TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Emily Joy Krohn 185198IADB

User Authentication Method for Distributed Networking Systems with Short Message Peer-to-Peer Protocol

Bachelor's thesis

Supervisor: Nadežda Furs
MBA

Tallinn 2021

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Emily Joy Krohn 185198IADB

Isiku tuvastamise meetod hajutatud võrgusüsteemideks Short Message Peer-to-Peer protokolliga

Bakalaureusetöö

Juhendaja: Nadežda Furs
MBA

Tallinn 2021

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Emily Joy Krohn

16.05.2021

Abstract

The aim of the current thesis is to create an authentication method for authenticating Short Message Peer-to-Peer sessions, thus ensuring security. The method is an interface to existing Short Message Peer-to-Peer systems, which allows the bind request handler to send credentials and receive a positive or negative response.

During the study, the existing problem and solutions to it in different companies across the globe will be thoroughly analysed.

During the development a module will be created, which handles users' data, compares it against existing data, and then allows or forbids access. The development has been split into four two-weeks parts, which handle stated tasks.

The emphasis on the study is agile planning and implementing requirements set by clients. The result of the development is a working module with an authentication method.

All figures in the study are created by the author.

This thesis is written in English and is 42 pages long, including 8 chapters, 37 figures and 0 tables.

Annotatsioon

Isiku tuvastamise meetod hajutatud võrgusüsteemideks Short Message Peer-to-Peer protokolliga

Käesoleva bakalaureusetöö eesmärgiks on luua autenteerimise meetod Short Message Peer-to-Peer sessioonide isiku tuvastamiseks, seejuures tagades turvalisust. Meetod on liideseks olemasolevatele Short Message Peer-to-Peer süsteemidele, mis võimaldab sidumise päringu käitlejal saata sisselogimisandmeid ning vastu saada positiivse või negatiivse vastuse.

Töö käigus analüüsitakse põhjalikult olemasolevat probleemi ning selle seniseid lahendusi erinevates firmades üle maailma. Samuti vaadeldakse hetkesitutuatsiooni firmas, kus autor töötab.

Arendusprotsessi käigus luuakse moodul, mis käsitleb kasutajate andmeid, võrdleb neid olemasolevate andmete vastu ning vastavalt võrdlusele lubab või keelab kasutajatel sessioone avada. Planeerimise käigus selgitatakse nõudeid ning grupeeritakse nõudeid ülesanneteks. Arendus on jagatud nelja kahenädalasse ossa, mis käsitlevad määratud ülesandeid, mille kaudu ehitatakse moodul tüki haaval.

Töös pannakse rõhku töö agiilsele planeerimisele ning nõuete implementeerimisele, mis on seatud klientide poolt. Samuti käsitleb töö testimist antud nõuete järgi ning vaadeldakse edasiseid arendusvõimalusi.

Arendusprotsessi tulemuseks on moodul töötava isiku tuvastamise meetodiga.

Kõik joonised antud töös on autori koostatud.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 42 leheküljel, 8 peatükki, 37 joonist, 0 tabelit.

List of abbreviations and terms

ASCII	American Standard Code for Information Interchange
API	Application Programming Interface
CIDR	Classless Inter-Domain Routing
CPaaS	Communications Platform as a Service
ESME	External Short Messaging Entity
GDPR	General Data Protection Regulation
IAM	Identity and Access Management
ID	Identifier
IP	Internet Protocol
MO	Model Object
PDU	Protocol Data Unit
PII	Personally Identifiable Information
RE	Routing Entity
SMPP	Short Message Peer-to-Peer
SMS	Short Message Service
SMSC	Short Messaging Service Center
SSL	Secure Sockets Layer
TLS	Transfer Layer Security

Table of contents

List of figures

1 Introduction

The SMPP protocol is being used as an industry standard for sending and receiving short message data between ESMEs, REs and SMSC. The use for the SMPP protocol is almost limitless, not only for SMS, but also for a variety of other messaging applications including messaging APIs.

With SMS gaining more popularity than ever in the form of marketing texts, booking confirmations and two-factor authentication, it is important that users would not receive unwanted messages from fraudulent sources posing as official service providers and therefore potentially become victims of phishing or clicking on malicious links. Specification version 3.4 of the SMPP protocol is inherently insecure by design, lacking a secure way for user authentication as an out-of-the-box feature, therefore making it easy for ill-natured actors to perform an attack on vulnerable services. The limit for a username is 16 characters and the limit for a password is 8 ASCII characters.

The given thesis will analyse the security issue the SMPP protocol enables and the current technical situation at the company the author works for. The offered solution to the problem is a module within the proxy server, which accepts decoded data from bind PDUs and checks the given system ID, password, and IP address against existing user data.

The author of the thesis works for a CPaaS company which offers SMS support for users, thus supporting the SMPP protocol. The author is also a member of a team, which is focused on improving and maintaining the SMPP side of SMS and recognizes the lack of security in the protocol to be a reason for customers to be wary of SMS. The author concludes the current problem can be solved by adding an extra layer of security in the form of user authentication.

1.1 Methodology

Firstly, the given thesis will describe the existing problem, the current technical situation in the CPaaS company the author works for and existing solutions throughout other

companies will be analysed, then a scope for the study will be set. Based on the available implementations, functional and non-functional requirements for the new system will be set.

It will be determined in which environment the new system will work in and which kind of suitable IT software will be developed. The choice of technologies will be analysed and justified based on their compatibility for the given requirements. A prototype will be developed during the work, which will also be tested based on functional requirements.

Tasks will be described, and sprints containing the aforementioned tasks will be planned and implemented. Lastly, future developments and additions that fell out of the thesis scope will be mentioned.

2 Overview of the problem

Protocols are a necessity in digital communication, as they determine how data must be transmitted, formatted, and sent. In a broader view, they can be viewed as languages. If two people speak the same language, they are able to communicate easily and effectively [1]. The same also applies to computer programmes, as a set of strict rules allows them to be written in such a way that data sent between them will hold a constant format and not change over time.

The SMPP protocol is an industry standard, open protocol which provides flexible data communications interface for handover of short message data between a Message Center and an SMS application [2, p. 8].

In order to send and receive data, an SMPP session needs to be initiated and established. This is done by the ESME establishing a connection with the SMSC and then issuing an SMPP Bind request. [2, p. 14] There are three different types of bind requests an ESME is able to send – *bind_transmitter*, *bind_receiver* and *bind_transceiver*. A transmitter is able to send short messages and receive the corresponding response PDU from the SMSC, a receiver is able to receive short messages and send the corresponding response PDU to the SMSC and a transceiver is able to send and receive both short messages and responses [2, p. 45].

The differences between the PDUs sent with each type of bind request are minimal – only the *command_id* in the PDU header of each request is set to the corresponding value of the specific request. Each bind request PDU describes the option to add a *system_id*, *password,* and *address_range* to the body of the request. Use of a *system_id* is recommended, while the use of *password* and *address_range* is implementation specific and can be set to NULL to establish an unsafe connection if the SMSC allows it [2, pp. 45-53].

2.1 SMPP protocol vulnerabilities

The SMPP protocol is intended as an application layer protocol and relies heavily on the underlying connection to provide secure data transfer, including error handling, packet encoding, windowing, and flow control. At application layer, SMPP does not define security and therefore suffers from multiple vulnerabilities [3, pp. 51-52].

1. Zero confidentiality – no encryption standard is described in the protocol and messages travelling between the ESME and SMSC are sent in plain text. It relies heavily on the underlying transport layer. Therefore, tools such as Wireshark [4] can be used by malicious actors to obtain information such as *system_id* and *password* [3, p. 52].
2. Weak Endpoint Authentication – attackers are easily able to obtain login credentials and authenticate themselves as genuine users, thus misusing the messaging service [3, p. 53]. In this case, fraudulent actors can appear as trusted sources and send deceitful links and begin phishing attacks on unsuspecting victims.
3. Man-in-the-Middle Attack – attackers are able to make independent connections with the victims and transfer messages, making the victim believe they have obtained a secure connection. In reality, all messages are orchestrated by the attacker [3, p. 51]. This kind of attack can be used for theft of information such as business and government secrets.
4. Message Tampering – as attackers are able to obtain credentials, authenticate themselves and pose as middlemen, they are able to adulterate messages before they arrive to the receiver [3, p. 51]. The protocol does not contain any kind of control value to compare against, therefore a customer would not be able to verify the originality of the incoming message.

Therefore, it is a necessity for software using the SMPP protocol to be implemented over a secure protocol such as TLS, where the client and server have access to a secret key with which they are able to send encrypted data. The authentication of the SMSC by the ESME is achieved through digital certificates provided during the handshake of TLS. In addition to this, the author has found it necessary to add one extra layer of security to assure all SMPP sessions are candid by implementing bind authentication.

2.2 Existing solutions

As the SMPP protocol is one of a kind in the industry, all companies who send and receive SMSs are bound to the protocol either directly or indirectly.

2.2.1 Current technological situation

The solution currently offered by the company the author works at follows the best practises for user authentication as proposed by and described in the protocol specification. The user's customer ID is checked against existing users and the user is billed according to use of the service. Up until recently, a large interest has not been taken towards the SMPP service, but with more customers showing curiosity towards the service, the company has taken the initiative to modernize how authentication within SMPP is implemented and take extra steps, therefore assuring clients the service they are using is trustworthy and safe.

2.2.2 Existing solutions in other companies

Many companies across the globe offer SMPP support, such as sms.to [5], Melrose Labs [6], GatewayAPI [7] and Elisa [8]. While all companies claim to offer secure delivery of messages, the descriptions of how this security is obtained is fairly vague. Melrose offers secure SMPP over TLS and mentions IP allow-listing [9]. Elisa has no notice of using TLS, but the implementation guide states that *system_id* and *password* are both necessary while sending a bind request [10], which could indicate they are using bind authentication. GatewayAPI's SMPP specific blogpost [11] and documentation [12] does not specify security, but their main page mentions GDPR compliance and secure connections over tough SSL and leading security measures [7]. Sms.to offers security by investing in security and certifications and are willing to offer transparency on products purchased from them [13].

2.3 Scope of the study

Faults in existing solutions within other companies are difficult to address as the technical implementations are not available to the public. As previously noticed, one can assume other companies use security layers such as TLS, bind authentication or a combination of both.

Therefore, the proposed solution to the problem is to implement an authentication module in the existing SMPP service, which can be used during the handling of bind requests.

The solution will be limited to building, implementing, and testing the bind authentication module and will leave opportunity for the authentication module to become a separate microservice in the future. Implementing a secure TLS connection, decoding bind requests, and handling them will fall outside of the study's scope, as these services already exist in the SMPP service today.

The goal of the solution is to offer customers of the service a greater feeling of security when sending and receiving messages so they can be sure no attacks can be performed on their data. This can also help avoid potential reputation damage to respectable businesses and public relations crises if data were to be compromised.

3 Analysis of the proposed solution

To properly analyse the proposed solution, requirements need to be set and the type of software needs to be determined.

3.1 Determining requirements

Determining requirements has been based on customers' requirements to the company the author works for and general code standard and maintainability has also been taken into account. Determined requirements will then be grouped into tasks, which can be managed in sprints.

3.1.1 Functional requirements

Functional requirements describe functions or features of the product which need to be implemented in order for a user to complete the task they are using the software for. Functional requirements generally describe the behaviour of the system under specific conditions [14].

Functional requirements for the proposed solution:

- The authentication module shall check if a *system_id* exists
- The authentication module shall check if there is a specified IP allow-list
- The authentication module shall check if the given IP is allowed to bind
- The authentication module shall check if the given password matches the stored password in the database
- If all checks pass, the authentication module shall respond with a custom successful response

- If any check fails, the authentication module shall respond with a custom unsuccessful response and update metrics corresponding to the unsuccessful attempts

3.1.2 Non-functional requirements

Non-functional requirements state how the given system should behave and demonstrate limits of its functionality. These kinds of requirements can also be defined as the given system's quality attributes [14].

Non-functional requirements for the proposed solution:

- A new authentication module is created that will be in charge of authentication and identity lookup
- The authentication module is initialized in the Dropwizard [15] application during start-up
- Models are created for communication between the SMPP server module and the authentication module
- Interfaces are created for the API between the SMPP server module and the authentication module
- A database table has been set up and created for storage of authentication credentials
- Credentials are populated into the database table
- Sensitive credentials are hashed in the database table using bcrypt
- Credentials are read into memory and made available to the authentication module through an authentication item
- The authentication logs must not contain PII

- The authentication module shall have different error codes per error that can happen
- The error codes shall be clear and understandable

3.2 Determining the type of software

The existing SMPP service in the company the author of the thesis works for is written in Java and uses Maven [16] for software management and comprehension. It contains many different microservices, one of them being a proxy service, which handles setting up the server, accepting incoming requests, decoding the requests, and sending them further down the pipeline, accepting responses from further down the pipeline and sending them back to the customer.

Java was deemed the best suitable language for the SMPP project due to its server capabilities and extensive choice of highly functional SMPP libraries, such as jSMPP [17] and OpenSMPP [18]. Maven had been selected mostly due to its wide variety of build life-cycle steps and its ability to integrate seamlessly with third-party tools.

The proposed solution will therefore be a separate Maven module in the proxy microservice, which will be able to communicate with the existing server module. It will be able to be called by the handler accepting bind requests in order to authenticate clients. The module will consist of an interface and the implementation of aforementioned interface for straightforward use in the bind request handler. The authentication module will have an authentication method (Figure 1) which will handle the incoming information and respond positively or negatively based on the provided credentials.

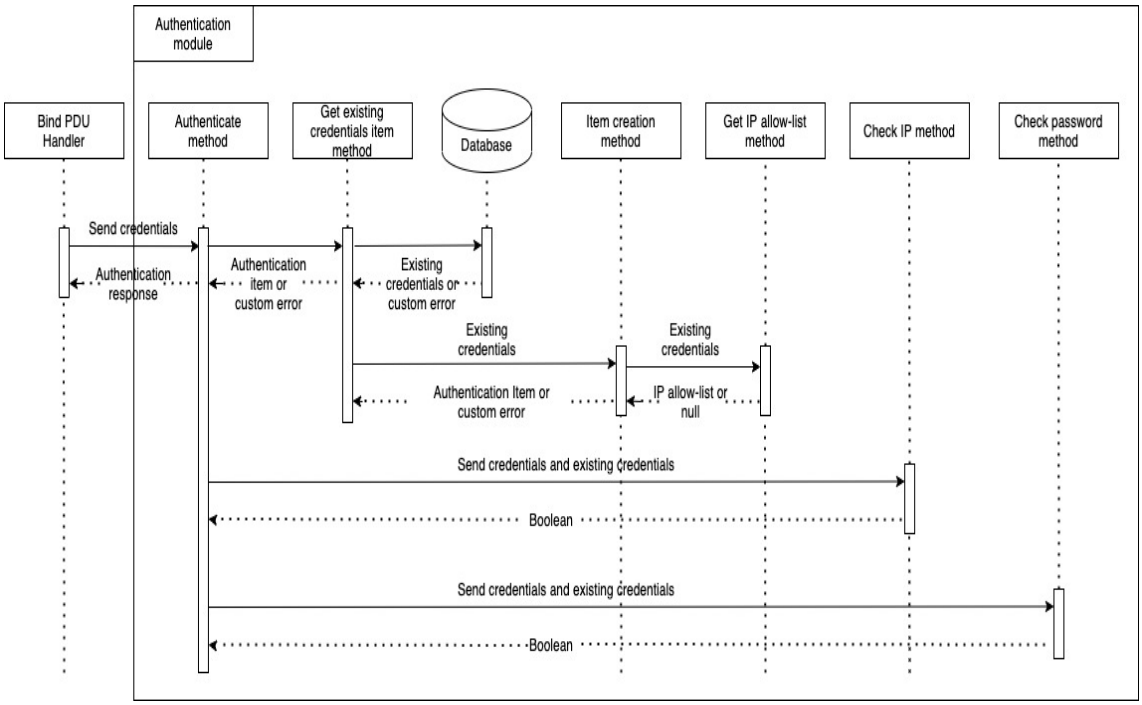

Figure 1. Proposed solution dataflow diagram

The bind PDU handler will be exposed to the authentication response (positive or negative) and based on that, send a bind response, or refuse to open a connection without exposing the customer to a specific reason. This way attempts of brute-force attacks will be made more complex for malicious actors.

4 Technical implementation planning

Planning the technical implementation is crucial for any kind of project. This way, one is able to fully understand what is required of them and how large the scope of the work is.

4.1 Creating issues

The author works in an agile team, therefore having epics containing available issues and planning the prognosed time to work on them for sprints is crucial. Tasks were created in Jira [19] based on the requirements for the new solution and an epic was created and populated with the issues.

4.1.1 Create the authentication module

The first issue associated with the authentication module epic was to create the authentication Maven module. The description states the authentication module is created and initialized during service start-up in the server module.

Acceptance criteria:

- A new authentication module is created that will oversee authenticating clients and identity lookup
- The identity module is initialized in the Dropwizard application during start-up. For this, an AuthenticationModuleBundle is created and initialized during service start-up

Next, the issue was groomed using the planning poker method. The method is used for release planning to get an approximate estimate of the effort to build the project, therefore determining the project scope [20]. The team the author works in determined this task to have an estimate value of 2 story points.

4.1.2 Create models for calls and responses

The second issue in the epic was to create MOs in the protocol module (used for models all modules can have access to) authentication requests and positive and negative authentication responses. The description states MOs (authentication request, authentication response (successful response or error response)) are created for information transport between the server and the authentication module.

Acceptance criteria:

- MOs are created for communication between the SMPP server module and the authentication module
- An authentication request model is created – used to authenticate a session. Suggested fields are *system_id*, *password*, and *IP*
- A successful authentication response model is created – used for sending the data wanted from the authentication request. Suggested fields are *system_id*, *customer_id* and *session_id* (auto generated)
- An unsuccessful authentication model is created – used to notify the service why a request failed. Suggested fields are *error_code* and *error_message*
- The MOs are created in the protocol module

The issue was groomed by the team and valued at an estimate of 2 story points.

4.1.3 Define the interface for authentication

The next task's description states interfaces are created for communication between the server and authentication modules. The interfaces should be written in such a way that it will be easy to split the service in the future.

Acceptance criteria:

- Interfaces and necessary methods are created and defined for authentication between the server module and the authentication module
- Interfaces exist both in the server and authentication module to ease a potential future split of the project
- Models defined in "Create models for calls and responses" for request and response are used

The team groomed the issue and estimated it at a value of 1 story point.

4.1.4 Credential storage

The description for this task states there should be support for storing credentials and actor identity in a data store. The data to be stored is *system_id*, bcrypt hashed *password*, *customer_id* and *ip_allow_list*. The credentials should be accessible by-code via the authentication module.

Acceptance criteria:

- A table in a data store is setup and used for SMPP credentials
  - A table is created using *system_id* as the key
  - The data model for the table is defined
- Credentials are populated
- Sensitive credentials are hashed using bcrypt
- The credentials are read into memory and made available to the authentication module

The task was groomed, and the team estimated it to be valued at 5 story points.

4.1.5 Implement session authentication

The final task associated with the epic was to implement session authentication. The description states the authentication flow is implemented using the authentication vectors available in the authentication model (“Create models for calls and responses”) over the authentication interface (“Define the interface for authentication”).

Acceptance criteria:

- The request interface (“Define the interface for authentication”) is implemented
- Credential lookups are done using supplied information
  - A check if the *system_id* is valid exists
  - A check if the IP is allowed to authenticate exists
  - A check if the hashed password is valid exists
- A successful request should send back a Successful Authentication Response model (“Create models for calls and responses”)

- An unsuccessful request should send back an Unsuccessful Response model (“Create models for calls and responses”)
    - Different error codes for all failures exist
    - Error messages are clear and understandable
- Metrics are implemented for both successful and unsuccessful authentication attempts

During grooming, this issue was estimated to have a value of 5 story points.

4.2 Sprint planning

15 story points are in total after adding the points together of all the issues in the epic. With the team’s velocity of slightly short of 5 story points per sprint per engineer, the issues have been divided over 4 sprints (Figure 2) with additional time in the last sprint to iron out any possible issues.

Figure 2. Sprint planning

4.2.1 First sprint planning

For the first 2-week effort, issues “Create the authentication module”, “Create models for calls and responses” and “Define the interface for authentication” were taken into consideration. It is expected for the first two aforementioned issues to be completed by the end of the sprint and work started on the third. Therefore, the sprint goal was worded as “The authentication module is created and intialized with models ready for calls and responses”.

4.2.2 Second sprint planning

“Define the interface for authentication” and “Credential storage” were added to the second sprint in hopes of the first issue being completed by the middle of the first week and the majority of the second issue completed by the end of the sprint. For these 2 weeks, the sprint goal was defined as “Interfaces are defined for communication between modules and a data store is up and running”.

4.2.3 Third sprint planning

The third sprint consisted of completing “Credential storage” within the first week and starting work on “Implement session authentication” in the remaining week. The sprint goal for the third fortnight was therefore described as “Credentials are available to the authentication module and initial authentication is possible”.

4.2.4 Fourth sprint planning

“Implement session authentication” was the only issue to be added to the fourth sprint, leaving a buffer zone for any unexpected problems to be ironed out. The sprint goal was worded as “Bind requests are able to be authenticated”.

Testing the authentication module’s functionality was to be implemented as part of the final issue “Implement session authentication”.

5 Technical implementation

The technical implementation of the proposed solution had been divided into two-week sprints, in which the author was able to handle the tasks according to planning.

5.1 Sprint 1

Sprint goal was “The authentication module is created and intialized with models ready for calls and responses”.

The first sprint began with the author getting acquainted with Maven, how modules work and how the *pom.xml* file is structured.

5.1.1 Create the authentication module implementation

The author decided to use IntelliJ IDEA [21] to work on the implementation, as it is a standard tool in the company they work for, they have prior experience with the software and IntelliJ IDEA offers a fully functional integration with Maven, which eases the automation of the building process [22].

Firstly, the Maven module was created using the IntelliJ IDEA new module wizard (Figure 3). The module was named authentication and the parent of the module was marked as the smpp-proxy service.

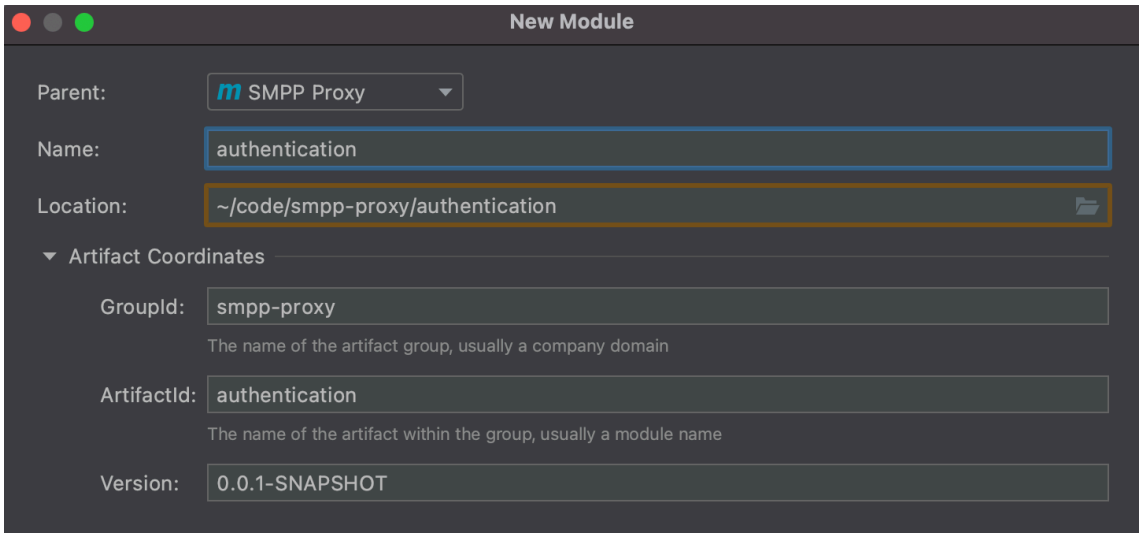

Figure 3. New Maven module creation in IntelliJ IDEA

The *pom.xml* file was therefore autogenerated (Figure 4) and an empty Maven module with the correct layout was created. The proper structure contains both a main and a test package, which both hold a java package, under which new packages and classes can be created [23].

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
       <artifactId>smpp-proxy</artifactId>
       <groupId>smpp-proxy</groupId>
       <version>0.0.1-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>
    <artifactId>smpp-proxy-authentication</artifactId>
    <name>Authentication</name>
</project>
```

Figure 4. Authentication module pom.xml example

The module was then added as a dependency and module to the parent (smpp-proxy) *pom.xml* (Figure 5).

```
<dependency>
    <artifactId>smpp-proxy-authentication</artifactId>
    <groupId>smpp-proxy</groupId>
    <version>${project.version}</version>
</dependency>

<modules>
    <module>identity</module>
</modules>
```

Figure 5. Adding the authentication module to the parent pom as a dependency and module

Next, a basic skeleton of the authentication module was created – an empty interface (Figure 6) and an empty class implementing the interface (Figure 7).

```
public interface AuthenticationServer {
}
```

Figure 6. Initial authentication module interface

```
public class AuthenticationServerImpl implements AuthenticationServer {
}
```

Figure 7. Initial authentication module interface implementation

It was then decided that initialization of the authentication module could only happen when there was an authentication configuration class, which would contain information about the datastore. Therefore, the ticket was closed and marked as done with a comment that initialization of the authentication module would be added to the "Credential storage" issue.

5.1.2 Create models for calls and responses implementation

For the creation of MOs, the author looked into using the Immutables library [24]. This way, the internal state of the MOs will always be constant throughout the entire project and the Immutables library also guarantees thread-safety while sharing the models [25] because the library generates an immutable version of the type with the *@Immutable* annotation [26].

Although Project Lombok [27] is generally the more popular as an annotation processor [28], it lacks in ease to create immutability and can be misused to create objects with invalid states [29]. Project Lombok is not intended as an immutability library and can add unnecessary risk.

The Immutables library was added to both *pom.xml* files of the parent and of the protocol module, as the MOs were to be created there as to ease communication between the server and authentication modules. The author created the *AuthenticationRequest* (Figure 8), *AuthenticationResponse* (Figure 9) and *UnsuccessfulResponse* (Figure 10) modules with the suggested fields named in the "Create models for calls and responses implementation" ticket.

```
@Immutable
public abstract class AuthenticationRequest {
    abstract String systemId();
    abstract String password();
    abstract String ip();
}
```

Figure 8. AuthenticationRequest

```
@Immutable
public abstract class AuthenticationResponse {
    abstract String systemId();
    abstract String sessionId();
    abstract String customerId();
}
```

Figure 9. AuthenticationResponse

```
@Immutable
public abstract class UnsuccessfulResponse {
    abstract String errorCode();
    abstract String errorMessage();
}
```

Figure 10. UnsuccessfulResponse

For use of the immutable classes, a Maven build step was to be added to the protocol module's *pom.xml* file, which enables immutable classes to be auto generated in the target folder (Figure 11).

```
<build>
    <plugins>
        <plugin>
            <groupId>org.codehaus.mojo</groupId>
            <artifactId>build-helper-maven-plugin</artifactId>
            <version>3.2.0</version>
            <executions>
                <execution>
                    <id>add-immutables-generated-sources</id>
                    <phase>generate-sources</phase>
                    <goals>
                        <goal>add-source</goal>
                    </goals>
                    <configuration>
                        <sources>
                            <source>target/generated-
sources/annotations</source>
                        </sources>
                    </configuration>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

Figure 11. Adding Maven build step to auto generate immutable classes

5.1.3 Define the interface for authentication implementation

For defining the interfaces as stated in the issue, the author looked into vavr's Either interface, which represents a value of two possible types, a right and a left value, where the right value is the success and the left value the failure case [30].

While initially it was planned to have an interface and implementation in both the authentication and server modules, implementing them this way became problematic in the form of circular dependencies. Therefore it was decided to move both interfaces to the protocol module (accessible by all modules) and have an implementation of the interfaces on either side.

AuthenticationServer (Figure 12) was created and implemented on the authentication module side by *AuthenticationServerImpl* (Figure 13), while *AuthenticationClient* (Figure 14) was created and implemented on the server module side by *AuthenticationClientImpl* (Figure 15).

```
public interface AuthenticationServer {
    Either<UnsuccessfulResponse, AuthenticationResponse> authenticate(final
AuthenticationRequest authenticationRequest);
}
```

Figure 12. AuthenticationServer interface

```
public class AuthenticationServerImpl implements AuthenticationServer {
    @Builder
    public AuthenticationServerImpl() {
    }

    public Either<UnsuccessfulResponse, AuthenticationResponse>
authenticate(final AuthenticationRequest authenticationRequest) {
    }
}
```

Figure 13. AuthenticationServer implementation

```
public interface AuthenticationClient {
    Either<UnsuccessfulResponse, AuthenticationResponse> authenticate(final
String systemId,
            final String passwordHash,
            final String remoteIp);
}
```

Figure 14. AuthenticationClient interface

As the "Create the authentication module" issue was reduced in workload, the author was able to complete the first 3 issues and bring in "Credential storage" to the sprint.

5.1.4 Credential storage initial implementation

For deciding on a data store, the key requirements were speed, availability, scalability, and maintainability. This led the author to look into non-relational type NoSQL databases such as MongoDB and DynamoDB. DynamoDB is fully managed solution with out of the box security and supports key-value queries, while MongoDB supports more complex

queries, supports mutable indexes, and offers a wider selection of data types and size [31]. DynamoDB also offers single-digit millisecond performance at any scale [32]. Taking into consideration the author's use-case, it does not require many different types of data and a single entry should not contain a large amount of data. Therefore, the decision was made to move forward with DynamoDB.

An identity MO (Figure 15) was created in the authentication module to represent the data to be stored in DynamoDB. For client comfort and flexibility, it was decided an IP allow-list is not mandatory, although reduces security. This way, any IP address is able to establish an SMPP session with the correct *system_id* and *password*. The name of the DynamoDB table was chosen to be *smpp.identity*. In the datastore *systemId*, *passwordHash* and *customerId* are stored as Strings, while the *ipAllowList* is a String of comma separated IP addresses in the CIDR notation.

```
@Immutable
public interface Identity {
    String systemId();
    String passwordHash();
    String customerId();
    Optional<Set<CIDR>> ipAllowList();
}
```

Figure 15. Identity MO representing database data

5.1.5 Sprint 1 conclusion

During the retrospective, with the reduced workload of the first issue, the sprint goal of "The authentication module is created and initialized with models ready for calls and responses" was marked as not entirely met, as the authentication module was not able to be set up due to lack of configuration. On the other hand, all issues accepted into the sprint were accomplished with work on an additional ticket started. The goal for sprint 2 was re-evaluated and stated as "Credentials are available to the authentication module and the authentication module is initialized".

5.2 Sprint 2

The initial sprint goal was “Interfaces are defined for communication between modules and a data store is up and running” but was later reworded as “Credentials are available to the authentication module and the authentication module is initialized”.

5.2.1 Credential storage final implementation

Having previously identified the data store and decided on a data model, configuration for DynamoDB and therefore the authentication module was implemented (Figure 16).

```
@Immutable
public abstract class IdentityDynamoDBConfiguration {
    public abstract int retries();
    public abstract boolean local();
    public abstract String endpoint();
    public abstract String region();
    public abstract String tableName();
}
```

Figure 16. DynamoDB configuration

To initialize the authentication module in the server, it was found to be more straightforward to add an authentication client to a channel handler (Figure 17), thus avoiding creating a separate bundle for the module. Configuration for the SMPP proxy microservice is given as a parameter to the method, from which the author was able to extract the identity configuration.

```
private static ServerBootstrap createServerBootstrap(final
SmppProxyConfiguration smppProxyConfiguration) {
    final var identityConfig = createIdentityConfig(smppProxyConfiguration);
    final var identityClient = createIdentityClient(identityConfig);
    final SmppChannelHandler smppChannelHandler =
createSmppChannelHandler(identityClient);
    return new ServerBootstrap().childHandler(smppChannelHandler);
}
```

Figure 17. Adding the authentication client to the server

The *DynamoDBUtils* class was created to handle the setup of the DynamoDB client, which contains methods to get the client and endpoint configurations, region, and number of retries allowed.

Thus, the author was able to add a DynamoDB client to the *AuthenticationServerImpl* (Figure 18). The identity configuration is added as a parameter, from which the DynamoDB configuration can be read.

```
@Builder
public AuthenticationServerImpl(final @NonNull IdentityConfiguration config)
{
    this.dynamoDBClient = createDynamoDBClient(config);
    this.tableName = config.dynamoDBConfiguration().tableName();
}

private static DynamoDB createDynamoDBClient(final IdentityConfiguration
identityConfiguration) {
    final var clientBuilder = AmazonDynamoDBClientBuilder.standard()
.withClientConfiguration(DynamoDBUtils.getClientConfiguration(identityConfigu
ration));
    if (DynamoDBUtils.isLocal(identityConfiguration)) {
clientBuilder.setEndpointConfiguration(DynamoDBUtils.getEndpointConfiguration
(identityConfiguration));
    } else {
clientBuilder.setRegion(DynamoDBUtils.getRegion(identityConfiguration));
    }
    return new DynamoDB(clientBuilder.build());
}
```

Figure 18. Creating DynamoDB client in AuthenticationServerImpl

While implementing the method to get data from DynamoDB, the author found using the *UnsuccessfulResponse* model to cause many repetitions and text-based code, making it prone to mistakes. The author then implemented an *SmppError* enum to carry the error messages and codes associated to them (Figure 19).

```
public enum SmppError {
    SMPP_3001("SMPP-3001", SmppError.AUTHENTICATION, "system_id does not
exist"),
    SMPP_3002("SMPP-3002", SmppError.AUTHENTICATION, "ip address does not
match with ip-allow-list"),
    SMPP_3003("SMPP-3003", SmppError.AUTHENTICATION, "invalid password"),
    SMPP_3004("SMPP-3004", SmppError.AUTHENTICATION, "unable to connect to
identity datastore"),
    SMPP_3005("SMPP-3005", SmppError.AUTHENTICATION, "necessary credentials
are missing");
    private static final String AUTHENTICATION = "authentication";
}
```

Figure 19. SmppError enum

Therefore the *UnsuccessfulResponse* model was updated to contain an *SmppError*.

The method to get credentials from DynamoDB and translate them into an *Identity* model (Figure 20) also covers the first authentication check, as the data is stored with the *system_id* as a key, if an entry with such a key does not exist, the provided *system_id* will be unable to bind. The method *fromDynamoDb* initially returned an empty optional.

```
Either<UnsuccessfulResponse, Identity> getCredentials(final String systemId)
{
    try {
        final var spec = new
GetItemSpec().withPrimaryKey(SYSTEM_ID_ATTRIBUTE, systemId);
        final var item =
this.dynamoDBClient.getTable(this.tableName).getItem(spec);
        if (item != null) {
            final var fromDynamo = fromDynamoDb(item);
            if (fromDynamo.isPresent()) {
                return Either.right(fromDynamo.get());
            } else {
                final var response =
ImmutableUnsuccessfulResponse.builder().error(SMPP_3005).build();
                return Either.left(response);
            }
        }
        final var response =
ImmutableUnsuccessfulResponse.builder().error(SMPP_3001).build();
        return Either.left(response);
    } catch (final RuntimeException e) {
        final var response =
ImmutableUnsuccessfulResponse.builder().error(SMPP_3004).build();
        return Either.left(response);
    }
}
```

Figure 20. Get credentials from DynamoDB

5.2.2 Sprint 2 conclusion

During the retrospective it was noted that the second sprint ended having implemented more than expected with work on “Credential storage” complete. Therefore, the sprint goal of “Credentials are available to the authentication module and the authentication module is initialized” was completed successfully. The third sprint’s goal was thus reworded as “Initial authentication is possible”.

5.3 Sprint 3

The initial sprint goal was “Credentials are available to the authentication module and initial authentication is possible” and with work done was later described as “Initial authentication is possible”.

5.3.1 Implement session authentication initial implementation

Having the first authentication check of *system_id* implemented in the "Credential storage" issue, IP addresses and passwords were yet to be checked. To get the credentials in a readable state to the authentication module, the *fromDynamoDb* method was implemented (Figure 21). In this method, if either *customerId* or *passwordHash* are null, the client will not be allowed to authenticate, as necessary credentials are missing. The parameter included in the method is an Amazon Web Services DynamoDB Item, which is a collection of attributes, each having a name and value [33].

```
static Optional<Identity> fromDynamoDb(final Item item) {
    final var systemId = item.getString(SYSTEM_ID_ATTRIBUTE);
    final var customerId = item.getString(CUSTOMER_ID_ATTRIBUTE);
    final var passwordHash = item.getString(PASSWORD_HASH_ATTRIBUTE);
    if ((null == customerId) || (null == passwordHash)) {
        return Optional.empty();
    }
    final var builder =
ImmutableIdentity.builder().systemId(systemId).customerId(customerId)
    .passwordHash(passwordHash);
    getIpAllowList(item).ifPresent(builder::ipAllowList);
    return Optional.of(builder.build());
}
```

Figure 21. fromDynamoDb method

In order to get a correctly formatted set of CIDR notated IP addresses, the method *getIpAllowList* was created and implemented (Figure 22). The parameter used in the method is an Amazon Web Services DynamoDB Item.

```
static Optional<Set<CIDR>> getIpAllowList(final Item item) {
    final var cidrString = item.getString(IP_ALLOW_LIST_ATTRIBUTE);
    if (null == cidrString) {
        return Optional.empty();
    }
    final var cidrSet = Arrays.asList(cidrString.split(","));
    final Set<CIDR> ipAllowList = new HashSet<>();
    cidrSet.forEach(rawCidr -> {
        ipAllowList.add(CIDR.newCIDR(rawCidr));
    }
    return Optional.of(ipAllowList);
}
```

Figure 22. Initial getIpAllowList method

For this, the Netty [34] framework was added to the *pom.xml* of the authentication module (Figure 23).

```
<dependency>
    <groupId>io.netty</groupId>
    <artifactId>netty</artifactId>
</dependency>
```

Figure 23. Netty framework in authentication module pom.xml

To check whether the given IP address would be allowed to authenticate, the *checkIpAddress* method was implemented (Figure 24). Here it can be seen a user is allowed to authenticate if they have not previously specified an IP allow-list. The two parameters in the given method are the identity object (created from the DynamoDB Item) and the authentication request sent by the client.

```
static boolean checkIpAddress(final Identity identity, final
AuthenticationRequest authenticationRequest) {
    final var ipAllowList = identity.ipAllowList();
    if (ipAllowList.isEmpty()) {
        return true;
    }
    final InetAddress requestIp;
    try {
        requestIp = CIDR.newCIDR(authenticationRequest.ip()).getEndAddress();
    } catch (final UnknownHostException e) {
        return false;
    }
    return ipAllowList.get().stream().anyMatch(cidr ->
cidr.contains(requestIp));
}
```

Figure 24. Initial checkIpAllowList method

The final credential to check was the password – the password stored in the data store had been encrypted using bcrypt, therefore the *checkPassword* method (Figure 25) had to have the ability to check the provided hash against the stored hash. To imitate the stored data in tests, the chosen library would have to be able to encrypt passwords effortlessly. For this use, mindrot.org's jBCrypt [35] was taken into use due to its methods such as *genSalt*, *checkpw* and *hashpw*.

```
private static boolean checkPassword(final Identity identity, final
AuthenticationRequest authenticationRequest) {
    final var passwordHash = identity.passwordHash();
    return BCrypt.checkpw(authenticationRequest.password(), passwordHash);
}
```

Figure 25. checkPassword method

With all the necessary checks implemented, the author was able to use them in the *authenticate* method (Figure 26), which sends back either an *AuthenticationResponse* or *UnsuccesfulResponse* to the bind PDU handler.

```
public Either<UnsuccessfulResponse, AuthenticationResponse>
authenticate(final AuthenticationRequest authenticationRequest) {
    final var identity =
this.getCredentials(authenticationRequest.systemId());
    if (identity.isLeft()) {
        return Either.left(identity.getLeft());
    }
    if (!checkIpAddress(identity.get(), authenticationRequest)) {
        final var response =
ImmutableUnsuccessfulResponse.builder().error(SMPP_3002).build();
        return Either.left(response);
    }
    if (!checkPassword(identity.get(), authenticationRequest)) {
        final var response =
ImmutableUnsuccessfulResponse.builder().error(SMPP_3003).build();
        return Either.left(response);
    }
    final var response = ImmutableAuthenticationResponse.builder()
            .systemId(identity.get().systemId())
            .sessionId(UUID.randomUUID().toString())
            .customerId(identity.get().customerId())
            .build();
    return Either.right(response);
}
```

Figure 26. authenticate method

5.3.2 Sprint 3 conclusion

With an implementation of authentication in place, the sprint was completed successfully, and during the retrospective it was found that the sprint goal had been met. The "Implement session authentication" ticket however was not closed yet, as implementing metrics and testing the solution had not been worked on yet.

5.4 Sprint 4

The sprint goal according to planning was “Bind requests are able to be authenticated”.

5.4.1 Implement session authentication final implementation

Adding metrics was the final acceptance criteria for the task. Using Micrometer Application Monitoring [36] for this is a standard in the company the author works for. The author added a counter for successful authentication attempts and a map of counters for unsuccessful attempts. For creating, adding, and incrementing the error counters, 2 methods were created in the *AuthenticationServerImpl – incrementErrorCounter* and *createErrorCounter* (Figure 27). The counters were added to the authenticate method for each different failure and success.

```
void incrementErrorCounter(final SmppError error) {
    if (this.errorCounterMap.containsKey(error)) {
        this.errorCounterMap.get(error).increment();
    } else {
        final var counter = createErrorCounter(error);
        this.errorCounterMap.put(error, counter);
        counter.increment();
    }
}

private static Counter createErrorCounter(final SmppError error) {
    return Counter.builder(AUTHENTICATION_CALLS)
            .tag(STATUS, UNSUCCESSFUL)
            .tag(ERROR, error.code)
            .register(Metrics.globalRegistry);
}
```

Figure 27. incrementErrorCounter and creteErrorCounter methods

During a code review from team members, it was recommended the Netty dependency in the authentication *pom.xml* (Figure 23) be reduced to be smaller and more specific. This led to the author realising the Netty CIDR class had been deprecated in Netty 4.x, which meant the methods regarding IP addresses had to be rewritten (Figure 28) and the type of *ipAllowList* in the identity MO had to be changed. This led the author to the Apache Commons Net library [37], which contains the SubnetUtils class. The author added functionality for subnets, where the subnet /32 implicitly defines there is 1 usable IP address in the range.

```
static Optional<Set<SubnetInfo>> getIpAllowList(final Item item) {
    final var cidrString = item.getString(IP_ALLOW_LIST_ATTRIBUTE);
     if (null == cidrString) {
        return Optional.empty();
    }
    final var cidrSet = Arrays.asList(cidrString.split(","));
    final Set<SubnetInfo> ipAllowList = new HashSet<>();
    cidrSet.forEach(rawCidr -> {
        try {
            final var trimmedCidr = rawCidr.contains("/") ? rawCidr.trim() :
(rawCidr.trim() + "/32");
            final var subnet = new SubnetUtils(trimmedCidr);
            subnet.setInclusiveHostCount(true);
            ipAllowList.add(subnet.getInfo());
        } catch (final IllegalArgumentException e) {
            log.warn("invalid CIDR. skipping", e);
        }
    });
    return Optional.of(ipAllowList);
}

static boolean checkIpAddress(final Identity identity, final
AuthenticationRequest authenticationRequest) {
    final var ipAllowList = identity.ipAllowList();
    if (ipAllowList.isEmpty()) {
        return true;
    }
    final var requestIp = authenticationRequest.ip();
    try {
        return ipAllowList.get().stream().anyMatch(cidr ->
cidr.isInRange(requestIp));
    } catch (final IllegalArgumentException e) {
        log.error("invalid IP Address supplied in the AuthenticationRequest",
e);
        return false;
    }
}
```

Figure 28. Rewritten getIpAllowList and checkIpAddress methods

Testing the new solution fell into the scope of “Implement session authentication” and is described in length in 6. Testing.

5.4.2 Sprint 4 conclusion

Initial planning stated there would be a buffer time for any issues that may have occurred, the issue around the CIDR class was luckily resolved during the extra time. With all of the tasks implemented and functional requirements tested, the final task was marked as done and the epic closed. The sprint was ended with the goal met successfully.

6 Testing

Test dependencies added to the project included Hamcrest [38] for its matchers, Mockito [39] to be able to mock classes and JUnit [40] for general testing functionality. In the context of the thesis, the author has implemented unit and smoke tests.

6.1 DynamoDBUtils class

The purpose of the *DynamoDBUtils* class is to get aspects of the DynamoDB configuration from a configuration class. In this sense, the most important aspect of the functionality comes from the configuration, therefore a configuration with dummy values was created (Figure 29).

```
private final IdentityConfiguration identityConfiguration =
ImmutableIdentityConfiguration.builder()
.dynamoDBConfiguration(ImmutableIdentityDynamoDBConfiguration.builder()
                .endpoint("endpoint")
                .local(false)
                .region("region")
                .retries(1)
                .tableName("tableName")
                .build())
        .build();
```

Figure 29. Configuration class with dummy values for testing

The first test, *testGetClientConfiguration* (Figure 30) oversees the *getClientConfiguration* method in the DynamoDBUtils class behaves correctly assigning the client a custom retry policy.

```
@Test
public void testGetClientConfiguration() {
    final var clientConfiguration =
DynamoDBUtils.getClientConfiguration(this.identityConfiguration);
    assertThat(clientConfiguration.getRetryPolicy().getMaxErrorRetry(),
is(equalTo(this.identityConfiguration.dynamoDBConfiguration().retries())));
}
```

Figure 30. Testing getClientConfiguration method

The next test, *testGetEndpointConfiguration* (Figure 31) confirms the behaviour of the *getEndpointConfiguration* method, which should return an *EndpointConfiguration* with a custom region and endpoint.

```
@Test
public void testGetEndpointConfiguration() {
    final var endpointConfiguration =
DynamoDBUtils.getEndpointConfiguration(this.identityConfiguration);
    assertThat(endpointConfiguration.getServiceEndpoint(),
is(equalTo(this.identityConfiguration.dynamoDBConfiguration().endpoint())));
    assertThat(endpointConfiguration.getSigningRegion(),
is(equalTo(this.identityConfiguration.dynamoDBConfiguration().region())));
}
```

Figure 31. Testing getEndpointConfiguration method

The last two tests of the class cover the methods' abilities to retrieve the signing region and whether the configuration has been set to local development (Figure 32).

```
@Test
public void testIsLocal() {
    final var isLocal = DynamoDBUtils.isLocal(this.identityConfiguration);
    assertThat(isLocal,
is(equalTo(this.identityConfiguration.dynamoDBConfiguration().local())));
}

@Test
public void testGetRegion() {
    final var region = DynamoDBUtils.getRegion(this.identityConfiguration);
    assertThat(region,
is(equalTo(this.identityConfiguration.dynamoDBConfiguration().region())));
}
```

Figure 32. Testing isLocal and getRegion methods

6.2 AuthenticationServerImpl class

In order for a client to authenticate themselves, the *authenticate* method will call the *getCredentials* method to get data from DynamoDB. To test written functionality, a spy of the real *AuthenticationServerImpl* class was created. This way, one is able to call methods in a normal way, but track and manipulate each step of the way as if it were a mock [41].

Tests will be described through functional requirements.

- The authentication module shall check if a *system_id* exists,
- The authentication module shall check if the given IP is allowed to bind,
- The authentication module shall check if the given password matches the stored password in the database and
- If all checks pass, the authentication module shall respond with a custom successful response (Figure 33)

```
@Test
public void testSuccessfulAuthentication() {
    Mockito.doReturn(Either.right(this.identity))
            .when(this.authenticationServer)
            .getCredentials(this.authenticationRequest.systemId());
    final var authentication =
this.authenticationServer.authenticate(this.authenticationRequest);
    final var successfulResponse = ImmutableAuthenticationResponse.builder()
            .systemId(this.authenticationRequest.systemId())
            .sessionId(authentication.get().sessionId())
            .customerId(this.item.getString("customer_id"))
            .build();
    assertThat(authentication,is(equalTo(Either.right(successfulResponse))));
}
```

Figure 33. Testing systemId exists and successful authentication

- The authentication module shall check if there is a specified IP allow-list (Figure 34). Here, a DynamoDB Item is created with firstly 2 and then 0 IPs in the IP allow-list.

```
@Test
public void testGetAllowedIpList() {
    final var allowList =
this.authenticationServer.getIpAllowList(this.item);
    assertThat(allowList.get().size(), is(2));
}

@Test
public void testNoIpInDynamo() {
    final var falseIp =
this.authenticationServer.checkIpAddress(this.authenticationServer.fromDynamo
Db(this.nullIpItem).get(),
this.authenticationRequest);
    assertThat(falseIp, is(true));
}
```

Figure 34. Testing IP allow-list

- If any check fails, the authentication module shall respond with a custom unsuccessful response and update metrics corresponding to the unsuccessful attempts (Figure 35). Here, separate Items were created with non-corresponding data to the stored credentials.

```
@Test
public void testIncorrectSystemId() {
Mockito.doReturn(Either.left(ImmutableUnsuccessfulResponse.builder().error(SM
PP_3001).build())).when(this.authenticationServer).getCredentials(this.authen
ticationRequest.systemId());
    final var authentication =
this.authenticationServer.authenticate(this.authenticationRequest);
    final var incorrectSystemIdResponse =
ImmutableUnsuccessfulResponse.builder().error(SMPP_3001).build();
    assertThat(authentication,
is(equalTo(Either.left(incorrectSystemIdResponse))));
    verify(this.authenticationServer,
times(1)).incrementErrorCounter(SMPP_3001);
}

@Test
public void testIncorrectPassword() {
    Mockito.doReturn(Either.right(this.identity))
        .when(this.authenticationServer)
        .getCredentials(this.incorrectPasswordRequest.systemId());
    final var authentication =
this.authenticationServer.authenticate(this.incorrectPasswordRequest);
    final var incorrectPasswordResponse =
ImmutableUnsuccessfulResponse.builder().error(SMPP_3003).build();
    assertThat(authentication,
is(equalTo(Either.left(incorrectPasswordResponse))));
    verify(this.authenticationServer,
times(1)).incrementErrorCounter(SMPP_3003);
}

@Test
public void testIncorrectIp() {
    Mockito.doReturn(Either.right(this.identity))
            .when(this.authenticationServer)
            .getCredentials(this.incorrectIpRequest.systemId());
    final var authentication =
this.authenticationServer.authenticate(this.incorrectIpRequest);
    final var incorrectIpResponse =
ImmutableUnsuccessfulResponse.builder().error(SMPP_3002).build();
    assertThat(authentication,
is(equalTo(Either.left(incorrectIpResponse))));
    verify(this.authenticationServer,
times(1)).incrementErrorCounter(SMPP_3002);
}
```

Figure 35. Testing incorrect credentials

6.3 Smoke testing

In order to verify if the implemented solution works as intended, the author performed smoke tests with different sets of credentials. The tests were divided into two groups – the first where an IP allow-list had been specified by the client and the second where it had not been. In both cases, the author attempted to authenticate with correct credentials, an incorrect *system_id*, an incorrect password and missing credentials. In the case of an IP allow-list being set, the author attempted to authenticate with an IP which was not in the list (Figure 36). For this, the author used an existing SMPP test client and created test credentials in the DynamoDB table. In all test cases, the taken steps were the following:

- Access the SMPP client
- Send a bind request with credentials
- Await a positive bind response or generic negative acknowledgement

In case of a generic negative acknowledgement:

- Check logs to identify specific custom *SmppError*

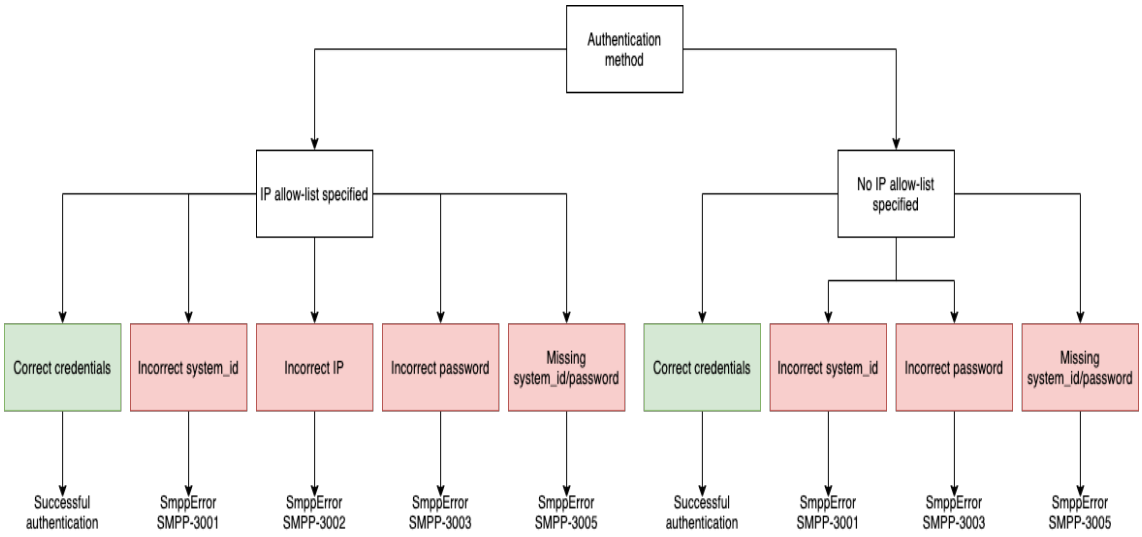

Figure 36. Smoke testing

The author was able to verify that both correct credentials cases returned the awaited positive outcome, therefore the smoke tests were considered successful. In cases of

incorrect or missing credentials the method returned the correct awaited custom error. As all smoke tests were successful, the author is sure the module works as intended.

7 Assessment of completed work

Assessment of the completed work can be monitored through the set requirements. The author was able to reach all goals related to both functional and non-functional requirements. The author is grateful for the extra allocated time for mistakes during the sprint planning, as the work was completed in exactly 4 sprints. Learning about SMPP and its vulnerabilities was a challenge for the author, as it is a protocol they had no prior knowledge about.

The authentication module was swiftly integrated into the bind PDU handler, where it has started to authenticate clients successfully.

7.1 Possibilities for future development

The authentication module was created in such a way, where it would be simple in the future to move it to a separate microservice. The authentication module can also be expanded to track a bind status and thus restrict the number of simultaneous open sessions per customer. A cap can be set on how many sessions a customer can open and when they reach the limit, authentication will be refused. This way, customers can be more aware as to how many resources they are using, because idle sessions send enquire links back and forth to insure the connection is still alive. After the authentication module is moved to a microservice and can handle bind state tracking, it can be published as an artifact in the Maven Central Repository (Figure 37).

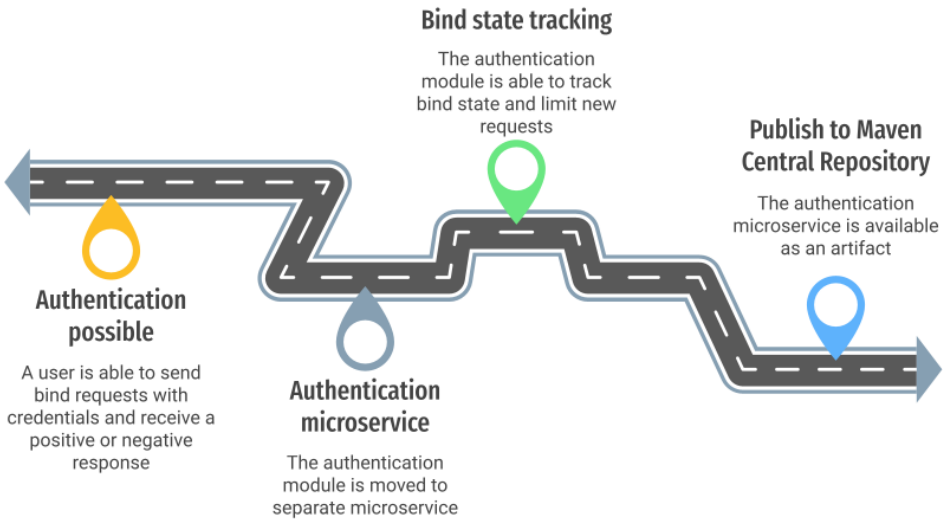

Figure 37. Possible future development roadmap

Furthermore, explanation work can be done to customers, why having an IP allow-list is better for security in the long run and the authentication method can be changed to not allow customers to authenticate with an empty allow-list.

8 Summary

The goal of the given thesis was to create a user authentication method for distributed networking systems with Short Message Peer-to-Peer protocol, which would enhance the current security. The initial scope of the work included creating a Maven module, which would be in charge of accepting credentials, checking them against existing credentials and sending a positive or negative response to the bind handler.

The analysis gave a good overview of the problem at hand, the existing technical situation and technical situations in other companies across the world. The technical implementation planning part of the thesis gave insight on how the work was divided into issues, how they were sized, and in which timeframe they could be implemented in. The technical implementation showed how the author approached the problem in an agile way and fulfilled sprint goals.

The authentication method can be called a success, because all functional and non-functional requirements were achieved, and the source code is available for viewing to the public. Although the problem at hand is well defined throughout many studies, there is a lack of public solutions. Therefore, as the world of SMS grows larger with each day, the authentication module is a contribution into a more secure future of SMPP.

The solution can be used by companies interested in creating support for SMS and SMPP, and private persons looking to send and receive secure messages via the SMS gateway.

References

[1] AfterAcademy, “AfterAcademy,” 16 December 2019. [Online]. Available: https://afteracademy.com/blog/what-are-protocols-and-what-are-the-key-elements-of-protocols. [Accessed 17 April 2021].

[2] SMPP Developers Forum, “Short Message Peer to Peer Protocol Specification v3.4,” 12 October 1999. [Online]. Available: https://smpp.org/SMPP_v3_4_Issue1_2.pdf. [Accessed 6 March 2021].

[3] S. Samanta, R. Mohandas and A. Pais, “Secure Short Message Peer-to-Peer Protocol,” International Journal of Electronic Commerce Studies, vol. 3, no. 1, pp. 45-60, 2012.

[4] The Wireshark Team, “Wireshark,” [Online]. Available: https://www.wireshark.org/. [Accessed 17 April 2021].

[5] Intergo Telecom Ltd, “sms.to,” [Online]. Available: https://sms.to/. [Accessed 17 April 2021].

[6] Melrose Labs Ltd., “Melrose Labs,” [Online]. Available: https://melroselabs.com/. [Accessed 17 April 2021].

[7] ONLINECITY.IO, “GatewayAPI,” [Online]. Available: https://gatewayapi.com/. [Accessed 17 April 2021].

[8] Elisa Eesti AS, “Elisa,” [Online]. Available: https://www.elisa.ee/. [Accessed 17 April 2021].

[9] Melrose Labs Ltd., “SMPP SMS Gateway,” [Online]. Available: https://melroselabs.com/services/smpp-sms-gateway/. [Accessed 17 April 2021].

[10] Elisa Eesti AS, “SMPP Implementation Guide,” 25 October 2016. [Online]. Available: https://www.elisa.ee/files/elisast/kasutusjuhendid/grupi-sms/SMPP%20Implementation%20Guide.pdf. [Accessed 17 April 2021].

[11] ONLINECITY.IO, “Get started with SMPP,” 23 April 2019. [Online]. Available: https://gatewayapi.com/blog/get-started-with-smpp-short-message-peer-to-peer/. [Accessed 17 April 2021].

[12] ONLINECITY.IO, “GatewayAPI docs,” [Online]. Available: https://gatewayapi.com/docs/smpp.html. [Accessed 17 April 2021].

[13] Intergo Telecom Ltd, “SMPP SMS Gateway API,” [Online]. Available: https://sms.to/smpp-api. [Accessed 17 April 2021].

[14] AltexSoft, “Functional and Nonfunctional Requirements: Specification and Types,” May 29 2018. [Online]. Available: https://www.altexsoft.com/blog/business/functional-and-non-functional-requirements-specification-and-types/. [Accessed 18 April 2021].

[15] Dropwizard Team, “Dropwizard,” [Online]. Available: https://www.dropwizard.io/. [Accessed 18 April 2021].

[16] The Apache Software Foundation, “Apache Maven Project,” [Online]. Available: https://maven.apache.org/. [Accessed 24 April 2021].

[17] jSMPP, “jSMPP,” [Online]. Available: https://jsmpp.org/. [Accessed 24 April 2021].

[18] OpenSMPP, “OpenSMPP,” [Online]. Available: http://opensmpp.org/. [Accessed 24 April 2021].

[19] Atlassian, “Jira Software,” [Online]. Available: https://www.atlassian.com/software/jira. [Accessed 24 April 2021].

[20] J. Grenning, “Planning Poker or How to avoid analysis paralysis while release planning,” April 2002. [Online]. Available: https://wingman-sw.com/papers/PlanningPoker-v1.1.pdf. [Accessed 24 April 2021].

[21] JetBrains s.r.o., “IntelliJ IDEA,” [Online]. Available: https://www.jetbrains.com/idea/. [Accessed 25 April 2021].

[22] JetBrains s.r.o., “Maven,” [Online]. Available: https://www.jetbrains.com/help/idea/maven-support.html. [Accessed 25 April 2021].

[23] baeldung, “Apache Maven Standard Directory Layout,” 3 July 2020. [Online]. Available: https://www.baeldung.com/maven-directory-structure. [Accessed 25 April 2021].

[24] Immutables, “Immutables,” [Online]. Available: http://immutables.github.io/. [Accessed 26 April 2021].

[25] baeldung, “Immutable Objects in Java,” 3 May 2020. [Online]. Available: https://www.baeldung.com/java-immutable-object. [Accessed 25 April 2021].

[26] baeldung, “Introduction to Immutables,” 24 April 2020. [Online]. Available: https://www.baeldung.com/immutables. [Accessed 25 April 2021].

[27] The Project Lombok Authors, “Project Lombok,” [Online]. Available: https://projectlombok.org/. [Accessed 26 April 2021].

[28] Awesome Java, “Lombok vs Immutables,” [Online]. Available: https://java.libhunt.com/compare-lombok-vs-immutables. [Accessed 26 April 2021].

[29] B. Pereira, “Why you should not use Lombok,” 24 September 2019. [Online]. Available: https://ppbruna.medium.com/why-you-should-not-use-lombok-f7556662e8c3. [Accessed 26 April 2021].

[30] D. Dietrich and R. Winkler, “Vavr User Guide,” 4 November 2020. [Online]. Available: https://docs.vavr.io/. [Accessed 26 April 2021].

[31] D. Tobin, “DynamoDB vs MongoDB: 5 Critical Differences,” 9 June 2020. [Online]. Available: https://www.xplenty.com/blog/dynamodb-vs-mongodb-differences. [Accessed 26 April 2021].

[32] Amazon Web Services, Inc, “Amazon DynamoDB,” [Online]. Available: https://aws.amazon.com/dynamodb/. [Accessed 26 April 2021].

[33] Amazon Web Services, Inc., “Working with Items and Attributes,” [Online]. Available: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/WorkingWithItems.html. [Accessed 27 April 2021].

[34] The Netty project, “Netty project,” [Online]. Available: https://netty.io/. [Accessed 27 April 2021].

[35] mindrot.org, “jBCrypt,” [Online]. Available: https://www.mindrot.org/projects/jBCrypt/. [Accessed 27 April 2021].

[36] Pivotal Software, Inc., “Micrometer Application Monitoring,” [Online]. Available: https://micrometer.io/. [Accessed 27 April 2021].

[37] The Apache Software Foundation, “Apache Commons Net,” 19 February 2021. [Online]. Available: https://commons.apache.org/proper/commons-net/. [Accessed 26 April 2021].

[38] hamcrest.org, “Hamcrest,” [Online]. Available: http://hamcrest.org/. [Accessed 27 April 2021].

[39] Mockito, “Mockito,” [Online]. Available: https://site.mockito.org/. [Accessed 27 April 2021].

[40] JUnit, “JUnit,” 13 February 2021. [Online]. Available: https://junit.org/junit4/. [Accessed 27 April 2021].

[41] E. Paraschiv, “Mockito - Using Spies,” 10 September 2020. [Online]. Available: https://www.baeldung.com/mockito-spy. [Accessed 27 April 20201].

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis[1]

I Emily Joy Krohn

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis “User authentication method for distributed networking systems with Short Message Peer-to-Peer protocol”, supervised by Nadežda Furs
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

16.05.2021

1 The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.

Appendix 2 – Source code relevant to authentication module

https://github.com/emilyjkrohn/smpp-authentication