

Test Security Report

Code Revision: 1.0.0.0

Company: Acme Inc.

Report: TEST201018

Author: [Name]

Date: [Date]

VWA Security Report

TEST20101801 - SQLi - SEVERITY	3
TEST20101802 - SQLi - SEVERITY	7
TEST20101803 - XSS - SEVERITY	10
TEST20101804 - XSS - SEVERITY	12
TEST20101805 - Broken Auth - SEVERITY	16
TEST20101806 - Broken Auth - SEVERITY	17

VWA Security Report

TEST20101801 - SQLi - SEVERITY

Vulnerability Exploited: SQLi

Severity: [Critical, High, Medium, Low, Info]

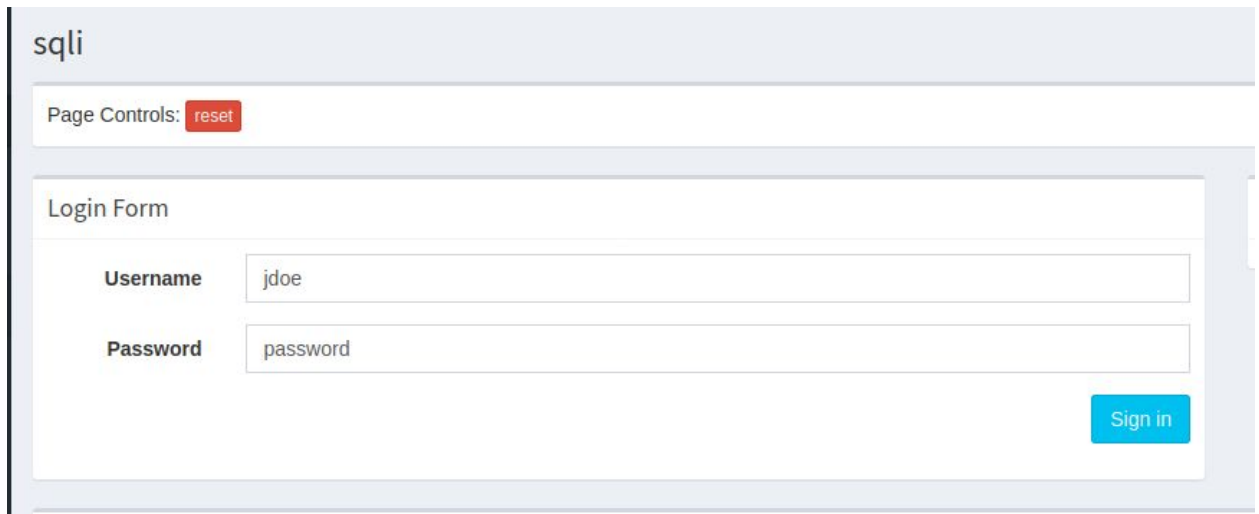
System: VWA Web Application

Vulnerability Explanation:

Summary of what was found

Vulnerability Walk-thru:

1. Got to the SQLi section of the application
2. On the page I noticed a login form



The screenshot shows a web application interface for the 'sqli' section. At the top, there is a header 'sqli'. Below it, there is a 'Page Controls' section with a 'reset' button. The main content area is titled 'Login Form' and contains two input fields: 'Username' with the value 'jdoe' and 'Password' with the value 'password'. A 'Sign in' button is located at the bottom right of the form.

3. After looking at the network traffic flow, I can see that the login page is posting data back to the following `http://0.0.0.0:3000/sql_i/login`.

VWA Security Report

Page Controls: [reset](#)

Login Form

Username

Password

[Sign in](#)

Login Results

2,user,john,doe,jdoe

The following SQL was I
SELECT id, role, firstna

Users List

Debugger [Network](#) [Style Editor](#) [Performance](#) [Memory](#) [Storage](#) [Accessibility](#) [Application](#)

File	Initiator	Type	Transferred	Size	Headers	Cookies	Request	Response	TI
login	jquery.js:9837 (xhr)	json	372 B	226 B	Filter Headers		POST http://0.0.0.0:3000/sql/login	Status 200 OK ?	

Login Form

Username

Password

[Sign in](#)

Login Results

2,user,john,doe,jdoe

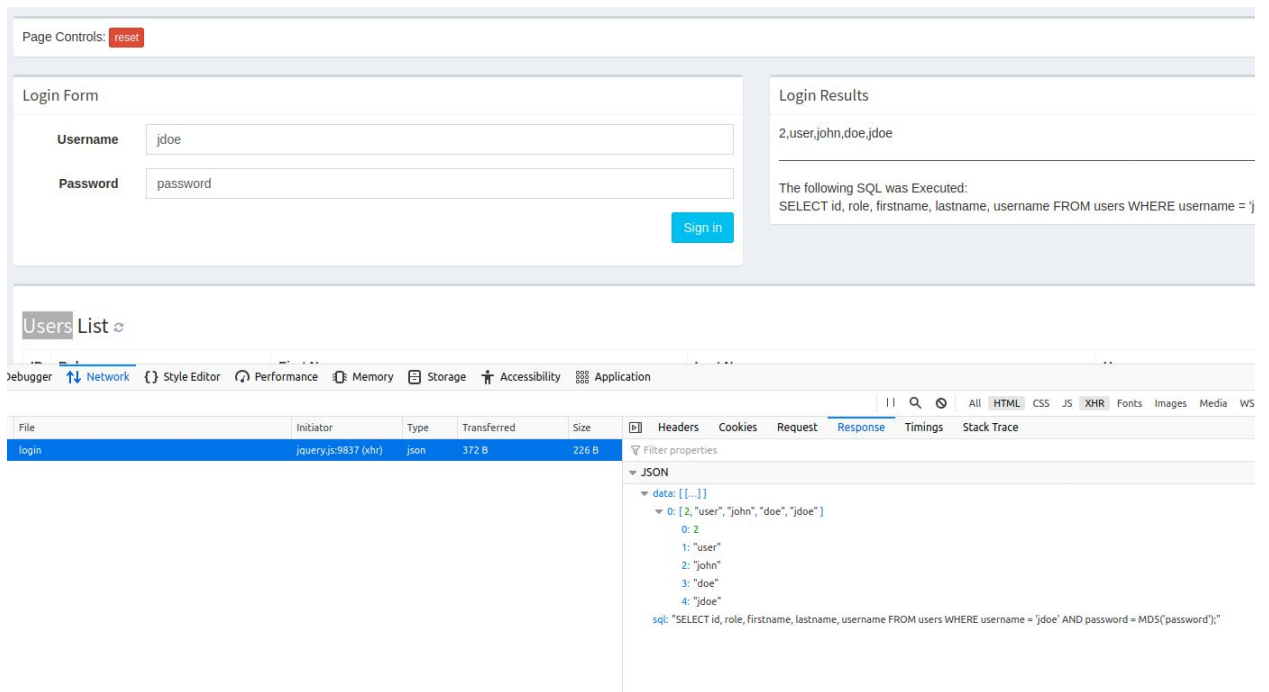
The following SQL was Executed:
SELECT id, role, firstname, lastname, username FR

Users List

Debugger [Network](#) [Style Editor](#) [Performance](#) [Memory](#) [Storage](#) [Accessibility](#) [Application](#)

File	Initiator	Type	Transferred	Size	Headers	Cookies	Request	Response	Timings	Stack Trace
login	jquery.js:9837 (xhr)	json	372 B	226 B	Filter Request Parameters		Form data username: "jdoe" password: "password" Request payload 1 username=jdoe&password=password			

VWA Security Report



4. Next I used a simple SQLi "password' or 1=1--" to see if the login form is exploitable. This injection didn't yield any results.
5. Next I used another SQLi "password') or 1=1--" to see if they are hashing or doing something else with that field.

VWA Security Report

The screenshot displays a web application security tool interface. At the top, the title 'sqli' is visible. Below it, the 'Page Controls' section includes a 'reset' button. The main area is divided into two panels: 'Login Form' on the left and 'Login Results' on the right. The 'Login Form' contains fields for 'Username' (filled with 'jdoe') and 'Password' (filled with 'password') or 1=1--', and a 'Sign in' button. The 'Login Results' panel shows the output of the SQL injection: '1,admin,flag,flag(548562),admin' and '2,user,John,doe,jdoe'. Below the login form, the 'Users List' section is visible. At the bottom, a network debugger window is open, showing a list of network requests. The selected request is a JSON response from 'jquery.js:9837 (xhr)' with a status of 200. The response body is a JSON array: '[[1, "admin", "flag", "flag(548562)", "admin"], [2, "user", "John", "doe", "jdoe"]]'. The SQL query executed is shown at the bottom: 'sql: "SELECT id, role, firstname, lastname, username FROM users WHERE username = 1=1--";'.

sqli

Page Controls: [reset](#)

Login Form

Username:

Password:

[Sign in](#)

Login Results

1,admin,flag,flag(548562),admin
2,user,John,doe,jdoe

The following SQL was Executed:
SELECT id, role, firstname, lastname, username FROM users WHERE username = 1=1--);

Users List

Debugger | Network | Style Editor | Performance | Memory | Storage | Accessibility | Application

File	Initiator	Type	Transferred	Size
login	jquery.js:9837 (xhr)	json	372 B	226 B
login	jquery.js:9837 (xhr)	json	300 B	154 B
login	jquery.js:9837 (xhr)	json	301 B	155 B
login	jquery.js:9837 (xhr)	json	475 B	329 B

Headers | Cookies | Request | **Response** | Timings | Stack Trace

Filter properties

JSON

data: [[...], [...]]

0: 1

1: "admin"

2: "flag"

3: "flag(548562)"

4: "admin"

1: [2, "user", "John", "doe", "jdoe"]

0: 2

1: "user"

2: "John"

3: "doe"

4: "jdoe"

sql: "SELECT id, role, firstname, lastname, username FROM users WHERE username = 1=1--);"

6. At this point I was able to inject the SQL and return all user info.

Recommendations:

Recommendation on either how to fix it or details on the best practice.

VWA Security Report

TEST20101802 - **SQLi** - SEVERITY

Vulnerability Exploited: **SQLi**

Severity: [Critical, High, Medium, Low, Info]

System: VWA Web Application

Vulnerability Explanation:

Summary of what was found

Vulnerability Walk-thru:

1. Got to the SQLi section of the application
2. Next I looked at the network flows and noticed that the userlist is using an async function to get the data it needed.

The screenshot displays a web application interface with a 'Users List' table. The table has columns for ID, Role, First Name, and Last Name. A single row is visible with ID 2, Role user, First Name john, and Last Name doe. Below the table is a copyright notice: 'Copyright © 2020 Test Company LLC. All rights reserved.'

Below the web application interface is a network debugger window. The 'Network' tab is active, showing a list of network requests. The selected request is a GET request to the URL 'http://0.0.0.0:3000/sqli/users/2'. The status is '200 OK' and the response size is '368 B (222 B size)'. The 'Response Headers' section shows 'Content-Length: 222'.

ID	Role	First Name	Last Name
2	user	john	doe

Copyright © 2020 Test Company LLC. All rights reserved.

Debugger | Network | Style Editor | Performance | Memory | Storage | Accessibility | Application

File	Initiator	Type	Transferred	Size
27_1603055390768	jquery.js:9837 (xhr)	json	368 B	222 B

Headers | Cookies | Request | Response | Timings | Status

Filter Headers

GET

Scheme: http
Host: 0.0.0.0:3000
Filename: /sqli/users/2

1603055390768

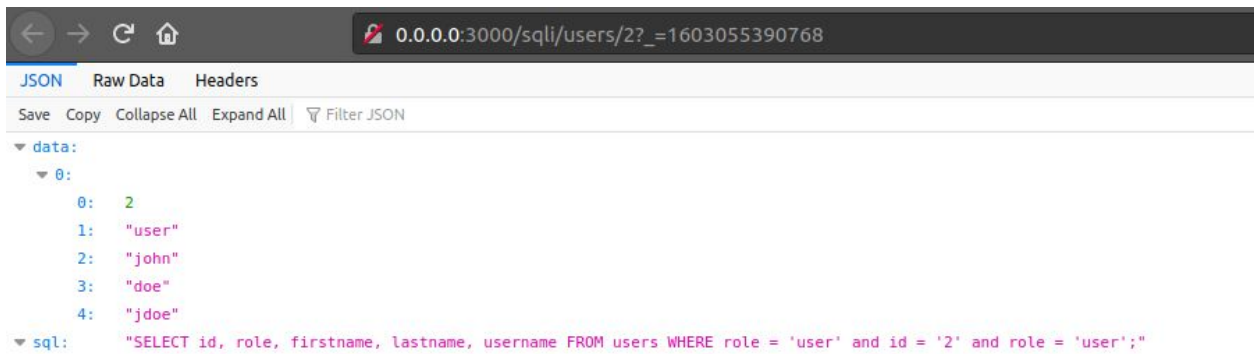
Address: 0.0.0.0:3000

Status: 200 OK
Version: HTTP/1.0
Transferred: 368 B (222 B size)
Referrer Policy: no-referrer-when-downgrade

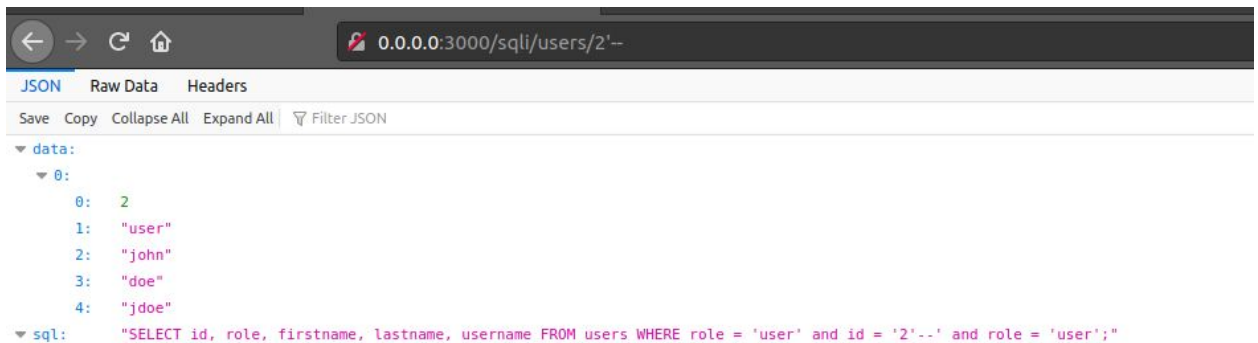
Response Headers (146 B)

Content-Length: 222

VWA Security Report



3. Then I did some basic tests to see if the async endpoint was vulnerable to SQLi.



4. After proving that this endpoint was injectable, I then crafted a SQLi that will expose all data in the table.



Recommendations:

VWA Security Report

Recommendation on either how to fix it or details on the best practice.

VWA Security Report

TEST20101803 - XSS - SEVERITY

Vulnerability Exploited: XSS

Severity: [Critical, High, Medium, Low, Info]

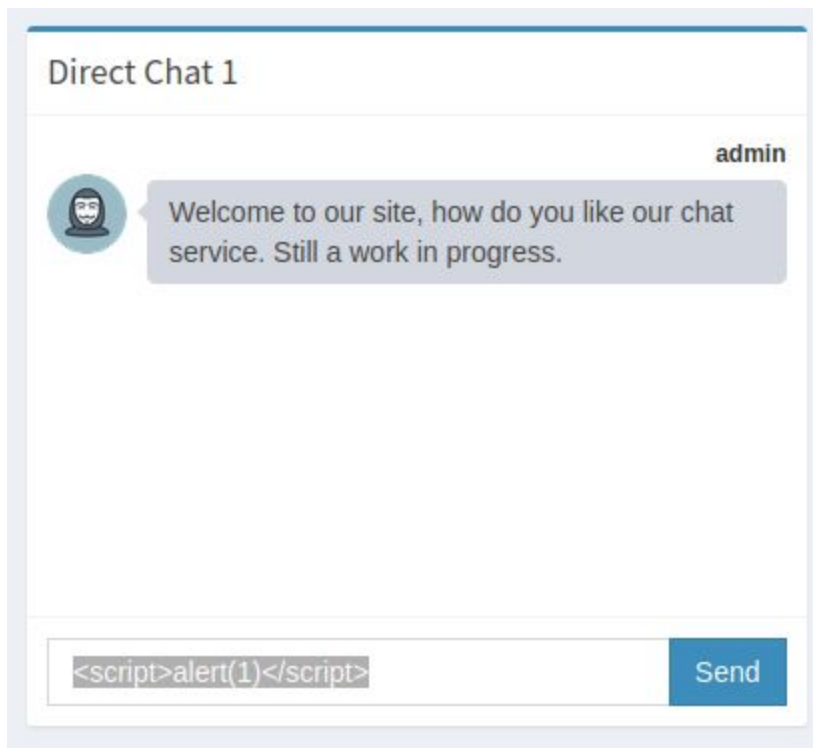
System: VWA Web Application

Vulnerability Explanation:

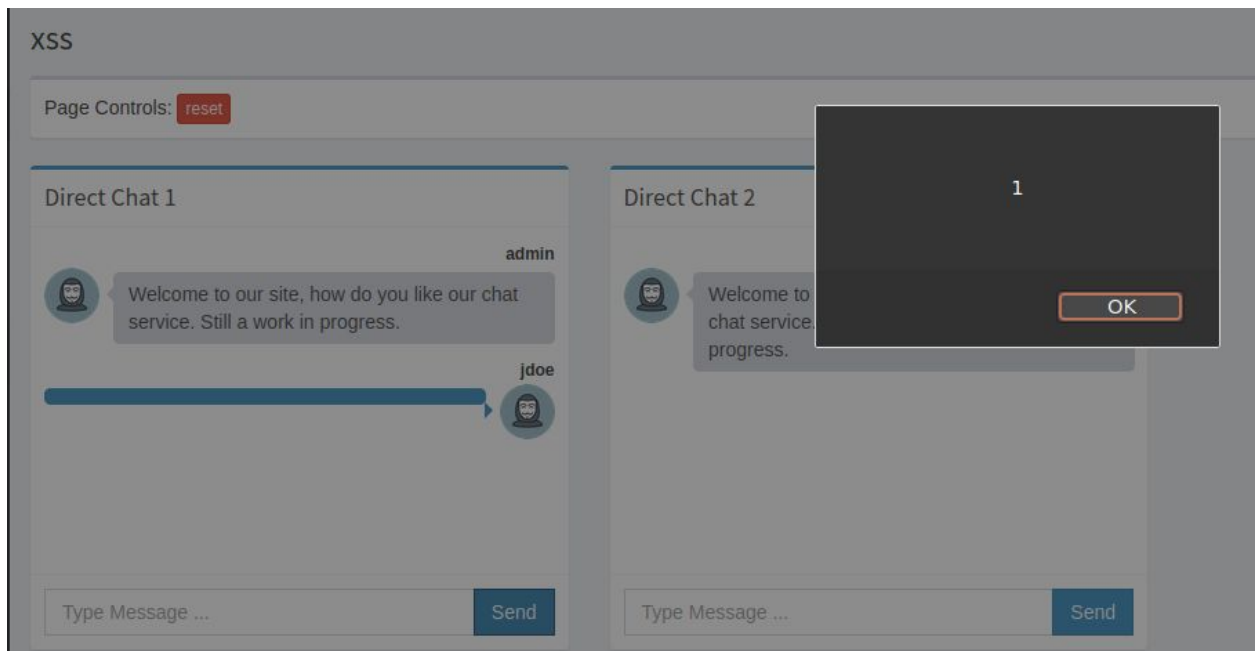
Summary of what was found

Vulnerability Walk-thru:

1. Go to the XSS Section
2. Next I check to see if Direct Chat 1 was exploitable to XSS
3. Using the following XSS I was able to inject a javascript alert in Direct Chat 1 "<script>alert(1)</script>".



VWA Security Report



Recommendations :

Recommendation on either how to fix it or details on the best practice.

VWA Security Report

TEST20101804 - XSS - SEVERITY

Vulnerability Exploited: XSS

Severity: [Critical, High, Medium, Low, Info]

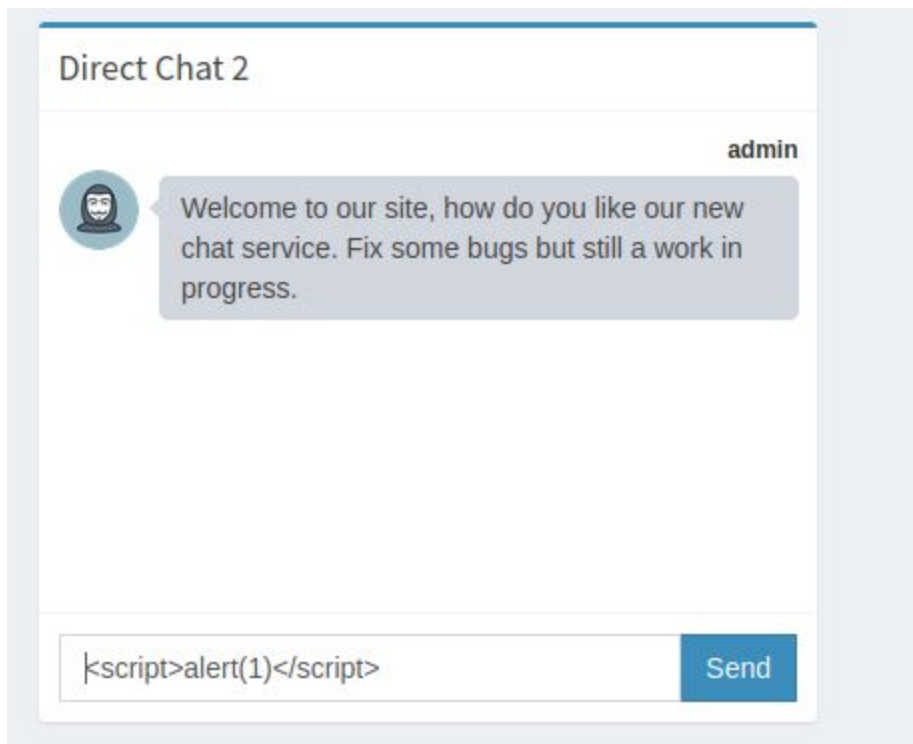
System: VWA Web Application

Vulnerability Explanation:

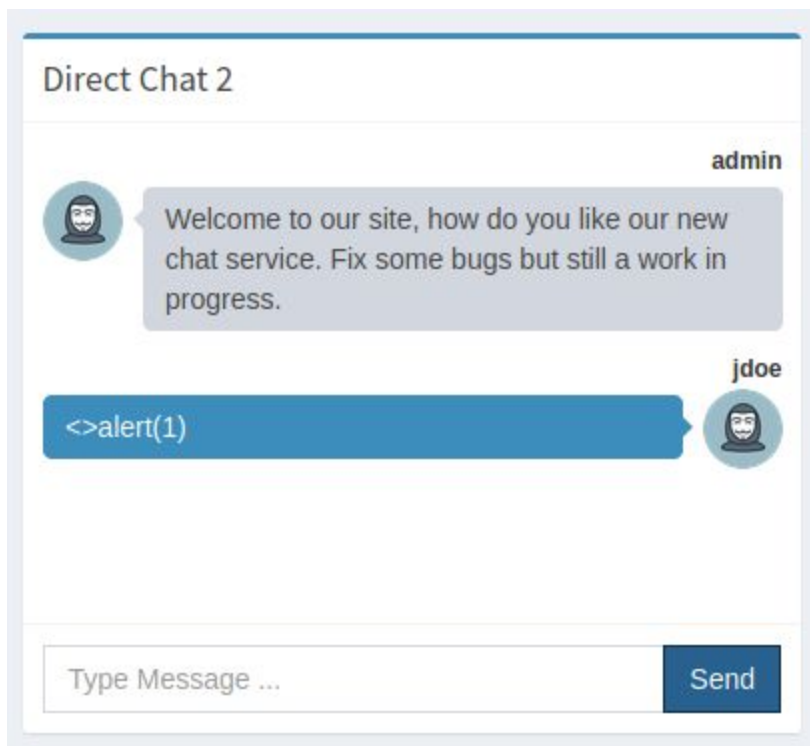
Summary of what was found

Vulnerability Walk-thru:

1. Go to the XSS Section
2. Next I check to see if Direct Chat 2 was exploitable to XSS
3. I first try a basic XSS "<script>alert(1)</script>" to see if the code is exploitable.

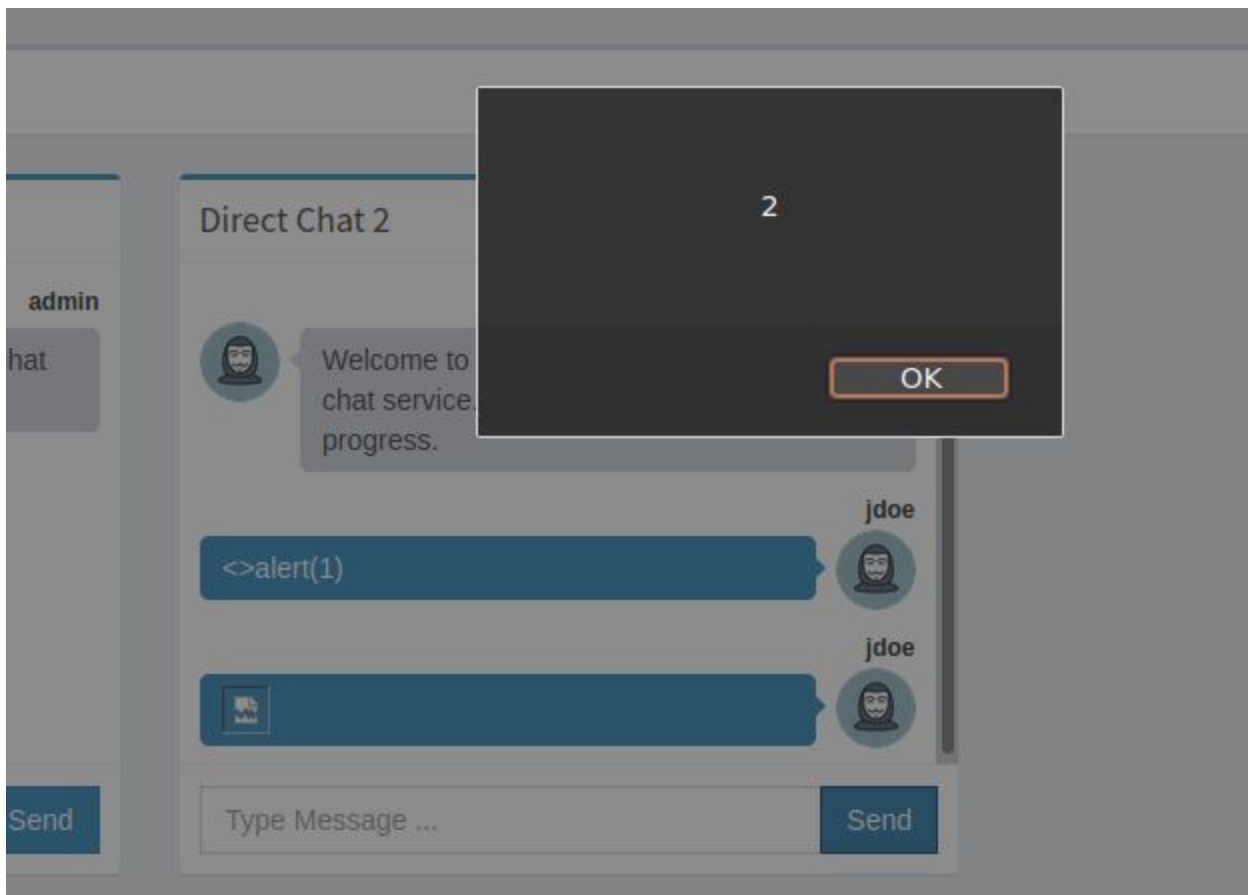
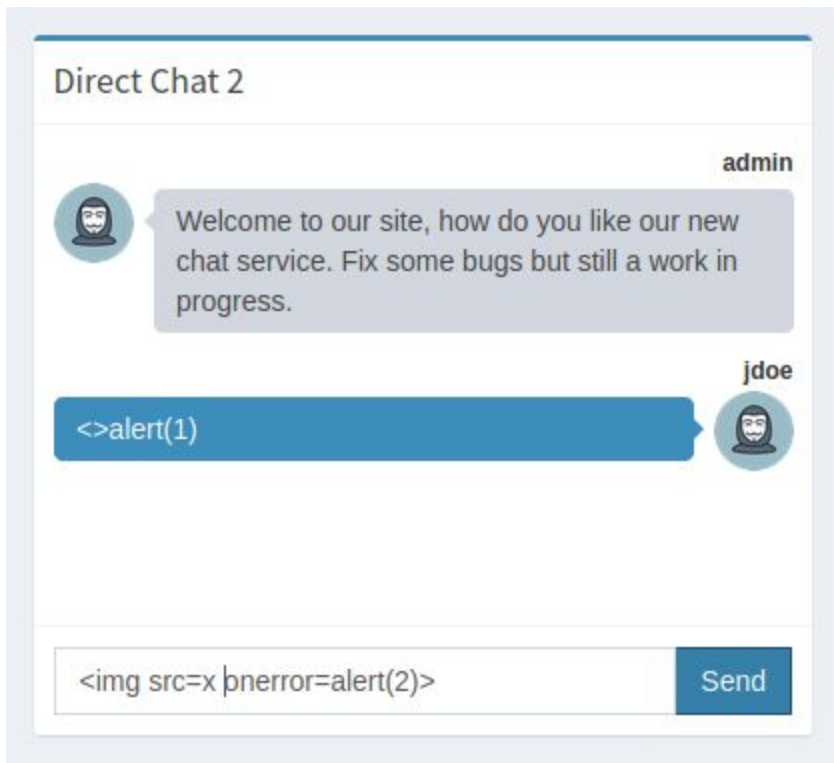


VWA Security Report



4. But the chat seems to be doing keyword replacement on the word "script".
5. Next I try a more advanced XSS, by inserting a im tag and setting the src to a non exist file with a onerror exec.

VWA Security Report



VWA Security Report

Recommendations:

Recommendation on either how to fix it or details on the best practice.

VWA Security Report

TEST20101805 - Broken Auth - SEVERITY

Vulnerability Exploited: Broken Auth

Severity: [Critical, High, Medium, Low, Info]

System: VWA Web Application

Vulnerability Explanation:

Summary of what was found

Vulnerability Walk-thru:

1. Go to the Broken Auth Section
2. Using a python script call "bruteforce.py"
3. I ran the following cmd.
 - a. python bruteforce.py -U test-username.txt -P test-password.txt -f False
<http://0.0.0.0:3000/brokenauth/>
4. After a min of running it was a find a working username and password combination that allowed me to login.

```
python bruteforce.py -U test-username.txt -P test-password.txt -f False http://0.0.0.0:3000/brokenauth/
[+] Login Found! {'username': 'test', 'password': 'klaster'}
This is a demo code used for this training.
```

Recommendations:

Recommendation on either how to fix it or details on the best practice.

VWA Security Report

TEST20101806 - Broken Auth - SEVERITY

Vulnerability Exploited: Broken Auth

Severity: [Critical, High, Medium, Low, Info]

System: VWA Web Application

Vulnerability Explanation:

Summary of what was found

Vulnerability Walk-thru:

1. Go to the Broken Auth Section
2. Using a python script call "bruteforce.py"
3. I ran the following cmd.
 - a. python bruteforce.py -U test-username.txt -P test-password.txt -f False
<http://0.0.0.0:3000/brokenauth/>
4. After a min of running it was a find a working username and password combination that allowed me to login.

```
python bruteforce.py -U test-username.txt -P test-password.txt -f False http://0.0.0.0:3000/brokenauth/login2  
[+] Login Found! {'username': 'user', 'password': 'dragon'}  
This is a demo code used for this training.
```

Recommendations:

Recommendation on either how to fix it or details on the best practice.