# 7.0 TESTING

## 7.1 TESTING PLAN

**1. Requirements Testing**

This includes the testing of the requirements implementation to ensure that the system is correctly and appropriately meeting the specified requirements as mentioned in the requirements specification.

**2. Implementation Testing**

This includes the testing of the implementation modules such that the flow of implementation is correct and appropriately working depending on the type of user and user input.

**3. Testing Process**

The modules are tested individually and then in an integrated manner in a definite order .i.e. in the order of design, implementation etc.

**4. Tested Items**

The tested items include user login and authorization, date validation, user input validation and generated files verification.

**5. Testing Schedule**

The testing is carried out first at the module level and on the implementation of all the modules, the testing is then carried out at the integration level.

## 7.2 TESTING STRATEGY

- **Unit testing**

  Unit testing or the testing of a single unit focuses verification effort on the smallest unit of software design - the software component or module.

- **Integration testing**

  Integration testing is a systematic technique for constructing the software architecture while at the same time conducting tests to uncover errors associated with interfacing.

- **Validation testing**

  Validation testing begins at the culmination of integration testing, when individual components have been exercised, the software is completely assembled as a package, and interfacing errors have been uncovered and corrected.

- **System testing**

  System testing is actually a series of different tests whose primary purpose is to fully exercise the computer-based system.

## 7.3 TESTING METHODS

The goal of testing is to find errors, and a good test is one that has a high probability of finding an error.

Types of testing:

- **Functional Testing**

  In the black-box testing approach, test cases are designed using only the functional specification of the software, i.e. without any knowledge of the internal structure of the software. For this reason, black-box testing is known as functional testing.

- **Structural Testing**

  In the white-box testing approach, designing test cases requires thorough knowledge about the internal structure of software, and therefore the white-box testing is called structural testing.

## 7.4TEST SUITE

1. **Registration of client**

- username
- email
- password

2. **Add/Edit Details by WAF admin**

- Add/Edit Client Details

3. **Add rules for customer by WAF**

- Allocate Customer Id
- URL
- Blacklist_ip
- Rules_configuration

## 7.5 TEST CASES

A test case has a component that describes an input, action or event and an expected response to determine if a feature of an application is working correctly.

**Test Suites No: 1**

Test Suite Detail: For Registration of client

Add details to register client to access client's dashboard

| Test Case ID | Function Name | Input | Expected Output | Actual Output | Pass/Fail |
|---|---|---|---|---|---|
| **1** | Register Client's Userid | User id: Yash1234 | Next Register user's userid To Add details | Saved successfully | pass |
| **2** | Enter client's email id | Email: Yash2554@gmail.com | Next Register user's emailid To Add details | Saved successfully | pass |
| **3** | Enter Client's Password | Password: ********** | Next Register user's password To Add details | Saved successfully | Pass |
| **4** | Security Question | Question: My first name? | Next Register user's question To Add details | Saved successfully | pass |
| **5** | Answer | Answer: Patel | Next Register user's answer To Add details | Saved successfully | pass |

Table 7.5.1 For Registration of client

**Test Suites No: 2**

Test Suite Detail: Maintain Clients Details by WAF admin

Valid Customer Id, Name, Address, Contact number, Date for Service, IP, URL.

| Test Case ID | Function Name | Input | Expected Output | Actual Output | Pass/Fail |
|---|---|---|---|---|---|
| 1 | Customer ID | 1 | Saved to database<br>Next add other details | Saved Successfully | Pass |
| 2 | Name | Yash Patel | Saved to database<br>Next add other details | Saved Successfully | Pass |
| 3 | Address | Valid - Address | Saved to database<br>Next add other details | Saved Successfully | pass |
| 4 | Contact Details | Mobile number:<br>9879316527 | Saved to database<br>Next add other details | Saved Successfully | Pass |
| 5 | Date | 1-April, 2015 | Saved to database<br>Next add other details | Saved Successfully | Pass |
| 6 | IP | Internal IP:<br>192.168.1.199 | Saved to database<br>Next add other details | Saved Successfully | Pass |
| 7 | URL | Demo.testfire.net | Saved to database<br>Next add other details | Saved Successfully | pass |

Table 7.5.2 Maintain Clients Details by WAF admin

**Test Suites No: 3**

Test Suite Detail: Add rules for customer

Add rules Configuration file assign a valid URL to proxy pass & IP list to black list those IPs for a specific URL/IPs.

| Test Case ID | Function Name | Input | Expected Output | Actual Output | Pass/Fail |
|---|---|---|---|---|---|
| 1 | Customer Id | 1 | Saved to database<br>Next add other details | Saved Successfully | Pass |
| 2 | URL | Demo.testfire.net | Saved to database<br>Next add other details | Saved Successfully | pass |
| 3 | Blacklist IP | 192.168.1.100 | Saved to database<br>Next add other details | Saved successfully | pass |
| 4 | Rule | Rule configuration file | Saved to database<br>Next add other details | Saved successfully | pass |

Table 7.5.3 Add rules for customer

# Test Case of vulnerable web application (without WAF)

**Test Suites No: 1**

Vulnerability Name: Cross Site Scripting (reflected Xss)

Description:
**Cross-site scripting** (**XSS**) is a type of computer security vulnerability typically found in Web applications. **XSS** enables attackers to inject client-side script into Web pages viewed by other users. A **cross-site scripting** vulnerability may be used by attackers to bypass access controls such as the same-origin policy.

Severity: High

**Proof of concept** – Observe the screenshot that application is vulnerable to (Cross Site Scripting) Xss attack.

POC 1: (snapshot1)

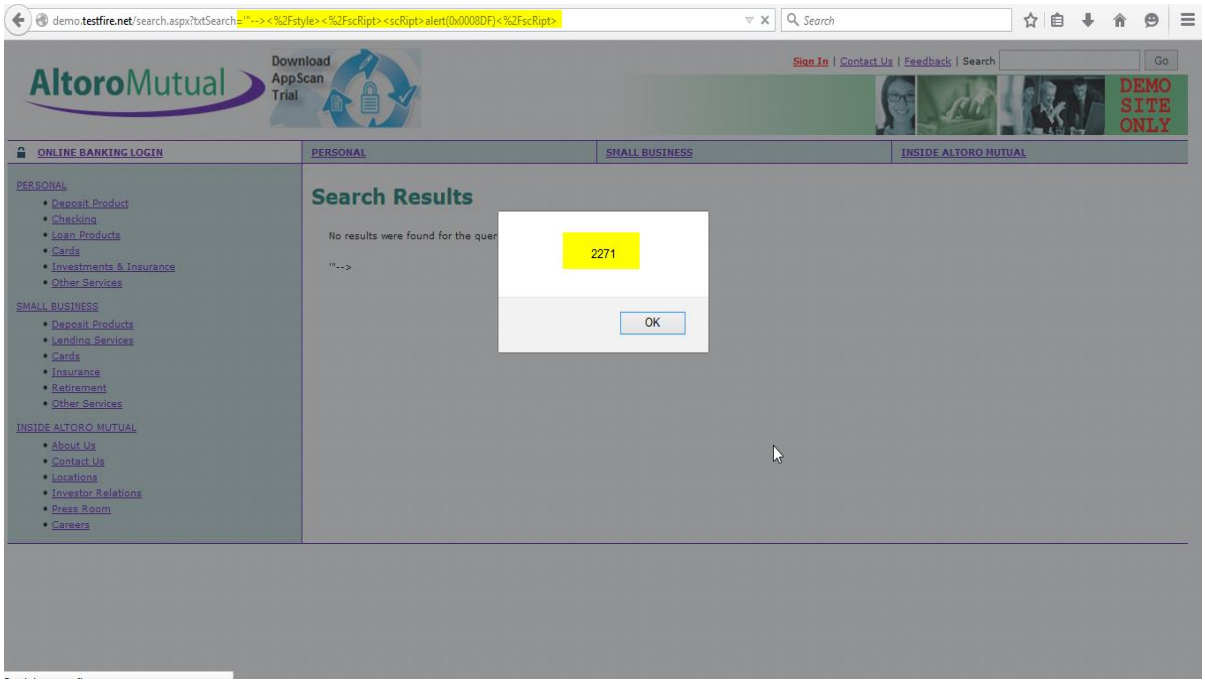


Fig 7.5.1 POC 1-Cross Site Scripting

POC 2: (snapshot2)

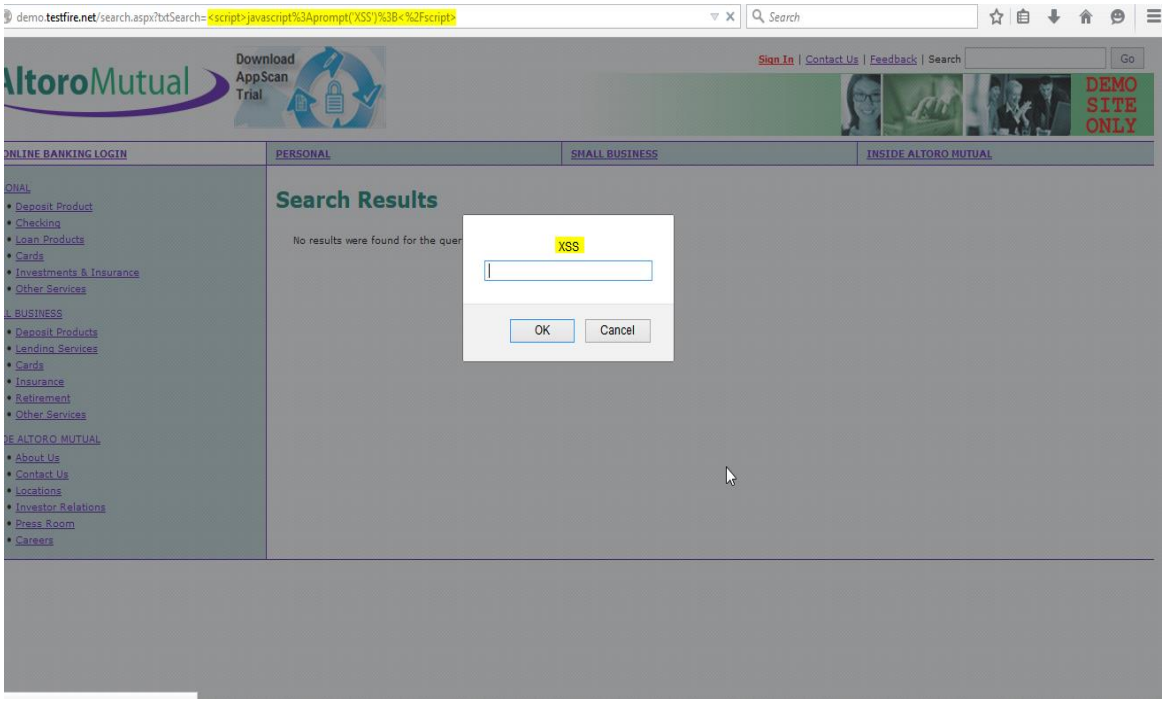


Fig 7.5.2 POC 2-Cross Site Scripting

POC 3: (snapshot3)

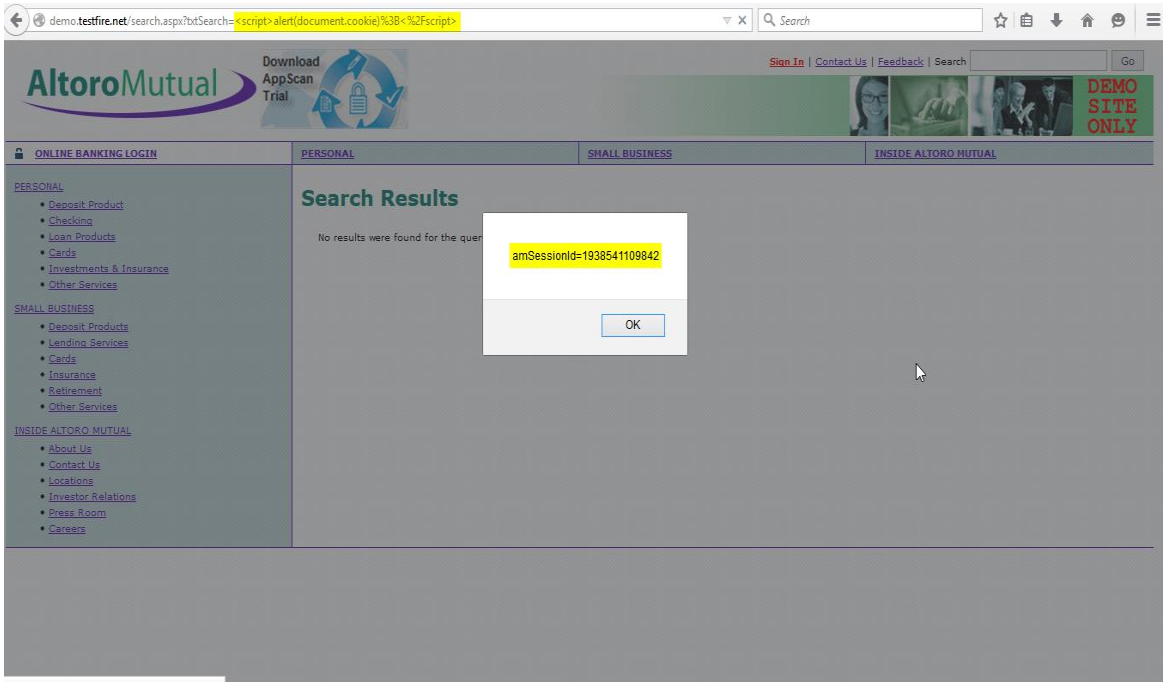


Fig 7.5.3 POC 3-Cross Site Scripting

**Test Suites No: 2**

Vulnerability Name: SQL Injection (sqlmap)

Description:

Sqlmap is an open source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over of database servers. It comes with a powerful detection engine, many niche features for the ultimate penetration tester and a broad range of switches lasting from database fingerprinting, over data fetching from the database, to accessing the underlying file system and executing commands on the operating system via out-of-band connections.

Severity: High

**Proof of concept** – Observe the screenshot that application is vulnerable to (sqlmap python script attack)

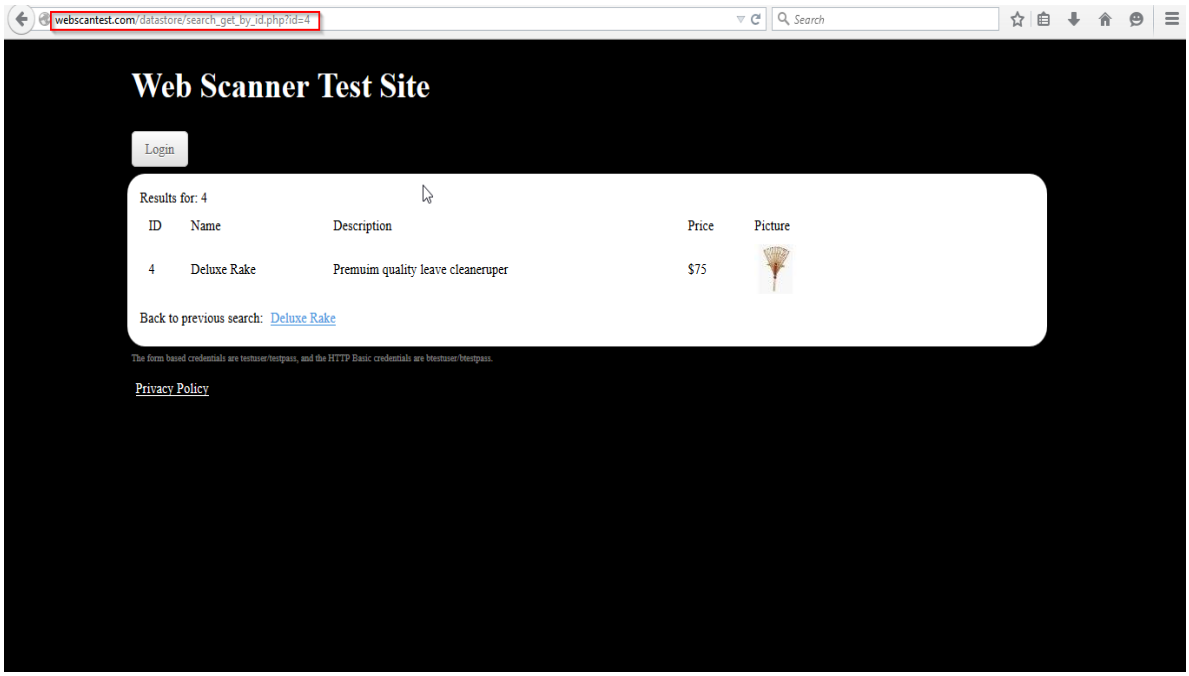


Fig 7.5.4 POC 4- parameter ID = ‘ Vulnerable

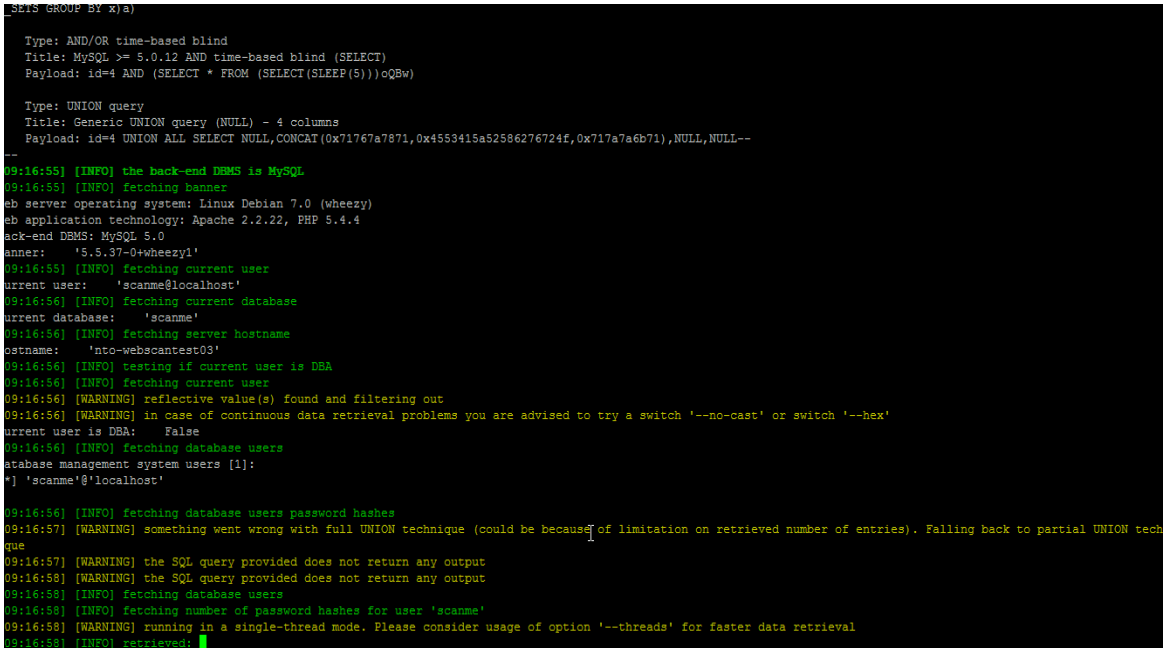


Fig 7.5.5 POC 5- sqlmap

**Sqlmap python script will exploited the database server**

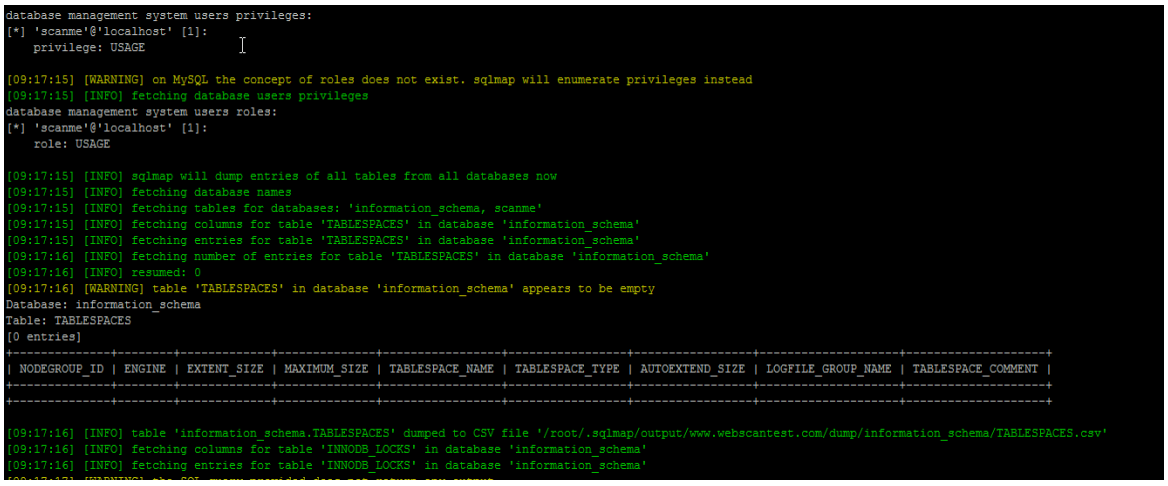


Fig 7.5.6 POC 6- sqlmap

**Test Suites No: 3**

Vulnerability Name: Click jacking

Description:

**Click jacking** (**User Interface redress attack**, **UI redress attack**, **UI redressing**) is a malicious technique of tricking a Web user into clicking on something different from what the user perceives they are clicking on, thus potentially revealing confidential information or taking control of their computer while clicking on seemingly innocuous web pages

Severity: High

**Proof of concept** – Observe the screenshot that application is vulnerable to Click jacking attack.

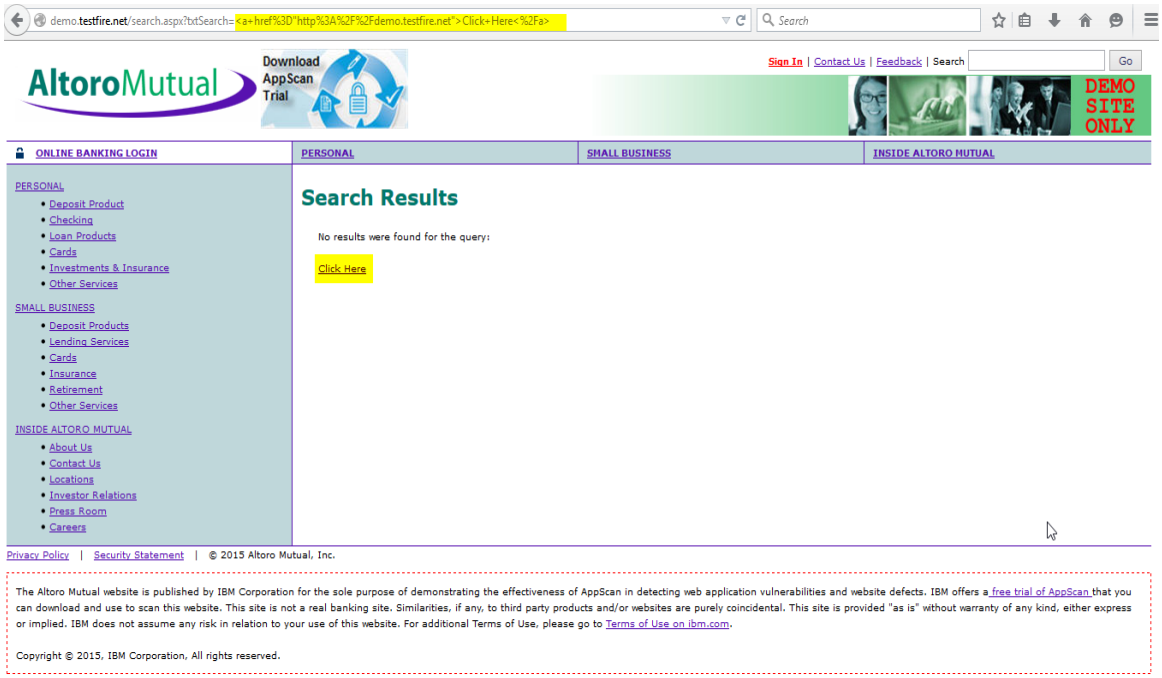


Fig 7.5.7 POC 7- click jacking

**Test Suites No: 4**

Vulnerability Name: Sql Injection

Description:

**SQL injection** is a code **injection** technique, used to attack data-driven applications, in which malicious **SQL** statements are inserted into an entry field for execution (e.g. to dump the database contents to the attacker).

Severity: High

**Proof of concept** – Observe the screenshot that application is vulnerable to sql injection attack

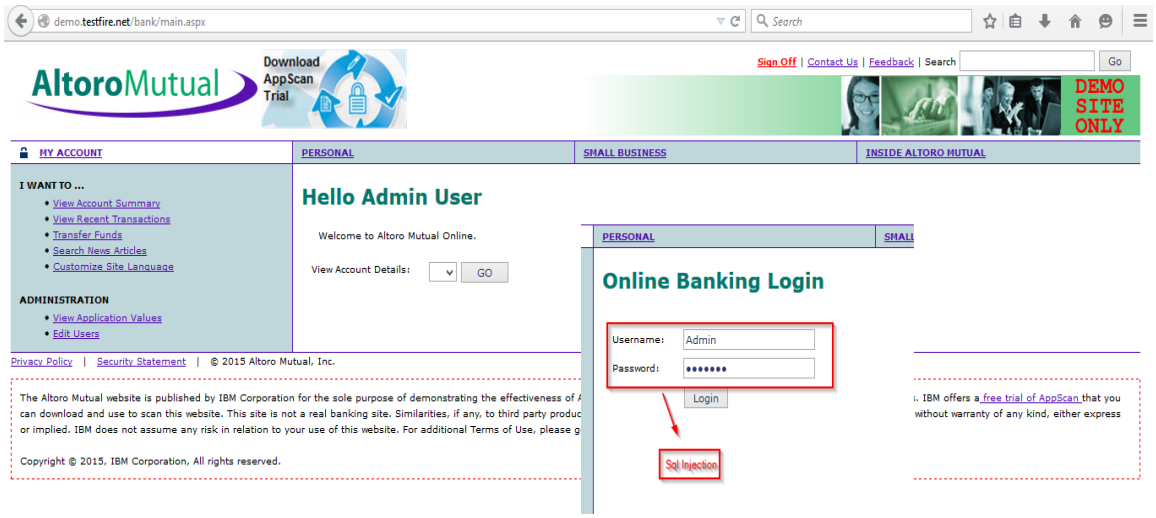


Fig 7.5.8 POC 8- sql injection

**Test Suites No: 5**

Vulnerability Name: Directory traversal

Description:

**Directory traversal** is a type of HTTP exploit that is used by attackers to gain unauthorized access to restricted **directories** and files. **Directory traversal**, also known as path **traversal**, ranks #13 on the CWE/SANS Top 25 Most Dangerous Software Errors.

Severity: High

**Proof of concept** – Observe the screenshot that application is vulnerable to Directory traversal attack


Fig 7.5.9 POC 9- Directory traversal

**Test Suites No: 6**

Vulnerability Name: Information Leakage

Description:

**Information leakage** happens whenever a system that is designed to be closed to an eavesdropper reveals some information to unauthorized parties nonetheless. For example, when designing an encrypted instant messaging network, a network engineer without the capacity to crack encryption codes could see when messages are transmitted, even if he could not read them.

Severity: High

**Proof of concept** – Observe the screenshot that application is vulnerable to information leakage

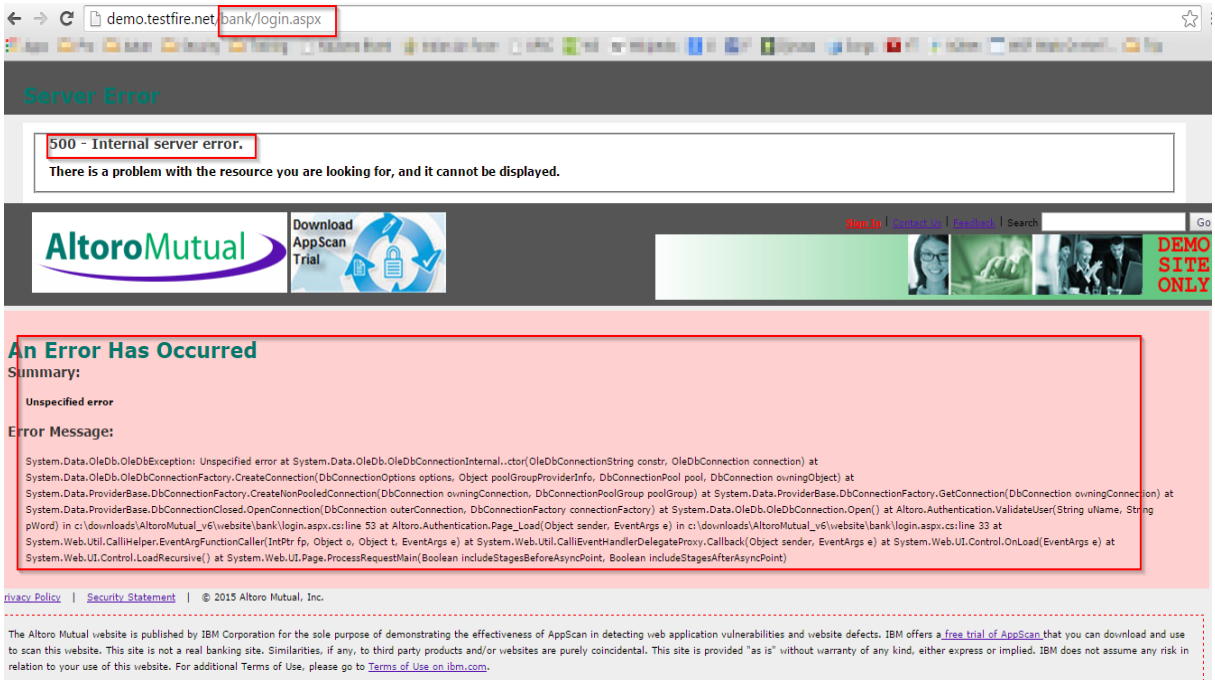


Fig 7.5.10 POC 10- Information Leakage

**Test Suites No: 7**

Vulnerability Name: Malicious File Upload

Description:

A **backdoor shell** is a malicious piece of code (e.g. PHP, Python, and Ruby) that can be uploaded to a site to gain access to files stored on that site. Once it is uploaded, the hacker can use it to edit, delete, or download any files on the site, or upload their own.

Severity: High

**Proof of concept** – Observe the screenshot that application is vulnerable to shell upload.

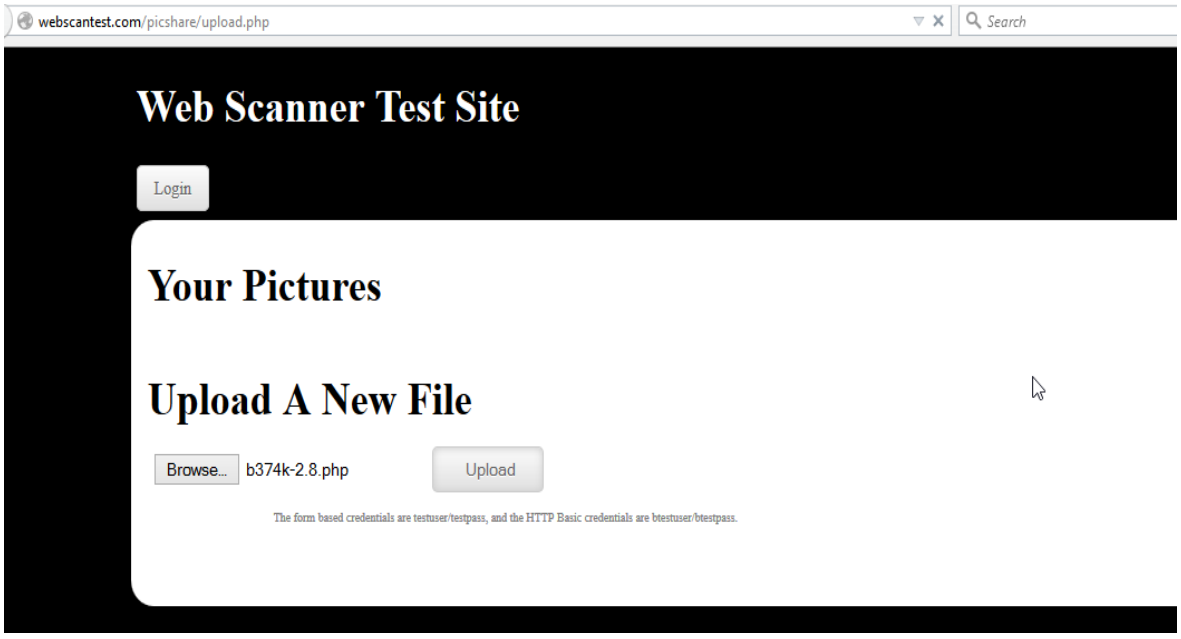


Fig 7.5.11 POC 11- Malicious File Upload

Shell exploited the web server & database server

Shell Upload (Snapshot 1)

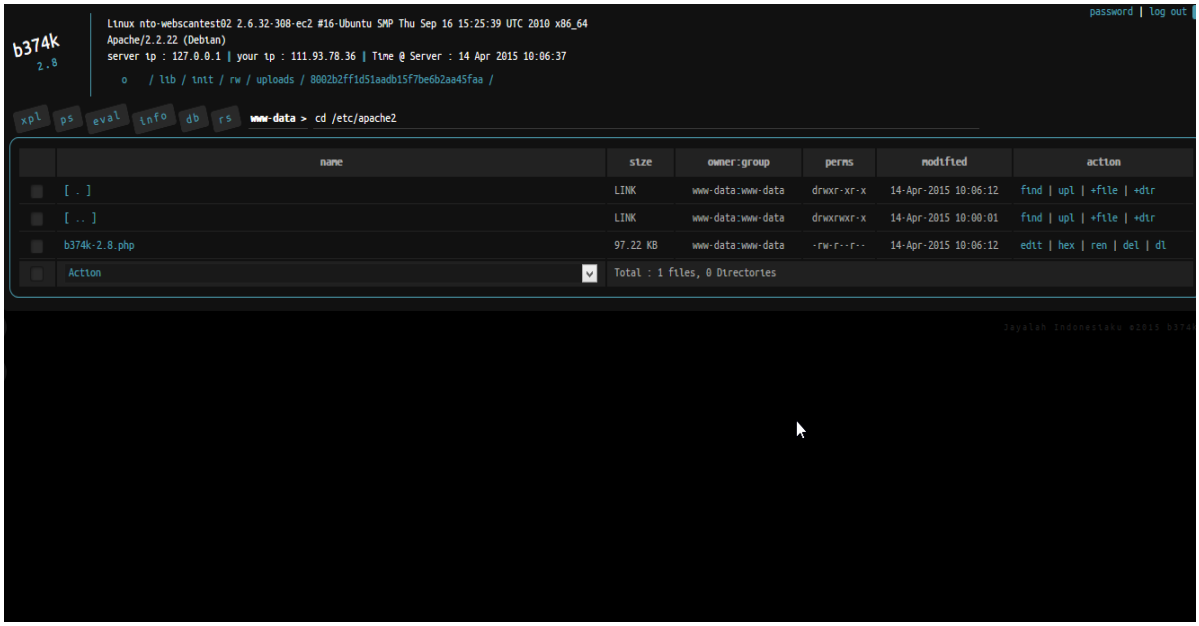


Fig 7.5.12 POC 12- Shell Exploited

Shell Upload (Snapshot 2)

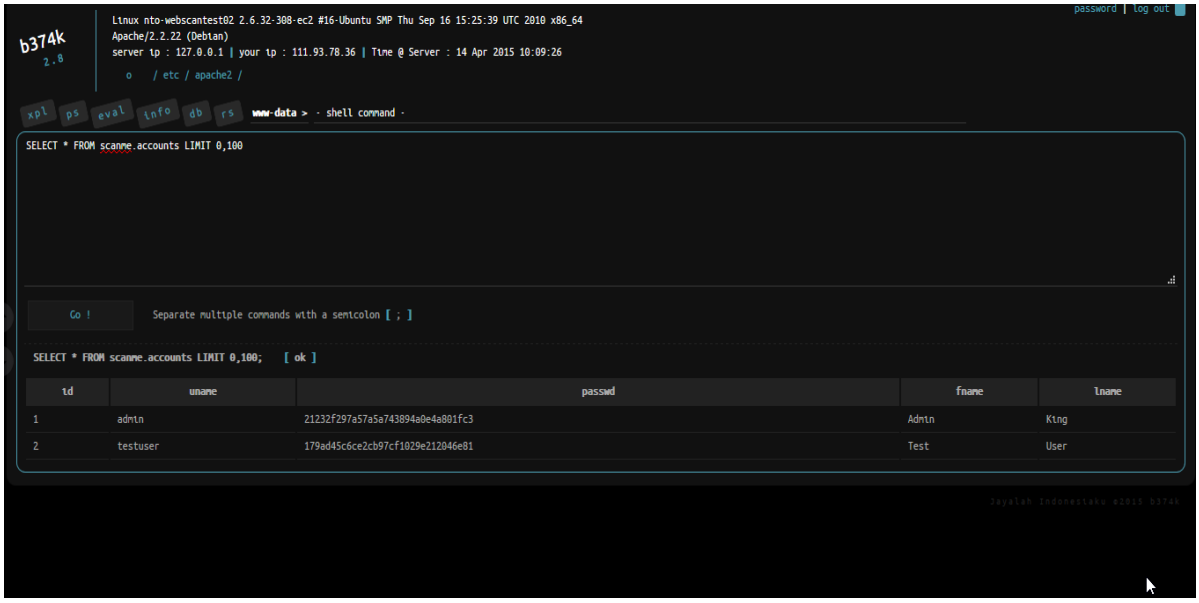


Fig 7.5.13 POC 13- Shell Exploited1

Shell Upload (Snapshot 3)

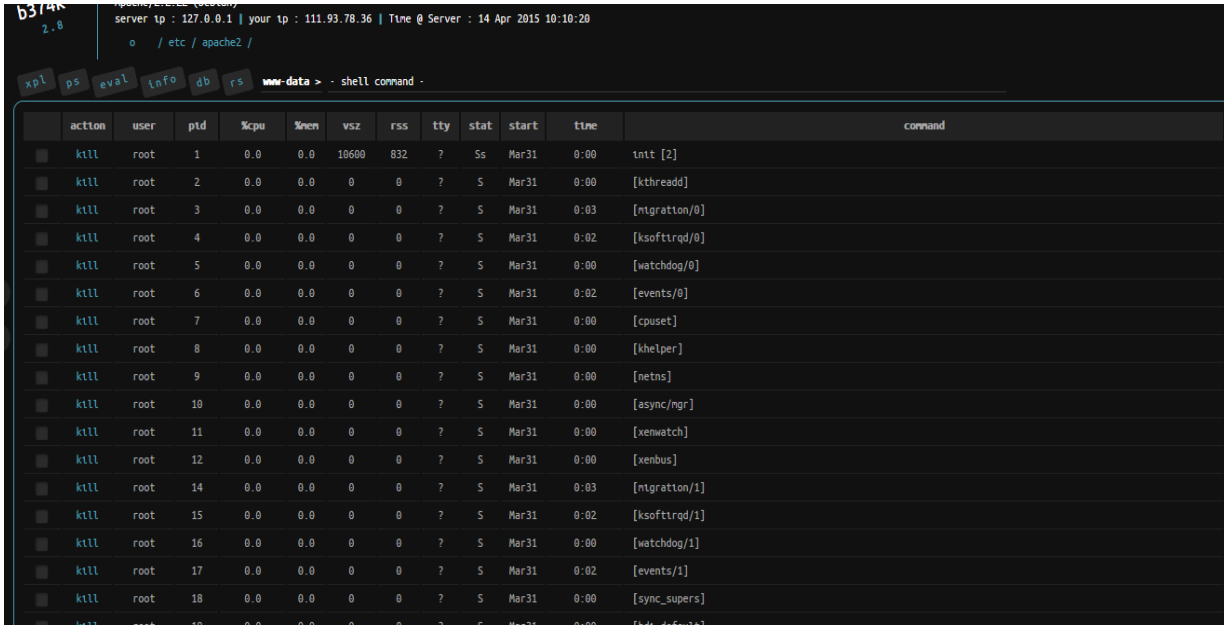


Fig 7.5.14 POC 14- Shell Exploited2

Shell Upload (Snapshot 4)

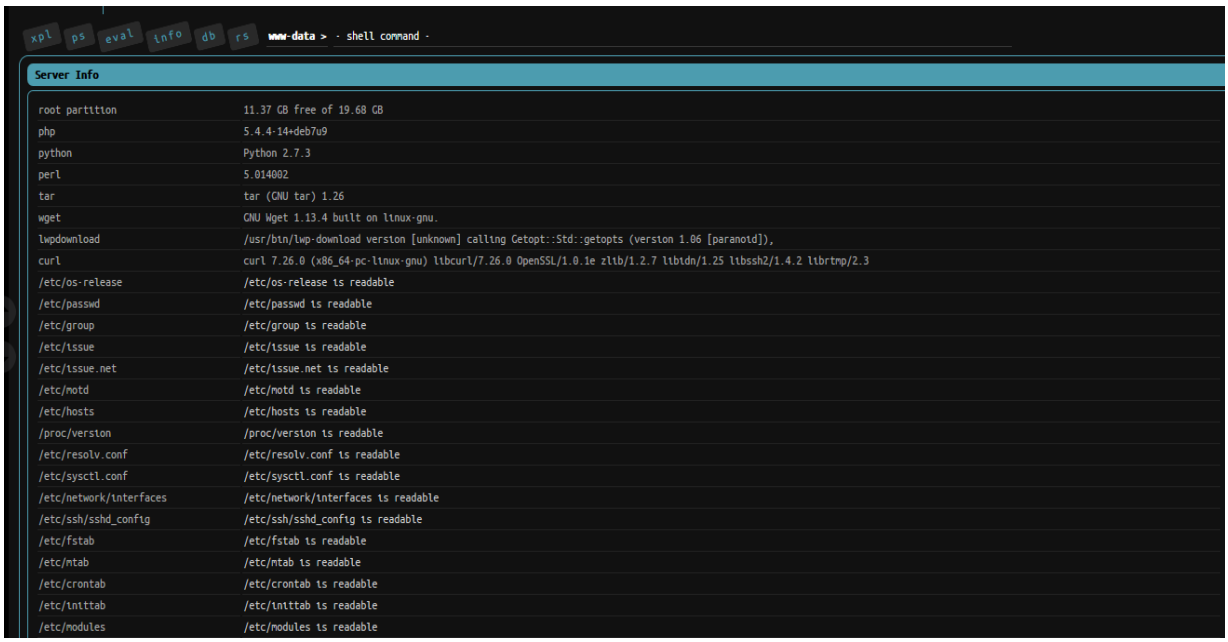


Fig 7.5.15 POC 15- Shell Exploited3

**WAF will protect this all attacks and this way how mod security will patch those vulnerabilities.**

FINAL POC: snap shot 1 (WAF)


Fig 7.5.16 POC 16- Vulnerabilities patched by WAF