Quanitative Security Assurance Metrics - REST API Case Studies

BASEL KATT, Norwegian University of Science and Technology, Norway
NISHU PRASHER, Statistics Norway, Division for Quality and Team Management., Norway

Security assurance is the confidence that a system meets its security requirements based on specific evidences that an assurance technique provide. The notion of measuring security is complex and tricky. Existing approaches do not conisder the relevance of the different security requirements to the evaluated application context. Furthermore, they are mostly qualitative in nature and are heavily based on manual processing, which make them costly and time consuming. Therefore, they are not widely used and applied, especially by small and medium-sized enterprises (SME), which constitute the backbone of the Norwegian economy, In this paper, we propose a quantification method that aims at evaluating security assurance of systems by measuring (1) the level of confidence that the mechanisms fulfilling security requirements are present and (2) the vulnerabilities associated with possible security threats are absent. Additionally, an assurance evaluation process is proposed. Two case studies applying our method are presented. The case studies use our assurance method to evaluate the security level of two REST APIs developed by Statistics Norway, where one of the authors is employed. One of the REST APIs is public and the other is private. Analyzes show 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.

ACM Reference Format:
Basel Katt and Nishu Prasher. 2018. Quanitative Security Assurance Metrics - REST API Case Studies. 1, 1 (June 2018), 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Security assurance is the confidence that a system meets its security requirements based on specific evidences that an assurance technique provide. Thus, the security assurance level of a system states how much confidence one can have in the system that is safe to use. Evaluation, on the other hand, is the process that is responsible for gathering evidences and assessing security "assurance" metrics to check if the security requirements are fulfilled. A security metric is often defined as a metric that depicts the security level, security performance or security strength of a system[Savola 2013].

Current evaluation methods and standards, like OpenSAMM and BSSIM maturity models, the Common Criteria (CC) [CCW 2012] and others, can be characterized as (1) qualitative in nature and tend to achieve their goals manually, to a large extend, which lead to an inaccurate and not repetitive security assurance levels; and (2) treating security requirements within one domain equally for all applications regardless of the context. The problem of the qualitative nature of security assurance metrics and the need for quantitative indicators has been identified in the literature [Ouedraogo et al. 2009]. Furthermore, recent efforts [Abdulrazeg et al. 2017; Thakurta 2013; Yoo et al. 2017] indicate that privatization of security requirements based on the application context is an important aspect during in the application life cycle. On the other hand, recent research work that applied quantitative methods are focusing mainly on vulnerabilities, like the work done by Joshi et al. in [Joshi and Singh 2017], but not security requirements.

We believe that security assurance metrics should be quantitative in nature and include both perspectives, the postive side of security, i.e., security requirement fulfillment, and the negative side of the security, i.e., vulnerability existence. Furthermore, the *importance*, or relevance, factor of security requirements, alongside the risk factor of security vulnerabilities should be taken into account.

This paper presents a quantitative approach for defining security assurance metrics that provides a high level security assurance evaluation and distinguishes two perspectives: protection metrics and vulnerability metrics. We adapt a similar process to the one presented in [Savola et al. 2010] for the development of the security assurance metrics.

Two case studies were carried out at Statistics Norway to validate the proposed method and verify its results. This include applying the assurance method to evaluate two operational REST APIs that belong and are used by *Statistics Norway*, which is the Norwegian national statistical institute and the main provider of official statistics. The first one, **TS-API**, is an internal private API which will be anonymized due to security reasons. This 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, such that the data must not be changed, before it is sent further[Metivier 2017].The other REST API, **PX-API** [1], is an open API which has a detailed user-documentation available on the website [Norway 2017]. This API lets the user create a customized dataset, based on queries made towards over 5000 StatBank tables Statistics Norway offer[Norway 2017].

Analyzes showed that the API with the most security mechanisms imple mented 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 rest of the paper is organized as follows. Section 2 discussed the related work and the main concepts discussed in this paper. Section 3 presents the security assurance process, while section 4 the different elements and indicators of our security assurance metrics are defined and discussed. In section 5 we present the case studies and results and finally in section 6 we analyze the results, conclude and present the plans for the future work.

Authors' addresses: Basel Katt, Norwegian University of Science and Technology, Norway, basel.katt@ntnu.no; Nishu Prasher, Statistics Norway, Division for Quality and Team Management. Norway, nishu.prasher@ssb.no.

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

2 RELATED WORK

Rsearch on security assurance metrics and evaluation methods is vast. The majority of work focus on security assurance throughout the software development life cycle [CC: 2006] process. This is a valid approach as dealing with security from the early stages of the development process will end up with more secure system. However, the emphasis on the process vs implementation leads to overlooking the actual practical security posture of the end product implementation and deployment of operational systems [Jansen 2009].

A various frameworks and standards have been developed for evaluating security assurance, such as for example the BSIMM [2] and OpenSAMM [3] maturity models, the OWASP Application Security Verification Standard (ASVS) [Manico 2016] and the Common Criteria (CC) [CCW 2012]. Maturity models provide qualitative frameworks to evaluate the security posture of the process and culture practiced in an organizations. ASVS collects a list of security assurance requirements and an associated qualitative evaluation scheme. The CC provides a framework that allows specifying security and assurance requirements that need to be evaluated to determine the security, or trust, level of a system. The main criticism against such standards is that they tend to focus on the process more than evaluating the implementation, and they are qualitative in nature and done manually to a large extend.

Spears et al. [Spears et al. 2013] examined assurance in a regulatory context. They aim at conceptualizing assurance by applying CMM to security processes. Still their work provide the abstract theoretical background on CMM and focuses on processes rather than implementation. Joshi et al. [Joshi and Singh 2017] proposes a framework that contributes 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, including threat identification, threat prioritization, and mitigation. This approach is quantitative in nature and results in prioritization of threats, but the main problem is the lacking of security requirement verification.

Such et al. [Such et al. 2016] suggests a framework with 20 "assurance techniques". The techniques are split over 5 high-level categories, review, observe, interview, test, and independent validation. Furthermore Such et al. conducted a survey on 153 industry practitioners where 81% had over 5 years of experience to study the characteristics those assurance techniques, like expertise required, number of people required, time required for completion, effectiveness and cost. The main finding of the work was to compute a measure of cost-effectiveness for each assurance technique, which is not the focus of the current work.

Tung et al. [Tung et al. 2016] has proposed a framework where they have applied security activities and practises of secure development life cycle to generate security guidelines and improving software security. This framework is an integrated security testing framework that particularly can be used while developing a software.

Hudic et al. [Hudic et al. 2017] offers a security assurance assessment methodology for hybrid clouds. Systems and services in the cloud are multi-layered and multi-tenant enviroments. 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. Further, the authors aggregated assessments into assurance levels for various groups, which is relevant for this thesis also. The focus of this work is mainly on security requirements and policy compliance.

Additionally, some initiatives aimed at developing operational methodologies for security assurance of IT infrastructures. Pham et al. [Pham et al. 2008] proposes an attack graph based security assurance method based on multi-agents. The authors defined an "attackability" metric for static evaluation and other metrics for anomaly detection at run time. Pendleton et al. discuss in their survey paper[Pendleton et al. 2016] how hard it is to develop security metrics. They used attack surface estimation to detect vulnerabilities within a system. In their study they define security metrics based on four key indicators: system vulnerabilities, the system defense strength, the attack (or threat) severity and (4) the system dimension or situation. Instead of evaluating the security directly, they estimate the number of access points to the subject system by counting available interfaces, supported protocols, open ports, open sockets, installed software, etc. The current work enacpsulates these factors into two main security assurance metric types, security requirement metrics and vulnerability metrics. Additionally, we consider the security requirement importance, besides the vulnerability risks.

BUGYO [Bulut et al. 2007; Haddad et al. 2011] can be cited as the first methodology and tool for continuous security assurance evaluation; security assurance evaluation in the context of BUGYO was aimed at probing the security of runtime systems rather than products. This work investigates a quantitative approach for defining security assurance metrics that provides an overall security assurance evaluation of a target of evaluation. Ouedraogo et al. [Ouedraogo et al. 2014] also advocate 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. Their paper analyze the practical challenges associated to the assessment of SA and shows when the assurance level eventually drop. The main focus of these work is on runtime monitoring, and not testing, to check the availability of security mechanisms.

Savola[Savola 2013] 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.

3 SECURITY ASSURANCE EVALUATION (SAE) PROCESS

We define a secruity assurance evaluation process as the process that defines the steps required to evaluate the security assurance level of the target of evaluation (ToE).The input is an operational system running a target of application to be evaluated and the output is the assurance level. Our proposed SEAP considers three main

[2]https://www.bsimm.com/
[3]http://www.opensamm.org/

types of metrics: *vulnerability metrics*, *security requirement metrics* and *assurance metrics*. The different types of assurance metrics will be discussed in the next section. Similar to the methodologies defined in [Haddad et al. 2011], the assurance process consists of five main activities: *application modelling, metric selection and test case definition, test case execution and measurement collection, assurance metrics and level calculation, evaluation and monitoring*.

3.1 Application and threat modelling:

The application modelling allows decomposing the application in order to identify its main components, its enviroment, its assumptions, and its critical assets. Security functions and threats related to the basic security concepts of the application and its environment will be analyzed. Besides the architecture of the ToE, this step results in the security requirements expected to be fulfilled and the set of threats to the target system.

There has been some confusion when checklist are defined to verify security requirement fulfillment and verify vulnerability existence. Insufficient TLS/SSL configuration is a vulnerability, while secure communication through HTTPS is a requirement. Further, if validation of the HTTP Certificate is deactivated it does not matter that the encryption is enforced, still it is possible to decrypt the information. In order to capture the whole picture, it is important to consider both security requirements and vulnerabilities.

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. 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[Felderer et al. 2016]. 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[De 2017]. For this study, all the vulnerabilities were obtained from the Owasp site[Oftedal et al. 2017]. The security requirements for this thesis work were compiled from the Owasp ASVS[Manico 2016].

3.2 Metric selection and test case definition:

A metric is based on the measurement of various parts or parameters of security functions implemented on the system with its service and operational environment. Depending on the measurements being performed, metrics can be classified as follows:

- *Security requirement metrics* relate to a measurement that evaluates whether security protection mechanisms exist and fulfill defined security requirements using the GQM method.
- *Vulnerability metrics* relate to a measurement that evaluates the weaknesses/severity and vulnerabilities existence in the systems using the CVSS and GQM methods.

Test cases for both metrics can be defined to test the vulnerabilities and verify the security requirements on the target of evaluation.

3.3 Test case execution and measurement collection:

Test case execution and measurement collection consist in deploying specific probes to implement the test cases on a target of evaluation and its operational environment. These probes can help to collect raw data from the system. This step will result in a measurement that will be normalized to produce an assurance level in step 3.4.

3.4 Assurance metrics and level calculation:

Once the security requirement and vulnerability metrics are determined in step 3.3, the overall security assurance of the target of evaluation can be calculated using equation 6 presented in section 4.

3.5 Evaluation and monitoring:

This step involves comparing the current value of the assurance level to the previous measure, or to a certain threshold and issuing an appropriate message. It can also providing a real time display of security assurance of the service to help the evaluator identify causes of security assurance deviation and assist him/her in making decisions.

4 SECURITY ASSURANCE METRIC

We will try in this section is to define meaningful and measurable metrics to capture the requirements and vulnerabilities that are present in the system. We assume that there are a finite set of m security requirements $r_1 ... r_m$, as well as a finite set of n potential vulnerabilities $v_1 ... v_n$, defined for the ToE. We assume that these sets are specified in the first step of the SAE procees, i.e., application and threat modeling. This can be done using various methods, like theat analysis [Myagmar et al. 2005], and standards, like the ASVS (Application Security Verification Standards) [OWASP 2015] for web application.

4.1 Requirement & Vulnerability

A security metric is often defined as a metric that depicts the security level, security performance or security strength of a system [Savola 2013]. We still use the term *metrics* despite the arguments [Savola 2013] that security cannot be measured as a universal concept due to the complexity and uncertainty during the testing. Further, according to NIST assurance is defined as following: *"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[Kissel 2013]."*

Fulfilling a security requirement through a countermeasure, or mechanism, 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 such security mechanism is present. While by doing security testing for vulnerabilities it is possible to check sufficient resistance to intentional penetration or bypass. Thus, we define the assurance metric of a ToE, AM as follows [4]

$$AM = RM - VM \tag{1}$$

such that RM is the requirement metrics and VM is the vulnerability metric.

[4]Another way to define the assurance metric is as a pair <RM, VM> instead of substracting the vulnerability metric from the requirement metric.

The previous discussion suggests that security requirement **fulfillment** and vulnerability **existence** factors are two important indicators that need to be considered when security assurance is evaluated. Additionally, the discussion in section 1 motivates that (1) security requirement importance, relevance, or **weight**, as we call it, and (2) vulnerability **risk** value should be considered as well. Putting all together we conclude that the security assurance metrics consists of two elements, security requirement metric and vulnerability metric. The first will be defined based on two main factors, fulfillment and weight, while the other will be defined based on two main factors, existence and risk.

4.2 Fulfillment an Existence

In order to map the fulfillment of security requirements and existence of security vulnerabilities into measurable evidences, we use the GQM (Goal Question Metric) approach [Basili et al. 1994]. It allows the to derive measurable metrics from defined conceptual goals by developing questions that cover the goals, for which the answers represent the measurable metrics. It is defined as tree structure consists of three levels [Basili et al. 1994]

- *Conceptual goal*: A goal in our case model the fulfillment of particular security requirement or the existence of particular vulnerability.
- *Operational question*: A question is used to define the way the achievement of a particular goal can be performed. A question in our context represent the test case that aims at checking whether a security requirement is fulfilled by a security mechanism(s) or a vulnerability exists within the operational context.
- *Quanitative answers*: An answer represent a metric that is associated with a particular question, i.e., test case, to check it in a measurable way.

If we assume that the overall goal is to check the security assurance level of the ToE. This main goal can be divided into sub-goals, each of associated with the fulfillment of one security requirement of the existence of one security vulnerability. After that we define the test cases that tries to check the fulfillment of a security requirement or the existence of a vulnerability. Finally we quanitify the results of the test cases and associate them to the metrics. For example, table 1 shows applying GQM for checking the fulfillment of the security requirement *user input should sanitzed.* Note that ASVS standard [OWASP 2015], among others, is used to construct the list of questions, i.e., test cases, associated with that security requirement.

4.3 Weight and Risk

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*[5]*."*

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 a 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.

On the other hand the term risk was defined as the probability and the consequence of an unwanted incident. . 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[NCSC 2016]. In our context we will use CVSS (Common Vulnerability Scoring System)[First.org [n. d.]] for calculating risks. 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[First.org [n. d.]]. The two latter metrics are optional to use depending on the system and context. Base metrics consist 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. Temporal metrics reflect if the characteristics of the vulnerability changes over time, and environmental metrics reflect characteristics of vulnerabilities that are unique to a particular user's environment.

4.4 Requirement & Vulnerability Metrics

Based on the previous discussion, we define a security requirement metric (Rm_i) for a given security requirement r_i at a specific time instance as:

$$Rm_i = (w_i \times \sum_{j=1}^{k} f_{ij}) \tag{2}$$

where k represent the number of test cases, or questions, defined for this security requirement, w_i is the weight of the security requirement and f_{ij} is the fulfillment factor of the j^{th} test case of the the i^{th}t security requirement. As mentioned before, GQM is used to measure the fulfillment of the security requirements.

As a result we define the accumulate security requirement metrics of an application at a specific time instance as:

$$RM = \sum_{i=1}^{m} (Rm_i) \tag{3}$$

where m represents the total number of security requirement defined for the ToE. Table 1 shows how to construct the test cases for a security requirement and link it to a measurable fulfillment factors.

[5]https://en.wikipedia.org/wiki/Weighting

Table 1. Example of applying GQM approach to quantify the fulfillment factor for the security requirement (sub-goal): *user input must be sanitized.*

Sub-goal	Question / Test case	Answer / metric
Use input sanitized	Do server side input validation failures result in request rejection and are logged?	1 (Full)
	Do input validation routines are enforced on the server side?	1 (Full)
	Are prepared statements used for protecting SQL queries, and others?	1 (Full)
	Do security controls preventing LDAP Injection enabled?	0.5 (Avarage)
	Does the ToE has defenses against HTTP parameter pollution attacks?	0 (Weak)
	Is client side validation used?	0 (Weak)
	Is positive validation (whitelisting) used for input data, like REST calls and HTTP headers?	0 (Weak)
	Is JSON.parse is used to parse JSON on the client.?	0 (Weak)

Similarly we define a vulnerability metric (Vm_i) for a given security vulnerability at a specific time instance as:

$$Vm_i = (r_i \times \sum_{j=1}^{p} e_{ij}) \quad (4)$$

where, p represent the number of test cases defined for this vulnerability type, r_i is the risk of the i^{th} vulnerability and e_{ij} is the existence factor for j^{th} test case defined for the i^{th} vulnerability. The existence factor can have three values, 0 means that the test case indicates no vulnerability, 1 indicates the existence of the vulnerability for the test case, and 0.5 indicates the partial existence of the vulnerability for the test case.

Thus, the vulnerability metrics of a system at a specific time instance can be calculated using the risk of vulnerabilities and their existence factor as follows:

$$VM = \sum_{k=1}^{n} (Vm_k) \quad (5)$$

where n represents the total number of vulnerabilities defined for the ToE.

Assurance metrics (AM) determine the actual confidence that deployed countermeasures protect assets from threats (vulnerabilities) and fulfill security requirements. We define assurance metrics as the difference between *security requirement metric* (RM) and *vulnerability metric* (VM). Thus, the assurance metrics can be calculated as follows:

$$AM = RM - VM = \sum_{i=1}^{m} (Rm_i) - \sum_{k=1}^{n} (Vm_k) \quad (6)$$

where, AM is the security assurance metrics at a given time instance, RM is the security requirement metrics at a given time instance, and VM is the vulnerability metrics at a time given instance.

From equation 6, it can be noticed that AM is *minimum* when the following two conditions are met:

- All security requirements are not fulfilled (RM becomes zero), which causes the value of the first term to be minimum (zero), and
- All possible vulnerabilities exist and all have a maximum risk value. This makes the second term to be maximum (VM).

AM, on the other hand, can be *maximum* if (1) VM is minimum for all vulnerabilities, and (2) the protection mechanisms have been found to be effective to fulfill the defined security requirements (RM is maximum) for all requirements.

4.5 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:

$$10 \times 10 = 100 \quad (7)$$

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

$$-9 \times 10 = -90 \quad (8)$$

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 [Borgatti [n. d.]; Delen et al. 2006].

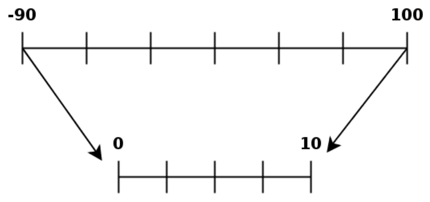

Fig. 1. Normalize to different scale

$$v' = \frac{v - min_A}{max_A - min_A}(newmax_A - newmin_A) + newmin_A \quad (9)$$

The min-max normalization[Delen et al. 2006, p.4] equation is shown in equation 9. 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[Khajvand and Tarokh 2011]. Quoting Shalabi & Shaaban, *"min-max normalization performs a linear transformation on the original data[Shalabi and Shaaban 2006]".*

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[NVD 2018]. However, my table is showing the opposite, thus *levels of security.* This table can be used to convert the score to textual representation.

5 CASE STUDY & RESULTS

This section describes how the quantification method was applied to two case studies, and the result that was obtained. The SAE process described before awasperformed on two different REST APIs, PX-API and TS-API respectively. The main activities that were performed.

(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.

5.1 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[Manico 2016].

- Authentication
- JWT security
- Access Control
- Input Sanitation
- Error Handling
- Data Protection
- Communication Security
- HTTP Security
- Web Services

5.2 List of Security Vulnerabilities

If a security mechanism is missing or not working properly it will lead to a security vulnerability. According to figure **??** 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[Oftedal et al. 2017] and other sites/blogs[Guru99 2017]. 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. The main vulnerability categories are gathered from[OWASP [n. d.]] are as following:

- Injection
- Broken Authentication
- Sensitive Data Exposure
- Broken Access Control
- Elevation of Privilege
- Cross-Site Scripting
- Cross-Site Request Forgery
- Parameter Tampering
- Man-in-the-middle-attack

5.3 API Security Testing

From the list of vulnerabilities and security requirements we applied the GQM approach to specify the test cases that can be used to check the fulfillment and existence of security requirement and vulnerability, respectively. The test cases were performed, some manually and some automatically. In general security testing validates if security requirements are implemented correctly. Black Box-fuzzing was performed on the APIs. API fuzzing is done by sending all possible types of combinations of input parameters and then inspect how the API responds. Mostly open-standard tools, those easy available on the internet, was used to security test the REST API, e.g., Sqlmap and Burp-Suite.

5.4 Results

The last two steps of the SAE process is about assigning of metrics and quantify assurance. The security assurance score can be calculated using the metrics **AM**. The value of VM metric was calculated as 2.93 and the value of the RM metric was 19.25. Hence, AM will give a value of:

$$19.25 - 2.93 = 16.32 \tag{10}$$

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. 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 : is the AM-value:16.32.

$$v' = \frac{16.32 - (-90)}{100 - (-90)}(10 - 0) + 0 \approx 5.60 \tag{11}$$

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 falls between 4.0-6.9, evaluated as *Moderate Security.*

The same was applied for the second REST API and the final result of the AM was 9.71, and for RM was 26.80. Thus AM was calculated as 17.09. After normalization the final value was **5.64**.

6 CONCLUSION AND DISCUSSION AND FUTURE WORK

PX-API had 2 vulnerabilities and 12 security requirements test cases fulfilled. TS-API on the other hand had 3.5 vulnerabilities and 17 security requirements present. The PX-API got the AM-score 5.60 and the TS-API got the AM-score 5.64. 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 it was expected 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[Metivier 2017]. 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. As future work we plan to work on automation of test cases and generation of test cases defintion based on abstract application and threat models.

REFERENCES

2006. Common Criteria for Information Technology Security Evaluation Part 3: Security Assurance components, Version 3.1 Rev 1.

2012. Common Criteria for Information Technology Security Evaluation.

Ala A Abdulrazeg, Norita Md Norwawi, and Nurlida Basir. 2017. RiskSRP: Prioritizing Security Requirements Based on Total Risk Avoidance. *Advanced Science Letters* 23, 5 (2017), 4596–4600.

Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. 1994. The goal question metric approach. *Encyclopedia of software engineering* 2, 1994 (1994), 528–532.

Steve Borgatti. [n. d.]. *Normalizing Variables (Handout).* Gatton College of Business and Economics. University of Kentucky.

Evren Bulut, Djamel Khadraoui, and Bertrand Marquet. 2007. Multi-agent based security assurance monitoring system for telecommunication infrastructures. In *Proceedings of the Fourth IASTED International Conference on Communication, Network and Information Security.* ACTA Press, 90–95.

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

Dursun Delen, Ramesh Sharda, and Max Bessonov. 2006. Identifying significant predictors of injury severity in traffic accidents using a series of artificial neural networks. *Accident Analysis & Prevention* 38, 3 (2006), 434 – 444. https://doi.org/10.1016/j.aap.2005.06.024

Michael Felderer, Matthias BÃijchler, Martin Johns, Achim D. Brucker, Ruth Breu, and Alexander Pretschner. 2016. Chapter One - Security Testing: A Survey. In *Advances in Computers*, Atif Memon (Ed.). Advances in Computers, Vol. 101. Elsevier, 1 – 51. https://doi.org/10.1016/bs.adcom.2015.11.003

First.org. [n. d.]. Common Vulnerability Scoring System. https://www.first.org/cvss/.

Guru99. 2017. What is Security Testing: Complete Tutorial. https://www.guru99.com/what-is-security-testing.html. Blog @Guru99.

S. Haddad, S. Dubus, A. Hecker, T. Kanstren, B. Marquet, and R. Savola. 2011. Operational Security Assurance Evaluation in Open Infrastructures. In *Proceedings of the 2011 6th International Conference on Risks and Security of Internet and Systems (CRiSIS) (CRISIS '11).* IEEE Computer Society, Washington, DC, USA, 1–6. https://doi.org/10.1109/CRiSIS.2011.6061831

Aleksandar Hudic, Paul Smith, and Edgar R. Weippl. 2017. Security assurance assessment methodology for hybrid clouds. *Computers and Security* 70, Supplement C (2017), 723 – 743. https://doi.org/10.1016/j.cose.2017.03.009

Wayne Jansen. 2009. Directions in security metrics research- NISTIR 7564. (2009).

Chanchala Joshi and Umesh Kumar Singh. 2017. Information security risks management framework âĂŞ A step towards mitigating security risks in university network. *Journal of Information Security and Applications* 35, Supplement C (2017), 128 – 137. https://doi.org/10.1016/j.jisa.2017.06.006

Mahboubeh Khajvand and Mohammad Jafar Tarokh. 2011. Estimating customer future value of different customer segments based on adapted RFM model in retail banking context. *Procedia Computer Science* 3 (2011), 1327 – 1332. https://doi.org/10.1016/j.procs.2011.01.011 World Conference on Information Technology.

Richard L. Kissel. 2013. Glossary of Key Information Security Terms. *NIST Pubs* (2013). http://ws680.nist.gov/publication/get_pdf.cfm?pub_id=913810.

Jim Manico. 2016. Open Web Application Security Project. https://www.owasp.org/images/3/33/OWASP_Application_Security_Verification_Standard_3.0.1.pdf.

Becky Metivier. 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.

Suvda Myagmar, Adam J Lee, and William Yurcik. 2005. Threat modeling as a basis for security requirements. In *Symposium on requirements engineering for information security (SREIS)*, Vol. 2005. Citeseer, 1–8.

NCSC. 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).

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

NVD. 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).

Erlend Oftedal, Andrew van der Stock, Tony Hsu Hsiang Chih, Johan Peeters, Jan Wolff, and Rocco Granitz. 2017. REST Security Cheat Sheet. https://www.owasp.org/index.php/REST_Security_Cheat_Sheet.

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

Moussa Ouedraogo, Haralambos Mouratidis, Djamel Khadraoui, Eric Dubois, and Dominic Palmer-Brown. 2009. Current trends and advances in IT service infrastructures security assurance evaluation. (2009).

OWASP. [n. d.]. OWASP top 10 project. https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project,[AccessedJuly2017]

OWASP. 2015. Application Security Verification Standard (ASVS). (2015).

Marcus Pendleton, Richard Garcia-Lebron, Jin-Hee Cho, and Shouhuai Xu. 2016. A Survey on Systems Security Metrics. *ACM Comput. Surv.* 49, 4, Article 62 (Dec. 2016), 35 pages. https://doi.org/10.1145/3005714

N. Pham, L. Baud, P. Bellot, and M. Riguidel. 2008. A Near Real-Time System for Security Assurance Assessment. In *2008 The Third International Conference on Internet Monitoring and Protection.* 152–160. https://doi.org/10.1109/ICIMP.2008.28

Reijo M. Savola. 2013. Quality of security metrics and measurements. *Computers & Security* 37 (2013), 78 – 90. https://doi.org/10.1016/j.cose.2013.05.002

R. M. Savola, H. PentikÃƒÂ£Â¡inen, and M. Ouedraogo. 2010. Towards security effectiveness measurement utilizing risk-based security assurance. In *2010 Information Security for South Africa.* 1–8. https://doi.org/10.1109/ISSA.2010.5588322

L. A. Shalabi and Z. Shaaban. 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. https://doi.org/10.1109/DEPCOS-RELCOMEX.2006.38

Janine L. Spears, Henri Barki, and Russell R. Barton. 2013. Theorizing the concept and role of assurance in information systems security. *Information and Management* 50, 7 (2013), 598 – 605. https://doi.org/10.1016/j.im.2013.08.004

Jose M. Such, Antonios Gouglidis, William Knowles, Gaurav Misra, and Awais Rashid. 2016. Information assurance techniques: Perceived cost effectiveness. *Computers and Security* 60, Supplement C (2016), 117 – 133. https://doi.org/10.1016/j.cose.2016.03.009

Rahul Thakurta. 2013. A value-based approach to prioritise non-functional requirements during software project development. *International Journal of Business Information Systems* 12, 4 (2013), 363–382.

Y. H. Tung, S. C. Lo, J. F. Shih, and H. F. Lin. 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. https://doi.org/10.1109/APNOMS.2016.7737238

Sang Guun Yoo, Hugo Pérez Vaca, and Juho Kim. 2017. Enhanced Misuse Cases for Prioritization of Security Requirements. In *Proceedings of the 9th International Conference on Information Management and Engineering.* ACM, 1–10.