Comments and Metadata for Information Leakage	Y	P0
OTG-INFO-006	Identify application entry points	Y	P0
OTG-INFO-007	Map execution paths through application	Y	P0
OTG-INFO-009	Fingerprint Web Application Framework	Y	P0
OTG-INFO-009	Fingerprint Web Application	Y	P0
OTG-INFO-010	Map Application Architecture	Y	P0
<b>Configuration and Deploy Management Testing</b>			
OTG-CONFIG-001	Test Network/Infrastructure Configuration	Y	NA
OTG-CONFIG-002	Test Application Platform Configuration	Y	NA
OTG-CONFIG-003	Test File Extensions Handling for Sensitive Information	Y	NA
OTG-CONFIG-004	Backup and Unreferenced Files for Sensitive Information	Y	P0
OTG-CONFIG-005	Enumerate Infrastructure and Application Admin Interfaces		NA
OTG-CONFIG-006	Test HTTP Methods	Y	NA

OTG-CONFIG-007	Test HTTP Strict Transport Security	Y	NA
OTG-CONFIG-008	Test RIA cross domain policy	Y	NA
<b>Identity Management Testing</b>			
OTG-IDENT-001	Test Role Definitions	Y	P0
OTG-IDENT-002	Test User Registration Process	Y	NA
OTG-IDENT-003	Test Account Provisioning Process	Y	NA
OTG-IDENT-004	Testing for Account Enumeration and Guessable User Account	Y	P0
OTG-IDENT-005	Testing for Weak or unenforced username policy	Y	P0
OTG-IDENT-006	Test Permissions of Guest/Training Accounts	Y	P0
OTG-IDENT-007	Test Account Suspension/Resumption Process	Y	P0
<b>Authentication Testing</b>			
OTG-AUTHN-001	Testing for Credentials Transported over an Encrypted Channel	Y	NA
OTG-AUTHN-002	Testing for default credentials	Y	NA
OTG-AUTHN-003	Testing for Weak lock out mechanism	Y	P0

OTG-AUTHN-004	Testing for bypassing authentication schema	Y	P0
OTG-AUTHN-005	Test remember password functionality	Y	NA
OTG-AUTHN-006	Testing for Browser cache weakness	Y	NA
OTG-AUTHN-007	Testing for Weak password policy	Y	NA
OTG-AUTHN-008	Testing for Weak security question/answer	Y	NA
OTG-AUTHN-009	Testing for weak password change or reset functionalities	Y	P0
OTG-AUTHN-010	Testing for Weaker authentication in alternative channel	Y	NA
<b>Authorization Testing</b>			
OTG-AUTHZ-001	Testing Directory traversal/file include	Y	NA
OTG-AUTHZ-002	Testing for bypassing authorization schema	Y	NA
OTG-AUTHZ-003	Testing for Privilege Escalation	Y	NA
OTG-AUTHZ-004	Testing for Insecure Direct Object References	Y	NA
<b>Session Management Testing</b>			
OTG-SESS-001	Testing for Bypassing Session Management Schema	Y	P0
OTG-SESS-002	Testing for Cookies attributes	Y	NA

OTG-SESS-003	Testing for Session Fixation	Y	NA
OTG-SESS-004	Testing for Exposed Session Variables	Y	NA
OTG-SESS-005	Testing for Cross Site Request Forgery	Y	NA
OTG-SESS-006	Testing for logout functionality	Y	NA
OTG-SESS-007	Test Session Timeout	Y	NA
OTG-SESS-008	Testing for Session puzzling	Y	P0
<b>Input Validation Testing</b>			
OTG-INPVAL-001	Testing for Reflected Cross Site Scripting	Y	P0
OTG-INPVAL-002	Testing for Stored Cross Site Scripting	Y	P0
OTG-INPVAL-003	Testing for HTTP Verb Tampering	Y	P0
OTG-INPVAL-004	Testing for HTTP Parameter pollution	Y	P0
OTG-INPVAL-006	Testing for SQL Injection	Y	P0
	Oracle Testing	Y	NA
	MySQL Testing	Y	P0
	SQL Server Testing	Y	NA
	Testing PostgreSQL	Y	NA
	MS Access Testing	Y	NA
	Testing for NoSQL injection	Y	NA
OTG-INPVAL-007	Testing for LDAP Injection	Y	NA

OTG-INPVAL-008	Testing for ORM Injection	Y	NA
OTG-INPVAL-009	Testing for XML Injection	Y	NA
OTG-INPVAL-010	Testing for SSI Injection	Y	NA
OTG-INPVAL-011	Testing for XPath Injection	Y	NA
OTG-INPVAL-012	IMAP/SMTP Injection	Y	NA
OTG-INPVAL-013	Testing for Code Injection	Y	NA
	Testing for Local File Inclusion	Y	NA
	Testing for Remote File Inclusion	Y	NA
OTG-INPVAL-014	Testing for Command Injection	Y	NA
OTG-INPVAL-015	Testing for Buffer overflow	Y	NA
	Testing for Heap overflow	Y	NA
	Testing for Stack overflow	Y	NA
	Testing for Format string	Y	NA
OTG-INPVAL-016	Testing for incubated vulnerabilities	Y	NA
OTG-INPVAL-017	Testing for HTTP Splitting/Smuggling	Y	
<b>Error Handling</b>			
OTG-ERR-001	Analysis of Error Codes	Y	NA
OTG-ERR-002	Analysis of Stack Traces	Y	NA
<b>Cryptography</b>			

OTG-CRYPST-001	Testing for Weak SSL/TSL Ciphers, Insufficient Transport Layer Protection	Y	NA
OTG-CRYPST-002	Testing for Padding Oracle	Y	NA
OTG-CRYPST-003	Testing for Sensitive information sent via unencrypted channels	Y	P0
<b>Business Logic Testing</b>			
OTG-BUSLOGIC-001	Test Business Logic Data Validation	Y	NA
OTG-BUSLOGIC-002	Test Ability to Forge Requests	Y	NA
OTG-BUSLOGIC-003	Test Integrity Checks	Y	NA
OTG-BUSLOGIC-004	Test for Process Timing	Y	NA
OTG-BUSLOGIC-005	Test Number of Times a Function Can be Used Limits	Y	NA
OTG-BUSLOGIC-006	Testing for the Circumvention of Work Flows	Y	NA
OTG-BUSLOGIC-007	Test Defenses Against Application Mis-use	Y	NA
OTG-BUSLOGIC-008	Test Upload of Unexpected File Types	Y	NA



OTG-BUSLOGIC-009	Test Upload of Malicious Files	Y	NA
<b>Client Side Testing</b>			
OTG-CLIENT-001	Testing for DOM based Cross Site Scripting	Y	NA
OTG-CLIENT-002	Testing for JavaScript Execution	Y	NA
OTG-CLIENT-003	Testing for HTML Injection	Y	P0
OTG-CLIENT-004	Testing for Client Side URL Redirect	Y	NA
OTG-CLIENT-005	Testing for CSS Injection	Y	NA
OTG-CLIENT-006	Testing for Client Side Resource Manipulation	Y	NA
OTG-CLIENT-007	Test Cross Origin Resource Sharing	Y	NA
OTG-CLIENT-008	Testing for Cross Site Flashing	Y	NA
OTG-CLIENT-009	Testing for Clickjacking	Y	NA
OTG-CLIENT-010	Testing WebSockets	Y	NA
OTG-CLIENT-011	Test Web Messaging	Y	NA
OTG-CLIENT-012	Test Local Storage	Y	NA

### Executive Summary

This project was contracted by client on to conduct a penetration test in order to determine its exposure to a targeted attack. All activities were conducted in a manner that simulated a malicious actor engaged in a targeted attack against BADESTORE.NET One with the goals of:

- Identifying if a remote attacker could penetrate BADESTORE.NET One's defenses
- Determining the impact of a security breach on:
- Confidentiality of the company's private data
- Internal infrastructure and availability of BADESTORE.NET One's information systems

Efforts were placed on the identification and exploitation of security weaknesses that could allow a remote attacker to gain unauthorized access to organizational data.

---

### Summary of Results

Initial reconnaissance of the BADSTORE.NET resulted in the discovery of a misconfigured directories listed to external network as shown in screenshots. The results provided us with a listing of specific hosts to target for this assessment. An examination of mentioned host revealed some specific folders in which accounts were stored at supplier module. After discovery of folders and directories, now we can able to bypass the authentication by accessing the admin module.

An examination of the administrative interface revealed that it was vulnerable to parameter modification. This initial compromise was escalated to administrative access due to a lack of appropriate system updates on the webserver.

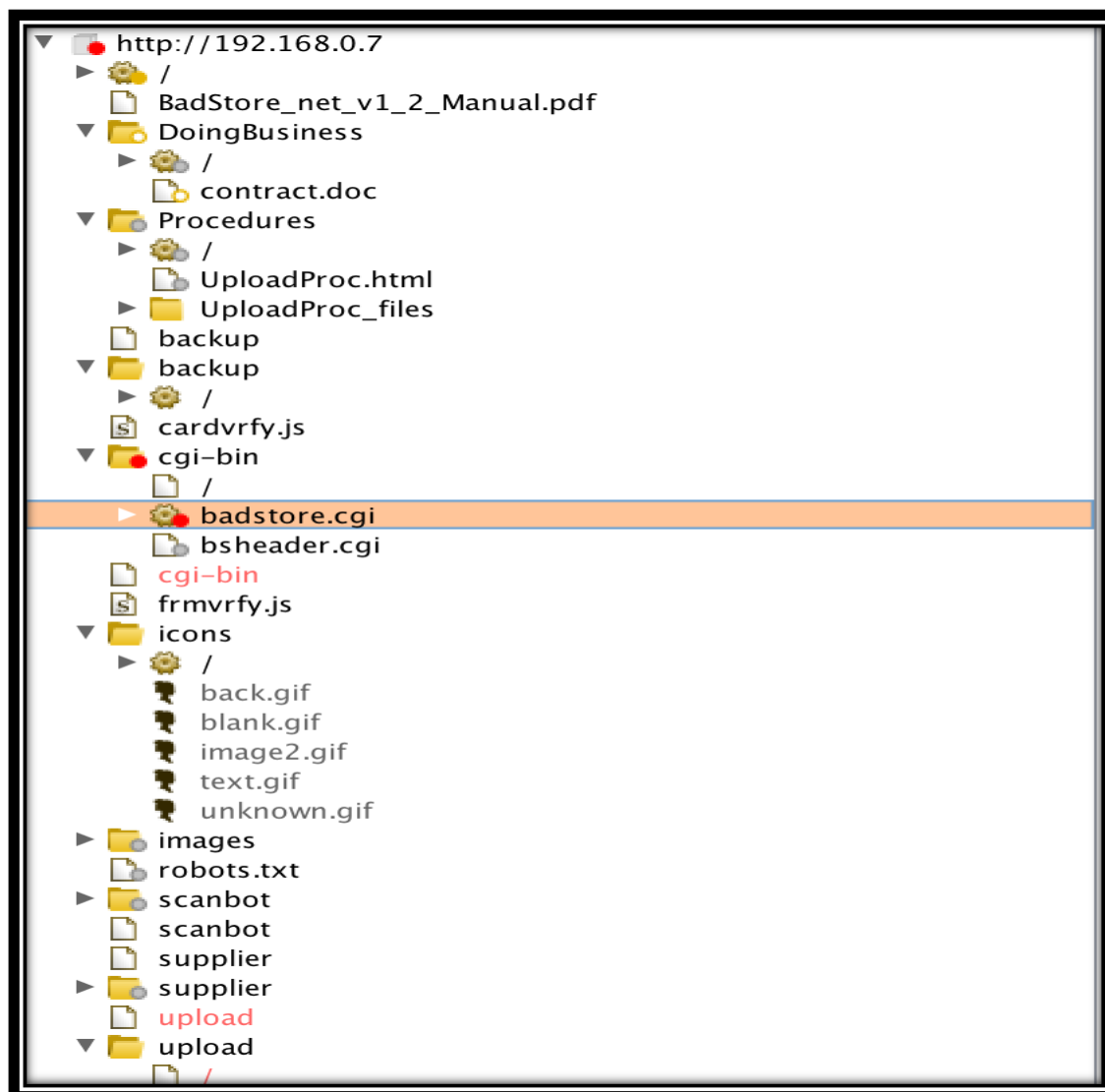
Now comes the part of using injecting the malicious characters into entry and exit points so we inserted the malicious character and we are discovered with SQL injection and cross site scripting attacks. Here every insertion point is vulnerable to cross site request forgery when change the email and password. Even the reset password functionality is showing decoded password in UI when reset the password for any listed user.

### Attack Narrative

#### Enumerate System Find:

For the purposes of this assessment, Discovery has been done for website enumeration as this is first part for enumeration and foot printing in which we have done spidering for provided URL. Reason for discovering the directories that most of the flaws is to find in discovering the directories.

In below screenshot, enumeration has been done for each and every directory, many of the directory and text files has been discovered like robots.txt file and some hidden URL like scanbot, supplier and upload directories has been found out.

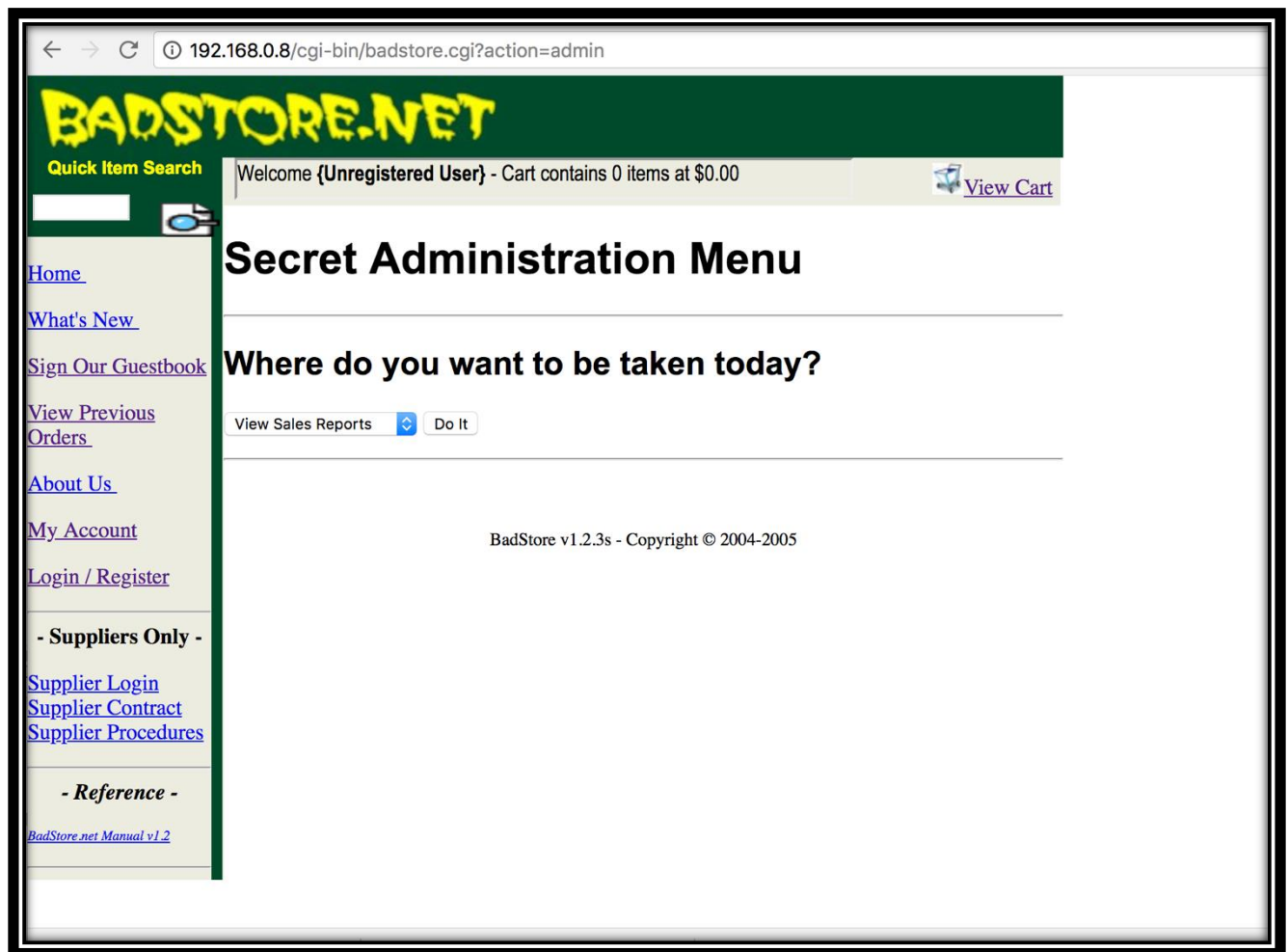


### Admin Webserver Interface Compromise

While most applications require authentication to gain access to private information or to execute tasks, not every authentication method is able to provide adequate security. Negligence, ignorance, or simple understatement of security threats often result in authentication schemes that can be bypassed by simply skipping the log in page and directly calling an internal page that is supposed to be accessed only after authentication has been performed.

In addition, it is often possible to bypass authentication measures by tampering with requests and tricking the application into thinking that the user is already authenticated. This can be accomplished either by modifying the given URL parameter, by manipulating the form, or by counterfeiting sessions.

If a web application implements access control only on the log in page, the authentication schema could be bypassed. For example, if a user directly requests a different page via forced browsing, that page may not check the credentials of the user before granting access. Attempt to directly access a protected page through the address bar in your browser to test using this method. This is a direct page request in which we are putting only "admin" parameter and we are getting direct access for admin panel. Please see the screenshot for more reference. **Here we are putting on action parameter as "admin" and it has successfully bypassed the authentication method through parameter modification.**



---

### Directory Listing for suppliers and other backup files/Apache Version Disclosure:

Web servers can be configured to automatically list the contents of directories that do not have an index page present. This can aid an attacker by enabling them to quickly identify the resources at a given path, and proceed directly to analyzing and attacking those resources. It particularly increases the exposure of sensitive files within the directory that are not intended to be accessible to users, such as temporary files and crash dumps. Directory listings themselves do not necessarily constitute security vulnerability. Any sensitive resources within the web root should in any case be properly access-controlled, and should not be accessible by an unauthorized party who happens to know or guess the URL. Even when directory listings are disabled, an attacker may guess the location of sensitive files using automated tools.

Here is the one of the URL which was getting exposed to external environment and it was showing all accounts information in BASE64 encoding under the supplier folder. Here are the encoded information has been

a3JvZW1lci9zM0NyM3QvZ29sZC8xMC4xMDAuMTAwLjE=

When decode above parameter then it shows the information as  
**kroemer/s3Cr3t/gold/10.100.100.1**

Above information clearly shows that information which is getting stored in accounts location, it clearly shows that  
Information is stored in not properly encrypted form or encoded form.

Vulnerability has been discovered in Apache Web Server that could allow for information disclosure. This vulnerability has been discovered in Apache Web Server is open source server software that is maintained by the Apache Software Foundation. Successful exploitation of this vulnerability could allow for unauthorized viewing of sensitive information. **Apache 1.3.28** version has been used which is an old server and latest Apache server 2.4 is available into Apache Software Foundation. Please below link for Apache server 1.3.28 vulnerabilities:

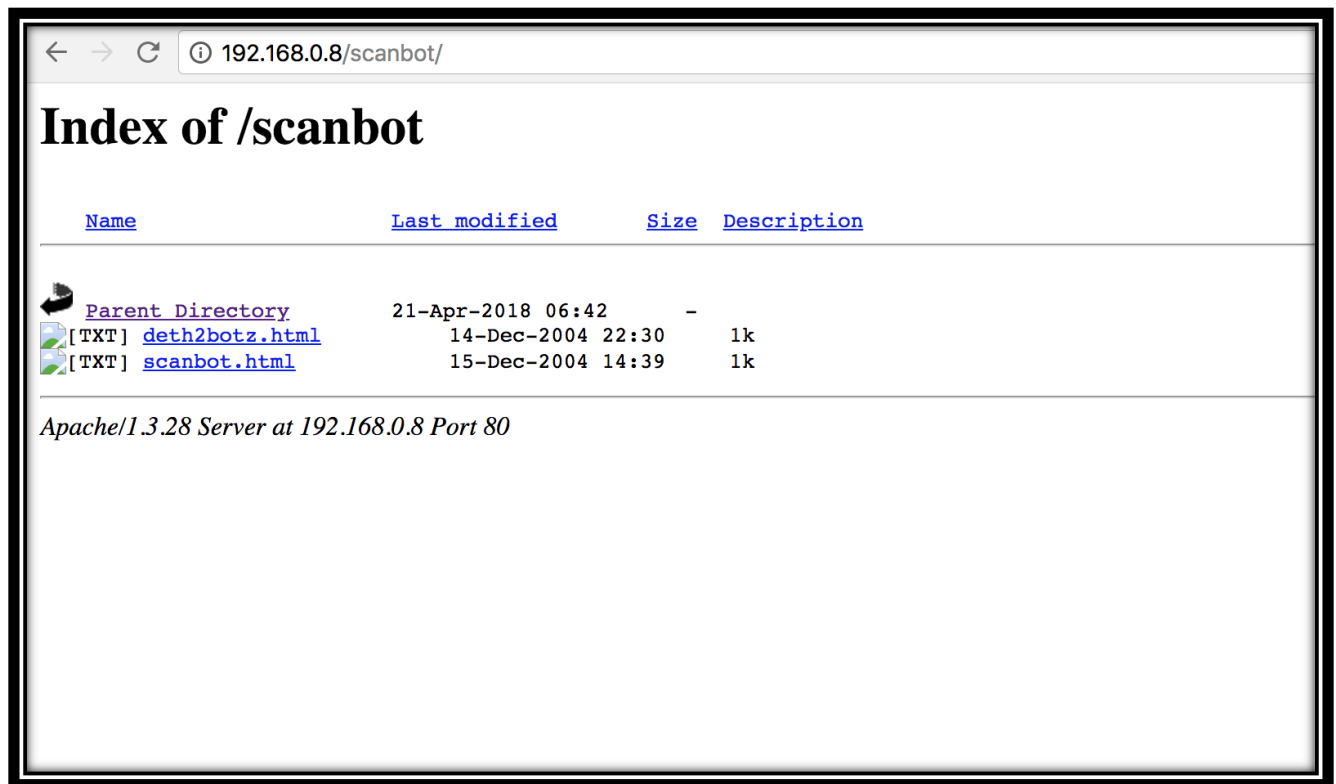
[https://www.cvedetails.com/vulnerability-list/vendor\\_id-45/product\\_id-66/version\\_id-3444/Apache-Http-Server-1.3.html](https://www.cvedetails.com/vulnerability-list/vendor_id-45/product_id-66/version_id-3444/Apache-Http-Server-1.3.html)

There is more than 33 vulnerability category which is present for Apache Server 1.3 version. All exploits are attached to CVE vulnerability.



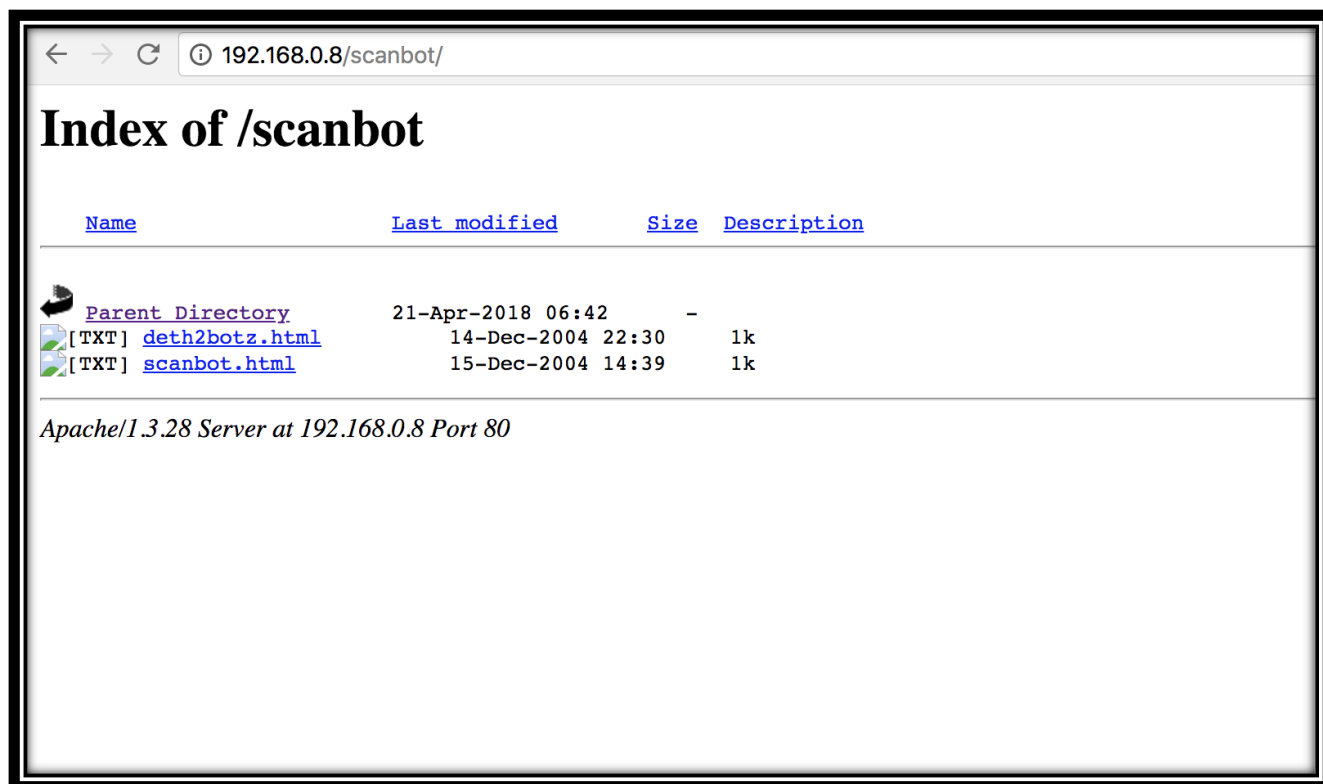
#### Screenshot for Account folder

Please see the backup and scanbot folder.



Using a custom word dictionary, it is possible to discover the administrative password for the “admin” folder.





Screenshot for Scanbot folder

### Cross Site Scripting Attack

Cross-site Scripting (XSS) refers to client-side code injection attack wherein an attacker can execute malicious scripts (also commonly referred to as a malicious payload) into a legitimate website or web application. XSS is amongst the most rampant of web application vulnerabilities and occurs when a web application makes use of invalidated or decoded user input within the output it generates.

By leveraging XSS, an attacker does not target a victim directly. Instead, an attacker would exploit vulnerability within a website or web application that the victim would visit, essentially using the vulnerable website as a vehicle to deliver a malicious script to the victim's browser. Here the all reflecting cross site scripting attack is getting reflected in response. **Below are links of vulnerable parameters:**

Email

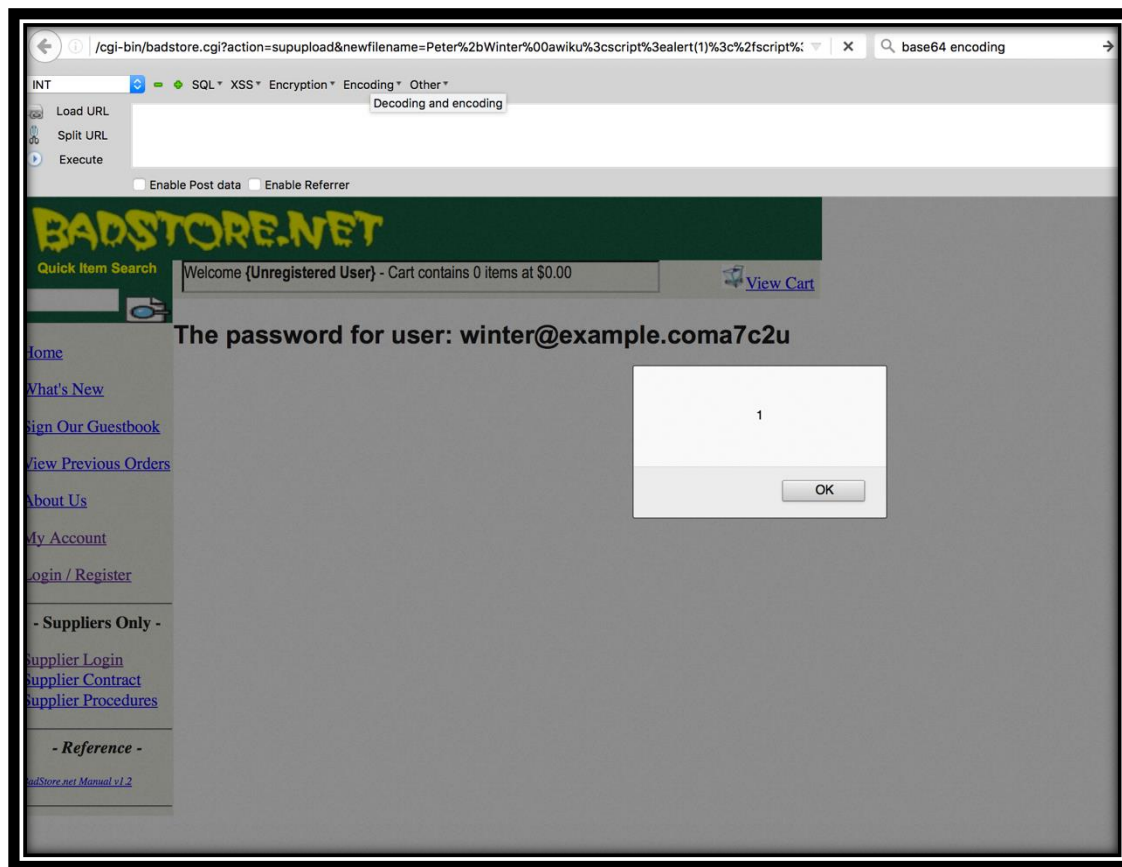
Searchquery

These are above parameters which are vulnerable to cross site scripting attacks

<http://192.168.0.8/cgi-bin/badstore.cgi?action=moduser&pwdhint=blue&DoMods=Reset+User+Password&email=winter@example.coma/c2u%3Cscript%3Ealert%281%29%3C%2Fscript%3Ex5086vhqn0e>

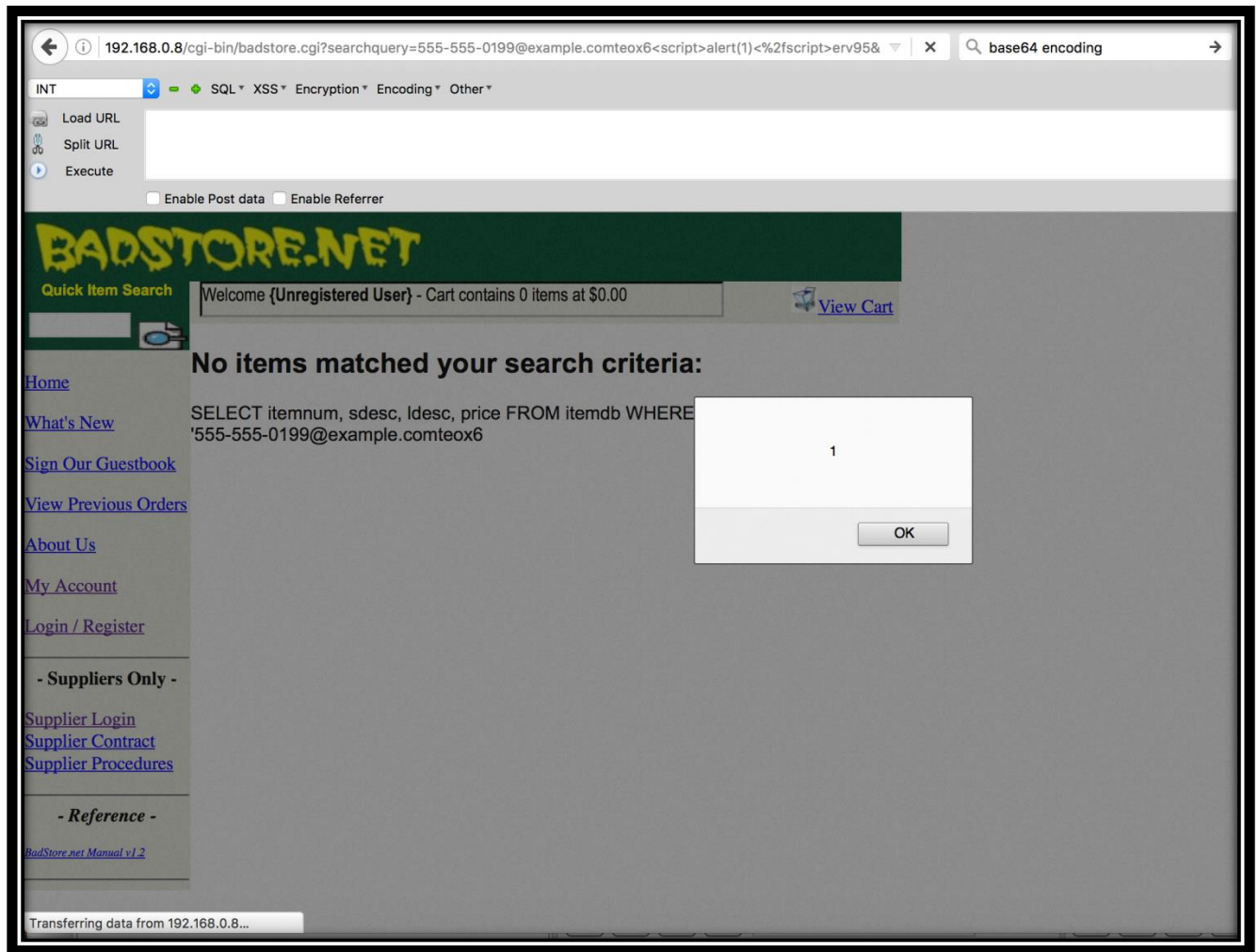
<http://192.168.0.8/cgi-bin/badstore.cgi?searchquery=555-555-0199@example.comteox6%3Cscript%3Ealert%281%29%3C%2Fscript%3EEerv95&action=search&x=1&y=1>

Please see the screenshot:



Screenshot for Cross Site Scripting

Please see another screenshot for cross site scripting attack:



Screenshot Cross Site Scripting Attack

---

## SQL Injection

SQL Injection (SQLi) refers to an injection attack wherein an attacker can execute malicious SQL statements (also commonly referred to as a malicious *payload*) that control a web application's database server (also commonly referred to as a *Relational Database Management System – RDBMS*). Since an SQL Injection vulnerability could possibly affect any website or web application that makes use of an SQL-based database, the vulnerability is one of the oldest, most prevalent and most dangerous of web application vulnerabilities.

By leveraging SQL Injection vulnerability, given the right circumstances, an attacker can use it to bypass a web application's authentication and authorization mechanisms and retrieve the contents of an entire database. SQL Injection can also be used to add, modify and delete records in a database, affecting data integrity.

To such an extent, SQL Injection can provide an attacker with unauthorized access to sensitive data including, customer data, personally identifiable information (PII), trade secrets, intellectual property and other sensitive information. **Below are the vulnerable parameters:**

**Cartitem (Intercepted Request from Burp)**

**Email Address (Field from UI)**

**Full Name (Field from UI)**

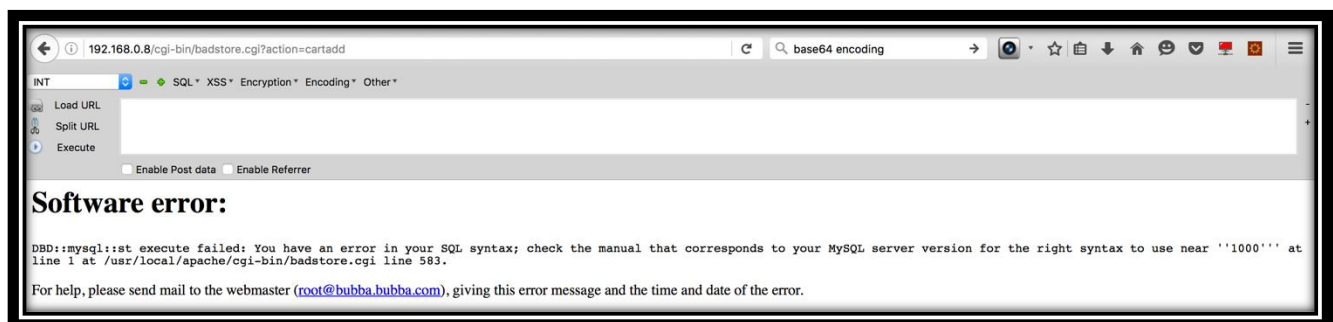
**Pwdhint (Intercepted Request from Burp)**

**Role (Intercepted Request from Burp)**

**Search Query(Passed in Parameter URL)**

Please see below the SQL injection error:

### Screenshot for SQL Injection



### Screenshot for SQL Injection at cartitem

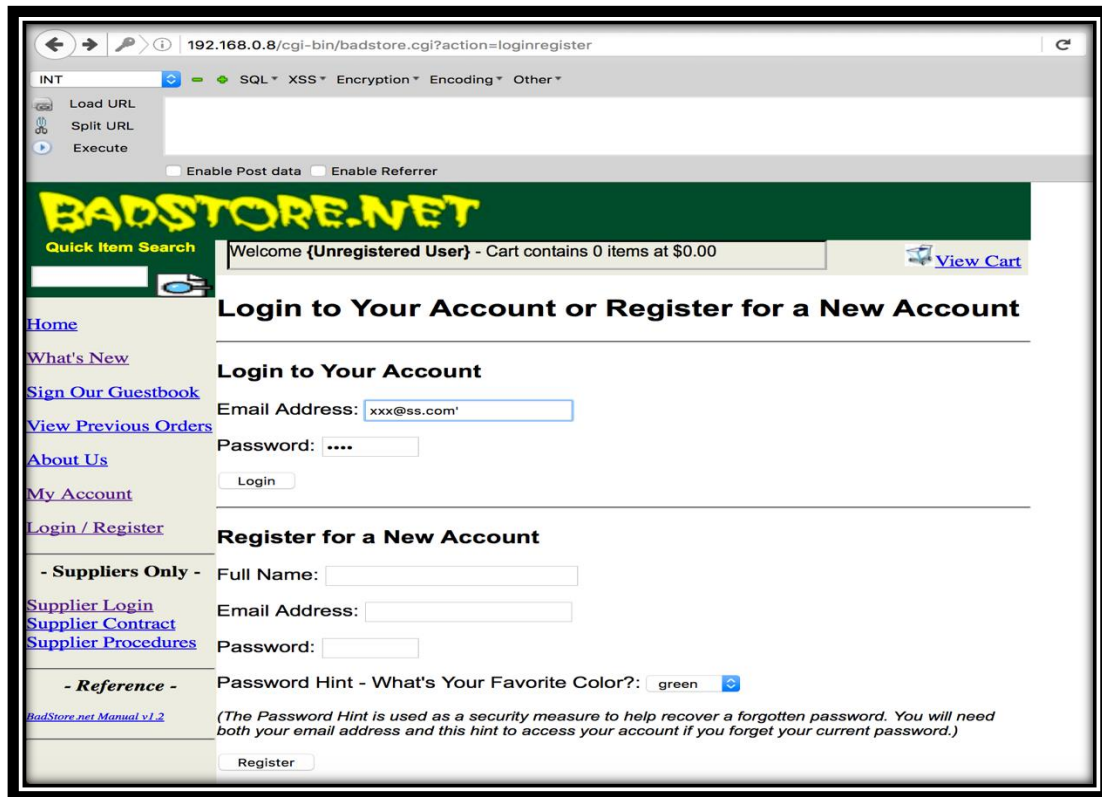
Request to http://192.168.0.8:80

Forward Drop Intercept is on Action

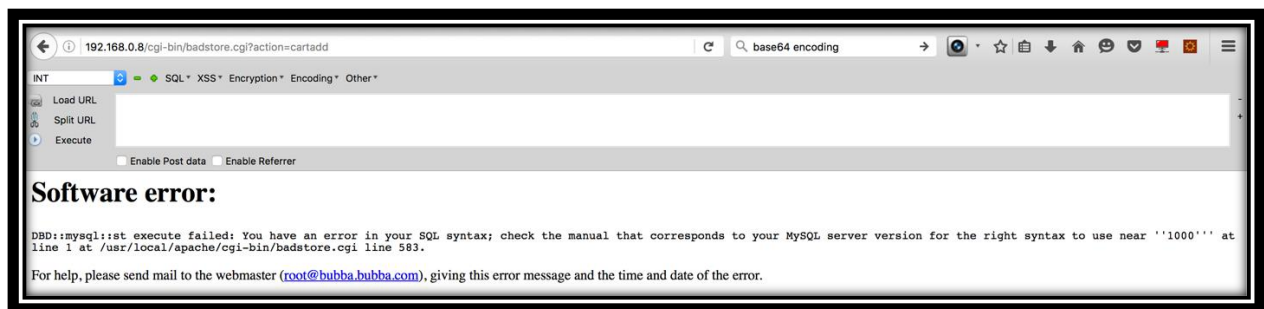
Raw Params Headers Hex

```
POST /cgi-bin/badstore.cgi?action=cartadd HTTP/1.1
Host: 192.168.0.8
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv:46.0) Gecko/20100101 Firefox/46.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.0.8/cgi-bin/badstore.cgi?action=whatsnew
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 49

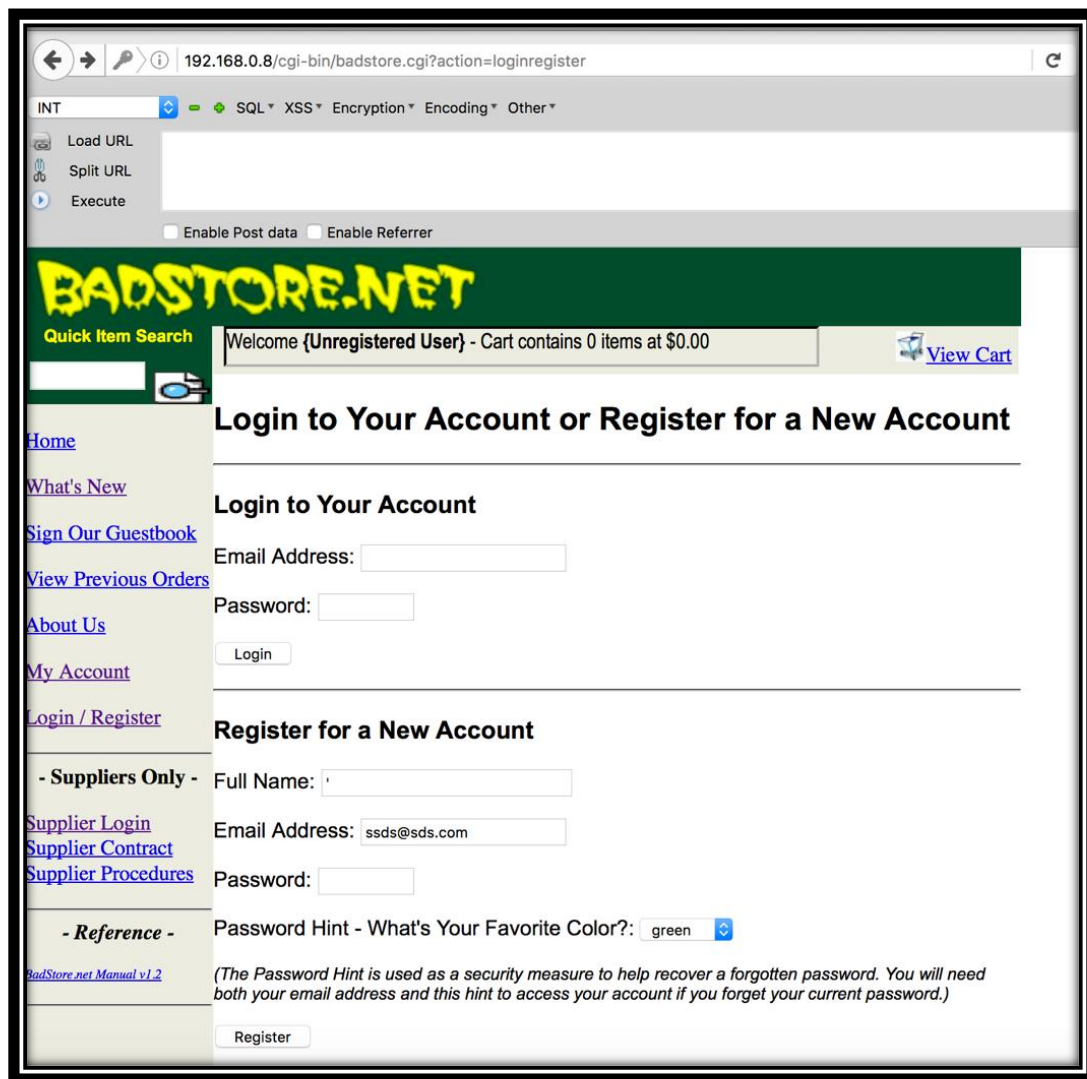
cartitem=1000'&Add+Items+to+Cart=Add+Items+to+Cart
```



## SQL EMAIL



## Screenshot for SQL Injection



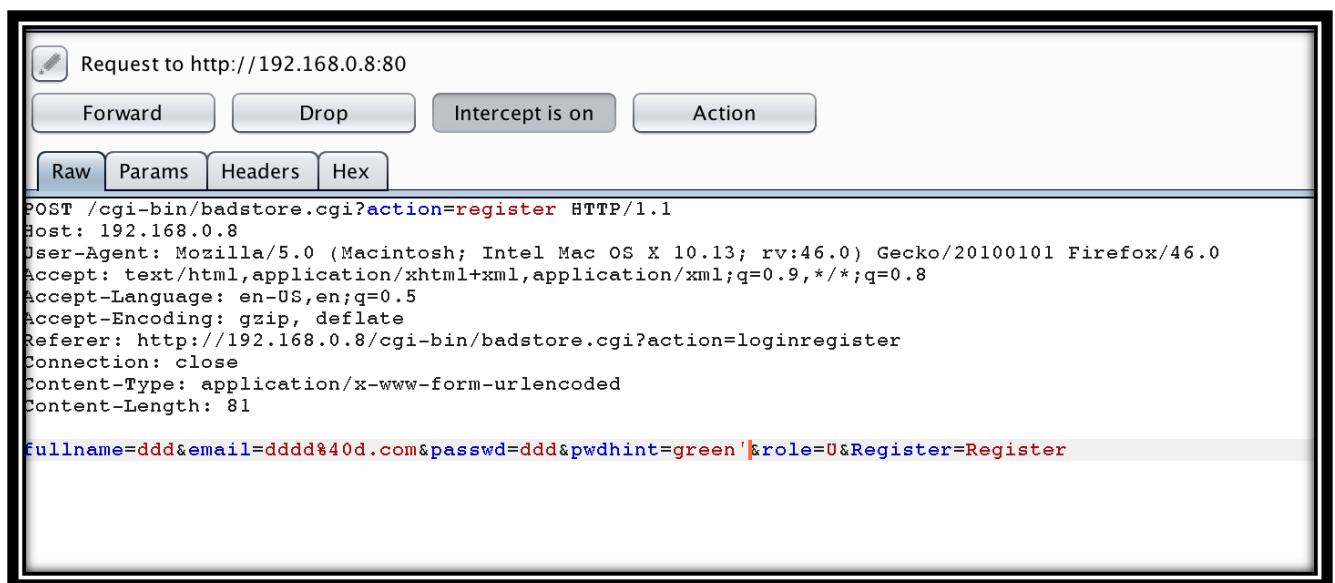
Screenshot for SQL Injection at Full Name



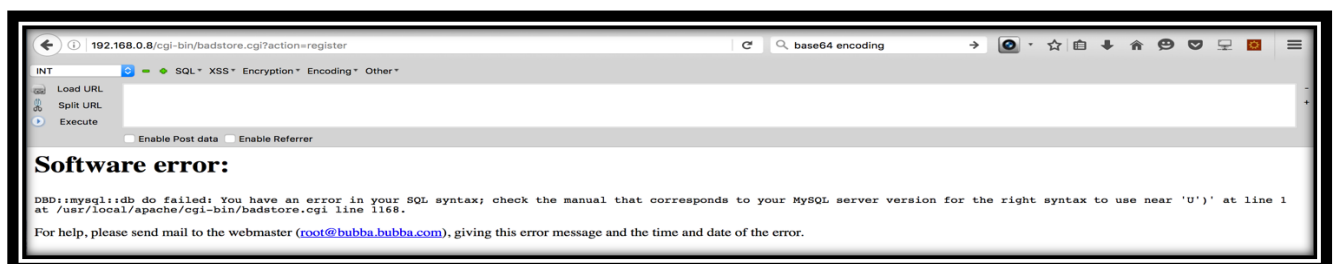
## Screenshot for SQL Injection Error

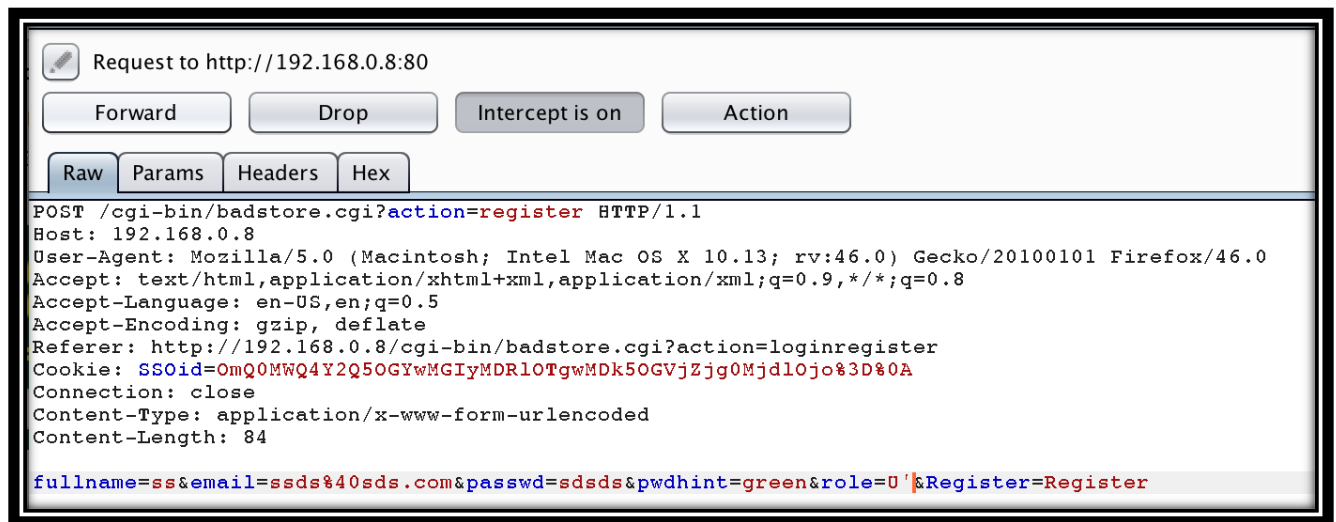


## Screenshot for SQL Injection at pwdhint

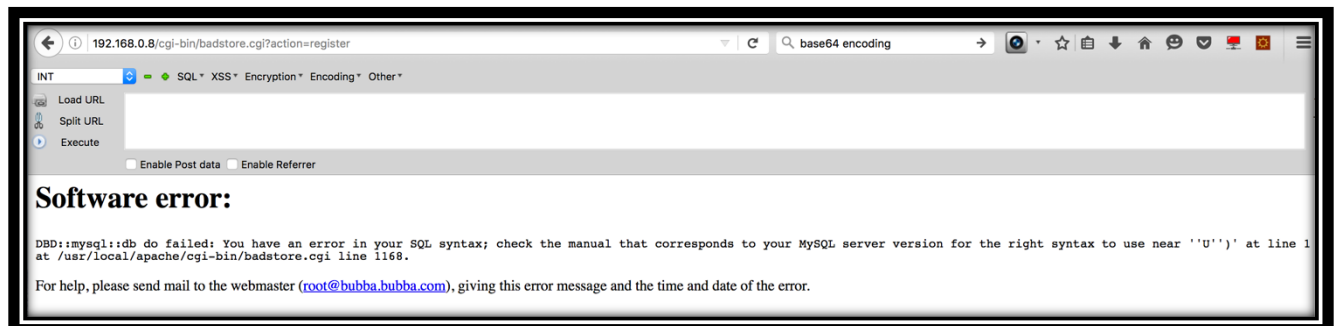


## Screenshot for SQL Injection error

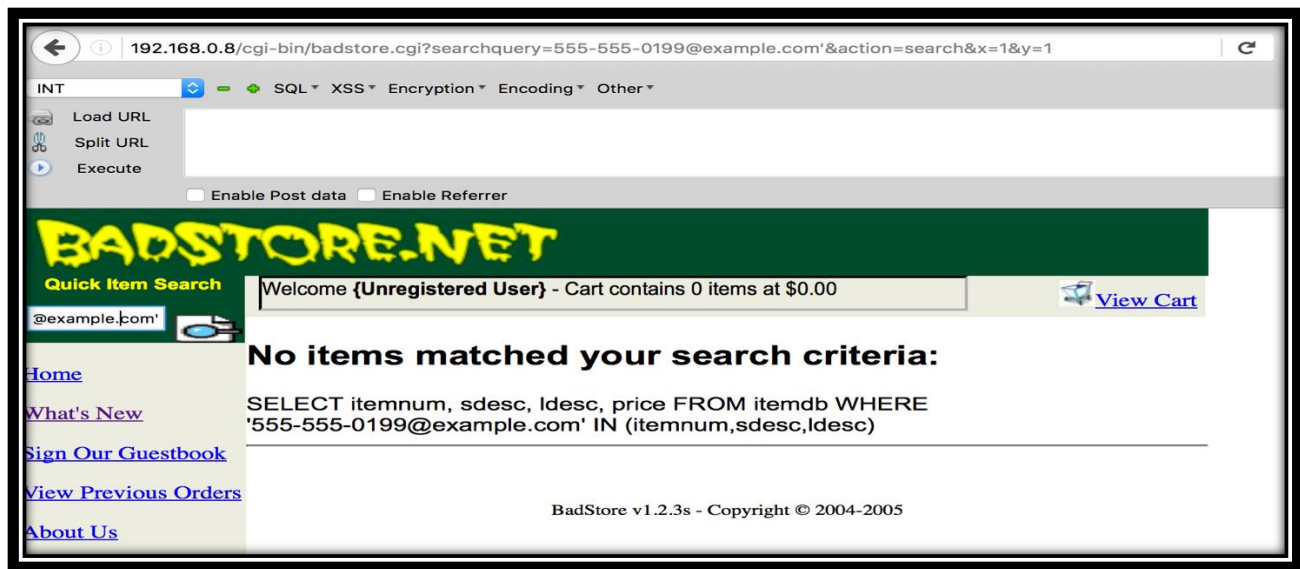




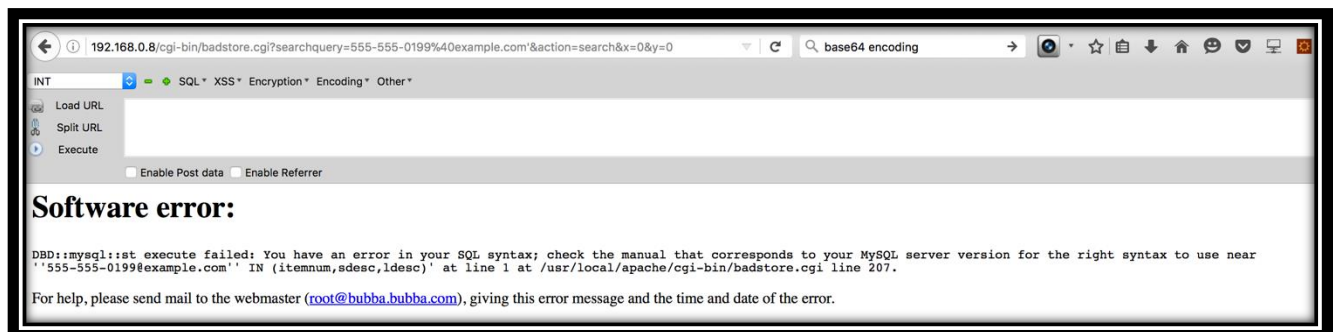
Screenshot for SQL Injection at role parameter



Screenshot for SQL Injection error



Screenshot for SQL Injection at SearchBox



Screenshot for SQL Injection Error

### Cross Site Request Forgery

Cross site request forgery (CSRF), also known as XSRF, Sea Surf or Session Riding, is an attack vector that tricks a web browser into executing an unwanted action in an application to which a user is logged in. A successful CSRF attack can be devastating for both the business and user. It can result in damaged client relationships, unauthorized fund transfers, changed passwords and data theft—including stolen session cookies. CSRFs are typically conducted using malicious social engineering, such as an email or link that tricks the victim into sending a forged request to a

server. As the unsuspecting user is authenticated by their application at the time of the attack, it's

impossible to distinguish a legitimate request from a forged one. In a Cross-Site Request Forgery attack, the attacker is exploiting how the target web application manages authentication. For CSRF to be exploited, the victim must be authenticated against (logged into) the target site. For instance, let's say examplebank.com has online banking that is vulnerable to CSRF. If I visit a page containing a CSRF attack on examplebank.com but am not currently logged in, nothing happens. If I am logged in, however, the requests in the attack will be executed as if they were actions that I had intended to take. In below screenshots, I have made one small script in which I have written the code for changing the password and here I am passing in which user is currently authenticated. This is mainly attempt to change the password.

Here email and password parameter both are vulnerable to CSRF attack.

If you can see in next two screenshots, password which was there, that was something different and now the password has been changed due to small script of CSRF.HTML

```
<html>
```

```
<!-- CSRF PoC - generated by Burp Suite Professional -->
```

```
<body>
```

```
<script>history.pushState("", "", '/')</script>
```

```
<form action="http://192.168.43.176/cgi-bin/badstore.cgi?action=moduser"
method="POST">
```

```
<input type="hidden" name="fullname" value="new&#32;email" />
```

```
<input type="hidden" name="newemail" value="newemails#64;gmail#46;com" />
```

```
<input type="hidden" name="newpasswd" value="432145" />
```

```
<input type="hidden" name="vnewpasswd" value="432145" />
```

```
<input type="hidden" name="role" value="U" />
```

```
<input type="hidden" name="email" value="newemail#64;gmail#46;com" />
```

```
<input type="hidden" name="DoMods" value="Change#32;Account" />
```

```
<input type="submit" value="Submit request" />
```

```
</form>
```

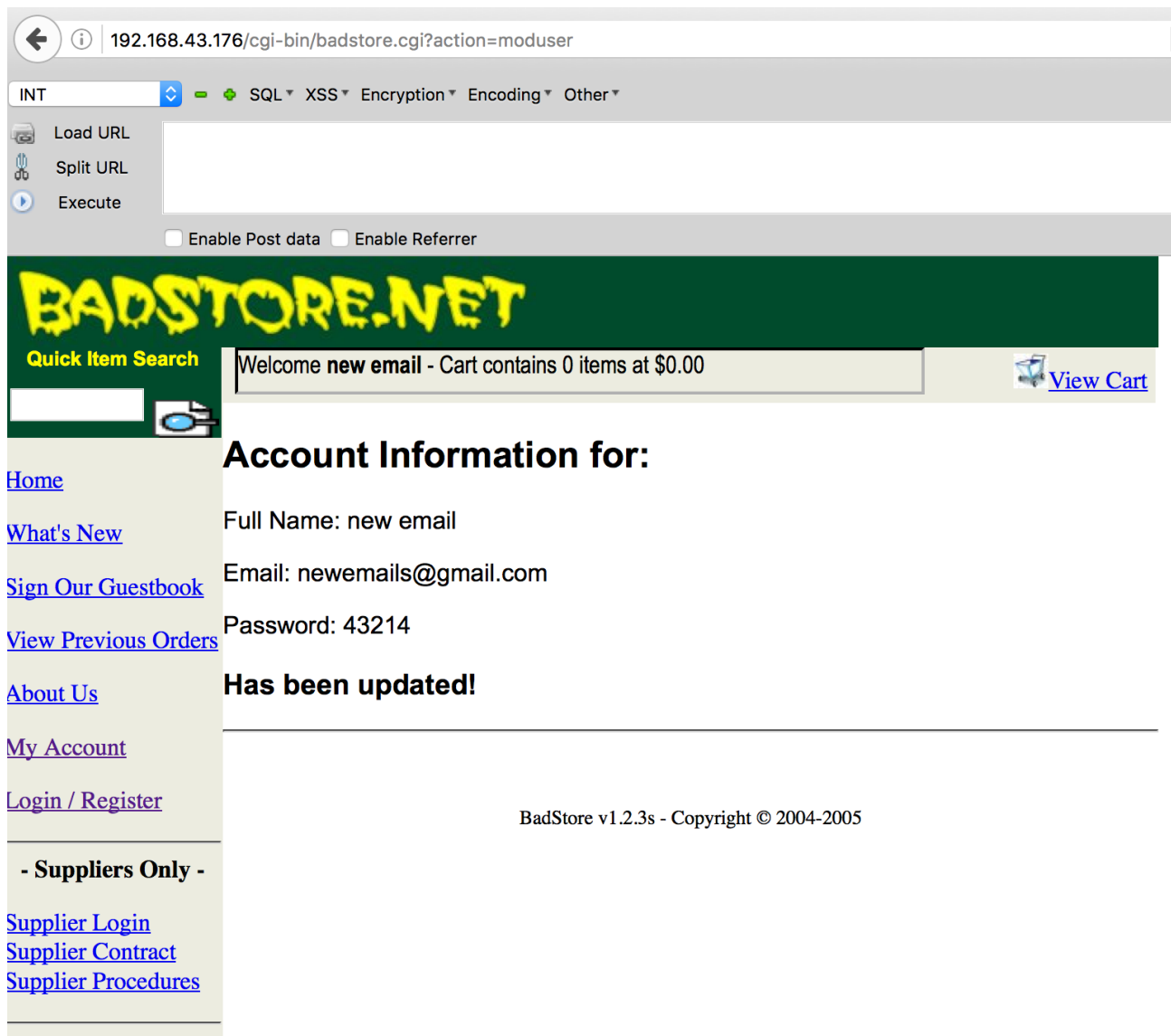
```
</body>
```

```
</html>
```

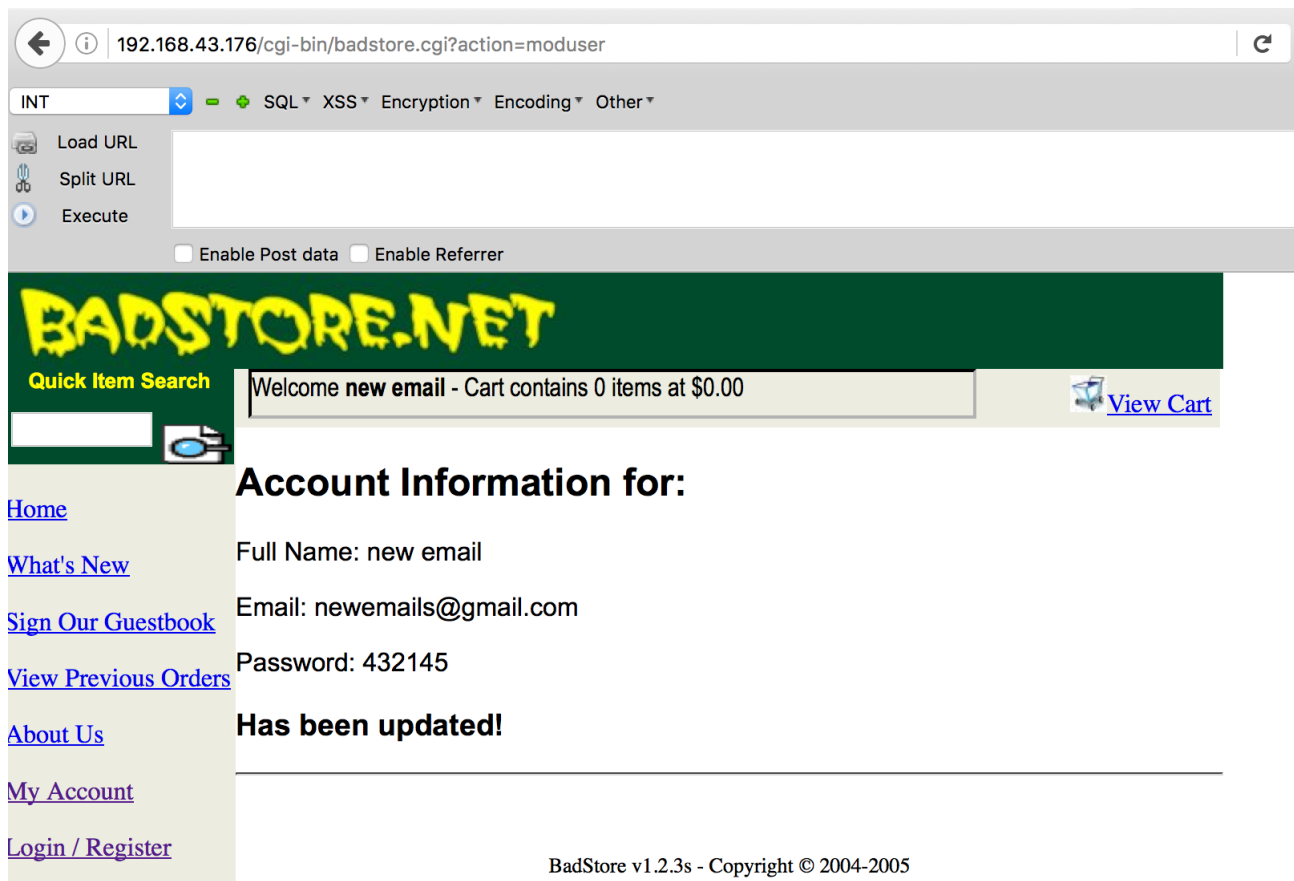
Please see below Screenshots:

```
1 <html>
2 <!-- CSRF PoC - generated by Burp Suite Professional -->
3 <body>
4 <script>history.pushState('', '', '/')</script>
5 <form action="http://192.168.43.176/cgi-bin/badstore.cgi?action=moduser" method="POST">
6   <input type="hidden" name="fullname" value="new&#32;email" />
7   <input type="hidden" name="newemail" value="newemails&#64;gmail&#46;com" />
8   <input type="hidden" name="newpasswd" value="432145" />
9   <input type="hidden" name="vnewpasswd" value="432145" />
10  <input type="hidden" name="role" value="U" />
11  <input type="hidden" name="email" value="newemail&#64;gmail&#46;com" />
12  <input type="hidden" name="DoMods" value="Change&#32;Account" />
13  <input type="submit" value="Submit request" />
14 </form>
15 </body>
16 </html>
17
```

### Screenshot for CSRF Code



Screenshot Before CSRF Injection



### Screenshot After CSRF Injection

#### Reset Password Functionality Vulnerability:

The password change and reset function of an application is a self-service password change or reset mechanism for users. This self-service mechanism allows users to quickly change or reset their password without an administrator intervening. When passwords are changed they are typically changed within the application. When passwords are reset they are either rendered within the application or emailed to the user. This may indicate that the passwords are stored in plain text or in a decryptable format. Most web applications allow users to reset their password if they have forgotten it, usually by sending them a password reset email and/or by asking them to answer one or more "security questions." In this test, we check that this function is properly implemented and that it does not introduce any flaw in the authentication scheme. We also check whether the application allows the user to store the password in the browser ("remember password" function). Password recovery functionalities can result in vulnerabilities in the same application they are intended to protect. Vulnerabilities such as username enumeration (showing different error messages when the user exists or not in the database), sensitive information disclosure (sending the password in clear-text by e-mail to user) and recover password message hijack (involving an attacker receiving a copy of the recover password message) are some common



vulnerabilities that may be found in a password recovery functionality. Various developers don't take into consideration the real implications of unsecure password recovery, and this blog post will show how things get complicated when developers don't apply basic security practices to this kind of functionality.

Here are below screenshots which are showing in screenshots as when the password has been reset and showing the whole password in UI screen and this is wrong practice for reset password implementation functionality. One screenshot is showing the default password of listed users for the website when reset it. Another screenshot is showing that when reset the password is "Welcome".


Implementation has been done wrong for reset password functionality when analyzing the results. This is we are generating and storing the password in one form as Welcome. This is not a proper implementation of reset password functionality. For more reference, please see the screenshot.

Please see the screenshots:

# 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](#)

The password for user:

...has been reset to: **Welcome**


BadStore v1.2.3s - Copyright © 2004-2005

Screenshot Password Visibility after Reset


← → ↻ 172.11.17.43/cgi-bin/badstore.cgi?action=doguestbook

# BADSTORE.NET

Quick Item Search



Welcome {Unregistered User} - Cart contains 0 items at \$0.00

 [View Cart](#)

## Guestbook

---

Wednesday, February 18, 2004 at 07:42:34: **Joe Shopper** [joe@microsoft.com](mailto:joe@microsoft.com)

*This is a great site! I'm going to shop here every day.*

---

Wednesday, February 18, 2004 at 11:41:07: **John Q. Public** [jqp@whitehouse.gov](mailto:jqp@whitehouse.gov)

*Let me know when the summer items are in.*

---

Friday, February 20, 2004 at 14:05:22: **Big Spender** [billg@microsoft.com](mailto:billg@microsoft.com)

*Where's the big ticket items?*

---

Sunday, February 22, 2004 at 06:16:05: **Evil Hacker** [s8n@haxor.com](mailto:s8n@haxor.com)

*You have no security! I can own your site in less than 2 minutes. Pay me \$100,000 US currency by the end of day Friday, or I will hack you offline and sell the credit card numbers I found on your site. Send the money direct to my PayPal account.*

---

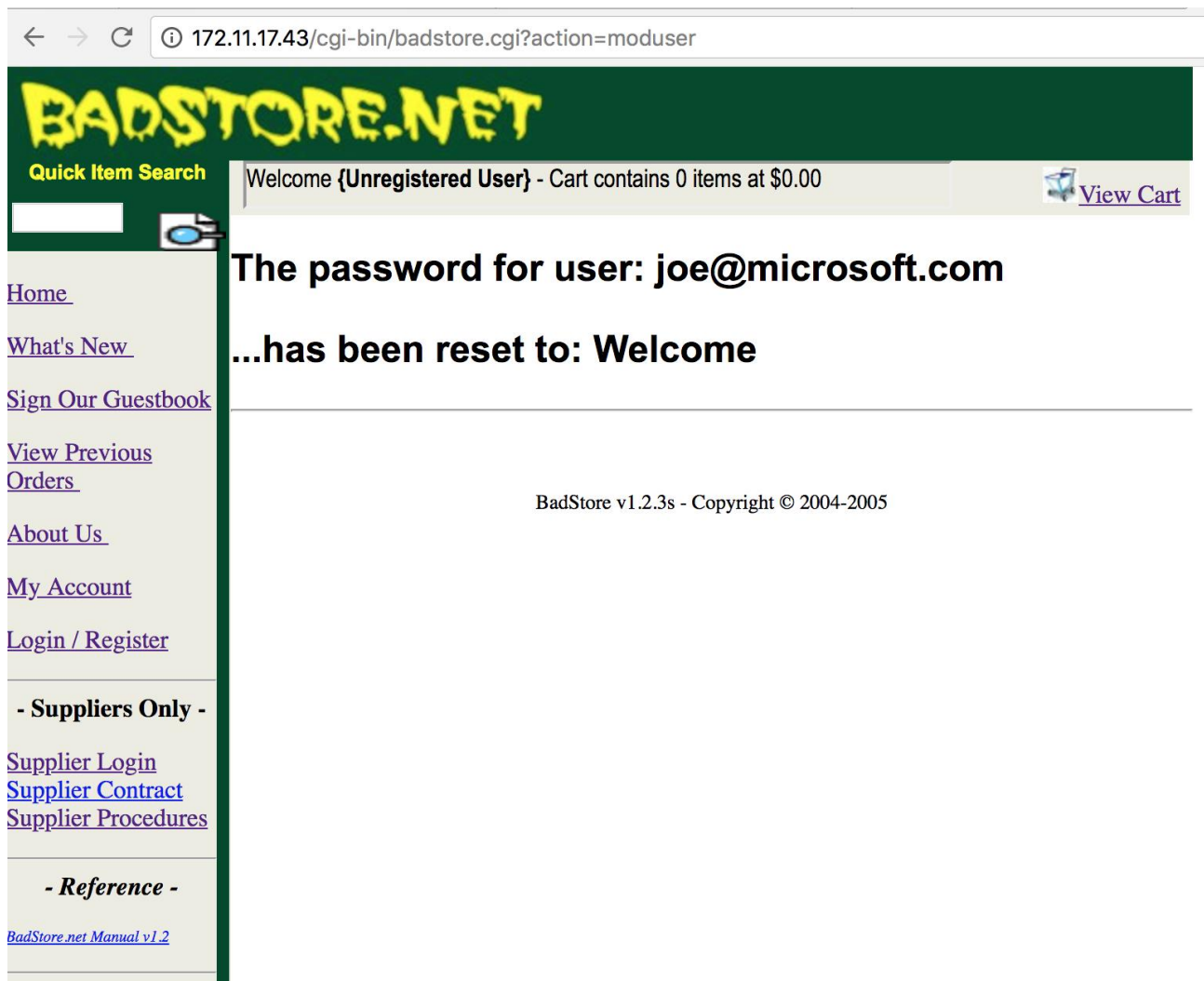
Saturday, April 21, 2018 at 14:17:06:

---

Saturday, April 21, 2018 at 14:17:32:

- [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](#)

Screenshot for UserName Enumeration



Screenshot Password Visibility after Reset

---

## Conclusion

BADSTORE.NET suffered a series of control failures, which led to a complete compromise of critical company assets. These failures would have had a dramatic effect on BADSTORE.NET, One operations if a malicious party had exploited them.

The specific goals of the penetration test were stated as:

- Identifying if a remote attacker could penetrate BADSTORE.NET One's defenses
- Determining the impact of a security breach on.
- Confidentiality of the company's information.
- Internal infrastructure and availability of BADSTORE.NET One's information systems

These goals of the penetration test were met. A targeted attack against BADSTORE.NET One can result in a complete compromise of organizational assets. Multiple issues that would typically be considered minor were leveraged in concert, resulting in a total compromise of the BADSTORE.NET One's information systems. It is important to note that this collapse of the entire BADSTORE.NET One security infrastructure can be greatly attributed to insufficient access controls at the network level.

## Recommendations

Due to the impact to the overall organization as uncovered by this penetration test, appropriate resources should be allocated to ensure that remediation efforts are accomplished in a timely manner. While a comprehensive list of items that should be implemented is beyond the scope of this engagement, some high-level items are important to mention.

Penetration Testing recommends the following:

**Stored procedures.** Stored procedures are the SQL statements defined and stored in the database itself and then called from the application. Developers are usually only required to build SQL statements with parameters that are automatically parameterized. However, it's possible for a developer to generate dynamic SQL queries inside stored procedures. Implement stored procedures safely by avoiding dynamic SQL generation inside.

**Input validation.** A common source of SQL injection is maliciously crafted external input. As such, it's always a good practice to only accept approved input—an approach known as input validation. To protect against it, there are two variants of input validation: blacklist validation and whitelist validation.

Blacklist validation tests the external input against a set of known malicious inputs. An application compiles a list of all malicious inputs, and then verifies the external input against the compiled list. Therefore, it's easy for an attacker to bypass blacklist validation since they can come up with multiple variants of malicious input that may not be part of the compiled blacklist.

Whitelisting is a much better approach to mitigate the risk of SQL injection. Whitelist validation tests an external input against a set of known, approved input. With whitelist input validation, the application knows exactly what's desired and rejects other input values that fail a test against the known, approved input.

**Principle of least privilege.** This is a standard security control that helps minimize the potential damage of a successful SQL injection attack. Application accounts shouldn't assign DBA or admin type access onto the database server. Additionally, depending on access requirements, they should be restricted to least privileged access. For example, accounts that only require read access are only granted read access to the table they need to access. This ensures that if an application is compromised, an attacker won't have the rights to the database through the compromised application.

**Sufficient authentication/authorization requires:** These are below items that needs to be addressed when check for Authentication and Authorization:

1. Ensuring that the strong passwords are required.
2. Ensuring granular access control is in place when necessary.
3. Ensuring credentials are properly protected.
4. Implement two factor authentications where possible.
5. Ensuring that password recovery mechanisms are secure.
6. Ensuring re-authentication is required for sensitive features.
7. Ensuring options are available for configuring password controls.
8. Ensuring credential can be revoked.
9. The app authentication is required.
10. The device authentication is required.
11. The server authentication is required.
12. Manage authenticated user id (credential info.) and the user's device id, the user's app id mapping table in the authentication server.
13. Ensuring that the authentication token/session key issuing to client is always different.
14. Ensuring that the user id, app id, device id is universally unique.

- **Filtering for XSS**

All XSS attacks infect your web site via some form of User Input. XSS attack code could come from a simple <FORM> submitted by your users, or could take a more complex route such as a JSON script, XML web service or even an exploited cookie. In all cases the web developer should be aware that the data is coming from an external source and therefore must not be trusted.

The simplest and arguably the easiest form of XSS protection would be to pass all external data through a filter which will remove dangerous keywords, such as the infamous <SCRIPT> tag, JavaScript commands, CSS styles and other dangerous HTML markup (such as those that contain event handlers.)

Many web developers choose to implement their own filtering mechanisms; they usually write server-side code (in PHP, ASP, or some other web-enabled development language) to search for keywords and replace them with empty strings. I have seen lots of code that makes use of Regular Expressions to do this filtering and replacing. This technique is in itself not a bad one, however unfortunately the hackers usually have more experience than the web developers, and often manage to circumvent simple filters by using techniques such as hex encoding, Unicode character variations, line breaks and null characters in strings. These techniques must all be catered for and that is why it is recommended to use some sort of library that has been tried and tested by the community at large.

Many libraries exist to choose from, and your choice will primarily depend on the backend technology that your web server uses. What is important is that you choose a library that is regularly maintained by a reliable source. XSS techniques keep changing and new ones emerge all the time so your filters will need to be updated periodically to keep abreast with the changing attacks.

If you are using Java, then a good place to go is XSS Protect, a project hosted on Google code. It claims to filter all “known” XSS attacks from HTML code. PHP boasts a more comprehensive library called HTML Purifier which licensed as Open Source and can be customised depending on your needs. HTML Purifier also boasts strict standards compliance and better features than other filters.

Another interesting library you can use is HTML Markdown which converts text from your users into standard and clean XHTML. This gives the advantage that minimal HTML Markup can exist in your user’s input (such as bold, underline and colours). HTML Markdown is a Perl library and does not explicitly advertise XSS prevention features so it probably should not be your only line of defence.

The side-effect with these filtering techniques is that legitimate text is often removed because it hits one or more of the forbidden keywords. For example, I would not be able to publish this article if the blogging software I used was filtering out all my HTML tags. I would not be able to write things like **<SCRIPT>** and **alert (‘you have been hacked’)** as these would be filtered out and you would not see them. If you want to preserve the original data (and its formatting) as best as possible you would need to relax your filters and employ HTML, Script and CSS Escaping techniques, all of which I explain in the next section.

- **Escaping from XSS**

This is the primary means to disable an XSS attack. When performing Escaping you are effectively telling the browser that the data you are sending should be treated as data and should not be interpreted in any other way. If an attacker manages to put a script on your page, the victim will not be affected because the browser will not execute the script if it is properly escaped.

Escaping has been used to construct this article. I have managed to bring many scripts into your browser, but none of these scripts has executed! The technique used to do that is called, escaping, or as the W3C calls it “Character Escaping”.

In HTML you can escape dangerous characters by using the &# sequence followed by the its character code.

An escaped < character looks like this: &#60. The > character is escaped like this: &#62. Below is a list of common escape codes for HTML:

"	---	&#34
#	---	&#35
&	---	&#38
'	---	&#39
(	---	&#40
)	---	&#41
/	---	&#47
;	---	&#59
<	---	&#60
>	---	&#62



Escaping HTML is fairly easy, however in order to properly protect yourself from all XSS attacks you require to escape JavaScript, Cascading Style Sheets, and sometimes XML data. There are also many pitfalls if you try to do all the escaping by yourself. This is where an Escaping Library comes useful.

The two most popular escaping libraries available are the **ESAPI** provided by OWASP and **AntiXSS** provided for Microsoft. ESAPI can plug into various technologies such as Java, .NET, PHP, Classic ASP, Cold Fusion, Python, and Haskell. AntiXSS exclusively protects Microsoft technologies and is therefore better suited in an all-Microsoft environment. Both libraries are constantly updated to keep up with the latest hacker techniques and are maintained by industry experts who understand changing tactics and emerging technologies such as HTML5.

### **When to Escape**

You cannot just simply escape everything, or else your own scripts and HTML markup will not work, rendering your page useless.

There are several places on your web page which you need to ensure are properly escaped. You can use your own escaping functions (not recommended) and you can use the existing ESAPI and AntiXSS libraries.

#### **Use HTML Escaping when...**

Untrusted data is inserted in between HTML opening and closing tags. These are standards tags such as <BODY>, <DIV>, <TABLE> etc...

For example:

```
<DIV> IF THIS DATA IS UNTRUSTED IT MUST BE HTML ESCAPED </DIV>
```

#### **Use JavaScript Escaping when...**

Untrusted data is inserted inside one of your scripts, or in a place where JavaScript can be present. This includes certain attributes such as STYLE and all event handlers such as ONMOUSEOVER and ONLOAD

For example:

```
<SCRIPT>alert("IF THIS DATA IS UNTRUSTED IT MUST BE JAVASCRIPT ESCAPED")</SCRIPT>
```

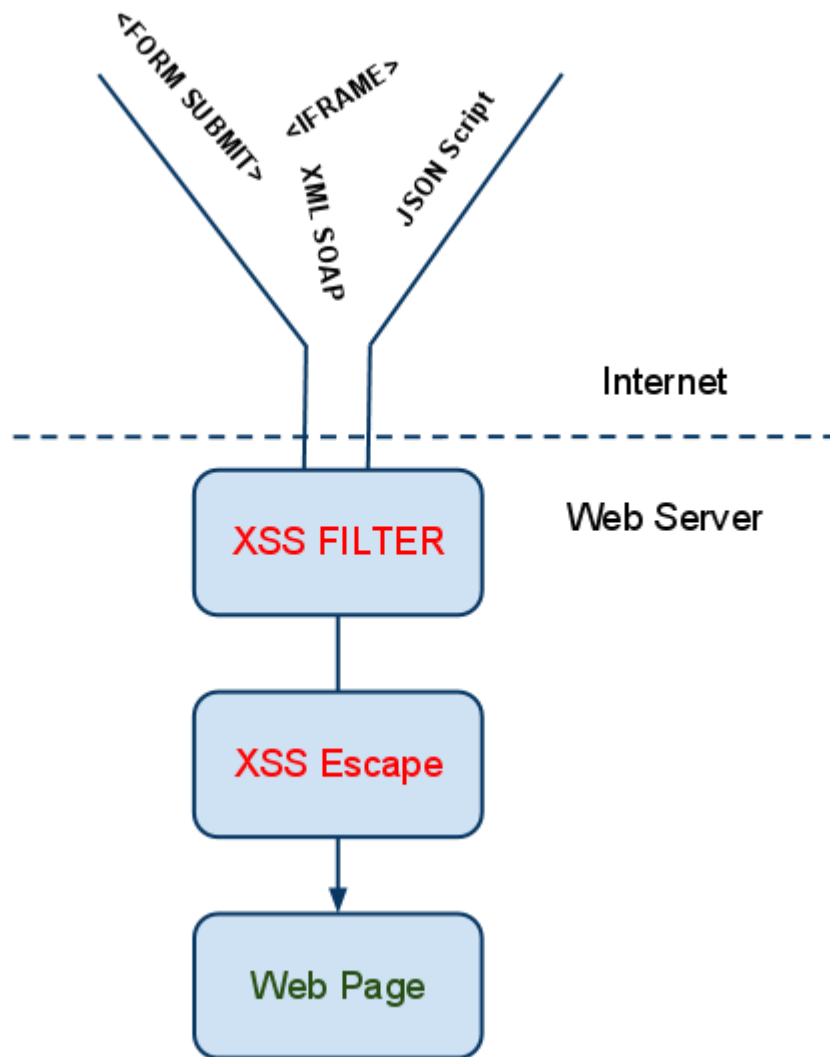
```
<BODY ONLOAD="IF THIS DATA IS UNTRUSTED IT MUST BE JAVASCRIPT ESCAPED">
```

#### **Use CSS Escaping when...**

Untrusted data is inserted inside your CSS styles. As you saw in the Attack Vectors examples, many CSS styles can be used to smuggle a script into your page.

For example:

```
<DIV STYLE="background-image: IF THIS DATA IS UNTRUSTED IT MUST BE CSS ESCAPED">
```



Above is a diagram visually representing the internet boundary and where filtering and escaping must happen to ensure XSS protection.

#### Preventing CSRF vulnerabilities

Although there have been a variety of proposed CSRF prevention mechanisms, not all of them are effective in all scenarios. The following implementations prove to be effective for a variety of web applications while still providing protection against CSRF attacks.

#### Anti-CSRF Tokens

The most popular implementation to prevent Cross-site Request Forgery (CSRF), is to make use of a challenge token that is associated with a particular user and can be found as a hidden value in

every state changing form which is present on the web application. This token, called a *CSRF Token* or a *Synchronizer Token*, works as follows:

- The web server generates a token
- The token is statically set as a hidden input on the protected form
- The form is submitted by the user
- The token is included in the POST data
- The web application compares the token generated by the web application with the token sent in through the request
- If these tokens match, the request is valid, as it has been sent through the actual form in the web application
- If there is no match, the request will be considered as illegal and will be rejected.

This protects the form against Cross-site Request Forgery (CSRF) attacks, because an attacker crafting a request will also need to guess the anti-CSRF token for them to successfully trick a victim into sending a valid request. What's more, is that this token should be invalidated after some time and after the user logs out.

For the anti-CSRF mechanism to be implemented properly, it will also need to be cryptographically secure, so that the token itself cannot be easily guessed, which is a possibility if the token is being generated based on a predictable pattern.

It is also recommended that you make use of the readily available option in popular frameworks that will defend against CSRF attacks for you, meaning that you should refrain from rolling your own anti-CSRF mechanism, if possible. This allows less room for error, while making the implementation quicker and easier.

### Same Site Cookies

CSRF attacks are only possible since Cookies are always sent with any requests that are sent to a particular **origin**, which is related to that Cookie. Due to the nature of a CSRF attack, a flag can be set against a Cookie, turning it into a same-site Cookie. A same-site Cookie is a Cookie which can only be sent, if the request is being made from the same origin that is related to the Cookie being sent. The Cookie and the page from where the request is being made, are considered to have the same origin if the protocol, port (if applicable) and host is the same for both.

A current limitation of same-site Cookies is that not all modern browsers support them, while older browsers do not cater for web applications that make use of same-site Cookies. At the moment, same-site Cookies are better suited as an additional defense-in-depth layer due to this limitation, while still making use of other CSRF protection mechanisms.

There are two common approaches:

1. Generate a new password on the server and email it
2. Email a unique URL which will facilitate a reset process

Despite plenty of guidance to the contrary, the first point is really not where we want to be. The problem with doing this is that it means a persistent password – one you can go back with and use any time – has now been sent over an insecure channel and resides in your inbox. Other options, we can take as

1. Ask user for email, check email is registered
2. Generate GUID, and send it to that email
3. Do not reset password yet
4. User clicks link, and then have to enter new pass
5. Reset password only after user is in your site, and have clicked reset button after typing new pass.
6. Make that GUID expirable within a short time period to make it safer.

---

### Risk Rating

The overall risk identified to BADSTORE.NET as a result of the penetration test is **High**. A direct path from external attacker to full system compromise was discovered. It is reasonable to believe that a malicious entity would be able to successfully execute an attack against BADSTORE.NET.

---

## Appendix A: Vulnerability Detail and Mitigation

### Risk Rating Scale

Exploited vulnerabilities are ranked based upon likelihood and impact to determine overall risk.

#### Directory Path Traversal/Enumerate System Find

**Rating:** **High**

**Description:** Directory Path Traversal/Enumerate System Find

**Impact:** Using common enumeration and brute-forcing techniques, it is possible to retrieve the all folders and robots.txt file which is most sensitive files with supplier/scanbot/upload folder which are containing information of encoding users with their private IP's.

**Remediation:** Ensure that all the directories must be provided with proper authorization must be provided with no access rule at server level and if any directory has to be access with known directory level then it should give the error 403 no directory found or file found. Authentication has to be check for direct parameter modification access.

#### Admin Webserver Interface Compromise

**Rating:** **High**

**Description:** Direct page access of Admin level through parameter modification of Action parameter.

**Impact:** Any malicious users can add users and can view other user details. This is like compromising the system with direct admin access. This is due to if any parameter modification has to be done then definitely default username will be taken into account like root, user, guest, admin or administrator.

**Remediation:** Ensure that all the access level or any user who is administrator should be provided with some long integer value so that it will be no way to determine the user access level or any user credentials. This is to be done from Server side and Database side. This parameter modification directly hits the database query and if whitelisting has not been done for user level,

then it can be easily to bypass the authentication, so whitelisting and providing parameter value like "4849303034344" is mandatory to prevent from authentication bypass.

### **Cross Site Scripting Attack**

**Rating:** **High**

**Description:** Cross site Scripting is present in vulnerable parameter as defined into report.

**Impact:** Session can be easily stolen through cross site scripting attack, we call as session hijacking and whole system can be compromised. If the token are not properly implemented then it can be easily to bypass through cross site scripting, generally if there is cross site scripting as reflected then it reflects only in response or if it storing into database then it must be persistent cross site scripting attack.

**Remediation:** Block all malicious characters or characters; this is best for mitigation of Cross Site Scripting Attack.

---

## **SQL Injection**

**Rating:** **High**

**Description:** Every parameter which has an entry point is affected with SQL injection.

**Impact:** If you can see above screenshots, every parameter is affected with SQL injection, reason is that input parameter is not sanitized at client level then server level which is hitting database directly. Queries are written in such a way that it is hitting database directly, whole DB can be compromised.

**Remediation:** Parameterized Queries, Writing Stored Procedures, Whitelisting of parameters, Escaping wild card characters are various ways for mitigation of SQL Injection.

---

## **Cross Site Request Forgery**

**Rating:** **High**

**Description:** Every parameter which has an entry point as email or password or first or last name, all are vulnerable to cross site request forgery.

**Impact:** Impact is simple on that, that it is vulnerability of account take over with known email id or with brute force attack, we can determine the pattern for whether it is first or last name or email or password. If any website is using with default parameters then they can change all password under one admin panel.

**Remediation:** Strong cookie or token mechanism, no token generation in mail for password reset functionality, Strong encryption requires for implementation and expiration for any token which is generated are various ways for mitigation of this vulnerability.

---

### **Reset Password Functionality Vulnerability**

**Rating:** **High**

**Description:** parameter where reset or forget password functionality is there, that are vulnerable to reset password functionality vulnerability.

**Impact:** Reset password are vulnerable to session hijacking or MITM attacks, if the cookies are not implemented. Malicious user can brute force for given list of mail ids and can reset the password. If the password is generated with base64 or any other encoded scheme than any other malicious user can decode all those schemes and can change the password. Impact is high if the sessions or tokens are not implemented properly.

**Remediation:** Generation of new URL from server side with strong token encryption implementation, multiple attempts for resetting password lock the account, 2FA implementation and strong password implementation are various ways for mitigation of reset password functionality vulnerability.