Security Assurance of REST API based applications

Nishu Prasher

Master in Information Security
Submission date: June 2018
Supervisor: Basel Katt, IIK

Norwegian University of Science and Technology
Department of Information Security and Communication Technology

Preface

This Master thesis is submitted to complete a three year part-time study program in Master of information security (MIS) at NTNU, Gjøvik. The project was carried out during the spring semester 2018, in cooperation with Statistics Norway. The idea of the project was suggested by my supervisor, Associate Professor Basel Katt at the Department of Information Security and Communication Technology. The topic was discussed with the management at Statistics Norway, and it was chosen after recognizing that the topic was complying with company's need and objectives. The intended readers for this thesis are security testers, security architects, developers and managers, particularly security managers. The readers should be familiar with basic concepts of security related terms and REST API technology.

01-06-2018

Acknowledgments

My foremost thanks to Almighty for giving me knowledge, strength and opportunity to undertake this research.

I would like to thank my supervisor Basel Katt, who has given me valuable guidance throughout the work. This thesis could not be completed without his help.

I would also like to thank my colleagues at Statistics Norway. From the management, thanks to Erik Øvrum for giving me the opportunity to complete my master's degree. Thanks to Even Lysen, who has contributed to the security testing process. Many thanks to Agata Wrzos-Kaminska and Hilde Madsen for the valuable discussions and inputs during the work. Thanks to Rune Johansen and other colleagues in the developer team for sharing their technical knowledge of the case studies.

Last but not the least I would like to express my gratitude towards my family for their support during this work. I have spent precious family time on week-ends and vacations to finish this work. Jeinender, you are my rock, this could not have been accomplished without your support. My daughters Argha & Kashvi, thank you for being patient and understanding. Thanks to my mother for feeding me dinners on busy days.

N.P.

Abstract

Security assurance is the confidence that a system meets its security requirements, based on specific evidences that an assurance technique provide. In this thesis, I have proposed a quantification method which aims to develop security assurance profiles by measuring the level of security of a REST API. The notion of measuring security is complex and tricky, existing approaches are often based on manual review and time consuming tasks. In addition, there is little research work done on quantification of security assurance for REST APIs.

A common perspective has been to focus on the vulnerabilities of a system while security testing. However, security requirements are not tend to get enough attention during a security test. The main approach of this thesis was to look at both requirements and vulnerabilities to accomplish a level of security assurance. Appropriate metrics were defined to reflect the *requirement fulfillment* and the *vulnerability presence*. The requirements were declared to be fulfilled if their associated security mechanisms were present. Vulnerabilities were on the other hand sorted into their relevant categories and assigned a risk score. The security assurance metric was defined as an equation where the vulnerability metric was subtracted from the requirement metric.

The case studies were carried out at Statistics Norway, where the author is employed. Analyzes showed that the API with the most security mechanisms implemented got a slightly higher security assurance score. This was due to the fact that the vulnerabilities were considered more harmful in one of the cases as the security objectives diverged.

The proposed quantification method can be re-used on any other domain, by altering the lists of requirements and vulnerabilities.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Topic Description

Security of sensitive information, whether it is personal or organizational data, is a crucial issue troubling public, developers and decision makers today. REST APIs is not any exception since they are part of critical systems that need protection against threats and security breaches. How can security testers proclaim with confidence that a particular API is safe to use? For this we need to *measure* security applied, with appropriate security metrics.

The topic of this thesis is to define a quantitative security assurance method, that will help to obtain a security assurance value which will reflect the security of a REST API. Security assurance standards, such as Common Criteria (CC), is often associated with a lot of manual work and time consuming tasks, which can lead to discouragement for the security tester. Hence, it is important to simplify the method, without compromising with the quality of the output value.

To assess the security of any application we need metrics which can be provided as "a yardstick". Hence, an evaluation method have to be created to calculate the overall risk. This thesis will offer security metrics that will help security testers to obtain security assurance profiles for a REST API based application. Hopefully, it will contribute to improve the efficiency of the security testing process and will subsequently also motivate the security testers to use this method.

For this thesis initially a literature review is needed to analyze if and how available standards such as *Application Security Verification Standard* from OWASP[2] can be used. The suggested quantitative method must be tested to evaluate if it is working as per the expectations and the intended purpose. The verification of the method can be done by applying it on two different APIs and compare the results. In addition a questionnaire will be sent to persons involved in the process to validate the method.

The security testing will be done with the help of security experts from **Statistics Norway**, since the author is employed there. The REST APIs included in the case studies also belongs to Statistics Norway. Further, the quantification method will be used to calculate the overall *security* of the REST APIs.

This thesis will be useful for security testers, security architects, developers and managers, particularly security managers, etc.

1.2 Keywords

- Security testing
- Assurance
- Information System Security
- Security risk
- Security quantification
- Security metrics

The collection of the keywords are specific and gives a good amount of reading material when searched alone and in combination.

1.3 Problem description

Security assurance is the confidence that a system meets its security requirements, based on specific evidences that an assurance technique provide. Evaluation, on the other hand, is the process that is responsible for gathering and analyzing evidences to check if the security requirements are fulfilled.

The current evaluation methods and standards, like the Common Criteria, tend to achieve their goals manually, to a large extend, and require a lot of resources in term of time, cost, and effort. Furthermore, such criteria can be found for a limited number of application domains.

In other words the motive is to measure security which is not a forthright task[3].

The main task of this thesis is to develop assurance profiles for general REST API based systems with support of a coherence framework proposed by the master thesis author.

An assurance profile is described as following by Haddad et al.

> " *The Assurance Profile provides a framework to the security expert community in order to collect descriptions and architectures of typical security mechanisms, and establish best practices on operational security assurance requirements and measurements for these architectures... An Assurance Profile (AP) is a formalization to define a common set of assurance measurments needs on an agreed Target Of Measurement (TOM) for a given service...*
> *...An AP represents a common and coherent understanding on how security and assurance should be addressed by a TOM"[4].*

According to the definitions above, the proposed framework will include the definition of security functional requirements, security vulnerabilities and the associated metrics. Furthermore, the framework will be tested against two case studies. The case study will be done on APIs at Statistics Norway, and with cooperation of colleagues at Statistics Norway.

1.4 Justification, motivation and benefits

There are very little research on any framework for security testing a RESTful API. However, there are a lot of research on frameworks for security testing other domains such as Cloud, networks, web applications etc. An API makes it easier for companies to offer services to their customers. With the growth of the API usage, it is also important to focus on the security of APIs, so that essential company data and customer data stay secure. According to a survey done by NorSIS, 8 out of 10 companies in Norway has 4 or fewer employees[5], hence it is challenging and not always a priority to appoint a security specialist in the team. It is also very common to outsource the IT-services, IT-security included. A framework can be beneficial for security testers so that they are sure that the process is compliant with the rules and objectives, in addition they can save time since by reusing generic modules. Other quantification methods only focus on the vulnerabilities, but my method is based on including the requirements, which will give a better understanding and insight of the total security applied.

1.5 Research questions

1. How can an assurance-level of a REST API be estimated based on a quantification method?
2. How can we automate the security control checking?
3. How can we check if the security testing process is working as per the expectations and fulfill its intended purpose?

1.6 Choice of methods

In this study both quantitative and qualitative methods will be used. Qualitative methods are mostly used to collect user experiences by doing in-depth interviews or taking surveys for capturing mostly the feeling of the subject. Quantitative methods implies taking objective measurements or numerical analysis of tests, which is suited for this study.

- **Q1. How can an assurance-level of a REST-API be estimated based on a quantification method?**
  Scientific research must be based on existing work. Thus, a literature review of assurance methods will be performed. It is essential to gather knowledge of similar frameworks which are available and getting an overview of current methods and tools. After that narrow down the research to REST APIs. Metrics to measure the security must be assigned, so that the security assurance can be calculated.

- **Q2. How can we automate the security control checking?**
  The definition of *automated* is described in the dictionary as "*something that is designed to replace or decrease human labour*"[6]. Everything that can be automated in the process will help to increase the efficiency of the method. If there is a list of requirements available, it is easier to automate the security control. The security tester know in advance what s/he need to look for. All of this will be done by reviewing literature and trying to find out what the common security issues of REST APIs are. Likewise, what are the needed requirements. For instance OWASP have a very comprehensive list of security requirements for web-applications. This list will be restricted for those security requirements that are particular for REST APIs. Likewise they have also a list of top 10 vulnerabilities for web-applications that will be used.

- **Q3. Does the security testing process meets specifications made and fulfill its intended purpose?**
  The security process can be evaluated by applying it on two different REST APIs and compare the results. Everyone involved in the process will be formally (and informally) asked about what they think about the process and the proposed method. The latter is defined as a qualitative research method.

1.7 Claimed contributions

- A definition of a method and an overall security testing process, which can be used as directions for security testers who wants to test a REST API.
- Creation of an evaluation method to calculate the overall assurance, based on proposed security metrics for *fulfilled requirements* and *found vulnerabilities*.
- Verification of the method by applying the method on two different APIs and compare their assurance value. In addition, a validation from co-workers in the form of questionnaire and informal discussions.

1.8 Chosen Case Studies

There are two REST APIs security tested for this thesis. Both of them belong to Statistics Norway, where the author is employed. On the official website it can be learnt more about the organization. But in short:

> "*Statistics Norway is the national statistical institute of Norway and the main producer of official statistics. We are responsible for collecting,*

producing and communicating statistics related to the economy, population and society at national, regional and local levels"[7].

The first one is an internal API which will be anonymized due to security reasons. The other REST API is an open API which has a detailed user-documentation available on the website. This API lets the user create a customized dataset, based on queries made towards over 5000 StatBank tables Statistics Norway offer[8].

A summary of StatBank is given below:

"StatBank is a database where the user can compile tables and create figures and maps. The database contains figures in tables that form part of and are linked to the statistical publications, and enables users to create more detailed tables than the published tables. The main source for the tables is Statistics Norway, but tables are also sometimes sourced from other producers of statistics in Norway who have given their consent to the inclusion of the figures"[9].

1.8.1 PX-API

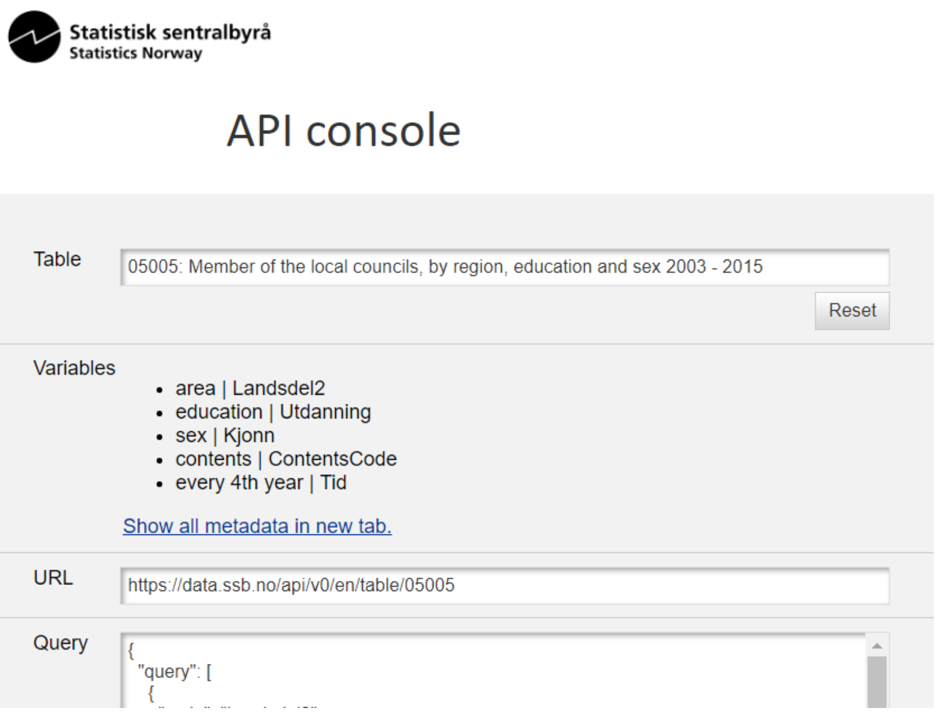
A short synopsis of the API will be given in this section, a detailed description of the API is available here[8]. The API URL is composed as following:

http://data.ssb.no/api-name/api-version/lang/

There are two ways to find tables in the API. One is from the console and second is the URL itself.

1. Via Console: http://data.ssb.no/api/v0/en/console/
2. Via URL: http://data.ssb.no/api/v0/en/table/

The data can only be extracted from the console, and from the URL its only possible to extract the metadata.


Statistisk sentralbyrå
Statistics Norway

API console

Table: 05005: Member of the local councils, by region, education and sex 2003 - 2015

Variables
- area | Landsdel2
- education | Utdanning
- sex | Kjonn
- contents | ContentsCode
- every 4th year | Tid

URL: https://data.ssb.no/api/v0/en/table/05005

Query:

```
{
 "query": [
  {
```

Figure 1: PX-API Console

The figure 1 is a snippet of the PX-API console on the web. In the console it is possible to POST your query to extract the desired data.

If we choose the second option (Via URL), it is possible to navigate through the StatBank tables. First, you get an overview of all the tables. Further, you can move yourself more and more downward to specific data/statistics. The structure of the URL is as following:

http://data.ssb.no/api/v0/no/table/(topic)/(subtopic)/(statistics)/(tablename)

The security objective of this API is mainly availability, since this is a public API with official data.

1.8.2 TS-API

TS is short for top secret. This is an API that must be anonymized because the management of the organization require that the functions and the security issues are not disclosed in this thesis. The API does not hold any data and it is used for transformation of the data. The security concerns are mostly related to integrity of the data and not the confidentiality. The integrity objective is that the data must not be changed, before it is sent further[10].

1.9 Thesis Outline

This thesis is divided into 8 chapters. The first chapter is the Introduction. Chapter 2 gives a background information necessary to understand the subsequent chapters, in addition there is a summary on the related work done on this topic. Next, there is a comprehensive description on the methodology used. Chapter 4 is a description of the quantification method and all the related parts. Followed by chapter 5 where I explain how the metrics were developed. In Chapter 6 I have analyzed the two case studies and the results that was acquired from the security testing. Then there is chapter 7 where I discuss the result, the method and feedback from the questionnaire. Finally, the last chapter 8 consist of a conclusion and suggested further work.

2 Background

This chapter presents an introduction of REST APIs, and the reason for their popularity. In addition, an introduction on the main security issues that are related to REST APIs. Further, a section on related work that has been done on security testing of REST APIs and other related domains. In addition related work done on quantification methods is included.

2.1 Definition of REST & API

API is an abbreviation for Application Programming Interface. An API is a software intermediary that lets two applications talk to each other[11]. Others [12] describe APIs as windows into applications. Mulesoft[11] gives a real-life example of an API. If you are using a website for instance for an online travel service such as Expedia, you search on your criteria for instance destination hotel type etc. The travel service have to talk with the airline database API to get all the information. The API is an interface which get requests from several travel services and acquire data from its database and then sends it back to the online travel service. The enormous growth of APIs are due to a technology shift from XML to JSON (Javascript Object Notation). Just to mention, Twitter went from using XML to JSON, and Oracle went from SOA (Service Oriented Architecture) to REST. An API is just a front end to your backend services[13]. It is a smart way of communicating because it provides an layer of security, since the client-data is never fully exposed to the server and the server is never fully exposed to the client[11]. We can compare an API[11] to a waiter taking orders from the guest at a restaurant and giving the order to the kitchen. The order is the request and the food which is sent back from the kitchen is the response. The guest and the kitchen never talks to each other.

Neeraj Khandelwal [13] talks about how important it is to focus on security of APIs. APIs don't only expand business, an API-breach can also collapse it instantly.

> *"As an example, the use of JSON in HTTP requests presents new conduits through which untrusted data can reach backend services or an end user's browser, where it is consumed. Similarly, would-be attackers can pass user inputs within the URL path (rather than URL query) with REST, breaking legacy security tools. Another challenge with APIs is service availability. APIs are exercised programmatically and can be extremely chatty. Unanticipated use, verbose applications or abusive partners can wreak*

havoc upon the API SLAs (Service Level Agreements), or even bring down the backend services, with severe financial implications[13]".

REST stands for REpresentational State Transfer, and is a architectural style developed by Roy Fielding[14, 15]. REST offers these key concepts for building a service over the web[16]:

- Resource Identification: A resource is a *unique thing* identified with a URI (Uniform Resource Identifier). It should have at least one representation, and a set of attributes to describe the resource and not only an ID. Examples of resources can be student, teacher, subject, book etc. Attributes describing a student can be name, studentnr, class etc.
- Uniform Resource Interface: All resources are accessible with the help of HTTP operations which can help retrieve, create, delete and update the resource. Roy Fielding noticed that four http verbs were used commonly (also called methods)[15]:
  - GET: Requests a representation of the resource specified
  - POST: Create a new resource
  - PUT: Update an existing resource or add a new
  - DELETE: Delete resource

  The idea is that the standard HTTP operations should be used for REST-based services[16].
- Links and Hypermedia: Resources are linked together with relation link types, so that client applications can navigate between different states of the same resource[16].

2.1.1 Design Responses

For each operation the composition of the response must be considered. The two aspects are the content of the response and secondly the fact that the responses must handle errors[16]. The latter is described first.

Status Code

It is recommended to use proper status codes so that the client can understand the interactions. Owasp[14] gives a guideline on how to use status codes. Paik et al.[16] recommends using minimum of three codes:

- 200: OK - Normal response
- 400: Bad Request - The client requested something with errors or
- 500: Internal Error - The server encountered internal erros during processing of the request.

Response Formats

JSON and XML is the most commonly used response formats. It is recommended to use response formats that represents simple objects, and a single result should return a single objects. However, a collection of the simple object should be returned if a response contains multiple objects[16].

2.1.2 HATEOAS

HATEOAS stands for Hypermedia As The Engine of Application State. For a web-service to be completely RESTful it should follow the HATEOAS principle. Which is explained as:

> *"The principle is that a client interacts with a network application entirely through hypermedia provided dynamically by application servers. A REST client needs no prior knowledge about how to interact with any particular application or server beyond a generic understanding of hypermedia"[16].*

In easy words HATEOAS let the API guide the user through the application by including relevant information about any potential next step with the returned responses[17].

2.1.3 Simple & Stateless

The main motive of REST is to simplify the URLs instead of using complex request body or POST parameters. This is done by specifying the URLs as objects/nouns instead of actions. This example is from[18] and describes it well:

- GET /tickets - Retrieves a list of tickets
- GET /tickets/12 - Retrieves a specific ticket
- POST /tickets - Creates a ticket
- PUT /tickets/12 - Updates ticket 12
- DELETE /tickets/12 - Delete ticket 12

It is recommended to use plural form of the noun to keep it consistent. In addition it is easier for the API provider to implement the API, when they do not need to deal with different forms (person/people, goose/geese)[18]. Other advantages of using REST is, that it is lightweight in communication, meaning that it acquire less bandwidth[19]. REST is also stateless implying that nothing is saved on the server, and the philosophy of REST is that the needed task is done in one request, addressed as an ideal RESTful service[20].

2.1.4 REST API Security

REST API has similar vulnerabilities as a web-application[12]. According to[14, 19]the top most security vulnerabilities a REST API may have are listed here:

- Injection Attacks and message Altering

- Authentication-based attacks
- Denial of service (DoS) and buffer overflows
- Cross-site scripting/cross-site request forgery
- Man-in-the-middle (MITM) attacks
- Replay attacks and spoofing
- Insecure direct object references
- Sensitive data exposure
- Missing function level access control
- Unvalidated redirects and forwards

Since REST APIs uses simple HTTP/HTTPS protocols it is inclined to all application layer security vulnerabilities, e.g. XSS and parameter tampering. Beer and Hassan also states in their paper:

> *"Generally REST deals with JSON format for data exchange; it is possible to subvert or disturb the application logic on the services by JSON injections, which may result in misbehaviour of services, data theft, resource deletion and malware execution"[19] .*

2.1.5 UnRESTful security measures

Oauth and OpenId are two security measures that are frequently used by many organizations. However, their philosophy are indeed the opposite of REST, since they are session based and REST is not supposed to maintain any user session. Still, they will be selected as a security requirement in this thesis work, as the majority of the security experts recommends these security measures for REST APIs[19].

2.2 Related Work

There is alot of research on security assurance framework for other domains but there is little work on framework for REST APIs. This section will document this and in addition it will document related work on quantification methods that has been published.

2.2.1 Various Security Assurance Frameworks

Joshi et al. [21] proposes a framework in their paper, which contributed to improvement of the security level of their University campus network. The model is a quantitative information security risk assessment model which uses Common Vulnerability Scoring System (CVSS). The framework is divided into three main phases:

1. Identify weak points (e.g. Identity Threat, Vulnerability Identification and Analyzing the effectiveness of controls)
2. Prioritize what matters most (e.g. consisting of Quantitative Risk Manage-

ment)
3. Improve University's Security Position (e.g. Creation of Actionable Remediation Plan)

These phases are again divided into sub-processes and examples are given in brackets. The framework is based on OCTAVE which is model based risk assessment methodology, the difference is that the proposed model is operational based. This is a framework that is too ample and less focused to use for testing a REST API, but parts of the methodology can be suitable.

Such et al. [22] suggests a framework with 20 "assurance techniques". The techniques are split over 5 high-level categories.

1. Review - Review of processes, Thereat Assessment, Source Code Review, etc.
2. Observe - Identification of real world deviations.
3. Interview - Interview of one or more persons about security related matters
4. Test - Vulnerability Sca, Penetrations Scan, Red Team Excercise, Social Engineering, etc.
5. Independent validation - Witnessed Test and Public review

This framework can really be helpful in the categorization of the tests, because the framework has a universal approach. All tests that are applicable to REST APIs can be extracted and reused in our framework. Furthermore Such et al. also has done a survey on 153 industry practitioners where 81% had over 5 years of experience. They were asked questions such as, how many people are required to perform the assurance techniques? What kind of expertise is required? How much time it is required to do the testing? etc. This information can be helpful when the testing of our framework will be conducted.

Ouedraogo et al. [23] advocates the need for a Security Assurance (SA) system which can be embedded within a current IT system. The purpose of the SA-system is to identify vulnerabilities and mitigate these by a so-called assurance-driven approach. Hence, the output is a set of assurance indicators of the system. Security tools such as anti-malware and intrusion detection systems must be strengthen with Security Assurance systems. Their paper analyze the practical challenges associated to the assessment of SA. Their SA-tool is not a operational-framework but a tool that helps to monitor the security assurance. It will show when the assurance level eventually drop. This is a little out of scope, however it may be useful on how its possible to combine the assurance indicators to the security mechanism. In addition, it is interesting to see how the monitoring works.

Spears et al. examined in their paper[24] assurance in a regulatory context. Their paper is mainly about how to conceptualize assurance by applying Capability Maturity Model (CMM) to security processes. The CMM is a framework for evolving a process from an ad hoc, less organized and less effective state to an highly effective state of security. The levels are called maturity levels. These levels can perhaps be useful in the thesis, for comparing with verification levels.

Tung et al. [25] has proposed a framework where they have applied security activities and practises of SSDLC to generate security guidelines and improving security software. This framework is an integrated security testing framework that particularly can be used while developing a software.

Hudic et al. [26] offers a security assurance assessment methodology for hybrid clouds. Systems and services in the cloud are multi-layered and multi-tenant viroments. Hence, the proposed methodology of Hudic et al. consist of identification and isolation of specific components that are of interest, where the independent assurance level is calculated for each of them. Clouds are more complex then APIs so this methodology is too obscure. However, in this paper the authors have aggregated assessments into assurance levels for various groups, which is relevant for this thesis also.

Gupta et al. has proposed a framework for security testing in their paper[27]. The main motive of their work was to check if the security mechanisms deployed was able to mitigate the threats to the vulnerabilities in the system. They suggest to take effective design decisions by choosing the most favorable security mechanism to implement security and then perform testing. Rather than performing the security testing after system implementation. By following their approach the security related issues can dealt with in early stages. This work is interesting, but it suits better for security testing during software development and not on later stages.

2.2.2 Security testing of REST API

This section is particularly on security testing of REST APIs and the related work that have been done in this field. As mentioned earlier there are no framework or a defined security testing process for a REST API. To define this we have to first know how the testing of a REST API is done. OWASP has a very informative website[14]. OWASP stand for Open Web Application Security Project, and it is an international nonprofit organization enganging of collecting security related incidents[19]. The information is updated regularly by the best experts in the world. To develop a framework of a process we have to first know how this is done. This also gives an initial overview over the different security controls that can be automated and those which will need manual considerations. CA technologies has published a white paper [12] which can give good technical insight into how to protect your APIs against attack and hijack. A noteworthy observation from that paper [12] is that even though REST API represents a new technology the threats are the same old which have troubled IT since the beginning,e.g. SQL injection, parameter attacks, man-in-the-middle attacks. etc. There are also other helpful "*articles*" [28, 29, 30, 31], often written in blog-form which can be valuable when we do a technical study of this kind.

2.2.3 Standards available

There are standards available such as *Owasp Application Security Verification Standard Project* [2], which can be used to establish a level of confidence that the security requirements are fulfilled. This particular standard is for web-applications, but it is likely that by carefully selecting only those verifications that are applicable for REST APIs, we can develop a security testing standard for our use.

Other standards can also be useful such as *Web Security Verification standard* by Mozilla[32]. This standard has not been updated lately, however it can be used to get an overview of verifications since it is briefer than the document from OWASP.

Srinivasan and Sangwan has published an article[33] where they have done a comparison and categorization of web app security-testing frameworks. Mainly five classification criteria were considered important:

- Which attack types a framework covers
- Testing approach (black box/white box)
- Automated/Manual or semi-automated method.
- Support of penetration testing for PCI (Payment Card Industry)
- Whether the framework mitigates false positives

This research is helpful for security testers/organizations looking for appropriate frameworks for their work. Likewise, it gave an overview of available frameworks during research work for this thesis.

2.2.4 Quantification methods

This subsection documents related work done on quantification methods. By quantification methods it is meant, methods to measure security of a particular system and development of associated security metrics.

In this paper[34] the authors discuss a different approach where they do not measure security by normal means, but instead use a decision-theoretic method. The proposed method is based on two options in terms of security sufficiency.

- "Send"-The system is secure enough to use
- "Hold"-The system is not secure enough to use. Reduce uncertainty to increase security and try again.

Further two simple indicators of security sufficiency are made, "Pass"/"Fail" . "Pass" means there will be no losses from a security breach, and "Fail" is the opposite. This research is a method to "measure" *how much security is enough?*, and it is usable for decision making before deployment of a system.

Pendleton et al. discuss in their paper[35] how hard it is to develop security metrics. In their study they [35] define security metrics based on four key dimensions/sub-metrics: a) metrics of system vulnerabilities b) metrics of system defense strength c) metrics of attack (or threat) severity d) metrics of system dimension or situations. Further, they try to investigate on the relationship between these four sub-metrics.

Savola[3] explains security metrics as a metric that illustrate the security level, security performance or the security strength of a system. Savola's paper is mainly about identifying quality criteria of security metrics. The result were three foundational quality criteria: correctness, measurability and meaningfulness.

In the master thesis[36] of Kahraman it is documented a process of measuring IT security and continuously validate the security level by setting up a Metrics Scorecard. The end product is a establishment of different metrics for three parts: Organizational, Minds (which I interpret as the human aspect) and Technical.

2.2.5 Summary of Related Work

None of the security assurance frameworks or quantification methods are able to answer my research questions. Each of these works are useful but either they test on other domains, different time of the process (mostly during the software development) or the method/framework is unspecific and wide.

3 Methodology

This study was done using both quantitative research methods and qualitative research. Quantitative research methods are methods that deal with measurements and numbers. While qualitative research is based on opinions and feelings.

3.1 Qualitative Methods

Qualitative research can help the researchers to get an idea of the basic opinions and motivations behind an issue. The researchers get a better insight into the problem and can contribute in developing new ideas and hypotheses. It is not possible to count qualitative data, since it can be chunks of videos, photos, text etc. However, the qualitative data can be transformed to quantitative data[37].

An extensive literature review of security assurance frameworks and REST APIs has been done. The case study was carried out at Statistics Norway and with assistance of colleagues from there. The test cases were chosen after discussion with colleagues and the management. First API, PX-API, was fine to security test since it did not have any confidential data. The other API, TS-API, required to be anonymized, since there were restrictions to publish security related information about it.

During the thesis work, I had several meetings with my colleagues to discuss my findings. Communication with the supervisor was also very important and helpful, and we had continuously discussions on questions that I had in mind.

Several persons with security expertise was involved in different parts of the work progress. Colleagues from the developer-team and IT-administration were beneficial to include, due to their expertise and advance knowledge in the chosen test cases. The main objective to involve these was capacity development and a better understanding of the test cases. The security testing of the REST APIs was done by a security expert in the company.

A questionnaire was sent to all colleagues that were involved in the process in mid-april. The questions can be found in the Appendix C. The questionnaire had a formal tone, but some of the questions were already informally discussed in meetings, hence the response rate was low. Ultimately, I still wanted to send a formal questionnaire to get a verification of the method and the findings, and also if there were additional comments.

3.2 Quantitative Methods

Quantitative research method implies the activity of collecting data and quantify the information that has gathered in order to come to a conclusion[38]. According to Leedy & Ormrod[39] the quantitative research can broadly be classified in three types: descriptive, experimental and casual comparative. Descriptive research describes the situation as it is. The experimental research is when some treatment is done and then the outcome is described. In the casual comparative the investigation is done on how independent variables are re affected by dependent variables and check the cause and affect relation between the variables[38].

The research question "*How can the assurance-level of a REST API be estimated based on a quantification method?*", must be answered by doing quantitative research.

The raw data consisted of a collection of requirements and vulnerabilities. Further, the analysis of the data was done by assigning metrics:

- Existence score
- Weights
- Risk Score

The motive was to find the total security applied to the REST API. At first only security issues related to REST APIs were investigated. Each vulnerability could be counted as something that reduces the security, thus a negative number. A security mechanism could be counted as something that increase the security, thus a positive number.

During that research I was looking at a specific requirement, enforcement of HTTPS. It was observed that the enforcement of HTTPS is dependent on certificate validation, which ensures that the sender is trustworthy and if its not validated it is possible to decrypt the data sent. Certificate Pinning gives an extra layer of security, by establish a trusted connection between the client and server. Quoted from Symantec: "*The client authenticate the server by validating that the server certificate was issued by a Certificate Authority that the client trusts*[40]".

A quantification of the situation above is as following:

Table 1: Quantification Example

Features of TLS/SSL	**Count**
HTTPS is enforced	+1
Certificate validation is deactivated	-1
Certificate Pinning is implemented	+1

The table 1 illustrates how enforcement of HTTPS can be futile if the validation of the certificate is not active since it will be possible to decrypt the data. Certificate Pinning on the other hand will give an extra layer of security. This example show how the idea came about; a security requirement add security and a vulnerability reduce security.

3.3 Case study

A case study is a research strategy that can be compared to an experiment or simulation of some kind[41]. Yin argues in his paper[41] that a case study does not need a specific type of data. Both qualitative and quantitative data can be used in a case study. The data can come from fieldwork, observations, verbal reports, archival records or any combination of these. Yin emphasize that the quantitative data "*should reflect meaningful events*"*[41, p.61]*. There is no restriction on using a particular data collection method either.

4 Design

This chapter documents the design of the proposed quantification method. The first section introduces on some basic concepts needed to better understand subsequent sections. Further, a description of the different steps of the proposed method is included.

4.1 Basic Concepts

To prevent confusion, definitions of some of the essential security related terms that have to be mentioned.

Souag et al. defines a security requirement as:

> "*Security is defined as a discipline which allows one to build reliable system that can face malice, errors or mischief... A requirement prescribes a property judged necessary for the system; security requirements engineering frameworks derive security requirements using security-specific concepts, borrowed from security engineering paradigm*"*[42]*.

While Felderer et al. states in their paper[43] that a security requirement can be specified as a *positive requirement* representing the expected security functionality of a security mechanism. Or it can be specified as a *negative requirement*, signifying what an application should not do. Further Felderer et al. gives explanation of other terms[43]:

- An asset is the element we want to protect.
- A fault is something that is wrong. A fault can also be found in dead code, so it is not necessary that a fault lead to an error.
- A vulnerability is special kind of a fault that is related to a security property. It is also described as a security mechanism that is missing or implemented in a wrong way.
- An exploit is where the vulnerability is used to do something malicious.
- A threat is something that can potentially harm the asset, such as hackers, malicious insiders or malware.
- A risk is the probability and the consequence of an unwanted incident.

Figure 2 shows relationship between security mechanism, security requirements and vulnerabilities.

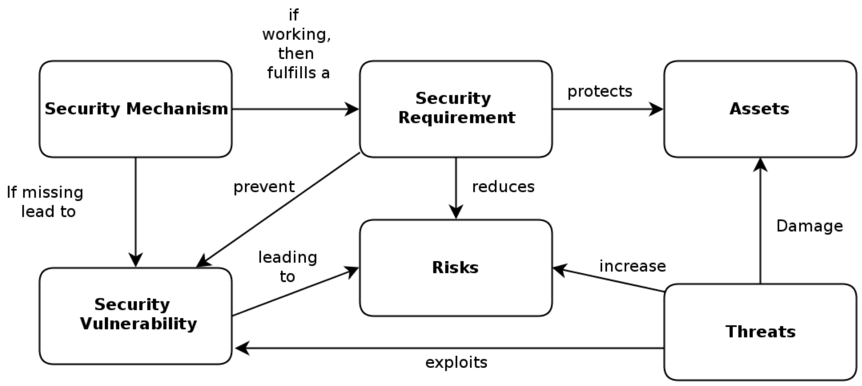


Figure 2: Relationship between security terms, adapted from[1]

From the figure we can see that a security requirement is dependent on the fact that a security mechanism is present and working properly. However, if it does not perform well or if it is completely missing it will lead to a security vulnerability. A security requirement reduces a risk and a security vulnerability lead to increased risk.

4.2 Process Development

The goal of this thesis was to develop a security assurance profile for a REST API based on a quantification method. To reach this goal several steps were needed:

1. Security testing to expose vulnerabilities
2. A draft of which security tests that must be sorted into manual and automated controls: Everything that can be automated can help the security testers to make the process itself more efficient. In addition list of tools is needed.
3. A checklist to tick of vulnerabilities that can be found.
4. A method for quantification of the security, with the help of risk scoring models. Along with a method to calculate the total security assurance level of the REST API.

Then all of these components could respectively added into a security testing framework. Figure 3 shows the proposed security process, which contain five steps. Each of the steps will be described in the next consecutive sections.

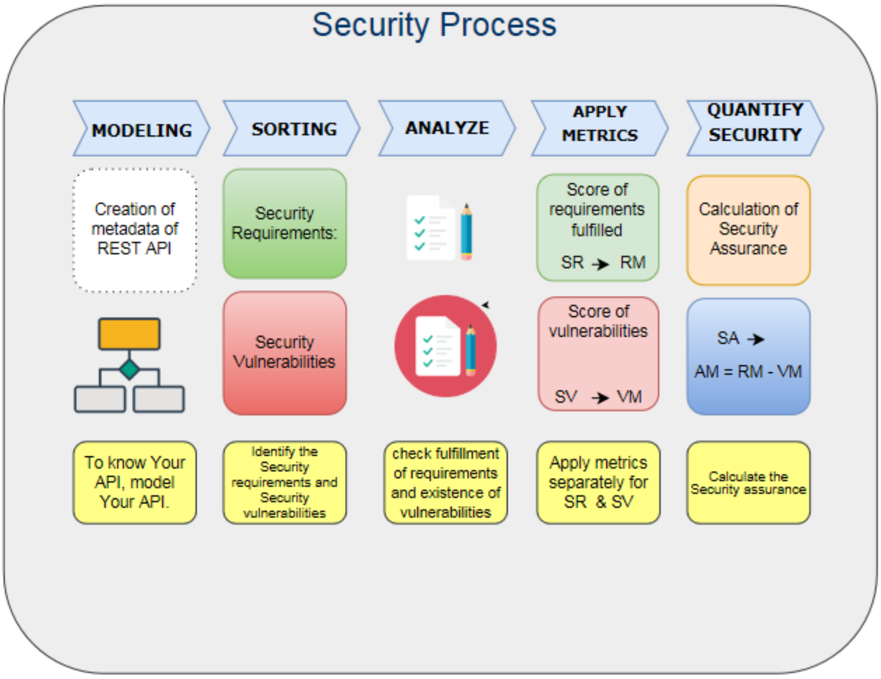

Security Process
MODELING
SORTING
ANALYZE
APPLY METRICS
QUANTIFY SECURITY
Creation of metadata of REST API
Security Requirements:
Score of requirements fulfilled
SR → RM
Calculation of Security Assurance
Security Vulnerabilities
Score of vulnerabilities
SV → VM
SA →
AM = RM - VM
To know Your API, model Your API.
Identify the Security requirements and Security vulnerabilities
check fulfillment of requirements and existence of vulnerabilities
Apply metrics separately for SR & SV
Calculate the Security assurance

Figure 3: Proposed Security Process

4.3 Modeling the API

As it is shown in figure 3. the process starts with creating a meta data model of the REST API. Because you as a security tester needs to know what you are going to test. Hence, the tester need to get acquainted with the API. Brajesh De gives an example in his book[44, p.155] where he has written some questions that can be used to investigate the API.

- What is the API behaviour when
  - no query parameters are passed
  - right query parameter is passed
  - parameter name passed is incorrect
  - multiple parameters are passed in both right & incorrect combinations are passed
- What is the default data format for the API response for both sucess and error conditions
- What is the API response status code for both success and error conditions
- What is the API response for unexpected HTTP methods, headers and URLs

The author takes up these questions to guide the reader about functional testing of an API, and it suits well for this particular step, as we are interested to know how the API behave. Generally API documentation tend to be available[44], so the consumers of the API know how to use it. This documentation can also be valuable for security testers when they want to know how an API acts.

To show how the REST API interacts an UML (Unified modeling language) model is chosen to help to illustrate the API functions.

4.3.1 Hidden APIs

Since this step is about gathering knowledge about the API, it is beneficial to know about how to find *hidden* information about REST APIs. A security tester should have the knowledge of where all the information of the API can be found. It is possible to find hidden APIs via the robots.txt. This file tells search engines whether to index or not to index different paths[45]. As an example:

```
Disallow: /images
Disallow: /scripts
Disallow: /secret/admins
Disallow: /api/admin/users
```

This information can often be misused by hackers, so it is important to restrict access to this file[45].

4.4 Eliciting Security Elements

A common perspective of security testers is focusing on the vulnerabilities, and working to find these during penetration testing, fuzzing, manual checks etc[42]. Souag et al. say in their paper

"Recent trends in security methodologies tend to consider that the best approach of security consists in starting from a risk analysis...[42, p.63]"

When the search for the security vulnerabilities started, it was observed that often security requirements and vulnerabilities were mentioned without classifying which category they belong to. It made the creation of the checklists a confusing task. *Insufficient TLS/SSL configuration* is a vulnerability, on the contrary the HTTPS is a requirement. Further, if validation of the HTTP Certificate is deactivated it does not matter that the encryption is enforced, still its possible to decrypt the information. To capture all these entities it was necessary to not only to focus on the security vulnerabilities but in addition create a checklist for security requirements. On the figure 3 we can see that under the *sorting* label the vulnerabilities and requirements are placed into two separate boxes. By doing this it was easy to get an understanding of what we expect from the API and what we don't expect from it[43]. Web applications- and APIs-security is often seen as similar. However, API-testing is different in the sense that it tests the interface which allows access to data and communication and not the application as whole[44]. For this study, all the vulnerabilities were obtained from the Owasp site[14]. The security requirements for this thesis work were compiled from the Owasp ASVS[2].

Finally the security testing provides a validation of whether various security requirements are implemented and/or security vulnerabilities are present, or not implemented or present[43]. More on the selection of the requirements and vulnerabilities will be discussed in the subsections below.

4.4.1 Selection of Requirements

Security requirements are often not prioritized due to several reasons. Stakeholders sometimes does not understand the threats that a system may be facing. It is difficult for developers to model what a system should not do instead of what it should do. In addition, a security requirement is seen as something that limits functionality or hinder usability[46]. Functional requirements can exhibit more variability than security requirements. If you look at the functional requirements for a website for an embedded avionics application and an e-commerce website they may have nothing in common. However their security requirements presents far less variability, both the avionics and ecommerce applications have to specify levels of identification, authentication, authorization, integrity , privacy, etc.[47]. Therefore, the security requirements: such as authentication, authorization, input validation will always be necessary for every domain. In chapter 5 the metrics for requirement is elaborated and definitions on how security mechanisms fulfill requirements is described in detail.

The security mechanisms that are compiled for REST APIs are modified from the requirements for web-applications. Those not suitable for APIs were discarded.

4.4.2 List of Requirements

Total of 10 requirements were considered as vital for a REST API. The 10 requirements have 53 security mechanisms associated with them.

Following is a description of every security requirement for the REST API according to Owasp that I have chosen as necessary for REST APIs[2].

- Authentication: Authentication is confirming that something or someone are who they say they are. Three common factors can be used for authentication, these are[48]:
  - Something you know e.g. passwords
  - Something you have e.g. a smartcard
  - Something you are e.g. biometric authentication (fingerprints, iris etc.)

.

- JWT: JWT stands for Json Web Tokens. JWTs are used for identity management and transmission of data between two parties. The JWT is sent as a compact and self-contained Json-object, which is digitally signed with HMAC or cryptographic keys[49].
- Access Control: The objective of access control is to verify that only those who are permitted to access a resource is allowed to do so, called authorization. This includes, persons are holding valid credentials, users are assigned to roles and metadata is protected from tampering.
- Input Sanitation: Input Sanitation is the activity of properly validating input from users or clients before it is processed. The rule is that incoming data should never be trusted.
- Error Handling: The API should provide a useful reaction to users. The logs must be protected if they contain sensitive information.
- Data Protection: The CIA triad must be enforced according to[2]: Confidentiality: The data should be protected against disclosure and unauthorised access both in transit and rest. Integrity: The data should be protected against alteration, deletion etc. Availability: The data should be available to authorised users.
- Communication Security: If there is sensitive data, TLS should be used. In addition strong ciphers and algorithms must be used.
- HTTP Security: The requirement regarding HTTP security is to verify that HTTP responses contain a safe character set in the content type header.
- Web Services: The requirement regarding Web services is to verify that all web-services has authentication, authorization, input validation etc enforced.

Appendix A shows the complete list of security requirements and their corresponding security mechanisms for a REST API.

4.4.3 Selection of Vulnerabilities

"Hackers are always in search of security holes through exposed APIs[44]"

Because unfortunately APIs are often overlooked when assessing the security of a web-application, since APIs dont have a visible front-end and is non-browseable. Also APIs are difficult to security test as compared to a web-application[50]. In addition, vulnerabilities that are found in APIs does not only affect one single project but spread to projects across global software ecosystem borders. Alqahtani et al. state that "*Tracing these vulnerabilities at a global scale becomes an inherently difficult task since many of the existing resources required for such analysis still rely on proprietary knowledge representation[51].*"
Hence securing the APIs should be a prioritized task for every organization.

If a security mechanism is missing or not working properly it will lead to a security vulnerability. According to figure 2 for a vulnerability to cause harm, presence of a threat is necessary to exploit it. For instance if no input sanitation is done it can lead to SQL Injection attack.

The most common vulnerabilities of a REST API was mainly found from Owasp[14] and other sites/blogs[28].

Just like every *requirement* is defined as a category with its related security mechanisms that can fulfill it, the vulnerabilities were defined as a category which was populated with related vulnerabilities. For instance all vulnerabilities that relate to broken authentication was placed into the same category.

- 2.1 Permits Credentials stuffing
- 2.2 Permits Brute Force
- 2.3 Permits Weak Passwords
- 2.4 Use plain text, encrypted or weakly hashed password
- 2.5 Has missing or ineffective multi-factor authentication
- 2.6 Exposes session id in url
- 2.7 Do not properly invalidate authentication tokens

All of these vulnerabilities are associated with authentication. Some vulnerabilities didn't fit into categories so they were defined as its own vulnerability category e.g. Elevation of Privilege.

4.4.4 List of Security Vulnerabilities

Total of 9 main vulnerability categories were considered to be crucial for a REST API. In detail 36 security vulnerabilities were added into their associated main categories. In Appendix B all of the security vulnerabilities is described in detail. The main vulnerability categories are gathered from[52, 14], and are as following:

- Injection: This vulnerability can be exploited by sending untrusted data into the API as part of a command or query. The input is then executed by the interpreter which can lead to an attacker getting unauthorized access to data or do other harm.
- Broken Authentication: Insufficient or missing authentication can lead to an attacker to compromise passwords, json web tokens, API keys etc. Only authenticated clients must be able to access the APIs. The APIs authentication and authorization requirements may be handled via API key, PKI, OAuth/OpenId tokens[44].
- Sensitive Data Exposure: Exposure of sensitive data due to absence of encryption in transit or at rest can lead to an attacker performing identity theft, card fraud etc.

- Broken Access Control: Insufficient or missing access control can allow the attacker to get access to other users account, change access rights, modify data etc.
- Elevation of Privilege: This attack is when an outsider tricks to get access rights they are not entitled to. This is mostly possible due to a bug that lets the attacker bypass the security.
- Cross-Site Scripting: This is an exploit where it is possible to inject malicious code into an API, due to improper input sanitation. The outcome is that it is possible to hijack user sessions or redirect the user to malicious pages.
- Cross-Site Request Forgery: This is a vulnerability where the exploit is performed by the authenticated user on the behalf of the victim. The target is not data theft but state changing request[53].
- Parameter Tampering: This is a vulnerability where the attacker is allowed to perform attacks e.g. model binding, mass assignment and http verb tampering.
- Man-in-the-middle-attack: Man-in-the-middle-attack is when an attacker place himself between two communicating systems and intercept the information sent between them[54].

4.5 API Security Testing

After the requirement and the vulnerability lists were created the security testing was performed. In general security testing validates if security requirements are implemented correctly[55]. According to Tian-yang et al.[56] there are two main approaches for security testers, these are security functional testing and security vulnerability testing. Tian-yang et al. state in their paper

> *"Software security testing can be divided into security functional testing and security vulnerability testing. Security functional testing ensures whether software security functions are implemented correctly and consistent with security requirements basing on security requirement specification. Software security requirements mainly include data confidentiality, integrity, availability, authentication, authorization, access control, audit , privacy protection, security management, etc. Security vulnerability testing is to discover security vulnerabilities as an attacker. Vulnerability refers to the flaws in system design, implementation, operation, management. Vulnerability may be used to attack, resulting in a state of insecurity, Security vulnerability testing is to identify software security vulnerabilities"[56].*

In this thesis I have combined these two methods and performed both of them.

The security functional testing is mainly done by checking manually if the security mechanisms exists or not. The security vulnerability testing on the other hand is partially done automatically. For instance Black Box-fuzzing was performed on the APIs. Black box, means that the attacker or in our case the tester does not have access to the source-code. API fuzzing is done by sending all possible types of combinations of input parameters and then inspect how the API responds. The goal of the attacker/tester is to understand the system behaviour by studying the error messages, and further use this information to get unauthorized access[44].

4.5.1 Tool selection

Mostly open-standard tools, those easy available on the internet, was used to security test the REST API.

Here is a list of the security testing tools used in this study:

- The tool Sqlmap is:

  > "*...an open source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over of database servers. It comes with a powerful detection engine, many niche features for the ultimate penetration tester and a broad range of switches lasting from database fingerprinting, over data fetching from the database, to accessing the underlying file system and executing commands on the operating system via out-of-band connections[57].*"

- Python (tool/scripting): Python is a simple and easy to learn programming language. Python code is turned into machine language and run as the script is executed. The code is not so fast as other languages e.g. C, Cobol, but it has better portability through heterogeneous environments, than other compiled codes[58]. Even though Python is slow, it is very popular. The reasons for its popularity is its massive user base, that can help other users when necessary. Other reasons are that it has clean syntax and an expansive library[59].
- Curl: Curl is open source software command line tool that is used to transfer data with URLs. It is used in many areas like, cars, routers, mobile phones, printers etc.[60]
- Burp-Suite: Burp-Suite is a web scanner that can find all web related vulnerabilities. It can be used directly to interact and manipulate streams of web traffic on the browser, and is a standard when it comes to such kind of transparent proxies[61, 58]
- Wireshark: Wireshark is a popular network protocol analyzer worldwide, it lets you see what is happening on your network. It is the standard across many non-profit and commercial organizations, government agencies and

educational institutions. Wireshark has features as: GUI interface, deep inspection of hundreds of protocols, multi-platform application, Live data extraction, multiple output filetypes(XML/CSV/plaintext), etc[62].

4.6 Apply Metrics & Quantify Assurance

The last two steps on the figure 3 is assigning of metrics and quantify assurance.

Security metrics in this step is applied to "measure" the security. There have been limited research on how to measure security, and conclusively the experience has been that it is a complex task[34, 3]. It is not a straight forward method to do this, because the problem is that to measure security we need quantifiable information like percentage, average or any numbers[36]. Chapter 5 contains a detailed explanation of how the security metrics were defined.

There is a need to define a metric that can capture the *total score of requirements*. This specific metric will have a positive (+) sign. Unlike the metric that captures the *total score of vulnerabilities*, which will have a negative sign (-). Another metric must be defined to capture the total overal security by combining the two last-mentioned metrics.

The "*counting*" of security mechanisms and security vulnerabilities is done separately. Thus, a distinctive scoring system for each of them was needed.

To evaluate a vulnerability's impact and probability I needed a risk scoring model. There are several risk scoring models available, which have been extensively researched on. Two risk models were evaluated for my work, and a detailed description of these models can be found in chapter 5.

A model for *security requirement fulfillment* was not found in the literature. Hence, I am proposing a new method to evaluate the fulfillment of a requirement.

5 Quantifiable Metrics

In this chapter a description of all metrics are included. First an introduction on the importance of metrics. Then the proposed method of calculating the security assurance of a system is described, together with the security requirement and vulnerability metrics. Finally, a section on the risk models that were evaluated for this study is depicted.

5.1 Importance of Metrics and Measurement

The security assurance level of a system state how much confidence one can have in the system that is safe to use. Instead of acknowledge this with an exclusively yes or no, the decision makers need to take informed decisions[63]. Because basically security is often seen as a costly expenditure which in addition interrupts the production line. Therefore security metrics can be used to base a security testers argument on evident data. Another benefit could be to look at security metrics over a time period, to evaluate the security performance of the security mechanisms[36]. A security metric is often defined as a metric that depicts the security level, security performance or security strength of a system[3]. However, Savola[3] argues that security cannot be measured as a universal concept due to the complexity and uncertainty during the testing, therefore terms as indicators or strength are more suitable to use for this purpose. Further Savola gives a definition of a measurment as "*a process of experimentally obtaining information about the magnitude of a quantity. Quantity is a property of a phenomenon, body, or substance to which a magnitude can be assigned. A measurement result indicates single-point-in-time data on a specific factor to be measured, while metrics are descriptions of data derived from measurements used to facilitate decision-making[3]*".

For a security metric to be meaningful and usable it must be *measurable*. Which is possible if the measurable information is available[3]. According to Savola "*measurability of a metric means that it has dimensions, quantity or capacity ascertained in the measurement architecture and the measurable information is attainable with sufficient precision[3].*"

The purpose of this chapter is to define meaningful and measurable metrics to capture the requirements and vulnerabilities present in the system.

5.2 Security Assurance Models

There are little research done on security assurance models. Especially quantifying security assurance is particularly difficult and diffuse.

According to NIST assurance is defined as following:

> "*(Security) Assurance - Grounds for confidence that the other four security goals (integrity, availability, confidentiality and accountability) have been adequately met by a specific implementation. "Adequately met" includes (1) functionality that performs correctly, (2) sufficient protection against unintentional errors (by users or software), and (3) sufficient resistance to intentional penetration or bypass[64].*"

A security requirement or a countermeasure will give protection (referring to point 2), and to check if the functionality performs correctly (referring to point 1) there is necessary to check if a security mechanism is present, which is doable because it is something that can be verified. While by doing security testing for weaknesses it is possible to check sufficient resistance to intentional penetration or bypass. Definition of the various metrics, that will measure the security assurance, is described in subsections below.

5.2.1 Requirement & Vulnerability Metrics

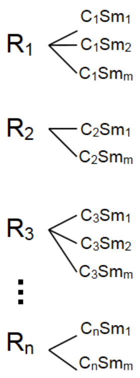


Figure 4: Requirements and their relation to security mechanisms

The figure 4 shows how requirement R_1 can be claimed to be fulfilled by checking the presence of the related security mechanisms C_1Sm_1, C_2Sm_2, etc. C is short for check and Sm is short for security mechanism.

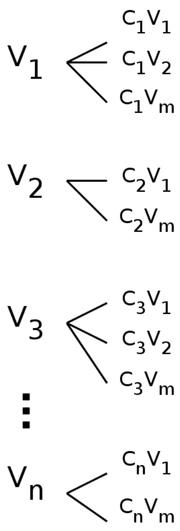


Figure 5: Related vulnerabilities are sorted into vulnerability categories.

The figure 5 is showing vulnerability categories with the related vulnerabilities included. C is short for check and the capital V is short for vulnerabiltiy category, while the lowcase v is short for vulnerability.

The metrics for measuring the score of Security Assurance can be defined as following.

- AM : Assurance Metric
- RM : Requirement Metric
- VM : Vulnerability Metric

The relation between these metrics is as following:

$$AM = RM - VM \tag{5.1}$$

To obtain AM, the VM must be subtracted from the RM. The RM and VM is defined below.

$$RM = \sum_{i=1}^{m} W_i \cdot \sum_{j=1}^{n} Cr_i Sm_j. \tag{5.2}$$

Equation 5.2 is the definition of RM, where **W** is weight. And *m* is the total number of requirements. **Cr** is abbreviated from check a requirement i, and each security mechanism j associated with the requirement i. Further *n* is the total number of security mechanisms. The variable in the figure 4 has been added in the equation 5.2 as $Cr_i Sm_j$. For each requirement I have to check all the mechanisms that are present. Next, each requirement must be weighted with a number reflecting the importance of the requirement.

$$VM = \sum_{i=1}^{m} R_i \cdot \sum_{j=1}^{n} Cv_i V_j. \tag{5.3}$$

In equation 5.3 VM is defined where **R** is the risk value for the specific vulnerability category, **Cv** is abbreviated from check vulnerability category, and for each category the associated vulnerabilities V_j must be checked if they are found. The variable *m* is the total number of vulnerability categories, and *n* is number of vulnerabilities in each category. Finally, each vulnerability category must be given a risk score.

5.2.2 Existence score

For each present mechanism an existence score must be assigned. A mechanism is either present or not present, at first a score of 0 and 1 were considered. However, in reality when the mechanisms was controlled it was acknowledged that there were mechanisms that were not performing as they should. To reflect this state of nature a score of *0.5* was included. The options for existence score were set to *P-score*={0, 0.5, 1}. P is short for *presence* (of a mechanism). The scores express:

- 0: Not Present
- 0.5: Partly Present
- 1: Present

Likewise, the vulnerability score *VE-score*={0, 0.5, 1} reflects the existence of the vulnerability. Zero signify that the vulnerability is not present, 1 is set for vulnerabilities that are found. Here 0.5 is applied if its dubious the vulnerability is able to be harmful.

5.2.3 Weights

Weighting is explained as

> *"The process of weighting involves emphasizing the contribution of some aspects of a phenomenon (or of a set of data) to a final effect or result, giving them more weight in the analysis. That is, rather than each variable in the data contributing equally to the final result, some data are adjusted to contribute more than others[65]."*

In relation to security, not all security requirements are equally important. That depends merely on the functions of the API and what it does and if it process sensitive data? The weights will express how important an security requirement is and it must be done according to what we want to protect.

A scale from 0-10 will aid to express the levels of importance. Where 0 is assigned to requirements that are meaningless and futile, and 10 is the maximum expressing a vital requirement.

5.3 Risk Models

The term risk was defined in the chapter 4. To pursue the definition, a risk is an impact of uncertainty on systems, organizations etc. Further risk management is the management of this impact, where the purpose is to protect against the threats. Whilst, risk assessment is the activity where the risks are identified, analyzed and articulated to the decision makers. There are several frameworks or methods for risk analysis, and organizations may choose their method depending on the type of risks they encounter or their business area[63].

Two risk models were considered for this thesis work, CVSS (Common vulnerability Scoring System) and DREAD-model.

5.3.1 DREAD

DREAD is a model for classifying and quantifying risk developed by Microsoft[66]. DREAD stands for the first letters of damage, reproducibility, exploitability, affected users and discoverability. Further in OWASP explains in detail[66]:

- **D**amage: How much damage will it create if a vulnerability is exploited.
    - 0=Nothing
    - 5=Individual user data is affected
    - 10=Complete system
- **R**eproducibility: This tells how easy it is to reproduce the threat exploit.
    - 0=Very hard or impossible, even for administrators of the application.
    - 5=One or two steps required, may need to be an authorized user.
    - 10=Just a web browser and the address bar is sufficient, without authentication.
- **E**xploitability: This tells what is needed to exploit the threat.
    - 0=Advanced programming and networking knowledge, with custom or advanced attack tools.
    - 5=Malware exists on the Internet, or an exploit is easily performed, using available attack tools.
    - 10=Just a web browser
- **A**ffected Users: How many users are affected.
    - 0=None

  - 5=Some users, but not all
  - 10=All users

- **D**iscoverability: How easy is it to discover this threat.
  - 0=Very hard to impossible; requires source code or administrative access.
  - 5=Can figure it out by guessing or by monitoring network traces.
  - 8=Details of faults like this are already in the public domain and can be easily discovered using a search engine.
  - 10=The information is visible in the web browser address bar or in a form.

According to various research work[67] it was noticed that there is little research done on DREAD. During this thesis work the most helpful article was from Owasp[66]. The algorithm in DREAD model uses, assigns a score from 0-10 to each five categories, and then getting the average score for that particular threat. See appendix for more information.
The risk score is calculated as following: $Risk\_DREAD = (Damage + Reproducibility + Exploitability + AffectedUsers + Discoverability)/5$

The category of Discoverability has been a discussion topic where many has debated its value. Discoverability says something about how easy it is to discover this threat. In some cases this has always been set to a constant 10, indicating that it will always be discovered while doing security review[66].

In other cases this category is not set to a constant[67]. Others has moved to DREAD-D, where the Discoverability has been omitted as they feel it encourage *Security by Obscurity*[68]. If we compare the DREAD model with the common risk matrix (impact*probability), the Damage potential and Affected users can be recognized as the Impact, and Reproducibility, Exploitability, and Discoverability can be recognized as the Probability. Notably Owasp[66] recommend to alter the formula, in a way that the impact score and the probability score is equal. If this is not done the probability score will weight more than the impact in the total score.

5.3.2 CVSS

CVSS stands for Common Vulnerability Scoring System[69], an it is a published standard used globally. It was designed by National Institute of Standard and Technology (NIST) with cooperation of industry partners.

CVSS helps to achieve a score, a decimal number, which reflects the vulnerability's severity, in the range of [0.0, 10.0]. The metrics are calculated in three different groups: base, temporal and environmental[69]. The two latter metrics are optional to use depending on the system and context.

1. Base metrics: Consists of two metrics, exploitability and impact metrics, which represents the intrinsic and fundamental characteristics of a vulnerability that do not change over time or across user environments. There is two groups within this metric. The first group focus on how the vulnerability is accessed[69]:

    - Attack vector (AV): measures if its possible to exploit the vulnerability locally or remotely. Options are [*network, adjacent, local, physical*].
    - Attack complexity (AC): measures what must exist beyond the attackers control to exploit the vulnerability e.g. software, hardware. Options are [*low,high*].
    - Privileges Required (PR): measures the privileges required to exploit a vulnerability. Options are [*none,low, high*].
    - User Interaction (UI): measures if user interaction is needed to exploit the vulnerability or the attacker alone can perform the attack. Options are [*none, required*].

    The other group of metrics focus on the impact of the vulnerability:

    - Scope: measures whether the exploit of the vulnerability will impact other components in addition to the vulnerable component. Options are [*Unchanged,Changed*].
    - Confidentiality impact (CI): measures the impact on confidentiality if the attack is successful. Options are [*none,low,high*].
    - Integrity impact (II): measures the impact on integrity if the attack is successful. Options are [*none,low,high*].
    - Availability impact(AI): measures the impact on availability if the attack is successful. Options are [*none,low,high*].

    The score 0.0 indicate None severity vulnerability. When the scale of the Base score is between 0.1-3.9, it indicates Low severity vulnerability. If its between 4.0-6.9 it indicates Medium severity vulnerability and 7.0-8.9 indicates High severity vulnerability. At the highest, when it is between 8.9-10.0 it is Critical severity vulnerability. The CVSS site[69] offers a calculator which can be used to calculate the score, so the analyst does not need to know the complex formulas in the background[69].

2. Temporal: This metric group reflects if the characteristics of the vulnerability changes over time[69].

- Exploit Code Maturity: measures the probability of the vulnerability to be attacked based on current state of exploit techniques and code availability. There are many type of malicious snippets available on the internet an attacker can use to exploit a vulnerability. Options are [*not defined, unproven,proof-of-concept, functional, high*]. The option *not defined* signify that the metric is not applicable and it will not affect base score if chosen.
- Remediation Level: This metric is important for indicating the prioritization of the vulnerability. Because when the vulnerability is first published, there is no fixed solution for it, and workarounds are often used as temporary remediation. The chosen option will reflect the urgency of the vulnerability. Options are [*not defined, official-fix, temporary-fix, workaround, unavailable*].
- Report Confidence: measures the degree of confidence that the vulnerability exists and the credibility of the known technical details. For instance sometimes the details of the vulnerability is not publicized because it is not known, only that the vulnerability exists. Later on it may worked on it and acknowledge it by the vendor etc. Options are [*not defined, unknown, reasonable, confirmed*].

3. Environmental: Reflects characteristics of vulnerabilities that are unique to a particular user's environment. This metric is composed by modified factors that control security controls that could reduce or raise the impact of the exploit. They can be categorized in two groups. The first group let the analyst to modify the base score depending on the importance of these requirements[69]:

   - Confidentiality
   - Integrity
   - Availability

   The other group is factors that need additional weighting to reflect the security of the specific environment. These factors are:

   - Modified Attack Vector
   - Modified Attack Complexity
   - Modified Privileges Required
   - Modified User Interaction
   - Modified Scope
   - Modified Confidentiality
   - Modified Integrity
   - Modified Availability

For instance the last three factors, modified confidentiality, integrity and availability were used in this study. The modification of the confidentiality requirement was done for the PX-API since its data was not private. Hence, the CVSS score decreased indicating a low threat to data corruption.

5.3.3 Evaluation of Risk models

At first the DREAD model was chosen due to its simplicity, as CVSS sounded complex and difficult to understand. However, it was done a risk-analysis with DREAD of the REST APIs with a manager at Statistics Norway and the experience was that DREAD is difficult to use in the real world. Especially the reproducibility and discoverability category was complex to evaluate for each vulnerability. It was necessary to convert the vulnerability to a threat since a vulnerability is not the same as a threat. The DREAD model is actually a threat-model. Based on this experience the CVSS was chosen instead.

CVSS base score, Temporal score and Environmental score was calculated for the both REST APIs. NIST has developed its own CVSS calculator[70] which was used in this study. The calculator makes it is easier to calculate the overall CVSS-score, because the temporal and environmental score is also included. Environmental score is usually added by the end-user[71], to adjust the score further respectively according to the user environment. This case will be explained in detail in chapter 6.

5.4 Scale Consideration & Normalization

The scale is dependent on number of requirements and vulnerabilities. The total number of requirements are 10 which has 53 mechanisms. Total number of vulnerability categories are 9 and containing 36 vulnerabilities. The maximum number of the weight a requirement can get is 10 and the maximum number of risk score is also 10. The maximum score an API can get is if every requirement is fulfilled:

$$10 \times 10 = 100 \tag{5.4}$$

For each vulnerability that is present, the score assigned to the vulnerability is subtracted from the total. If every vulnerability is present, and each of them is assigned max risk score, the total score will be

$$-9 \times 10 = 90 \tag{5.5}$$

Consequently, the minimum score is -90. The scale will range from -90 up to 100.

However, such large range is difficult to work with and interpret, so the score must be normalized to a common domain. A common domain used in scoring models in security, e.g. risk models is 0-10, therefore this range was chosen. It is possible to shrink a large scale to fit into a new, smaller scale. The method used here is min-max normalization [72, 73].

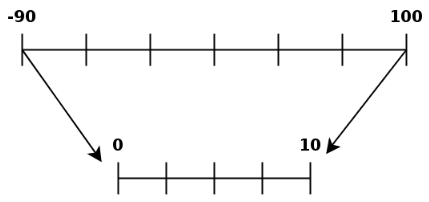


Figure 6: Normalize to different scale

$$v' = \frac{v - min_A}{max_A - min_A}(newmax_A - newmin_A) + newmin_A \tag{5.6}$$

The min-max normalization[73, p.4] equation is shown in equation 5.6. The min_A and max_A are minimum and maximum of an attribute, A, refering to the original variable A (in our case A=-90 to 100). The value v is mapped to value v' in the range of [$newmin_A$,$newmax_A$] using min-max normalization[74]. Quoting Shalabi & Shaaban, "*min-max normalization performs a linear transformation on the original data[75]*".

Table 2: Degree of Security

Score	**Degree of Security**
0.0 - 0.99	None Security
1 - 3.9	Low security
4.0 - 6.9	Moderate Security
7.0 - 8.9	Very Good Security
9.0 - 10.0	Excellent Security

The table 2 is adapted from the table of severity for vulnerabilities[70]. However, my table is showing the opposite, thus *levels of security*. This table can be used to convert the score to textual representation.

6 Case study & Results

A detailed description of the methodology can be found in the methodology chapter 3. This chapter describes how the quantification method was applied to two case studies, and the result that was obtained.

Following procedure is performed on two different REST APIs, PX-API and TS-API respectively. (Except sketching of the UML-diagram, which is only done for PX-API.)

1. Learning about the API. Sketching a UML-diagram of the API, that shows what the requests and responds look like.
2. Manual check: First a existence score that express whether a requirement is present or not by checking the related security mechanism.
3. Weighting: Each requirement get a *weight* that will reflect the importance of that requirement depending on the functionality of the API, and the security requirement.
4. Analysis of security testing result: Security vulnerability score will be checked after security testing both of the REST APIs.
5. Risk score: A risk analysis is done to get a vulnerability score.
6. Security Assurance: A total score of security assurance is calculated for the REST API.

When we have the two security assurance scores, we can compare these two values to evaluate how the quantification method worked.

6.1 Case study 1: PX-API

This section is particularly dedicated to PX-API. Referring to the section on case study 3.3 in chapter 3 where it is stated that the case study can use both quantitative and qualitative data. The quantitative data is for instance the existence score (P,VE) of requirements and vulnerabilities. However, the assessment of the risks and the importance of requirement is of a qualitative nature.

6.1.1 UML Diagram

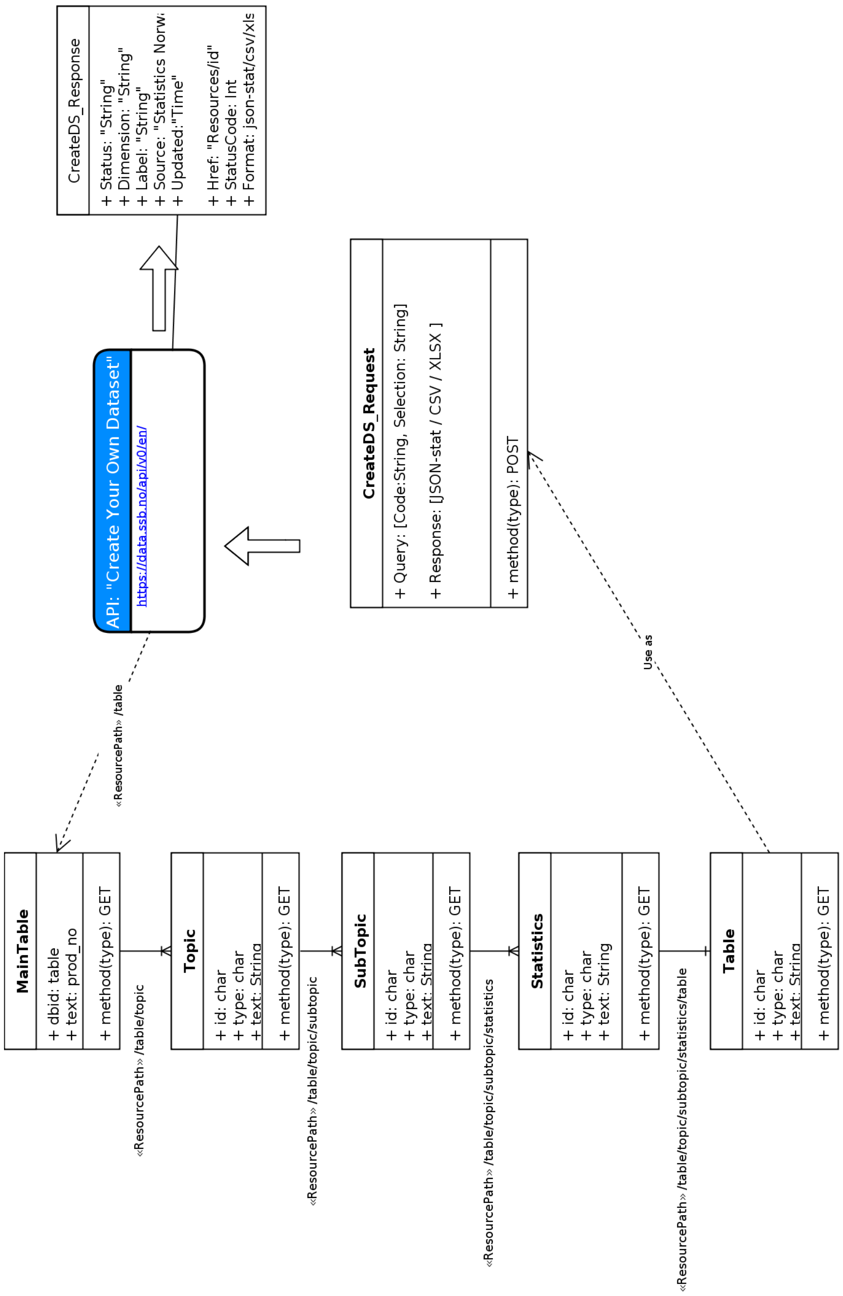


Figure 7: UML diagram of PX-API

Figure 7 is an UML diagram of the PX-API, and shows what is sent in the "request" and how the response looks like. The figure is adapted from the diagram for the twitter's REST API[76]. My UML-diagram mainly focuses on the request and response of the API. It is possible to navigate through the API using the URL. The structure is in general: table-topic-subtopic-table-statistics-table. For instance the url, http://data.ssb.no/api/v0/en/table/be list all the subtopics in the topic *be* which is *population*. The drilling is done with GET-method all the way down. That is shown on the left side of the diagram. If you know the table number, it is possible to acquire the metadata of that table. For instance, for table number 05000 write: http://data.ssb.no/api/v0/en/table/05000.

As mentioned earlier in the introduction chapter 1 there are two ways to use the API, the other way is to use the API-console. In the console you can run the POST query. It is only possible to acquire data with a POST query, GET method will only give metadata.

There are three possible output format to choose from:

- JSON-stat
- CSV
- XLSX

The response will deliver a table based on the query you build with the chosen variables. The diagram gives valuable information that can aid in the security testing process, as the diagram shows the API methods, input parameters, requests and response, as well as output format.

6.1.2 Requirements review of PX-API

For each requirement, their corresponding mechanism was checked mostly manually in cooperation with system architects and developers. Table 3 shows all the requirements and the associated security mechanisms that were present. P-Score is short for *Presence score*. As explained earlier for mechanisms that were present, a score of 1 is assigned. Zero was assigned for those which were not there. The numbered list of security mechanisms can be looked up in the Appendix A. For the first requirement *Authentication* there are no security mechanisms present. The reason behind this is simply because no authentication is done, due to the fact that this is an API open for public. The data that is acquired through the API is official statistics available for anyone.

Several security mechanisms are available for Input Sanitation, Error Handling and Communication Security, all of them has got a score of 1. The description of 7.2 can be found in the Appendix A stating:

> *"Verify that TLS is used for all connections (including both external and backend connections) that are authenticated or that involve sensitive data or functions,*

and does not fall back to insecure or unencrypted protocols. Ensure the strongest alternative is the preferred algorithm."

This has got 0.5 because the users are not redirected over to https, both http and https works.

Table 3: Displaying presence of security mechanisms for PX-API

Security Requirement	Security Mechanism(s)	P-Score
Authentication	1.1 -1.7	0
JWT	2.1 -2.3	0
Access Control	3.1-3.5	0
	3.6	1
Input Sanitation	4.1	1
	4.2	1
	4.3	1
	4.4-4.8	0
Error Handling	5.1	1
	5.2	1
	5.3	1
	5.4	0
Data Protection	6.1-6.6	0
Communication Security	7.1	1
	7.2	0.5
	7.3, 7.5 & 7.6	0
	7.4	0.5
HTTP Security	8.1	0.5
	8.2	1
	8.3-8.7	0
	8.8	0.5
Web Services	9.1-9.4	0
Cross Site-Scripting	10.1	1

Each requirement have different number of security mechanisms and this can give a skewed score if its not normalized. For all the requirements to weight equally, average of the score is used[77]. The average of the scores is then multiplied with the weight. The weight reflects the importance of that requirement. The table 3 is a table showing the total score with weights.

Table 4: Total Requirement Score (RM) for PX-API

Security Requirement	Tot. Score	Tot. Mechanisms	Average (A)	Weight (W)	Result (A*W)
Authentication	0	7	0.00	0	0.00
JWT	0	3	0.00	0	0.00
Access Control	1	6	0.17	5	0.83
Input Sanitation	3	8	0.38	8	3.00
Error Handling	3	4	0.75	7	5.25
Data Protection	0	6	0.00	0	0.00
Communication Security	2	6	0.33	5	1.67
HTTP Security	2	8	0.25	6	1.50
Web Services	0	4	0.00	0	0.00
Cross Site-Scripting	1	1	1.00	7	7.00
Sum					19.25

Application of Weights

Table 4 is showing the total score and total mechanisms for each requirement, average, weight and the result. (Total mechanisms is total security mechanisms that is possible to obtain for each requirement, this can be looked up in Appendix A) The average is presented as rounded numbers, but the result is calculated based on the decimal number. This also applies to the numbers in table 9. Although the *authentication* was initially considered for PX-API by the developers, it was discarded as it was considered to be less user friendly due to the fact that the data is official. The security requirement *authentication* get 0 weight score for this API. The API also do not need data protection as the data is not confidential official data, hence the weight here is also 0. *JWT* also gets a 0 weight. All of the above mentioned controls got 0, because they are considered as irrelevant. However, it was discussed whether it could be useful if there were some security mechanisms implemented for *Web Services*. But since there were none, the weight is deliberately set to 0. It is worth mentioning that even though the weight would have been higher it would not matter since there is no security mechanisms present.

The requirement *Access Control* has a weight of 5, again the reason for the low weight is because the data is official.

Input Sanitation and Error Handling are two important security requirements. Input must be validated before it is used, to confirm that it does not contain malicious code which can be harmful to the system[2]. The intention behind *Error Handling* is to give useful reaction to users and administrators in case of errors[2]. These requirements get 8 and 7 respectively.

The requirement of communication security is given the weight 5 because in communication security mostly the countermeasures involves around the enforcement of TLS-protocol. Since the data is not confidential the weight is low.

Next, HTTP Security is essential for any API sine API interface is based on the HTTP protocol[78]. But since the API has official data therefore the weight is 6 for this requirement.

Last the Cross site scripting defence is important because even though the API itself does not have a confidential data, a XSS-attack can inject malicious script to the web-page and acquire other confidential information retained by the browser[79]. The weight is 7 for this requirement.

The requirement fulfillment score for the different requirements will certainly differ depending on the weight given to each of them.

6.1.3 Vulnerability inspection of PX-API

The table 5 displays all the vulnerabilities found for the PX-API after completing the security testing process. The VE-score is the existence score for a vulnerability, that either can be 0, 0.5 or 1. During a security test the tester can find a vulnerability or s/he don't find any, sometimes the vulnerability is found but the vulnerability is not fully exploitable.

Table 5: Displaying vulnerabilities for PX-API

Vulnerability Category	Vulnerability Description	VE-Score
Injection Attack	1.1-1.3	0
Broken Authentication	2.1-2.7	0
Sensitive Data Exposure	3.1 3.2 3.3	1 1 0
Broken Access Control	4.1-4.12	0
Elevation of Privilege	5.1	0
Cross Site Scripting	6.1-6.3	0
Cross Site Request Forgery	7.1	0
Parameter Tampering	8.1-8.5	0
Man-in-the-middle-attack	9.1	0

The vulnerabilities found was mainly misconfiguration of TLS protocol. The VE-score was set to 1 for vulnerabilities that exists and 0 for those which were not found.

The details explaining the uncovered vulnerabilities can be looked up in Appendix B.

6.1.4 Risk Modeling of PX-API

After the vulnerability inspection the risk scoring is performed. There is not any fixed answer to risk appetite. Every organization need to define the level of risk they are ready to accept. The skill to quantify risk, will depend on the knowledge and experience of the team performing the risk scoring[80]. The risk score can be reduced, but that does not mean that the risk itself has reduced. It is essential to understand the causes behind the risk score. Ultimately the real risk must be tackled[81].

Table 6: Risk Scores for PX-API

Metrics		**Vulnerability**
		Sensitive Data Protection
Exploitability Metrics	AV	Network
	AC	Low
	PR	None
	UI	None
Impact Metrics	Scope	Unchanged
	C	Low
	I	None
	A	None
Temporal Metrics	E	High
	RL	Official Fix
	RC	Confirmed
Environmental Metrics	CR	Low
CVSS Base Score		5.3
CVSS Temporal Score		5.1
CVSS Environmental Score		4.4
CVSS Overall		4.4

Table 6 shows risk analysis of the exposed vulnerabilities. The abbreviated row names (AV,AC,PR,UI,etc.) are explained in the subsection 5.3.2. Just to explain, for the Exploitability Metric the attack vector (AC) is set to Network, Attack Complexity (AC) is set to Low, etc. The impact on confidentiality (Impact Metrics, Scope:C) is set to *low* since the API data is public. It is possible to decrypt the data, since encryption is not enforced. However, the data captured by the attacker will not be of sensitive nature. The temporal metrics does impact the overall score because mainly a fix is available. The score would have increased if the exploit did not have any fixes available. Since the requirement of confidentiality for this API was low, I adjusted the score further with the help of the Environmental Metrics (CR=low).

The CVSS calculator[69] specify this option as:

> *"There is some loss of confidentiality. Access to some restricted information is obtained, or the amount or kind of loss is constrained. The information disclosure does not cause a direct, serious loss to the impacted component."*

Table 7: Total Vulnerability score (VM) for PX-API

Vulnerability	**VE-score**	**Tot. vul. in cat.**	**Average**	**Risk**	**Average*Risk**
Sensitive data protection	2	3	0.67	4.4	2.93

Table 7 is compiled from table 5 and 6, showing the average VE-score and the risk attached to these vulnerabilities. The second column shows total number of vulnerabilities in the category. The risk is set to 4.4 for the vulnerability category because the impact of these risks is low due to the data is not confidential. As we can see the score 4.4 is obtained from the CVSS overall score.

6.1.5 Security Assurance Score (AM) for PX-API

The security assurance score can be calculated using the metrics **AM**. The defintion can be found in the section 5.2.1. To summon the AM was calculated by subtracting the VM metric from the RM metric, as also depicted on the figure 3. Hence, It will give a value of:

$$19.25 - 2.93 = 16.32 \tag{6.1}$$

On the original scale [-90,100] the PX-API gets a value 16.32. This score need to be normalized to a new range from 0 to 10, as explained in chapter 5.

The values that needs to be filled in is as following, min_A : -90, max_A = 100, new_max$_A$:10, new_min$_A$:0. And υ : is the AM-value:16.32.

$$\begin{aligned}\upsilon' &= \frac{16.32 - (-90)}{100 - (-90)}(10 - 0) + 0 \\ &= \frac{106.32}{190} \times 10 \\ &= 0.5595 \times 10 \\ &= 5.595 \approx 5.60\end{aligned} \tag{6.2}$$

All calculations has been done with decimal numbers, even if they are shown as rounded numbers in the equation. The PX-API got 5.60, which in our table of security degree 2 falls between 4.0-6.9, evaluated as *Moderate Security*.

6.2 Case Study 2: TS-API

This section address the case study of TS-API.

6.2.1 Requirement check

TS-API is different from the PX-API, hence its security requirements are not weighted similarly. First of all it is not a public API. An application use this API to transform data, and the API does not have data of its own. The API has authentication controls and Access Controls enforced as it is needed for countermeasure against unauthorized users. Many of the mechanisms for authentication requirement is fulfilled, but not all. The most important security objective of this API is *integrity*. The reason is the data processed in the API is not confidential but if the data is changed and its recipients (users/applications) are changed, it can cause a lot of damage, including damaging the company's reputation. Input sanitation is equally important requirement for this API, due to the same reason as mentioned above. It should not be possible to change the data.

All mechanisms for Error Handling is on place, which is a good thing. Often this requirement is not taken seriously, but it is as important as other requirements. Not just to know the status of the request, but also that the error can be fixed in time. If there is no feedback from the system, the user can assume that the request has been successfully executed. This is particularly important when the API is suppose to handling request from applications. Else the whole system can hang, and it is difficult for the developers to locate the error[82].

Data protection is less important because there is no confidential data. However, all data is sent to the server in the HTTP message body or headers because of the integrity requirement.

Table 8: Displaying presence of security mechanisms for TS-API

Security Requirement	**Security Mechanism(s)**	**P-Score**
Authentication	1.1	1
	1.2-1.4 & 1.7	0
	1.5	1
	1.6	0.5
JWT	2.1 -2.3	0
Access Control	3.1	1
	3.2	1
	3.4	1
	3.3, 3.5, 3.6	0
Input Sanitation	4.1	0.5
	4.2	1
	4.3-4.7	0
	4.8	1
Error Handling	5.1	1
	5.2	1
	5.3	1
	5.4	1
Data Protection	6.1	1
	6.2-6.6	0
Communication Security	7.1-7.6	0
HTTP Security	8.1, 8.3, 8.5-8.8	0
	8.2	1
	8.4	1
Web Services	9.1, 9.3	0
	9.2	1
	9.4	1
Cross Site-Scripting	10.1	0

Table 8 shows the presence of the different security requirements. The score 1 is given if the requirement is present and functioning properly. Some of the requirements are given 0.5, because there is an uncertainty regarding the requirement's functionality. For instance for *4.1: "Verify that server side input validation failures result in request rejection and are logged."* It is not sure that it is logged adequately. For them to achieve score of 1, some improvement is needed in the implementation.

Table 9: Total Requirement score (RM) for TS-API

Security Requirement	Total Score	Total mechanisms	Average (A)	Weight (W)	Result (A*W)
Authentication	2.5	7	0.36	10	3.57
JWT	0	3	0.00	0	0.00
Access Control	3	6	0.50	10	5.00
Input Sanitation	2.5	8	0.31	9	2.81
Error Handling	4	4	1.00	10	10.00
Data Protection	1	6	0.17	4	0.67
Communication Security	0	6	0.00	0	0.00
HTTP Security	2	8	0.25	7	1.75
Web Services	2	4	0.50	6	3.00
Cross Site Scripting	0	1	0.00	0	0.00
Sum					26.80

Table 9 is showing the total security mechanisms present, and the given weight for each of the security requirement. The weight is given according to the importance of the requirement. For instance Authentication and Access Control is one of the most essential parts needed for protection. Data Protection has a low score because the data itself is not of confidential nature. Even if an outsider is able to get access to this data, s/he will not be able to do any harm.

6.2.2 Vulnerability check

Table 10 is showing the vulnerabilities that were found for TS-API during the security testing. Three main vulnerability categories are affected: broken authentication, broken access control and elevation of privilege.

The vulnerability 4.12:API key compromise has got 0.5 as score. Similarly to the requirement score this depicts as weak vulnerability, meaning not fully executable. It was found that even if it was possible to aquire the API key and get access to the API. It was not possible to change the data/recipient and act further.

6.2.3 Risk Modeling

The description of the vulnerability category *Broken Authentication* can be found in the Appendix B. The vulnerabilities found in this category were:

2.2 Permits Brute Force
2.5 Has missing or ineffective multi-factor authentication

They have the same risk score.

Table 10: Displaying vulnerabilities for TS-API

Vulnerability Category	Vulnerability Description	VE-score
Injection Attack	1.1-1.3	0
Broken Authentication	2.1, 2.3, 2.4, 2.6-2.7	0
	2.2	1
	2.5	1
Sensitive Data Exposure	3.1-3-3	0
Broken Access Control	4.1-4.11	0
	4.12	0.5
Elevation of Privilege	5.1	1
Cross Site Scripting	6.1-6.3	0
Cross Site Request Forgery	7.1	0
Parameter Tampering	8.1-8.5	0
Man-in-the-middle-attack	9.1	0

Table 11: Risk Scores for TS-API

Metrics		**Vulnerability**		
		Broken Authentication	Broken Access Control	Elevation of Privilege
Exploitability Metrics	AV	Network	Network	Local
	AC	Low	Low	Low
	PR	None	Low	Low
	UI	None	None	None
Impact Metrics	Scope	Unchanged	Unchanged	Unchanged
	C	None	None	None
	I	High	High	High
	A	None	None	None
Temporal Metrics	E	High	High	Functional Exploit exists
	RL	Official Fix	Official Fix	Official Fix
	RC	Confirmed	Confirmed	Reasonable
Environmental Metrics	IR	High	High	High
CVSS Base Score		7.5	7.5	5.5
CVSS Temporal Score		7.2	7.2	5.1
CVSS Environmental Score		8.9	8.9	6.8
CVSS Overall		8.9	8.9	6.8

The two vulnerabilities must be seen together to evaluate the risk because they are in the same category. A brute force attack[83]is when an attacker can send all kind of combinations of passwords to forcefully get access to an account. This is possible because there is no account lockout policy enforced and/or two-factor authentication. Next vulnerability is missing two-factor authentication, which again makes it possible to do a brute force attack or simple phishing[84]. This is a good example of how risk is evaluated based on the category instead of a single vulnerability.

Table 12: Total Vulnerability Score (VM) for TS-API

Vulnerability	**VE-score**	**Tot. vul. in cat.**	**Average**	**Risk**	**Average*Risk**
Broken Authentication	2	7	0.29	8.9	2.54
Broken Access Control	0.5	12	0.04	8.9	0.37
Elevation of Privilege	1	1	1.00	6.8	6.8
Sum					9.71

Table 12 is showing calculation of the Vulnerability score.

6.2.4 Security Assurance Score (AM) for TS-API

As mentioned earlier the score of AM is found by subtracting the VM metric from the RM metric. In this case it will give a value of:

$$26.80 - 9.71 = 17.09 \tag{6.3}$$

On the original scale [-90,100] the TS-API gets a value 17.09 Now we want to plot the AM value to a scale of 0-10. Just like the first API.

$$v' = \frac{v - min_A}{max_A - min_A}(new\_max_A - newmin\_A) + new\_min_A \tag{6.4}$$

The values that needs to be filled in is as following, min_A : -90, max_A = 100, new_max$_A$:10, new_min$_A$:0. And v :17.09 is the AM-value.

$$\begin{aligned} v' &= \frac{17.09 - (-90)}{100 - (-90)}(10 - 0) + 0 \\ &= \frac{107.09}{190} \times 10 + 0 \\ &= 0.5636 \times 10 \\ &= 5.636 \approx 5.64 \end{aligned} \tag{6.5}$$

All calculations has been done with decimal numbers, even if they are shown as rounded numbers in the equation. The TS-API got 5.64, which in our table of security degree 2 falls between 4.0-6.9, evaluated as *Moderate Security*.

6.3 Summary Result

PX-API had 2 vulnerabilities and 12 security mechanisms present. TS-API on the other hand had 3.5 vulnerabilities and 17 security mechanisms present. The PX-API got the AM-score 5.60 and the TS-API got the AM-score 5.64. A detailed discussion on the results can be found in chapter 7.

7 Discussion

In this chapter I will discuss the framework, the process and the result I got.

7.1 Definitions

In chapter 4, security requirement is defined as a representation of the expected security functionality of a security mechanism. Whilst a vulnerability was defined as a special fault in the security mechanism that was able to be exploited and lead to a risk.

It can be argued that there is no difference in a missing requirement and a present vulnerability. So it is futile to check both. If one missing requirement is recognized as a present vulnerability, a present requirement can be recognized as a vulnerability that does not exist.

This can be true in some sense but there are some aspects that should be reflected on. It was observed that the relation between a security requirement and a vulnerability was not a 1 to 1 relation. One security requirement may fix one to many vulnerabilities, and a single vulnerability may needs many requirements to get fixed.

Feng et al. confirm in their paper

> *"...there often exist complex interactions among the components of information systems. Therefore, any single vulnerability may have multiple propagation paths, leading to different security risks in information systems"[85].*

Lets take one security requirement as an example. TLS protocol is an security requirement. Further, missing or wrongly configuration of this security requirement can lead to a man-in-the-middle attack, a cross-site scripting attack. etc. But a man-in-the-middle attack is not only possible because of missing a TLS, there are several methods to do so such as sniffing, packet injection, session hijacking and SSL-hijacking, etc.[54]. Depending on various vulnerabilities e.g. missing HTST, deactivated certificate validation[86, 54], etc.

7.2 Framework & Method

In chapter 2 I have rendered a section on *Related Work*. What I have concluded with that it is little or no research on how to security test, quantify or measure security of

REST APIs. In general there are little research on quantification of security, where security of a system is measured. I have observed that a lot of authors have suggested security testing or security application as early as possible in the software development cycle. Which is a positive trend, but the pitfall is that the security assurance is not given the attention it needs on later stages. Because with new technologies comes new vulnerabilities and attacker tries to exploit these all the time. The framework I have proposed consisting of a method to quantify security provides helpful metrics. According to Kahra[36] metrics can be used to observe the *security* over a time period and the security performance. This framework is not bound to just facilitate a REST API. In general, it can be utilized for any system or domain. Provided that the list of requirements and vulnerabilities is adjusted according to the system.

The framework differs also in the chosen approach and perspective. While other security testing methods aim to address the vulnerabilities, in my thesis I have suggested to include verification of requirement fulfillment. As I see it, there are several advantages of doing so. The presence of requirements are checked manually before the security testing is done. Afterwards the security tester can cross-check the vulnerabilities with the related requirement(s) and conclude that they are not working as expected.

I also observed that other methods only include risk analysis in their process and often they lack requirement analysis. Mostly, requirement analysis is done early in the software development process, and it is not a part of the security testing. I experienced that by adding requirement checking it helped to prioritize what should be protected. The weight could give an additional indication on which requirement was important for the particular API. In total, all of this could be acknowledged as a basis on what to prioritize in implementing the countermeasures. After all the security requirements and vulnerabilities are aspects of the same picture depicting the security assurance level of a system.

7.3 Results

After the requirements were counted and before the result of the calculation was done. The expectation was that the more of the requirements were on place the more secure the REST API is. Hence I was expecting a higher score for TS-API. For instance, it has authentication which is an important requirement that gives that extra layer of security. It was not found a lot of vulnerabilities either, only three. While the PX-API had less requirements on place and it was found two vulnerabilities for this API. This was before any weighting was done or risk analysis. The weighting and risk score is necessary contributors to the score, because the weight portray the importance of the requirement. The risk score on the other hand illustrate how harmful the vulnerability is. This is due to the fact that the security objective can differ from one system to another. Information security has three main objectives: confidentiality, integrity and availability[10]. The PX-APIs main security objective is the availability of the data. The data should be available to the user. There is no confidentiality needed as the data is public. The main security objective for the TS-API is the integrity of the data. Any attacker should not be able to change the data. The data itself is not confidential.

In a range of 0-10 where 0 is minimum and 10 is maximum, PX-API got 5.60 and TS-API got 5.64. According to this result it is indicating that the TS-API is more secure than the PX-API. To evaluate this claim it is necessary to look at all the factors that contributes to the score.

Many factors will impact the score. The presence of the security requirements and security vulnerabilities are counted. The accuracy of this counting depends on which scoring model for requirements and risk models for the vulnerabilities are used.

Requirements are assigned weights according to their importance. The approach used in this thesis was that every requirement was not equally important. For a REST API that for instance process confidential data, needs protection so the data is not leaked. Hence, the requirement for data protection and authentication is more important for the TS-API used in this study than the other one which is a public API. This is an important aspect of the score because the weight will not only change the score total but value an existence score more than others.

The weight assigned indicate how security is emphasized. If authentication is necessary to make a specific API secure, the importance of that will also be higher than other, hence the weight is also higher. For a requirement that does not consolidate the security, or its presence is not vital, it will get a lower weight. The weights I have given are making an impact on the score. If I change the weight the score will change accordingly. The range, under which the score will fluctuate, can be calculated. In our original approach the requirements are weighted differently according to their importance. In the example below I have changed the weighting to max, which represent the situation where each requirement is weighted equally (abbreviated as Eq. W in the table column).

Table 13: Comparison between original RM with distinct weights and new RM with equal weights for PX-API

Security Requirement	Average (A)	Weight (Org/Eq. W)	Original value	Eq. W
Access Control	0.17	5/10	0.83	1.67
Input Sanitation	0.38	8/10	3.00	3.75
Error Handling	0.75	7/10	5.25	7.50
Communication Security	0.33	5/10	1.67	3.33
HTTP Security	0.25	6/10	1.50	2.50
Cross Site - *	1.00	7/10	7.00	10.00
Sum			19.25	28.75

Table 14: Comparison between original RM with distinct weights and new RM with equal weights for TS-API

Security Requirement	Average (A)	Weight (Org/Eq. W)	Original Value	Eq. W
Authentication	0.36	10/10	3.57	3.57
Access Control	0.50	10/10	5.00	5.00
Input Sanitation	0.31	9/10	2.81	3.13
Error Handling	1.00	10/10	10.00	10.00
Data Protection	0.17	4/10	0.67	1.67
HTTP Security	0.25	7/10	1.75	2.50
Web Services	0.50	6/10	3.00	5.00
Sum			26.80	30.86

Next, we can find the relative difference between the original value and the maximum value in percentage using this equation[87, 88] (Relative Percentage Difference):

$$RPD = \left(\frac{X_1 - X_2}{X_{mean}}\right) \times 100 \tag{7.1}$$

X_1 and X_2 are the values we want to compare. X_{mean} is the average of the two values.

$$\begin{aligned} RPD_{PX} &= \left(\frac{28.75 - 19.25}{\frac{28.75+19.25}{2}}\right) \times 100 \\ RPD_{PX} &= 39.58\% \end{aligned} \tag{7.2}$$

$$\begin{aligned} RPD_{TS} &= \left(\frac{30.86 - 26.80}{\frac{30.86+26.80}{2}}\right) \times 100 \\ RPD_{TS} &= 14.08\% \end{aligned} \tag{7.3}$$

For PX-API the RPD is 39.58 %. For TS-API the RPD is 14.08 %. With these percentages, I know that the original weights for TS-API is closer to the value where requirements are weighted equally.
It is interesting to find out the span of the percentage difference. If the score scale is from 0-10, we must insert 0 and 10 into the equation.

$$\begin{aligned} RPD_{span} &= \left(\frac{10 - 0}{\frac{10+0}{2}}\right) \times 100 \\ RPD_{span} &= 200\% \end{aligned} \tag{7.4}$$

This means that the span of the percentage difference will be from [0%,200%].
As assigning of weights on the requirements contributes in disparity in the score, likewise the risk scoring of the vulnerabilities will do the same. In this study I have used CVSS model for risk analysis. The DREAD model was initially used which gave almost the same scores. However, the DREAD model was challenging in the sense that it was challenging to understand the terms *reproducibility* and *exploitability*. I felt that a better technical insight of the exploit was necessary in producing a reliable score for DREAD. The result of the DREAD model is given below.

Table 15: PX-API Risk Analysis Result with DREAD

Vulnerability:	Sensitive Data Protection
Damage	2
Reproducibility	7
Exploitability	7
Affected Users	1
Discoverability	10
Dread_Score	5.4

Table 16: TS-API Risk Analysis Result with DREAD

Vulnerability:	Broken Authentication	Broken Access Control	Elevation of Privilege
Damage	8	8	8
Reproducibility	9	5	8
Exploitability	9	4	8
Affected Users	10	10	10
Discoverability	10	10	10
Dread_Score	9.2	7.4	8.8

Table 15 and 16 are showing the scores derived from the DREAD-model, which does not differ so much from the score I got by using CVSS-model.

The CVSS score acquired from table 6 and 11, were:

- PX-API:
  - Sensitive Data Protection: 4.4
- TS-API:
  - Broken Authentication: 8.9
  - Broken Access Control: 8.9
  - Elevation of Privilege: 6.8

7.3.1 AM-score with Equal Weights

The AM-score with equal weights can also be interesting to calculate. In this example I have used the vulnerability score as it is, and maximum requirement score.

- PX-API: 28.75 - 2.93 = 25.82, plotted into range 0-10, using equation 5.6, gives new value 6.10
- TS-API: 30.86 - 9.71 = 21.15, plotted into range 0-10, using equation 5.6, gives new value 5.85

If I use equal weighting the PX-API gets a higher score than the TS-API. In this example I can aslo change the risk score, but I have deliberately kept it as it is. For instance I could have used the value 10 as risk for PX-API, but that would have given a wrong picture of the risk. The risk is low for PX-API due to the fact that the data is not confidential. The question is if its correct to use equal weighting if the security objectives does not require to do so? In this example it is proven that equal weighting of the requirements will give another score than expected. Does this means that the PX-API is more secure than TS-API? It is debatable, since PX-API needs less security the requirements were originally weighted with lower score than the TS-API. The theory is that requirements cannot be weighted equally, since the weighting must be done according to the security objectives of the system. For instance, PX-API does not have confidential data so the weight for data protection cannot be the same as requirements for availability which is the main security objective of the PX-API.

7.3.2 Human Factor

The conclusion on the result section is that the human factor is one of the most important part that contributes to the score. The score will fluctuate according to how the requirements are weighted and how the vulnerabilities are interpreted by the person evaluating. In addition the person searching for the requirements and vulnerabilities must have the knowledge and the right tools available to perform the testing to provide the raw data.

7.4 Scale Consideration

In chapter 5 I have explained the choice of normalize the score to a 0-10, instead of using the original, large scale from -90 to 100. I had a discussion with colleagues on which scale that was better suited for this work. Whatever scale is chosen, it should not be confusing to understand the score. First of all it is necessary to define what the scale range is, without it the score in itself does not make any sense. For instance if I say that the AM-score is 10.

To interpret the score you need to relate it to something, and figure out what does the score represent. If the scale is from 0 to 10, then 10 is max, hence it is the maximum possible number, great! If the is scale from 0 to 100, then 10 is not so impressing. On the original scale (-90 to +100) however, 10 is more than half of possible scores.

Other ranges were also discussed, such as a scale of -5 up to 5. Where 5 is max. Zero (0) is the middle value hence this can be defined as the middle value. A negative score can therefore be interpreted as alarming, and can be used to create attention among decision makers. The downside of using this scale is having 0 as a middle value. Zero is too often interpreted as nothing. As appealing it was to use negative numbers to display critical security, I did not like the idea of getting a *Security Assurance* score of zero, as it will lead to confusion.

Therefore in this study I chose to use a range from 0-10, because it is the most common scale in people's mind.

7.5 Analogy Example

Lets think of the REST API as a particular house. A thief can be compared to a malicious attacker.

An attack surface is the points where any thieves can enter the building, which in the analogy is the door and windows. Authentication is verifying that you are the person you are claiming to be. By having a key to your house you will have access to the house. This is one type of authentication mentioned earlier, you enter by using something you own.

The PX-API can be compared to an open building, like for instance a library that anybody can come inside. People come in and read the newspapers they are interested in etc. Nobody knows how many visited the library. May be not so important, some may argue. But it could be an advantage to know how many guests usually come on weekdays compared to the week-ends. In API world this can be related to availability of the service.

On the other hand TS-API is a house with a key that need authorized persons to enter. Since the API does not have confidential data there is nothing private inside it. So TS-API can be compared to a show house which a real estate agent use to show for potential buyers. The house does not have anything "private" in it. The house is decorated with beautiful furniture. Remembering that the TS-API had integrity requirements, the same should be applied here in the analogy. The furniture should not be changed or damaged. Because it can affect the visitors' opinion on the house, and thus the selling rate. Compared to the library the real estate agent knows how many has visited the house.

Input Sanitation is an important security requirement. Sanitizing is checking what is sent as a parameter into the API. In the analogy this can be compared to a potential customer visiting the show house. S/He is allowed to take his/her partner along, children etc. But it is obvious he cannot bring along a cigarettes, lighters, or other hazardous items that can deteriorate the indoor environment.

The same applies to the library. If a guest brings a lighter and burn down the library there is no newspaper available to read. Therefore though it is a public place, there should be some access control and input sanitation.

What can be derived from the above example is that if there is some control that checks who is entering the house or library it is definitely a safer place for everyone. In public places such as libraries there is a need for authentication and/or access control that can help controlling visitors. There is also possibility to install a visitor counting system especially made for libraries and museums. This can be compared to API keys or Access Tokens in APIs.

To conclude in general the security requirements such as Authentication, Access Control and input sanitation, are essential security requirements.

7.6 Qualitative Validation

To evaluate the method a formal questionnaire was sent to those who were involved in the process. In addition to the formal interview there were also done informal discussions during the process. In this section I have summarized the feedback I got from both the formal questionnaire and the informal discussion sessions. The answers from the respondents were quite similar.

7.6.1 Framework

Do you think the developed security assurance framework based on quantitative security assurance metrics is helpful, why and when? The response on this question was that a framework for security testing a REST API is definitely seen as something useful for many people in an organization. For a developer it can help as a guide on how to develop a secure REST API (by using the requirement list), and provide a secure product to the end user. A security tester can use the lists to perform the testing and hereby reduce the manual work. The lists can also be used to build expertise. In addition the organization can use them as a standard to follow for everyone. Thus, the different stages of the process can be completed separately or in succession as desired. Most importantly the end result, which this thesis has revolved around can be used to report to the management.

7.6.2 Requirements and Vulnerability List

Do you think that you will use the list of requirement/vulnerability when you security test/develop a REST API in the future? and why? A very positive feedback I got from management is that in future all REST APIs developed in the organization has to follow the security requirements in the list proposed in this thesis. Some requirements may not suit the API due to its functionality, but they can be omitted. Another feedback I got was that it was beneficial and agile of checking the requirements during security testing, which was not common. This new approach, in addition to the requirement list were highly appreciated in the organization.

7.6.3 Metrics

Do you think that dividing the assurance metrics into requirement and vulnerability elements is sound? Is it possible that a security requirement can be only partially fulfilled? (For the whole question see C) The response about the metrics revolved around that a metric that is able to measure the security can help the management to make better decisions. The management is aware of the difficulties around measuring security.

When requirements were checked it was discussed on how to assign a value on whether a particularly security mechanism was present or not. At first only 0 and 1 was included, because either a mechanism is present or it is not present. However, it was necessary to set some of the presence score to 0.5, because they were not functioning properly or as expected. I got confirmation from colleagues that this was a good solution when such cases were discussed. Also, it was pointed out that in this case a smaller scale with three options (0, 0.5, 1) is easier to assign. On a larger scale it is difficult to interpret what the value means.

7.6.4 Other feedback

Do you have any other feedback to the developed security assurance framework. The feedback I got from the management was that the proposed framework was satisfactory and future aim for the security work is to develop similar frameworks with requirement lists and vulnerabilities for other domains.

Prior to the testing the expected security level of the REST APIs was discussed. The TS-API with authentication controls were likely to be more secure, but it was also emphasized that even though there is no confidential data in the PX-API, the API should be secure enough so that it live up to the company's credibility. A similar AM-score for the two REST APIs intend to express exactly this scenario.

8 Conclusion

The main goal of this work has been to design a quantification method which can help to estimate assurance levels of REST APIs. The current evaluation methods and standards, like the Common Criteria, tends to achieve their goals manually, to a large extend, and require a lot of resources in term of time, cost, and effort. Furthermore, there has been little research on REST APIs. The quantification method proposed in this thesis consist of an semi-automated process where the security tester can follow a step-by-step guide and achieve a score that will reflect the security assurance level of the REST API.

The first research question aim to answer how the security assurance value can be calculated with the help of a quantification method. Chapter 4 documents the complete process on how the quantification is done with the help of metrics. Chapter 5 specifically deals with the metrics. A security metric is a metric that depicts the security level, security performance or security strength of a system. Usually expressed by a numerical score. The desired security assurance value is calculated with the help of an assurance metric (AM).

The AM reflects the security confidence a system may possess, and it is defined as:

$$AM = RM - VM$$

RM is the requirement score corresponding to the security mechanisms that are present and the weight assigned to the particular requirement. VM is the vulnerability score, comprised by whether the vulnerabilities exist and a risk score that shows the harmfulness of these. It was observed that the RM varied according to the weight applied. Likewise for the VM, which was impacted by the risk score.

The second research question focus on whether it is feasible to automate the security control process. It was recognized that the manual labour could be reduced by complementing the process with the lists of requirements and vulnerabilities. The security requirements and vulnerabilities for REST APIs were compiled from security issues for web-applications that were pertinent for REST APIs. It was specified 53 security mechanisms for 10 requirements, and 36 vulnerabilities which were sorted into 9 categories.

The third and last research question concentrate on if the method met its intended purpose. To answer this question, two REST APIs were tested and their scores were compared to each other. In addition, a questionnaire was sent to validate the method.

It was expected that the TS-API will achieve a higher score than the PX-API, since it had more security mechanisms implemented. However, the difference in the score was not as significant as expected. The TS-API got 5.64 while the PX-API got 5.60. The reason was that even though the TS-API had more security mechanisms the vulnerabilities found for this API was categorized as more harmful than the vulnerabilities for PX-API. This lead to a reduction in the total score. In addition, PX-API had public data that were not confidential, and therefore the requirement of the confidentiality had been adjusted down with the *environmental score* in the CVSS score model. In the example with application of equal weighting it is demonstrated that the requirements cannot be weighted equally, if it is not required to do so. The weights must be assigned according to the security objectives of the system.

To validate the method a questionnaire were sent to those involved. These were security testers, developers and managers. The framework was assessed to be quite useful, both in parts and also as a complete process. The requirement list compiled from the OWASP was highly appreciated, and it was recommended as a guidance to help security testers to check if their REST API is up-to-date on security trends. The approach of testing the requirements in addition to the vulnerabilities were seen as a smart and useful way of performing the security testing. Finally, it was acknowledged that a metric that was able to *measure* security could be useful to make better decisions. Prior to the testing the expected security level of the REST APIs was discussed. The TS-API with authentication controls were likely to be more secure, but it was also emphasized that even though there is no confidential data in the PX-API, the API should be secure enough so that it live up to the company's credibility. A similar AM-score for the two REST APIs intend to express exactly this scenario.

In general, this process can be used on any system, since the process itself is not domain specific. Only the list of requirements and vulnerabilities are.

8.1 Future Work

During scientific research many questions are answered but also new one rises. Some of the question raised in this paper will be summarized here. This research was done on two REST APIs that had quite different security objectives. In short:

- Authentication: PX-API is an open API. TS-API need authentication to access.
- Data Protection: PX-API had open public data. The TS-API did not have data, but instead had integrity requirements
- Communication Security: Only TS-API had TLS enforced, even though it was recommended for PX-API also.

Due to the fact that the AM-score were not distinctively far from each other. It would be very interesting to see how another test cases with same security objectives will influence the AM-score. Ideally, there could be test cases where both has same security requirements (e.g. authentication controls, access controls etc.).

Another inquisitive test could be to modify the requirement and vulnerability lists for a different application domain. In my opinion with little effort the method can be customized to any system.

It was observed during the sorting of the vulnerabilities in categories that the vulnerabilities could be sorted further into subcategories. This work could help to ease the work of the security tester to evaluate the risk of the vulnerabilities.

Even though it is not feasible to map a requirement to each and every related vulnerability. A research on mapping requirements to vulnerability categories could be beneficial for the quantification method.

Bibliography

[1] Newby, G. February 2002. The orange book and common criteria. https://www.petascale.org/inls187/notes/Spring2002/February20/orange.html.

[2] Manico, J. July 2016. Open web application security project. https://www.owasp.org/images/3/33/OWASP_Application_Security_Verification_Standard_3.0.1.pdf.

[3] Savola, R. M. 2013. Quality of security metrics and measurements. *Computers & Security*, 37, 78 – 90. URL: http://www.sciencedirect.com/science/article/pii/S0167404813000850, doi:https://doi.org/10.1016/j.cose.2013.05.002.

[4] Haddad, S., Dubus, S., Hecker, A., Kanstrén, T., Marquet, B., & Savola, R. Sept 2011. Operational security assurance evaluation in open infrastructures. In *2011 6th International Conference on Risks and Security of Internet and Systems (CRiSIS)*, 1–6. doi:10.1109/CRiSIS.2011.6061831.

[5] NorSiS. October 2017. Nasjonal sikkerhetsmåned 2017. https://www.youtube.com/watch?v=X-DvBaXs4FY. Youtube video from conference.

[6] Merriam-Webster. Thesaurus - automated (adjective). https://www.merriam-webster.com/thesaurus/automated. Merriam-Webster Thesaurus since 1828.

[7] Statistics-Norway. May 2018. About us. https://www.ssb.no/en/omssb/om-oss.

[8] Norway, S. *StatBank API User Guide*. Statistics Norway, April 2017. http://www.ssb.no/en/omssb/tjenester-og-verktoy/api/px-api/_attachment/248250?_ts=15b48207778.

[9] Statistics-Norway. January 2014. General terms of use for statbank. https://www.ssb.no/en/informasjon/om-statistikkbanken/general-terms-of-use-for-statbank.

[10] Metivier, B. April 2017. Fundamental objectives of information security: The cia triad. https://www.sagedatasecurity.com/blog/fundamental-objectives-of-information-security-the-cia-triad. Sage Data Security.

[11] Mulesoft. 2018. What is an api? (application prorgramming interface). https://www.mulesoft.com/resources/api/what-is-an-api.

[12] technologies, C. September 2014. Protecting your apis against attack and hijack. https://www.ca.com/content/dam/ca/us/files/white-paper/protecting-your-apis-against-attack-and-hijack.pdf.

[13] Khandelwal, N. July 2015. Opinion: Gaping holes in security of apis. https://securityledger.com/2015/07/opinion-gaping-holes-in-security-of-apis/. The Security Ledger.

[14] Oftedal, E., van der Stock, A., Chih, T. H. H., Peeters, J., Wolff, J., & Granitz, R. November 2017. Rest security cheat sheet. https://www.owasp.org/index.php/REST_Security_Cheat_Sheet.

[15] Andrew, H. September 2012. Beginners guide to creating a rest api. http://www.andrewhavens.com/posts/20/beginners-guide-to-creating-a-rest-api/.

[16] Paik, H.-y., Lemos, A. L., Barukh, M. C., Benatallah, B., & Natarajan, A. *Web Services – REST or Restful Services*, 67–91. Springer International Publishing, Cham, 2017. URL: https://doi.org/10.1007/978-3-319-55542-3_3, doi:10.1007/978-3-319-55542-3_3.

[17] Baeldung. July 2017. An intro to spring hateoas. http://www.baeldung.com/spring-hateoas-tutorial.

[18] Sahni, V. June 2015. Best practices for designing a pragmatic restful api. https://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api#restful.

[19] Beer, M. I. & Hassan, M. F. Nov 2017. Adaptive security architecture for protecting restful web services in enterprise computing environment. *Service Oriented Computing and Applications*. URL: https://doi.org/10.1007/s11761-017-0221-1, doi:10.1007/s11761-017-0221-1.

[20] StackExchange. March 2012. Are authentication cookies compatible with the rest philosophy, and why? https://softwareengineering.stackexchange.com/questions/141019/should-cookies-be-used-in-a-restful-api.

[21] Joshi, C. & Singh, U. K. 2017. Information security risks management framework – a step towards mitigating security risks

in university network. *Journal of Information Security and Applications*, 35(Supplement C), 128 – 137. URL: http://www.sciencedirect.com/science/article/pii/S2214212616301806, doi:https://doi.org/10.1016/j.jisa.2017.06.006.

[22] Such, J. M., Gouglidis, A., Knowles, W., Misra, G., & Rashid, A. 2016. Information assurance techniques: Perceived cost effectiveness. *Computers and Security*, 60(Supplement C), 117 – 133. URL: http://www.sciencedirect.com/science/article/pii/S0167404816300311, doi:https://doi.org/10.1016/j.cose.2016.03.009.

[23] Ouedraogo, M., Kuo, C. T., Tjoa, S., Preston, D., Dubois, E., Simoes, P., & Cruz, T. Aug 2014. Keeping an eye on your security through assurance indicators. In *2014 11th International Conference on Security and Cryptography (SECRYPT)*, 1–8.

[24] Spears, J. L., Barki, H., & Barton, R. R. 2013. Theorizing the concept and role of assurance in information systems security. *Information and Management*, 50(7), 598 – 605. URL: http://www.sciencedirect.com/science/article/pii/S037872061300089X, doi:https://doi.org/10.1016/j.im.2013.08.004.

[25] Tung, Y. H., Lo, S. C., Shih, J. F., & Lin, H. F. Oct 2016. An integrated security testing framework for secure software development life cycle. In *2016 18th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, 1–4. doi:10.1109/APNOMS.2016.7737238.

[26] Hudic, A., Smith, P., & Weippl, E. R. 2017. Security assurance assessment methodology for hybrid clouds. *Computers and Security*, 70(Supplement C), 723 – 743. URL: http://www.sciencedirect.com/science/article/pii/S0167404817300627, doi:https://doi.org/10.1016/j.cose.2017.03.009.

[27] Gupta, D., Chatterjee, K., & Jaiswal, S. 2013. A framework for security testing. In *Computational Science and Its Applications – ICCSA 2013*, Murgante, B., Misra, S., Carlini, M., Torre, C. M., Nguyen, H.-Q., Taniar, D., Apduhan, B. O., & Gervasi, O., eds, 187–198, Berlin, Heidelberg. Springer Berlin Heidelberg.

[28] Guru99. November 2017. What is security testing: Complete tutorial. https://www.guru99.com/what-is-security-testing.html. Blog @Guru99.

[29] Mytton, D. October 2015. Api security risks and how to mitigate them. https://blog.serverdensity.com/5-api-security-risks-and-how-to-mitigate-them/. Blog @Server Density.

[30] Wasson, M. November 2012. Authentication and authorization in asp.net web api. https://docs.microsoft.com/en-us/aspnet/web-api/overview/security/authentication-and-authorization-in-aspnet-web-api. Microsoft Docs.

[31] Lensmar, O. November 2014. Api security testing - how to hack an api and get away with it. https://blog.smartbear.com/apis/api-security-testing-how-to-hack-an-api-and-get-away-with-it-part-1-of-3/. Blog @Smartbear.

[32] Wiki, M. April 2012. Webappsec/web security verification. https://wiki.mozilla.org/WebAppSec/Web_Security_Verification.

[33] Srinivasan, S. M. & Sangwan, R. S. Jan 2017. Web app security: A comparison and categorization of testing frameworks. *IEEE Software*, 34(1), 99–102. doi:10.1109/MS.2017.21.

[34] Port, D. & Wilf, J., eds. *A Decision-Theoretic Approach to Measuring Security*, January 2017. http://hdl.handle.net/10125/41901.

[35] Pendleton, M., Garcia-Lebron, R., Cho, J.-H., & Xu, S. December 2016. A survey on systems security metrics. *ACM Comput. Surv.*, 49(4), 62:1–62:35. URL: http://doi.acm.org/10.1145/3005714, doi:10.1145/3005714.

[36] Kahraman, E. Evaluating it security performance with quantifiable metrics. mathesis, Institutionen för Data- och Systemvetenskap, 2005.

[37] Dovetail. April 2018. The difference between qualitative and quantitative research. https://dovetailapp.com/guides/qual-quant.

[38] Williams, C. 01 2007. Research methods. *Journal of Business & Economic Research*, 5.

[39] Leedy, P. D. & Ormrod, J. E. 2001. *Practical Research Planning and Design*. Upper Saddle River, N.J Merrill Prentice Hall 2001.

[40] Symantec. 2017. Certificate pinning. https://www.symantec.com/content/dam/symantec/docs/white-papers/certificate-pinning-en.pdf. White Paper.

[41] Yin, R. K. 1981. The case study crisis: Some answers. *Administrative Science Quarterly*, 26(1), 58–65. URL: http://www.jstor.org/stable/2392599.

[42] Souag, A., Salinesi, C., & Comyn-Wattiau, I. 2012. Ontologies for security requirements: A literature survey and classification. In *Advanced Information Systems Engineering Workshops*, Bajec, M. & Eder, J., eds, 61–69, Berlin, Heidelberg. Springer Berlin Heidelberg.

[43] Felderer, M., Büchler, M., Johns, M., Brucker, A. D., Breu, R., & Pretschner, A. 2016. Chapter one - security testing: A survey. In *Advances in Computers*, Memon, A., ed, volume 101 of *Advances in Computers*, 1 – 51. Elsevier. URL: http://www.sciencedirect.com/science/article/pii/S0065245815000649, doi:https://doi.org/10.1016/bs.adcom.2015.11.003.

[44] De, B. *API Testing Strategy*, 153–164. Apress, Berkeley, CA, 2017. URL: https://doi.org/10.1007/978-1-4842-1305-6_9, doi:10.1007/978-1-4842-1305-6_9.

[45] Hunt, T. May 2013. Hack yourself first - how to go on the offence before online attackers do. https://www.troyhunt.com/hack-yourself-first-how-to-go-on/.

[46] da Silva, M. A. & Danziger, M. May 2015. The importance of security requirements elicitation and how to do it. In *Requirements Management*, EMEA, London, England. Newtown Square. Project Management Institute. Paper presented at PMI Global Congress 2015.

[47] Firesmith, D. January 2004. Specifying reusable security requirements. *Journal of Object Technology*, 3(1), 61–75. URL: http://www.jot.fm/issues/issue_2004_01/column6.pdf.

[48] Gibson, D. June 2011. Understanding the three factors of authentication. http://www.pearsonitcertification.com/articles/article.aspx?p=1718488.

[49] Righi, R. October 2017. Using jwt to secure a stateless api world. https://dzone.com/articles/using-jwt-to-secure-a-stateless-api-world.

[50] O'Leary, R. October 2017. Can you hear me now? apis are vulnerable! https://www.whitehatsec.com/blog/api-vulnerabilities/. White paper.

[51] Alqahtani, S. S., Eghan, E. E., & Rilling, J. March 2017. Recovering semantic traceability links between apis and security vulnerabilities: An ontological

modeling approach. In *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, 80–91. doi:10.1109/ICST.2017.15.

[52] OWASP. 2017. Owasp top 10 - 2017. the ten most critical web application security risks. https://www.owasp.org/images/7/72/OWASP_Top_10-2017_

[53] Owasp. March 2018. Cross-site request forgery (csrf). https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF).

[54] Thakkar, J. February 2018. What is a man-in-the-middle-attack? how do you prevent one? https://www.thesslstore.com/blog/man-in-the-middle-attack/. hashedout by The SSL Store.

[55] Felderer, M., Zech, P., Breu, R., Büchler, M., & Pretschner, A. 2015. Model-based security testing: a taxonomy and systematic classification. *Software Testing, Verification and Reliability*, 26(2), 119–148. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/stvr.1580, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/stvr.1580, doi:10.1002/stvr.1580.

[56] Tian-Yang, G., Yin-sheng, S., & You-yuan, F. 2010. Research on software security testing. *International Journal of Computer and Information Engineering*, 4(9). World Academy of Science, Engineering and Technology.

[57] sqlmap.org. April 2018. sqlmap: Automatic sql injection and database takeover tool. http://sqlmap.org/. Opensource tool.

[58] Duffy, C. September 2015. *Learning Penetration Testing with Python*. PACKT Publishing Ltd.

[59] Kolakowski, N. June 2016. Why is python so popular. https://insights.dice.com/2016/06/01/why-is-python-so-popular/.

[60] Stenberg, D. April 2018. curl:// command line tool and library for tranferring data with urls. https://curl.haxx.se/. Copyright owner is Daniel Stenberg.

[61] Portswigger. April 2018. Burp suite edition. https://portswigger.net/burp.

[62] Wireshark. April 2018. About wireshark. https://www.wireshark.org/.

[63] NCSC. September 2016. Risk management and risk analysis practice. https://www.ncsc.gov.uk/guidance/risk-management-and-risk-analysis-practice. National Cyber Security Centre (UK Government).

[64] Kissel, R. L. 2013. Glossary of key information security terms. *NIST Pubs*. http://ws680.nist.gov/publication/get_pdf.cfm?pub_id=913810.

[65] Wikipedia. December 2017. Weighting. https://en.wikipedia.org/wiki/Weighting.

[66] Owasp. July 2. Threat risk modeling. https://www.owasp.org/index.php/Threat_Risk_Modeling.

[67] Astafyeu, A. Information security risk assessment methodologies in vulnerability assessment of information systems. Master's thesis, Technical University of Denmark, 2015. URL: http://www2.imm.dtu.dk/pubdb/views/edoc_download.php/7059/pdf/imm7059.pdf.

[68] Wikipedia. April 2017. Dread (risk assessment model). https://en.wikipedia.org/wiki/DREAD_(risk_assessment_model).

[69] First.org. Common vulnerability scoring system. https://www.first.org/cvss/.

[70] NVD. April 2018. Common vulnerability scoring system calculater version 3. https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator. National Vulnerability Database (National Institute of Standards and Technology).

[71] Cisco. 2017. Common vulnerability scoring system q & a. https://www.cisco.com/c/en/us/about/security-center/cvss-q-a.html#6.

[72] Borgatti, S. *Normalizing Variables (Handout)*. Gatton College of Business and Economics. University of Kentucky.

[73] Delen, D., Sharda, R., & Bessonov, M. 2006. Identifying significant predictors of injury severity in traffic accidents using a series of artificial neural networks. *Accident Analysis & Prevention*, 38(3), 434 – 444. URL: http://www.sciencedirect.com/science/article/pii/S0001457505001879, doi:https://doi.org/10.1016/j.aap.2005.06.024.

[74] Khajvand, M. & Tarokh, M. J. 2011. Estimating customer future value of different customer segments based on adapted rfm model in retail banking context. *Procedia Computer Science*, 3, 1327 – 1332. World Conference on Information Technology. URL: http://www.sciencedirect.com/science/article/pii/S1877050911000123, doi:https://doi.org/10.1016/j.procs.2011.01.011.

[75] Shalabi, L. A. & Shaaban, Z. May 2006. Normalization as a preprocessing engine for data mining and the approach of preference matrix. In

2006 International Conference on Dependability of Computer Systems, 207–214. doi:10.1109/DEPCOS-RELCOMEX.2006.38.

[76] Visual-Paradigm. March 2015. How to design rest api? the twitter example. https://www.visual-paradigm.com/tutorials/rest-api-design-twitter-example.jsp.

[77] Wikipedia. May 2018. Weighted arithmetic mean. https://en.wikipedia.org/wiki/Weighted_arithmetic_mean.

[78] Irom, A. & Serota, J. August 2017. Best practices for securing your apis. https://www.incapsula.com/blog/best-practices-for-securing-your-api.html.

[79] Owasp. March 2018. Cross-site scripting (xss). https://www.owasp.org/index.php/Cross-site_Scripting_(XSS).

[80] RSA. Cyber risk appetite. Technical report, RSA Security LLC, 2016. Published in the USA. 05/06 Whitepaper H15150.

[81] Rochford, O. May 2017. Settling scores with risk scoring. https://www.csoonline.com/article/3195244/security/settling-scores-with-risk-scoring.html. IDG Communications, Inc.

[82] Katarzyna. 2018. Error handling why it is crucial in api design process. https://www.polidea.com/blog/Error_Handling_why_it_is_crucial_in_API_design_process/. Polidea.

[83] Owasp. January 2016. Brute force attack. https://www.owasp.org/index.php/Brute_force_attack.

[84] Griffith, E. February 2018. Two-factor authentication: Who has it and how to set it up. https://www.pcmag.com/feature/358289/two-factor-authentication-who-has-it-and-how-to-set-it-up.

[85] Feng, N., Wang, H. J., & Li, M. 2014. A security risk analysis model for information systems: Causal relationships of risk factors and vulnerability propagation analysis. *Information Sciences*, 256, 57 – 73. Business Intelligence in Risk Management. URL: http://www.sciencedirect.com/science/article/pii/S0020025513001539, doi:https://doi.org/10.1016/j.ins.2013.02.036.

[86] CWE. March 2018. Cwe-295: Improper certificate validation. https://cwe.mitre.org/data/definitions/295.html. Common Weakness Enumeration.

[87] MathisFun. 2017. Percentage difference. https://www.mathsisfun.com/percentage-difference.html.

[88] Sluiter, A., B.Hames, R.Ruiz, C.Scarlata, J.Sluiter, D.Templeton, & D.Crocker. Determination of structural carbohydrates and lignin in biomass. Technical report, National Renewable Energy Laboratory, July 2008. A national laboratory of the U.S. Department of Energy Office of Energy Efficiency & Renewable Energy.

A Appendix: Security Requirement List

Table 17: Security Requirements

#	Requirement Description
1.0	**Authentication**
1.1	Verify API endpoint by default require authentication except those specifically intended to be public (Principle of complete mediation).
1.2	Verify all authentication controls are enforced on the server side.
1.3	Verify all authentication controls fail securely to ensure attackers cannot log in
1.4	Verify that account passwords are one way hashed with a salt, and there is sufficient work factor to defeat brute force and password hash recovery attacks.
1.5	Verify that credentials such as passwords, security tokens and API keys dont appear in URL.
1.6	Verify API keys are not accessible to others. Transmitted securely and saved securely
1.7	Verify OpenId is present
2.0	**JWT (Session)**
2.1	Verify JWT (Json) tokens are handled in a secure way
2.2	Verify JWT (Json) tokens are invalidated after logout
2.3	Verify JWT in cookies is only accessible on the user domain
3.0	**Access Control**
3.1	Verify that access to sensitive records is protected, such that only authorized objects or data is accessible to each user (for example, protect against users tampering with a parameter to see or alter another user's account).
3.2	Verify that access controls fail securely
3.3	Verify that the same access control rules implied by the presentation layer are enforced on the server side.

Continued on next page

Table 17 –*Continued from previous page*

#	**Requirement Description**
3.4	Verify that all access control decisions can be logged and all failed decisions are logged.
3.5	Verify Oauth2 protocol is present
3.6	Verify CORS configuration is proper
4.0	**Input Sanitation**
4.1	Verify that server side input validation failures result in request rejection and are logged.
4.2	Verify that input validation routines are enforced on the server side.
4.3	Verify that all SQL queries, HQL, OSQL, NOSQL and stored procedures, calling of stored procedures are protected by the use of prepared statements or query parameterization, and thus not susceptible to SQL injection
4.4	Verify that the application is not susceptible to LDAP Injection, or that security controls prevent LDAP Injection.
4.5	Verify that the application has defenses against HTTP parameter pollution attacks, particularly if the application framework makes no distinction about the source of request parameters (GET, POST, cookies, headers, environment, etc.)
4.6	Verify that client side validation is used as a second line of defense, in addition to server side validation.
4.7	Verify that input data such as REST calls, query parameters, HTTP headers, cookies, batch files, RSS feeds, etc; using positive validation (whitelisting), then lesser forms of validation such as greylisting (eliminating known bad strings), or rejecting bad inputs (blacklisting).
4.8	Verify when parsing JSON in browsers, that JSON.parse is used to parse JSON on the client. Do not use eval() to parse JSON on the client.
5.0	**Error Handling**
5.1	Verify that the application does not output error messages or stack traces containing sensitive data that could assist an attacker, including session id, software/framework versions and personal information
5.2	Verify that error handling logic in security controls denies access by default

Continued on next page

Table 17 –*Continued from previous page*

#	**Requirement Description**
5.3	Verify that security logs are protected from unauthorized access and modification
5.4	Verify that the API does not log sensitive data such as sensitive authentication data that could assist an attacker, including user's session identifiers, passwords, hashes, or API tokens
6.0	**Data Protection**
6.1	Verify that all sensitive data is sent to the server in the HTTP message body or headers (i.e., URL parameters are never used to send sensitive data).
6.2	Verify that the list of sensitive data processed by the application is identified, and that there is an explicit policy for how access to this data must be controlled, encrypted and enforced under relevant data protection directives.
6.3	Verify that the application sets appropriate anti-caching headers as per the risk of the application, such as the following: Expires: Tue, 03 Jul 2001 06:00:00 GMT Last-Modified: now GMT Cache-Control: no-store, no-cache, mustrevalidate, max-age=0 Cache-Control: post-check=0, pre-check=0 Pragma: no-cache
6.4	Verify that data stored in client side storage (such as HTML5 local storage, session storage, IndexedDB, regular cookies or Flash cookies) does not contain sensitive data or PII.
6.5	Verify accessing sensitive data is logged, if the data is collected under relevant data protection directives or where logging of accesses is required.
6.6	Verify that sensitive information maintained in memory is overwritten with zeros as soon as it no longer required, to mitigate memory dumping attacks.
7.0	**Communication Security**
7.1	Verify that a path can be built from a trusted CA to each Transport Layer Security (TLS) server certificate, and that each server certificate is valid
7.2	Verify that TLS is used for all connections (including both external and backend connections) that are authenticated or that involve sensitive data or functions, and does not fall back to insecure or unencrypted protocols. Ensure the strongest alternative is the preferred algorithm.
7.3	Verify that backend TLS connection failures are logged

Continued on next page

Table 17 –*Continued from previous page*

#	**Requirement Description**
7.4	Verify that certificate paths are built and verified for all client certificates using configured trust anchors and revocation information.
7.5	Verify that TLS certificate public key pinning (HPKP) is implemented with production and backup public keys. For more information, please see the references below
7.6	Verify that HTTP Strict Transport Security headers are included on all requests and for all subdomains, such as Strict-Transport-Security: max-age=15724800; includeSubdomains
8.0	**HTTP Security**
8.1	Verify that the application accepts only a defined set of required HTTP request methods, such as GET and POST are accepted, and unused methods (e.g. TRACE, PUT, and DELETE) are explicitly blocked
8.2	Verify that every HTTP response contains a content type header specifying a safe character set (e.g., UTF-8, ISO 8859-1).
8.3	Verify that a suitable X-FRAME-OPTIONS header is in use for sites where content should not be viewed in a 3rd-party X-Frame.
8.4	Verify that the HTTP headers or any part of the HTTP response do not expose detailed version information of system components
8.5	Verify that all API responses contain X-ContentType-Options: nosniff and Content-Disposition: attachment; filename="api.json" (or other appropriate filename for the content type)
8.6	Verify that the X-XSS-Protection: 1; mode=block header is in place to enable browser reflected XSS filters.
8.7	Verify CORS headers are disabled if cross-domain calls are not supported
8.8	Verify the origins of cross-domain calls is specific enough in CORS configuration
9.0	**Web Services**
9.1	Verify that the same encoding style is used between the client and the server.
9.2	Verify that XML or JSON schema is in place and verified before accepting input.

Continued on next page

Table 17 –*Continued from previous page*

#	**Requirement Description**
9.3	Verify that the REST service is protected from Cross-Site Request Forgery via the use of at least one or more of the following: ORIGIN checks, double submit cookie pattern, CSRF nonces, and referrer checks.
9.4	Verify the REST service explicitly check the incoming Content-Type to be the expected one, such as application/xml or application/json.
10.0	**Cross Site**
10.1	Verify that there are security mechanisms against Cross-site Scripting

B Appendix Security Vulnerability List

Table 18: Security Vulnerabilities

#	Vulnerability Description
1.0	**Injection Attack**
1.1	SQL Injection
1.2	NoSQL Injection
1.3	LDAP Injection
2.0	**Broken Authentication**
2.1	Permits Credentials stuffing
2.2	Permits Brute Force
2.3	Permits Weak Passwords
2.4	Use plain text, encrypted or weakly hashed password
2.5	Has missing or ineffective multi-factor authentication
2.6	Exposes session id in url
2.7	Do not properly invalidate authentication tokens
3.0	**Sensitive Data Exposure**
3.1	Old or weak crypto keys is used
3.2	Encryption is not enforced (HTTPS is not enforced) Insufficient SSL/TLS configuration
3.3	User agent does not verify if the received server certificate is valid or not
4.0	**Broken Access Control**
4.1	Bypassing access control checks by modifying the URL
4.2	Allowing the primary key to be changed to another users record, permitting viewing or editing someone else's account.
4.3	Metadata manipulation, such as replaying or tampering with a JSON Web Token (JWT) access control token or a cookie or hidden field manipulated to elevate privileges, or abusing JWT invalidation

Continued on next page

Table 18 –*Continued from previous page*

#	**Vulnerability Description**
4.4	JWT insecurity (missing cryptographic signature or MAC)
4.5	Replaying or tampering with JSON Web Token
4.6	Abusing JWT invalidation
4.7	Replaying or tampering with Cookie
4.8	CORS misconfiguration allows unauthorized API access.
4.9	Force browsing to authenticated pages as an unauthenticated user or to privileged pages as a standard user.
4.10	Accessing API with missing access controls for POST, PUT and DELETE.
4.11	API keys revocation is not done
4.12	API key compromise
5.0 5.1	**Elevation of privilege** Elevation of privilege. Acting as a user without being logged in, or acting as an admin when logged in as a user.
6.0	**Cross-site Scripting**
6.1	Reflected XSS: The application or API includes unvalidated and unescaped user input as part of HTML output. A successful attack can allow the attacker to execute arbitrary HTML and JavaScript in the victim's browser. Typically the user will need to interact with some malicious link that points to an attackercontrolled page, such as malicious watering hole websites, advertisements, or similar
6.2	Stored XSS: The application or API stores unsanitized user input that is viewed at a later time by another user or an administrator. Stored XSS is often considered a high or critical risk.
6.3	DOM XSS: JavaScript frameworks, single-page applications, and APIs that dynamically include attacker-controllable data to a page are vulnerable to DOM XSS. Ideally, the application would not send attacker-controllable data to unsafe JavaScript APIs
7.0	**Cross-Site Request Forgery**
7.1	Cross-Site Request Forgery
8.0	**Parameter Tampering**
8.1	Model Binding
8.2	Mass Assignement

Continued on next page

Table 18 –*Continued from previous page*

#	**Vulnerability Description**
8.3	HTTP Verb Tampering
8.4	Insecure Direct Object Reference
8.5	Unvalidated redirects and forwards
9.0	**Man-in-the-middle attack**
9.1	Man-in-the-middle attack

C Questionnaire

Here is the questions that were asked in the interview.

1. Do you think the developed security assurance framework based on quantitative security assurance metrics is helpful, why and when?
2. Do you think that you will use the list of requirement/vulnerability when you security test/develop a REST API in the future? and why?
3. Do you think that dividing the assurance metrics into requirement and vulnerability elements is sound?
4. Is it possible that a security requirement can be only partially fulfilled? If yes, which scale can represent the fulfilment confidence, e.g., scale 0..1, 0..2, 0..3? (For this thesis work this scale has been used. 0=not present, 0.5=partly present, 1=present)
5. Do you have any other feedback to the developed security assurance framework.