Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

InIT Institut für angewandte
Informationstechnologie

Bachelor Thesis Computer Science

Dynamic Trust Monitoring of Containerized Services in Network Functions Virtualization Infrastructure

[illegible] V. De Riggi

Autoren	Raphael Vogt Valeria De Riggi
	[illegible]
Hauptbetreuung	Gürkan Gür
Nebenbetreuung	n/a
Datum	10. June 2022

Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

DECLARATION OF ORIGINALITY

Bachelor’s Thesis at the School of Engineering

By submitting this Bachelor’s thesis, the undersigned student confirms that this thesis is his/her own work and was written without the help of a third party. (Group works: the performance of the other group members are not considered as third party).

The student declares that all sources in the text (including Internet pages) and appendices have been correctly disclosed. This means that there has been no plagiarism, i.e. no sections of the Bachelor thesis have been partially or wholly taken from other texts and represented as the student’s own work or included without being correctly referenced.

Any misconduct will be dealt with according to paragraphs 39 and 40 of the General Academic Regulations for Bachelor’s and Master’s Degree courses at the Zurich University of Applied Sciences (Rahmenprüfungsordnung ZHAW (RPO)) and subject to the provisions for disciplinary action stipulated in the University regulations.

City, Date:	**Name Student:**
Gähwil, 10.06.2022	Raphael Vogt
Oberburg, 10.06.2022	Valeria De Riggi

Abstract

Although the topic of Network Function Virtualization (NFV) and containerized services embedded therein is an already active research field and is becoming increasingly widespread in practice (e.g., 5G networks), the challenges in terms of security and threats still deserve more attention and research efforts. To counteract the issues to this aspect, this thesis deals with the question of whether and how the issue of trust assessment can be addressed in such infrastructures. Different trust models are reviewed, and the trust attributes used in the literature are analysed in more detail and evaluated based on the knowledge gained. The parameters are subsequently included in a trust calculation for the confidence analysis. Thresholds for the assessment into trustworthy and non-trustworthy will be defined. The approach of developing a Dynamic Trust Monitoring (DTM) solution that supervises the trustworthiness of containerized services in an NFV infrastructure was implemented using a DTM prototype. This prototype is able to perform evaluations based on the results of NFV infrastructures. By collecting and processing the trust parameters, the infrastructures are evaluated according to their trustworthiness. Tests on the monitored system provided information regarding the practical applicability of our prototype. It was found that the DTM is suitable as a basis for further development. Nevertheless, improvements are necessary to be able to use the system in a productive environment.

Keywords: *Trustworthiness, Network Function Virtualization (NFV), Containerized Services, Microservices, Dynamic Trust Monitoring (DTM), Security, Network Monitoring*

Zusammenfassung

Obwohl das Thema der Netzwerkfunktionsvirtualisierung (NFV) und die darin eingebetteten containerisierten Dienste bereits ein aktives Forschungsfeld sind und sich in der Praxis immer weiterverbreiten (z.B. 5G-Netze), bedürfen die Herausforderungen in Bezug auf Sicherheit und Bedrohungen noch grösserer Aufmerksamkeit und Forschungsaufwand. Um den Problemen in diesem Bereich entgegenzuwirken, beschäftigt sich diese Arbeit mit der Frage, ob und wie das Thema der Vertrauensbewertung in solchen Infrastrukturen angegangen werden kann. Es werden verschiedene Vertrauensmodelle betrachtet und die in der Literatur verwendeten Vertrauensattribute näher analysiert und auf Basis der gewonnenen Erkenntnisse bewertet. Die Parameter werden anschliessend anhand einer Vertrauensberechnung für die Analysen einbezogen. Es werden Schwellenwerte für die Bewertung in vertrauenswürdig und nicht vertrauenswürdig definiert. Der Ansatz zur Entwicklung einer dynamischen Vertrauensüberwachung (DVÜ), die die Vertrauenswürdigkeit von containerisierten Diensten in einer NFV-Infrastruktur überwacht, wurde mithilfe eines DVÜ Prototyps umgesetzt. Dieser Prototyp ist in der Lage, Auswertungen auf Basis der Ergebnisse von NFV-Infrastrukturen durchzuführen. Durch die Erfassung und Verarbeitung der Vertrauensparameter werden die Infrastrukturen hinsichtlich ihrer Vertrauenswürdigkeit bewertet. Tests am überwachten System gaben Aufschluss über die Praxistauglichkeit unseres Prototyps. Es wurde festgestellt, dass das System als Grundlage für die weitere Entwicklung geeignet ist. Dennoch sind Verbesserungen notwendig, um die DVÜ in einer produktiven Umgebung einsetzen zu können.

Acknowledgements

We would like to take this opportunity to thank our supervisor Dr. Gürkan Gür, who has been a great help in our work at all times by providing us with important inputs, information, and his general support on the topic. Through regular exchange and feedback, he helped us to complete our thesis and always pointed us in the right direction.

Furthermore, we would like to express our gratitude to Onur Kalinagac, who supported us regarding the test infrastructure as well as in terms of the development of the DTM. We would also like to thank the ZHAW Institute of Applied Information Technology (InIT) and the Greek National Centre of Scientific Research "DEMOKRITOS" (NCSRD), especially Maria Christopoulou, Themistoklis Anagnostopoulos and George Xylouris, for providing us with the test environment.

Lastly, we would like to express our gratitude to Michael Simpson, who gave us many important inputs and suggestions for improvement from a linguistic point of view in the context of the ZHAW "Schreibberatung".

Table of Contents

1 Introduction

Companies have been able to reduce capital expenditure (CapEx) and operating expenditure (OpEx) costs with the help of virtualised networks [1]–[3]. From an economic point of view, hardware-free system environments are lucrative. On-premises hardware does not need to be purchased, licensed, maintained, or serviced, and electricity costs and network cabling for in-house datacentres can be reduced. System engineers and application developers do not have to deal with fundamental hardware-related questions and trust, as do the end-users, in the security and trustworthiness of the providers and their systems. However, virtualised network environments, which are used today in cloud environments, mobile networks such as 5G networks, or internal corporate environments that manage system communications with the help of Software Defined Networking (SDN) have the common feature of being complex environments [4]–[6]. This is especially true since they are pervasive distributed systems. As virtualised system environments are highly software-based, they offer a larger attack surface in terms of software vulnerabilities [7]. Alenezi and Zarour [8] discussed the relationship between software complexity and security and listed additional sources identifying the same correlation such as in [9], [10]. Nevertheless, the security challenge is not the only parameter to consider when measuring the trustworthiness of NFV infrastructures and containerized services deployed there. The deployment of 5G networks for instance found numerous use cases in the health sector, be it remote robotic surgery, augmented reality and virtual reality assistance for distraction and rehabilitative therapy, or remote patient monitoring [11], [12]. A smart health medical device that is connected via a 5G network for real-time monitoring should be able to always deliver correct data, have no losses in terms of availability as well as act reliably. If the application uses an alert function to send an alarm when the heart rate drops below a certain threshold, help should be provided immediately. Apart from the health sector, similar challenges can also be found in the areas of manufacturing or transport systems, for example in autonomous vehicles as well as in public safety for surveillance in smart cities [13]. These requirements lead to the question of how to dynamically monitor and appropriately measure the trustworthiness of containerized services and Network Functions Virtualization (NFV) infrastructures. This topic has already been addressed in many scientific papers and led to various frameworks proposing different approaches as demonstrated in [14]–[18]. The main objective of this work is to address the question of how the trustworthiness of these complex virtual network environments can be evaluated. The assignment of the task is described in Appendix B: Annex, Project Work Description. We will first assess similar publications and explain and concretize basic terms as they are used in this thesis. Subsequently, we will address theoretical aspects of trustworthiness and present a Dynamic Trust Monitoring (DTM) solution. The DTM is then run in a test environment which presents a reference NFV framework and some deployed microservices. The gathered data will be subsequently evaluated and assessed in terms of effectiveness. Finally, a review of the results obtained is described and conclusions are drawn.

2 Background and Related Work

Depending on the publication, the definition of trust varies and so is the way in which trust is implemented as demonstrated in [19]–[22]. Before discussing the definitions of the individual trust attributes, various concepts of trust from different related work are provided. The main objective of this description is to collect these divers formulations, recognize patterns from the definitions and provide a consistent picture of the term trust.

Azzedin and Maheswaran describe trust as a belief with no fixed value tied to a context and time:

> Trust is the firm belief in the competence of an entity to act as expected such that this firm belief is not a fixed value associated with the entity but rather it is subject to the entity's behaviour and applies only within a specific context at a given time [19, p. 1].

The authors assign dynamic values of this "firm belief" using a trust range consisting of six gradations. As a further factor of trustworthiness, certain time units, as well as contexts, are included. For example, a program can access memory resources in a specific directory, but this does not necessarily mean that this authorization will be valid a year later. Moreover, trustworthiness can decay over time: If x should trust y at t_1, the level of trust will be set lower a year later at a time t_2, assuming that no more interactions have taken place between x and y during this time.

Goyal et al. [20] use Azzedin and Maheswaran's definition of trust as a basis for their Quality of Service (QoS) based trust model. In that regard, they describe trust as "an entity based on reliability and firm belief based on attribute of the entity" [20, p. 2]. Describing the definition of trust such as "an entity based on [...] firm belief", the authors refer to this very definition in [19]. Though, Goyal et al. add the *reliability* attribute to their trust definition and expand the concept. The authors state that a lot of publications consider behaviour-based algorithms as well as rank-based trust models to map their models to real paradigms but datacentre and QoS parameters are not considered.

Denko and Sun [21] present a different approach of a trust model. While not going into details of the parameters to be used for the trust calculation, they present a new trust model approach to ensure security in pervasive computing environments. Their definition of the trust that device A places in a device B is described as "the level that A believes B will implement the desired operations and will not initiate or transfer attacks on Device A or a system that runs on Device A" [21, p. 1]. Furthermore, the authors present a model that is as lightweight as possible to primarily fit on battery-powered pervasive devices. Three stages are shown in the model, where the first two stages are for calculating and periodically updating the trust value, and the last is to delete expired entries. The standard procedure for calculating the trust value is called *Direct Computation*. This method is used when enough information about the device, which should be used to establish a trusted connection, could be collected. If this is not the case, a combination of *Direct Computation* and *Indirect Computation* is used. The procedure of the *Indirect Computation* is to ask other devices in the network whether they can provide trust information related to the very device, with which a trusted connection should be established. The *Indirect Computation* value is finally computed using a combination of the received recommendations and *Direct Computation*. The notations used for the trust value computations are depicted in Table 1.

Table 1: Trust Notations according to Sun and Denko (2007) [21, p. 2]

Term	Description
$\boldsymbol{T_A(B)}$	Trust-value device A looks at device B
$\boldsymbol{O_A(B)}$	Observation value derived from the direct observation of device B by device A
$\boldsymbol{R_A(B)}$	Recommendation value derived from the recommendations to A by other devices regarding B

The trust value ranges from -1 to 1, where the more a device trusts another device the higher the trust value is. A trust value of 0 indicates that no trust information is available. This scale is depicted in Table 2.

Table 2: Trust Range according to Sun and Denko (2007) [21, p. 2]

Trust Value	Description
$\boldsymbol{T_A(B) = -1}$	Complete mistrust
$\boldsymbol{T_A(B) < 0}$	Device A considers device B to be untrustworthy to a certain degree
$\boldsymbol{T_A(B) = 0}$	Device A has no trust information about device B
$\boldsymbol{T_A(B) > 0}$	Device A considers device B to be trustworthy to a certain degree
$\boldsymbol{T_A(B) = 1}$	Complete trust

The direct computation is represented by *Eq. 2-1* [21]:

$$T_A(B) = f\big(O_A(B)\big)$$

Eq. 2-1

[21]

where f represents the particular function transferring the direct observation value to the corresponding trust value.

The indirect computation is represented in *Eq. 2-2*:

$$T_A(B) = \begin{cases} \omega_1 * f\big(O_A(B)\big) + \ \omega_2 * R_A(B) \\ \quad if\ O_A(B)\ \neq 0, \qquad where\ \omega_1 + \omega_2 = 1, \omega_1 > 0, and\ \omega_2 > 0 \\ \omega_3 * R_A(B) \\ \quad if\ O_A(B)\ = 0, \qquad where\ 0 <\ \omega_3 \leq 1, and\ \omega_3 \geq \omega_2 \end{cases}$$

Eq. 2-2

[21, eq. (1)]

where ω_1 and ω_2 represent the weights by which the observation and recommendation value scale. The authors suppose $\omega_1 > \omega_2$ to show that direct observation outweighs neighbours' recommendations. $R_A(B)$ represents the recommendation value and results from the average of trust values given to device B by all neighbouring devices of A as depicted in *Eq. 2-3*:

$$R_A(B) = \frac{1}{n}\sum_{i=1}^{n} T_{D_i}(B)$$

Eq. 2-3

[21, eq. (2)]

where $T_A(D_i) > 0$. This is to prevent device A from accepting recommendations from untrustworthy devices.

Furthermore, the authors suggest a formula for trust update and maintenance after the initial trust computation. The trust value should be updated periodically for reliability and safety reasons. Nevertheless, the authors also mention choosing the update interval carefully to reduce communication overhead. The update calculation is based on the current behaviour and previous trust value as shown in *Eq. 2-4*.

$$T_A(B) = \begin{cases} 1 & if\ T'_A(B) + C_A(B) \geq 1 \\ T'_A(B) + C_A(B) & \\ -1 & if\ T'_A(B) + C_A(B) \leq -1 \end{cases}$$

Eq. 2-4
[21, eq. (3)]

where $T_A(B)$ represents the new trust value after re-computation and $T'_A(B)$ the old trust value. $C_A(B)$ is a customizable parameter based on the current behaviour of device B: $-1 \leq C_A(B) \leq 1$.

Li et al. [23] describe two different types of trust: trust in performance and trust in relationship. Trust in performance describes a direct trust relationship between trustor A to trustee C whereas trust in relationship describes an indirect trust relationship between trustor A to a third-party B, which also trusts trustee C. Both concepts are depicted in Figure 1.

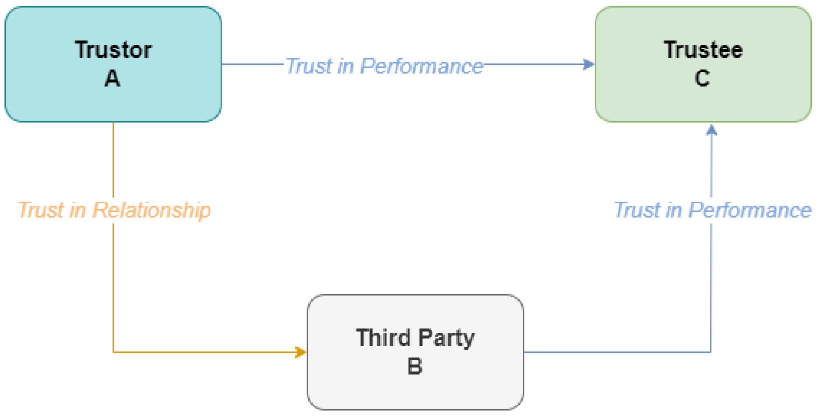


Figure 1: Types of Trust by Li et al. (2021) [23, Fig. 1]

Govindaraj and Jaisankar [22] describe the relationship between trustor and trustee by means of an example in a cloud computing environment. In this client-provider relationship, the service customer represents the trustor and the service provider the trustee. Furthermore, after reviewing several publications dealing with the subject of trust, the authors collected different definitions and interpretations of trust and trustworthiness and were able to summarise them according to Figure 2. They found that the definition of trust can be divided into two main groups: Based on Trustor Expectancy and Based on Trustor Experience. Furthermore, trust Based on Trustor Expectancy can be divided into Trust in performance and Trust in belief. These two terms were defined in [24]. The same applies to trust Based on Trustor Experience, which can be divided into Direct trust, being the trust based on own experiences with other entities, and Recommended trust, which is based on third-party recommendations when there is no direct interaction between two entities as mentioned in [25]. Trust in performance is represented in [24] with the term $trust\_perform$ which is defined by (t, e, p, c) where t represents the trustor's trust in trustee e concerning e's performance p in circumstance c. Trust in belief is represented using the term $trust\_b$ which is defined by (t, e, b, c) where t represents the trustor's trust in trustee e concerning e's belief of b in circumstance c.

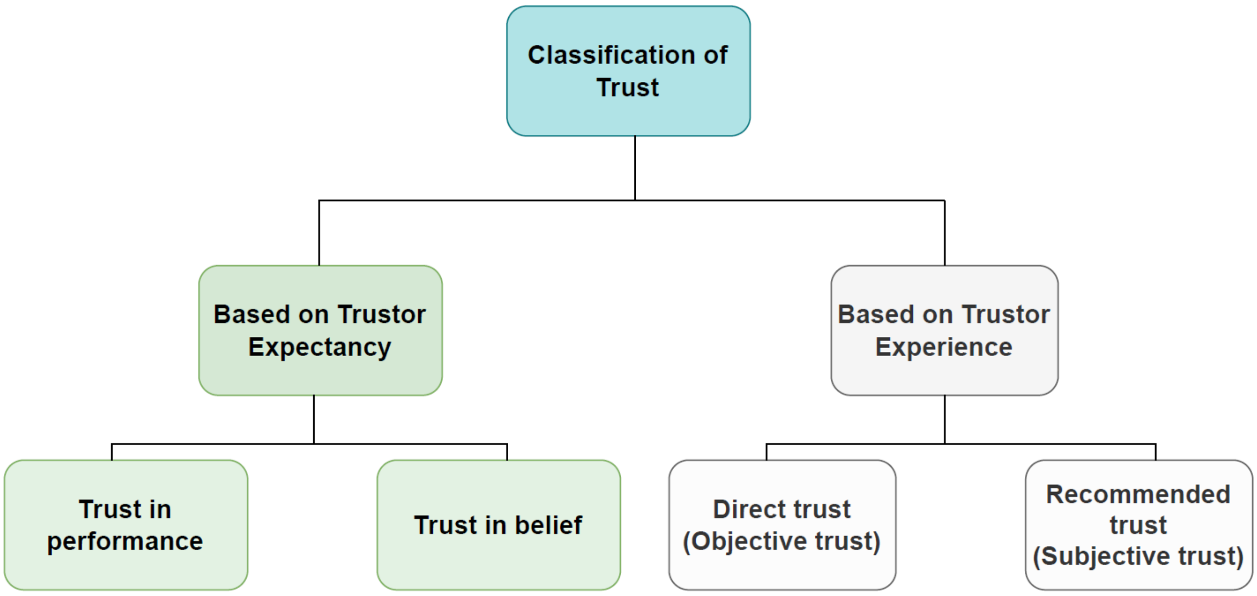

Classification of Trust
Based on Trustor Expectancy
Based on Trustor Experience
Trust in performance
Trust in belief
Direct trust (Objective trust)
Recommended trust (Subjective trust)

Figure 2: Classification of Trust by Govindaraj and Jaisankar (2016) [22, Fig. 1]

Compared to the previously mentioned definition of trust by Li et al. their trust definition is related to the trust group Based on Trustor Expectancy where Trust in Belief can be seen as a synonym for Trust in Relationship.

2.1 Trust Models

According to [22] a trust model is a model to evaluate the process of system trust. It is designed to analyse services based on their properties [26], i. e., in our case in the context of trustworthiness.

In [22] different trust model proposals are categorized into three categories, namely SLA Based Trust Model, Recommender Based Trust Model, and Reputation Based Trust Model, which are depicted in Figure 3. Those models are described in more detail in the following sub-chapters.

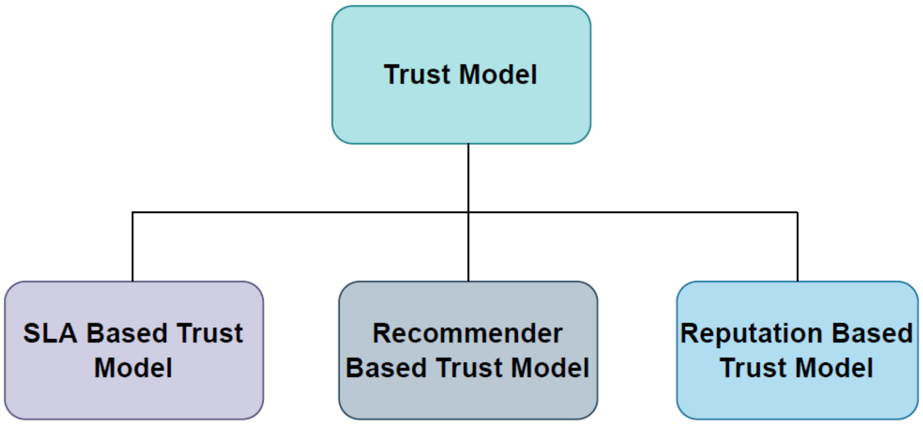


Figure 3: Trust Model Classification Govindaraj and Jaisankar (2016) [22, Fig. 2]

SLA Based Trust Model: As the name already suggests, this trust model is created based on contracts, Service Level Agreements (SLA) being the most frequently used between the service providers and service customers. SLA is an agreement that offers the list of services as well as the quality that the provider needs to provide to the customer. It consists of QoS parameters and security documents. Alhamad, Dillon, and Chang [27] proposed an SLA-based trust framework specially designed for selecting cloud providers for critical business applications requiring a high level of trustworthiness using various criteria. The SLA-based trust model consists of an SLA agent which performs tasks such as monitoring SLA parameters and business activities for consumers and selecting cloud providers based on pre-defined requirements. Additionally, a trust model is included in the SLA-based trust model. The interaction between these two models results in a cloud provider ranking suggesting the best provider for the pre-defined functional and non-functional requirements.

Recommender-Based Trust Model: Recommended trust is similar to an indirect trust, which has already been described earlier in this chapter. If two parties have no direct interactions, the trust may be established through a third-party recommended trust. Various frameworks described in [22] specify implementation scenarios of a recommender-based trust model such as in [25], [28]–[30] users may directly rate requested services.

Reputation-Based Trust Model: A reputation-based trust model is based on the estimation of the public about the cloud provider. Therefore, trustworthiness depends on the reputation of the provider. This is reflected by various QoS, and security parameters offered by the cloud service providers themselves. Significant work on this topic includes Huang and Nicol [24], Abawajy and Goscinski [31], and Borowski et al. [32]. This enumeration is not complete and is based on [22].

2.2 Terms and Definitions in the Context of Trust and Trustworthiness

As the previous sections have shown, trust depends on several properties. These properties differ depending on the reference which defines trust and related concepts. Becker et al. [14] describe the trustworthiness of software systems with the help of five key attributes. Those key attributes are determined by the characteristics mentioned in each box in Figure 4. The attribute QoS is subdivided into three quality characteristics.

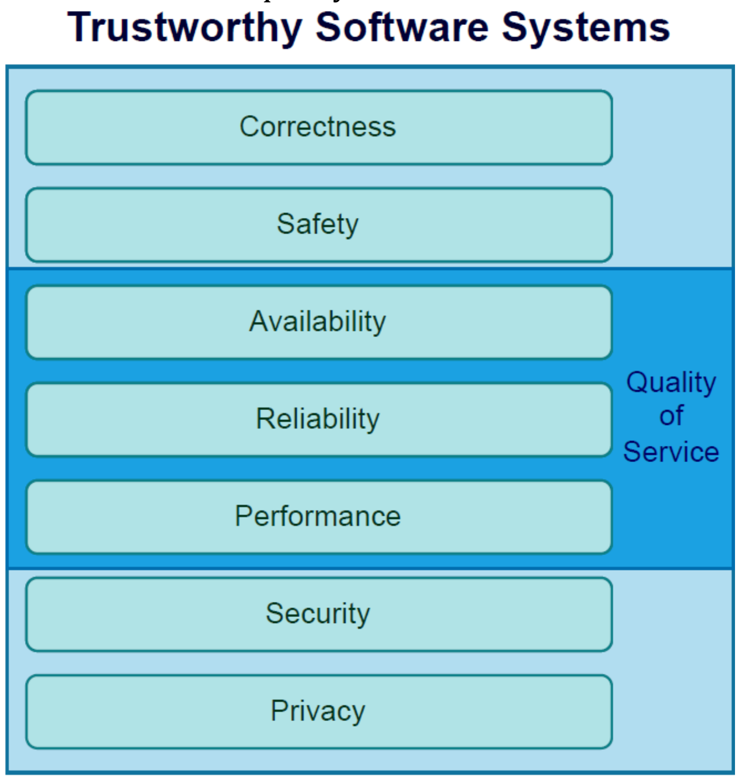


Figure 4: Trustworthiness of Software Systems Characteristics by Becker et al. (2006) [14, Fig. 3]

The characteristics depicted in Figure 4 are now described in more detail. For further exemplifications and unless otherwise stated, reference is made to [14].

Correctness

The term correctness describes the degree to which a system satisfies its requirements. With the help of verification and validation, the correctness of a software system is reviewed whether it meets the specified requirements respectively if the system fulfils the expectations of its users. Following Boehm's [33] definition verification and validation describe aspects for verifying whether a product meets the requirements accordingly, here in the context of software and services. Verification verifies whether the product has been correctly developed or meets the user specifications, while validation verifies whether the product really does what the user needs. Romdhani et al. [18] paraphrased correctness as a so-called data quality factor named *data freshness*. The authors describe data freshness as data that is valuable and trustworthy compared to outdated data because of its timelessness. John Joseph and Mariappan [17] define the term *credibility* as the measure of *accuracy* and *effectiveness*. Accuracy describes the degree of correctness of the provided data and effectiveness as the usability of this precise data by other platforms.

Safety

Safety, in general, implies that nothing bad happens. The term bad in this context describes a measure of the probability and severity of harm. Lowrance [34] referred to this definition as the severity of harm to human health. However, instead of referring to humans, this definition is also applicable in a context of a software-oriented environment. Meaning that safety describes a measure of the severity of harm to a user or its environment.

Quality of Service is a term used to quantify the quality of a provided service consisting of three key attributes: *availability*, *reliability,* and *performance*. These attributes are now explained in more detail.

Availability

Availability is the probability that a system can provide and fulfil a certain service at a certain time without errors or interruptions. One is primarily interested in the result after a longer period of time. Generally, availability can be described with the help of two Boolean operators: up and down. Habib, Ries, and Muhlhauser [15] report about cloud providers using the term availability to indicate to their customers the level of reliability of the cloud services. The authors describe availability as the *reachability* and *success rate* of the transactions to a cloud service. Further, the statements made by cloud providers are criticised. Typically, the availability is indicated with percentages such as 99.99%, also referred to as four-9's. However, the customer does not know whether this information applies to a single virtual server instance or all servers of a data centre in different regions of the world.

For Hassan et al. [16] the term availability describes both, the *accessibility*, and the *operability* of a system meaning that the system should not only be up and running but the service must also be ready to process requests. The latter is according to Becker et al. [14] and Romdhani et al. [18] not described with the term availability but with *reliability* respectively *Task Success Ratio (TSR)*:

> Sometimes, data services may be available and accessible but are unable to deliver data to data consumers due to network failures, timeouts set by the consumer, etc. The task success ratio of a service measures successful data delivery in response to accepted requests [18, p. 4].

In [17] *reputation* is described as a cumulative measure of *information availability* (IA) and *compatibility* whereas IA is related to the definition of availability given in [14], and the term compatibility refers to the synchronisation capability of a certain platform.

Reliability

Software and hardware reliability depend on how likely it is that the system will generate errors or that the system will fail completely with reference to how long it will take to recover the system. The reliability of a system can be influenced by a wide variety of factors. Among other things, poor hardware, and software design, but also external factors such as environmental factors, constant high system load, or high system age could lead to problems. In [17] the term persistence is described to be measured through stability and consistency where both terms are related to reliability.

Performance

The term performance refers to the ability of a system to process a certain request in a given unit of time. In the literature, the synonyms *efficiency*, *responsiveness*, and *scalability* are sometimes used to describe performance. According to [14], the ISO 9126-3 standard considers *resource utilisation* and *time behaviour* as two disjunctive quantities, which, when combined, result in *efficiency*. In [15] performance measurements are described with the following attributes: *latency, bandwidth, availability, reliability,* and *elasticity*. Yet, the authors do not go into the details of the individual parameters. As seen in [18] the term *Time Efficiency* (TE) is used as a synonym for performance and stated that a data service with a high response time violation leads to users not being able to rely on a service provider. According to [16] performance is described as a QoS attribute including a*vailability, response time,* and *turnaround time* whereas in [17] the term *competence* is used to define performance.

In contrast to [14], Goyal et al. [20] suggest the use of the parameters *Initiation Time*, describing the time to deploy a virtual machine (VM), the hourly costs for using a VM from a provider as a Software as a Service (SaaS) provider, the processing time of the VM, the fault rate and bandwidth as QoS parameters for their trust calculation. These parameters are listed in Table 3.

Table 3: Datacentre and QoS Parameters by Goyal et al. (2012) [20, p. 3]

Parameters	**Description**
Initiation Time	Time to deploy a VM.
Price	The amount a SaaS provider must pay per hour for using a VM from a provider.
Processing Speed	Processing time of the VM.
Fault Rate	Number of faults in a given period of time.
Bandwidth	Amount of data that passes the VM in a given period of time.

Furthermore, as demonstrated in [20] a formula is proposed for the initialization of the trust value based on some parameters listed in Table 3.

Eq. 2-5

$$Initial\_Trust = Default\_Value \; + \; CPU$$

[20, p. 3]

Where $Default\_Value$ describes the trust values initialized on a newly introduced datacenter and CPU describes the clock rate in GHz. After a fixed period of time, these values are updated.

According to [16] SLA parameters are defined, namely *availability, reliability, data integrity*, and *turnaround efficiency*. Based on these initially defined parameters, the authors developed the requirements for the QoS attributes which are depicted in Table 4.

Table 4: QoS Attributes by Hassan et al. (2020) [16, Tab. 1]

Parameters	**Description**
Performance	Availability (%)
	Response time (sec)
	Turnaround time (sec)
Capacity	Number of Central Processing Units (CPUs) (4 cores each)
	CPU speed (GHz)
	Disk (TB)
	Memory (GB)
Cost	Costs per task ($)
Security	Data confidentiality & integrity (boolean)
Network QoS	Latency (milliseconds)
	Bandwidth (Gbit/s)

Furthermore, the scenario of a cloud user defining requirements for weighted QoS attributes is demonstrated in [16, Tab. 2]. The user receives recommendations from cloud providers, whereof the user can select suitable candidates based on his requirements.

Security

In the common view security is the defence against attacks from outside, however, a safe system also has to be safe from unintentional and intentional attacks from inside. The most used definition for security is the CIA triad, which stands for confidentiality, integrity, and availability. This means, that data should be secured from unauthorised read access, unauthorised modification, and available when required. In [15] security measurements such as crypto algorithms and key management, physical security, network, and data security support are used. The authors do not go into the details of the individual parameters.

Privacy: In recent years, data protection has increasingly moved under public discussion which led to new laws like the GDPR in the EU [35]. Anderson defines privacy as follows:

> Privacy is the ability and/or right to protect your personal secrets; it extends to the ability and/or right to prevent invasions of your personal space (the exact definition varies quite sharply from one country to another). Privacy can extend to families but not to legal persons such as corporations [36, pp. 13–14].

According to [17] the term *integrity* with the attributes *confidentiality* and *availability* is used to measure the non-disclosure of the information between the sender and receiver. The terms *confidentiality, integrity,* and *availability*, i.e., the CIA triad.

As suggested in [15] aggregations of different parameters from various sources are used to calculate a trustworthiness score. For example, customer recommendations as subjective and response time as objective trust parameter. Further hard as well as soft trust contexts should be used such as certificates, user feedback, and highlighting competencies as, for example, servers using Trusted Platform Module (TPM). Furthermore, the definition of trustworthy servers can vary, especially with subjective parameters. For example, one user may place more value on high-quality customer support, while another customer may prefer highly secure server environments.

The key attributes of trustworthiness that have just been discussed in detail are now listed in Table 5. As mentioned before, the parameters relevant for our work refer to [14]. Accordingly, all trust attributes could be provided with a description in this row. Habib, Ries, and Muhlhauser [15] dealt with the three parameters *availability*, *reliability,* and *security* in their work. However, their definition differs from Becker et al.'s [14] insofar the term *reliability* was equated with *availability*. Thus, these two attributes cannot be assigned different meanings. The term *security*, however, was used in the same way in both works, namely with the widespread CIA triad. In comparison to [15], John Joseph and Mariappan [17] used more comprehensive terms to describe the concept of trust. As a synonym for *correctness*, the term *data credibility* is used, which is described as a measure of *accuracy*, i. e. the degree of *correctness* of the data from the platform, and *effectiveness*, which is the usability of the data provided by the platform. Furthermore, in [17] *availability* is described very similarly to [14] and [15]. *Reliability*, however, is again used as an actual term of its own, which is defined as the *stability* and *consistency* of the platform. *Performance* was also described as a parameter, but as a synonym for the term *competence*. It is measured using the average of the response time and the average of the turn-around time. The latter is described as the total time taken by the platform from the submission of the request to the completion of the task. *Privacy* can be interpreted as a slimmed-down variant of the CIA triad, i.e., without the concept of integrity, which normally belongs to the security parameter. In [16] only two of the trust attributes defined by [14], namely *availability* and *performance* are used. *Availability* is being described similarly as in [14], [15], [17], but without mentioning the faultlessness of the system. The error-free property is defined in the parameter *performance*, which is paraphrased as a combination of availability, response time, and turnaround time. Finally, the terms in [18] are listed. The term *data freshness* is introduced to describe the parameter *correctness* as in [14]. *Reliability* is described as the ability to deliver data to consumers, which could have been interpreted in [14], [15], [17] as a paraphrase for the term *availability*. Further, *performance* is defined as the time efficiency of the system. For the sake of simplicity additional trust parameters mentioned by these and other papers have been omitted. The corresponding publications are listed in the rows. If no definition is available, the corresponding cell is characterized with a hyphen.

Table 5: Comparison of Trust Attributes by Publication

	Correctness	Safety	Availability	Reliability	Performance	Security	Privacy
Becker et al. (2006) [14]	Degree of fulfilled system requirements	The absence of bad things to happen to a user or a system environment	Error-free and uninterrupted service at a given time	Recovery time in case of system errors or failure	Ability to process requests in a given unit of time	confidentiality, integrity, and availability	Confidentiality of personal data
Habib, Ries, Muhlhauser (2010) [15]	-	-	Reachability and success rate of transactions	Availability	-	confidentiality, integrity, and availability	-
John Joseph & Mariappan (2018) [17]	Data credibility	-	The tendency of a platform to respond towards a request	Stability and consistency	Competence	-	Confidentiality and availability
Hassan et al. (2020) [16]	-	-	Accessibility and operability of a system	-	Availability, response time, and turnaround time	-	-
Romdhani et al. (2021) [18]	Data freshness	-	-	Ability to deliver data to consumers	Time Efficiency	-	-

3 Theoretical Basis

In this chapter, we will examine the theoretical foundations of virtualised network environments since we address trust in such an environment. First, we will describe the background to the development of network virtualisation and outline its historical aspects. Subsequently, we will mention theoretical aspects of example infrastructures that work with virtualised networks, such as microservices and cloud-native architectures, which also include 5G. Afterwards, the introduction to trust management and monitoring is introduced, following the investigation on the topic of how to best measure the parameters discussed in the previous chapter and how they can later be used or integrated for the quantification of trust. Moreover, important terms for the evaluation and experimentation phase will also be introduced.

3.1 Network Virtualization

Traditionally, network administrators manage switches, routers, and firewalls individually. When a new subnetwork is needed, the administrator connects directly to the router to configure the new implementation directly on the command-line interface (CLI). Small mistakes could lead to the collapse of an entire company network. The idea behind Software-Defined Networking (SDN) is to make administration more flexible and easier to manage [37]. With the help of a central management unit, administrators maintain an overview of all networking devices including changes in the configuration [38]. This is achieved by decoupling the control plane from the data plane. In short, hardware resources are virtualised, which means that the network functionality is delivered with the help of software resources. However, this is not a novel invention and virtualised networks do not necessarily have to be related to SDNs. For example, configuring Virtual Local Area Networks (VLANs) on a switch to be able to separate Local Area Networks (LAN) from each other without having to purchase additional hardware is also related to virtualised networks. Thus, the concept of virtualised networks covers a wide range of implementations. Nevertheless, the concept of virtualising network services is referred to as NFV [39]. With the same idea in mind, several telecom providers teamed up in 2012 to address the problem of traditional, error-prone administration of new network functions [40], [41]. Since the topic of virtualisation was not a new technological invention, one logical step was to virtualise network infrastructures and devices. To conceptualise this idea together, the NFV Industry Specification Group (ISG) under the European Telecommunications Standards Institute (ETSI) was formed. This has resulted in widespread requirements and architectural concepts for NFV today. As services, devices for performing network functions are considered, i.e., switches, routers, firewalls, and so on. The most frequently mentioned advantages are the advantages of generally virtualised environments. However, faster scalability increased security, and reduced CapEx and OpEx expenses are listed among other benefits [42].

3.2 Microservices and Cloud-Native Architectures

The term microservices describes an architectural pattern composed of loosely coupled and separate services [43]. The main objective is to assemble a distributed application. Whilst monolithic applications put all their functionalities into a single process, each functionality of a microservices application is put into a separate service. Those services are deployed independently and each one has an individual lifecycle [44]. Microservices offer clear advantages over monolithic applications. As an example, code can be updated more easily. New features can be implemented without much effort, as the entire application does not have to be touched as a whole. Individual services can also be scaled more efficiently. If a single service, such as the database service, is short of resources, this single component can be scaled individually. Thus, costs can be saved because the resources can be distributed more efficiently [45]. However, as microservices are applications composed of multiple instances, a failure in one instance may have a big impact on the entire system. Bruce and Pereira describe the term *cascading failures* which means failures that "increase the magnitude of the initial disturbance" [46]. These failures are common in distributed applications and are an example of positive feedback, i. e. with increasing effect.

Luong et al. [47] suggest using cloudification and autoscaling orchestration for container-based mobile networks. The idea is to make mobile networks such as 5G more flexible, dynamic, and faster adaptable. A similar approach is presented by Liu et al. [48]. A 6G network architecture termed SOLIDS is proposed. Among other changes compared to the 5G architecture, the implementation of a cloud-native-enabled and microservice-based architecture is suggested. A similar vision is presented by the 5G Infrastructure Association:

> In 6G, we expect direct integration of different resources from networking to computation and sensing. For that reason, we extend the scope [...] to the terminals and data centres and insist on a full, end-to-end resource awareness of the 6G system. In this way, the functionalities and services can be provided as microservices where needed, while ensuring fully trustworthy services [49, p. 36].

In [49] the further proliferation of cloud-based services in 6G mobile networks is also addressed. A list of the changes from 5G to 6G with regard to virtualised network infrastructure is listed in Table 6. Topics not covered in this thesis have been omitted from the list. CP refers to the control plane and 5GC to 5G core network, which is the brain of the 5G mobile communication network. It is responsible for managing and controlling the entire 5G network [50].

Table 6: Envisioned main Differences between 5G and 6G Network Architecture [49, p. 32]

Parameters	**5G**	**6G**
Cloud-Native	Only CP in 5GC	end-to-end (E2E) and cross-plane
Microservices	No	Yes, E2E, all planes
Trustworthiness	Trustworthy nodes	Trustworthy adaptive services/network of networks

As we can see in the comparison in Table 6, it is assumed that the focus in the architecture of 6G networks will most likely be on cloud-native and microservices architecture. In addition, infrastructure trustworthiness is mentioned as one of the key features:

> Key features of 6G will include intelligent connected management and control functions, programmability, integrated sensing and communication, reduction of energy footprint, trustworthy infrastructure, scalability, and affordability [49, p. 2].

3.3 Trust Monitoring

Trust management aims to create confidence and faith in providers, overcome inelasticity in supporting complex trust relationships in large networks and eliminate the complexity of different policies regulating authorisation rules and security policies [22], [51], [52]. Trust management techniques have been used to identify untrusted behaviour or to form trusted connections to influence the future behaviour of interactions between two or more devices [53]. According to [52] the trust management approach in distributed environments, such as cloud services, Internet of Things (IoT), or dynamic 5G networks, was developed to simplify distributed-system security and to get rid of the inadequacy of traditional authorization mechanisms. This approach reduces the workload of administrators in authorising countless devices and increases flexibility. The trust negotiation is left to a software application, which helps the individual devices in the distributed network to negotiate a trust connection among themselves on a predefined parameter basis. Such parameters were described in the previous Section 2.2.

3.4 Trust Parameters

As stated earlier, the degree of trust depends on various parameters. To be able to use these parameters later for trust calculations, it must first be defined how these parameter values can be categorised as trustworthy or untrustworthy. For example, it would be necessary to find out when a high processor load is too high or which factors are decisive for a system or a system landscape to be considered insecure. This will now be recorded for all main attributes defined in Table 5.

Correctness: If an application does not function properly, this has an impact on user satisfaction. As an end-user, one normally assumes that an application performs the functions correctly in the background and that one can trust the respective result. As an example, one should be able to assume that a smart health medical device, that is connected via a 5G network for real-time monitoring, provides correct values at all times. If the application uses an alert function to send an alarm when the heart rate drops below a certain threshold, help should be provided immediately. If there are difficulties in the transmission of the values, be it the incorrect or even non-transmission of data, or the delayed data transmission, there can be serious consequences. Hence, it can be said that the quality of the software is closely linked to correctness; good software quality leads to fewer errors and, accordingly, to more correctness in software-intensive systems such as virtualised network infrastructure in 5G and future 6G networks. The simplest form to prove software correctness is with the help of functional correctness behaviour [54]. In this active technique, an application is given various inputs. It is then verified if the produced output satisfies the specification respectively if the output fulfils correctness. Based on tests with inputs and outputs, a connection between software correctness and data correctness emerges. In today's world, data is an important commodity [55]–[57] and it is therefore highly relevant to process and store data correctly and consistently in order to avoid errors in further processing. Data quality metrics are important components to measure the extent of errors in data sets. Such data quality metrics are mentioned in [58]. For our work, relevant metrics are listed in Table 7.

Table 7: Data Quality Measures by Tozzi (2021) [58]

Metric	**Description**	**Calculation**
Ratio of data to errors (R_{de})	The number of errors relative to the size of the data set.	The total number of errors divided by the total number of items.
Number of empty values (N_e)	Empty values indicate information is missing from a data set.	The number of empty fields within a data set.
Data transformation error rates (D_t)	The number of errors arising when converting information into a different format.	The number of failed attempts when converting data.

Furthermore, six dimensions of data quality have been defined as demonstrated in [59]. These dimensions are listed in Table 8.

Table 8: Six Data Quality Dimensions by Sarfin (2021) [59]

Metric	**Description**
Accuracy	The information reflects reality.
Completeness	The information is comprehensive enough to be understood.
Consistency	The information is unambiguous and consistent everywhere where stored.
Timelessness	The information is available and accessible when needed.
Validity	The information is saved in a usable format and follows business rules.
Uniqueness	The information is not redundant.

As can be seen in Table 7 and Table 8, the data quality dimensions are closely linked to quality measurements. Thus, one can deduce that if the consistency dimension is not fulfilled, the ratio of data to errors will increase compared to a strictly consistent data set. A similar situation can be seen with data that does not fulfil the validity dimension, which causes the data transformation error rates to also increase. As mentioned earlier, Table 7 is not a complete representation of all the metrics listed in [58]. Nevertheless, this excerpt is a good representation to show the relationship between the data quality dimensions and their corresponding metrics.

Safety: Safety is defined as a measure of the severity of harm to a user or its environment. It could be assumed that system safety is measured in the same way as workplace safety, which consists of the number of hours worked per year in relation to the number of experienced injuries [60]. If we would project this concept one-to-one to system safety, we could say that the number of hours worked would be the system uptime and the number of experienced injuries can be transferred analogously to system failures. Other definition approaches also describe system safety as minimizing hazards. It is independent of whether safety-critical events occur in elements of life-critical systems or others. The resulting risks should be acceptable and the probability predictable [61]. The safety risks are typically presented in the form of risk assessment matrices and classified according to the extent of the hazard and its probability of occurrence. Accordingly, it is difficult to give a generally valid formula for the calculation of system safety. The risks that occur when a system fails can be interpreted in many ways depending on the scenario or use case. Moreover, risks can be categorised differently, depending on the perspective. For example, a scenario in which a system failure leads to a large loss of money in a company would be categorised as very high, whereas high risk could also include the failure of a life-critical system, such as a network slice serving autonomous robots provided by a 5G mobile operator. The assessments of the safety criteria must therefore be made individually for each system and application. These risks should also be assessed individually, and measures formulated according to the scenario. As unique as the individual risks are, the protective measures must also be adapted accordingly.

Availability: As described earlier in the sub-chapter *Availability*, the parameters uptime and availability are often equated. Nevertheless, when talking about availability in the context of SLAs, uptime is almost always mentioned as an important QoS parameter. This is especially common in telecommunication and hosting environments. An almost perfect system availability uptime of 100% is associated with a lot of redundancy, not only in terms of servers, i.e., services distributed on two or more servers. With mobile and cloud providers, where virtualised networks have become indispensable, uptime relates to the accessibility from the internet, which means that users can access the server from the internet. As an example, what if an Internet Service Provider (ISP) experiences outages? For full redundancy, technically speaking, a second internet connection would have to be provided for accessibility. To assume the worst-case scenario, this would be best done by another provider who does not rent the same lines as the first provider. Furthermore, the server hard disks must also be redundant, which means that regular backups of all the data on all the disks should be made. Additionally, one or more data centres could be damaged by natural disasters, so the redundancy would have to be spread over several locations. For 5G mobile operators, the base stations may be redundant connections to the backhaul network to avoid connectivity problems. Power failures must also be considered, so all server and network components must be connected to an uninterruptible power supply (UPS) in the edge, core, and cloud segments. Completing this enumeration exhaustively would go beyond the scope of this thesis. Therefore, we will focus on the points that can be reasonably implemented in a test environment with virtualised resources and services within the scope of what is possible. For us, this means that we will focus on the measurement of trustworthiness in relation to the availability of servers and services. Specifically, this means microservices, virtual machines, and applications running on them.

Lerner [62] stated that the average cost of network downtime is 5 600$ per minute extrapolating to 300 000$ per hour. Considering that this is a blog post from 2014, it is reasonable to assume that the costs may be higher in today's market and the proliferation of digitally connected services. However, a 100% uptime is currently an idealistic goal that is unlikely to be included in any SLA. As a customer downtime must therefore be expected in practical conditions. Typical so-called availability nines are listed below in Table 9.

Table 9: Availability "Nines" with anticipated Downtime by Stanford, Wray, and Millsap (2021) [63]

Uptime	**Downtime per Year**
90%	36.53 days
99%	3.65 days
99.9%	8.76 hours
99.95%	4.38 hours
99.99%	52.56 minutes
99.999%	5.26 minutes

The Australian telecommunication provider Telstra Corporation guarantees in their SLA a 5G network uptime of 99.9% during business hours [64], i. e. 8.76 hours downtime per year. Thus, planned outages outside these time ranges are excluded. In 2020 cloud providers such as Amazon Web Services (AWS) and Microsoft Azure promise a monthly uptime of 99.95% respectively 99.99% in their SLA [65]. Nevertheless, it is important to note that the classification into trusted and untrusted availability depends on the specific use cases. For example, an uptime of 99.999% may be desired for a high-risk application such as in healthcare but for an application that is only used from Monday to Friday, it is perfectly acceptable that the application is unavailable at the weekend. The deployment of 5G networks in healthcare has already found acceptance, such as for continuous monitoring of electronic medical devices or in the case of real-time care during an emergency to bridge the gap until help arrives. There are also use cases for the deployment of 5G networks in the energy sector as demonstrated in [66], [67]. For example, for the remote reading of gas, water, or electricity meters as shown in [68]. In this way, resources can be used in a more environmentally friendly way, namely only when they are actually needed. In the second example, high availability is also indispensable, but not to such a critical extent as in the first example. It is, therefore, necessary to assess how critical the systems really are and, accordingly, whether a three or even four nines availability is actually required. This can be particularly important for smaller companies, as a high uptime is automatically associated with higher costs.

Reliability: As mentioned earlier in the section *Reliability*, reliability can depend on various factors. A high system age, poor hardware, or software design as well as external factors such as environmental factors or constant high system load may have a negative impact. Some of these factors can be determined from information in log files. For instance, information that can be traced back to a high system load is written to log files by certain applications. Hardware and software errors are also responsible for failures in the application, which in turn should also be visible in log files. Nevertheless, there are many different log files except for application-specific ones. Some files log information about events such as system error messages, shutdowns, changes in system-relevant configuration files, and so forth. Some log files include so-called log levels, which are messages that indicate to the administrator how relevant a log entry is. Depending on the severity of an event, such distinctions can be powerful for filtering important messages. Lonvick [69] describes in RFC 3164 that each Syslog message should have a severity level indicator. In Table 10 those severity levels are depicted including the corresponding descriptions.

Table 10: Severity Levels for Syslog Messages by Lonvick (2001) [69, Tab. 2]

Severity	Description
Emergency	System is unusable
Alert	Action must be taken immediately
Critical	Critical conditions
Error	Error conditions
Warning	Warning conditions
Notice	Normal but significant condition
Informational	Informational messages
Debug	Debug-level messages

Nowadays many applications use their own logging framework, but the log levels are typically consistent [70]. In some cases, other terms are used, or individual severity levels have been omitted. Common terms are listed in Table 11.

Table 11: Commonly used Logging Levels by Kuć (2020) [70]

Log Level	Description
FATAL	One or more key functionalities are not working.
ERROR	One or more functionalities are not working.
WARN	Unexpected system behaviour inside the application.
INFO	An informative event happened. Can be usually ignored.
DEBUG	Events considered to be useful during debugging.
TRACE	Showing step-by-step execution, useful during debugging.

Apart from log files, however, there are other factors that influence reliability, such as the previously mentioned issue of poor hardware and software design. The usual way to fix software bugs and general functional and security issues is through software updates or patches. A system that has not been patched regularly is considered high risk [71]. This is a common issue for systems like IoT [72], [73]. The reasons for this are varied and depend on the type of IoT devices. For example, Industrial IoT (IIoT) devices are designed to work with obsolete components. In some cases, IIoT devices are also designed to be long-lasting or are already obsolete themselves. Additionally, there are no real standards to ensure security on IoT devices, leaving manufacturers on their own.

Performance: As we have seen in section Performance, the term is described amongst others as a measure of capacity, power, and elasticity. To go through all the possibilities of performance measurement would go beyond the scope of this thesis, so we will focus on what we consider to be the most important parameters for measurements in virtualised environments. Especially in distributed systems, resource allocation is an important issue that has already been addressed in [74]–[76]. The hardware resources that are allocated to the system should be reasonably assigned. If a single machine experiences a high work load the allocation of the appropriate resources should happen fast and reliably. Furthermore, monitoring system utilization is essential to guarantee performance. Depending on the applications, processes, or services running on a system, the workload of the individual system resources may vary. Ellingwood [77] describes general but crucial parameters such as typical host-based metrics used in monitoring tools that are coupled to the operating system or hardware. Such metrics may be the CPU, memory, disk, or processes running. In addition, application metrics focus on the health and load of the application running on the machine respectively the operating system. They usually monitor error and success rates, resource usage, or latency of response. Network and connectivity metrics focus on the measurement of packet loss, bandwidth utilization, and latency among other things.

In Table 12 three software products are compared based on their default CPU utilization thresholds. Ipswitch's WhatsUp Gold, an IT infrastructure and network monitoring software, set the default threshold to alert when the CPU utilization exceeds 90% for more than 30 minutes [78]. Compared to the other two products, the monitoring software has only one alert parameter configured. The 5950 FlexFabric Hewlett Packard Enterprise (HPE) switches are configured with three default gradings where 99% is classified as severe, 90% as minor, and 60% as recovery CPU usage threshold [79]. According to the configuration guide of the Huawei Service Routers, the NE05E series are configured with a default overload threshold of 95% and an alarm recovery threshold of 75% CPU usage [80]. In summary, Table 12 shows that values above 90% are generally considered critical and require immediate action. CPU utilisation values between 60% and 75% and below are generally considered non-critical.

Table 12: CPU Utilisation Thresholds

	Ipswitch WhatsUp Gold [78]	**Hewlett Packard Enterprise 5950 FlexFabric [79]**	**Huawei Service Routers NE05E [80]**
Severe/Overload	>90%	>99%	>95%
Minor/Medium	-	>90%	-
Recovery	-	>60%	>75%

Cooke [81] emphasises that generally speaking, CPU usage spikes may regularly appear on high workloads, for example over a longer period of time when running backups overnight or when rendering video content, depending on what kind of server we are dealing with. In addition to CPU metrics, monitoring memory utilisation is also a key component in many monitoring applications [82]–[84]. For example, a steady increase in memory utilisation can be an indicator of a memory leak. In other words, the allocated memory resources are not released when the processes end. This could degrade system performance over time. The CPU and memory utilization are similarly calculated as there is a general formula to calculate utilization: required resource at a given time divided by the total capacity of the system [85]–[87]. The result is multiplied by 100 to get the utilization as a percentage. The following equations show how to calculate memory utilization.

$$U_t = \frac{R_t}{T} * 100$$

Eq. 3-1 [86]

where U_t is the total memory utilization and R_t the required memory at time t, which can also be described as a subtraction of the $free\ memory\ F$ at time t from the $total\ memory\ T$:

$$R_t = T - F_t$$

Eq. 3-2 [86]

F_t consists of the $total\ memory\ T$ minus the sum of the consumed as C_1, buffered as B and cached as C_2 memory at time t:

$$F_t = T - C_{1,t} + B_t + C_{2,t}$$

Eq. 3-3 [86]

The CPU utilization may also be calculated using the CPU idle time:

$$U_t = 100 - I_t$$

Eq. 3-4 [88]

where I_t refers to the idle time at time t.

As an example, the endpoint agent TrendMicro monitors host memory utilisation where the default threshold of the 'Warning' alert is set to 70% and ‘Critical' when usage has exceeded 85% [89]. The default value for a high memory utilization in Ipswitch’s WhatsUp Gold is set above 90%, where everything between 80% and 90% is similar to the ‘Warning’ alert in TrendMicro. Everything below 80% is rated as acceptable [90]. For example networking gear, Huawei Service Routers the NE05E series is configured with a default overload threshold of 95% and an alarm recovery threshold of 75% memory usage, exactly as the default thresholds of the CPU usage according to their documentation [80]. In Table 13 these tools are compared based on their default memory utilization thresholds.

Table 13: Memory Utilization Thresholds

	TrendMicro DeepSecurity [89]	**Ipswitch WhatsUp Gold [90]**	**Huawei Service Routers NE05E [80]**
Critical/Overload	>85%	>90%	>95%
Warning/Medium	70% - 85%	80% - 90%	-
Acceptable	<70%	<80%	>75%

Apart from memory utilisation, another important performance metric is disk access [91]–[93]. Since large amounts of data cannot simply be cached or stored in Random Access

Memory (RAM), this is an important requirement for ensuring workflow. Observing disk performance is not only important for plain database systems, but also server systems or hosts in general. The cooperation between CPU, memory, and disk usage should not contain any serious bottlenecks in order to ensure consistently good system performance. When it comes to disk metrics, it is important to distinguish between physical and logical disks [94]. The physical disk refers to the complete physical disk while the logical disk refers to the disk partitions, which is useful to monitor or for troubleshooting when having more than one logical partition on a disk.

Another important term to consider is Input/Output Operations Per Second (IOPS), which is a performance metric that measures the input and output commands per second [95]. A distinction is made between read and write IOPS, which indicates the average number of outputs respectively inputs per second. The higher the IOPS value, the faster the data carrier. Another important disk performance metric is byte throughput, which is the effective bandwidth expressed in MB/s [96]. Both the logical as well as the physical disk can be measured using those metrics. Nevertheless, there are other metrics to determine disk bottlenecks, namely latency and queue length [96]. Latency refers to the average number of milliseconds it takes to get a response from the disk. Here we also differentiate between read and write latency. The disk queue length refers to the number of input and output (I/O) requests in the queue and therefore waiting to be sent to the storage system. According to Grande [96], the disk latency should be below 15ms. Noticeable performance issues are caused by a disk latency of above 25ms. Nagy [97] states that an average disk read and write access up to 10ms is good and it is also acceptable if it is less than 20ms. Any higher value needs further investigation. In Linux systems *iostat* is a common monitoring tool for command-line CPU and I/O statistics for devices and partitions [98]. It measures among other things the read and write requests metrics. Therefore, the time spent on the requests in the queue and the time to serve them. They are referred to as *r_await* and *w_await* and are given in milliseconds. Petrovic [94] defined recommended average disk sec/read values. These values are depicted in Table 14.

Table 14: Recommended average Disk Sec/Read Values according to Petrovic (2014) [94]

Value (ms)	Performance
< 8	Excellent
8 - 12	OK
12 - 20	Fair
> 20	Bad

Furthermore, according to [94] during excessive I/O operations the maximum peak may reach up to 25ms but constant high values above 20ms should be considered bad and indicate poor performance. Disk access values above 25ms are generally considered as bad. As described in [96] disk latency should be below 15ms for good performance and bad performance is noticeable with rates above 25ms. According to [94], [97] values that can be considered good are lower in contrast to [96], namely below 8ms and below 10ms respectively. The comparison of the metrics regarding disk I/O access is listed in Table 15.

Table 15: Comparison of Disk I/O Access Values according to [94], [96], [97]

Performance	Petrovic (2014) [94]	Grande (2019) [96]	Nagy (2011) [97]
Good	< 8ms	< 15ms	< 10ms
OK/Fair	8ms – 20ms		< 20ms
Bad	> 20ms	> 25ms	> 20ms

Security: Security is an important benchmark for building trust. If end-users do not feel safe in the corresponding environment, trust will decrease accordingly. However, achieving 100% security is illusory, so system administrators and application developers must make compromises in certain areas. Attempts must be made to offer the correct level of security; too much, for example, could be detrimental to user-friendliness and usability, while too little could lead to a great extent of loss. Moreover, there are contractual agreements such as Security Service Level Agreements (SSLAs) which must be fulfilled by service providers.

The question is how to implement a sufficient level of security so that users can build up enough trust in the system environment and accordingly ensure that the systems are well enough protected against potential attacks. In the sub-chapter *Security*, we talked about the CIA triad which stands for confidentiality, integrity, and availability, i. e. data should be secured from unauthorised read access, unauthorised modification, and available when needed. As this definition can be very comprehensive depending on the system, we focus on network slices in NFV infrastructure. In terms of a safe and secure environment keeping systems up to date regardless of the environment is a key task [71], be it related to network devices, servers, IoT whether on hard- or software, in the cloud, or on-premises. Up to date means patching the operating system or, depending on the system, the firmware as well as the corresponding applications. In addition, it is important to ensure encrypted communication [99]–[102]. This depends on the system and application. With an encrypted client-to-server or server-to-server tunnel, for example, this is possible in the application layer with Hypertext Transfer Protocol (HTTP), Internet Message Access Protocol (IMAP), Post Office Protocol (POP), or Simple Mail Transfer Protocol (SMTP). In the remote maintenance or data transmission area with Secure Shell (SSH), in the radio network area with Wi-Fi Protected Access 2 (WPA2) and WPA3, and in Internet Protocol (IP) networks with Internet Protocol Security (IPSec). According to Firoozjaei et al. [103], security risks that need to be considered in the area of NFV include side-channel attacks and shared resource misuse attacks. Yang and Fung [104] summarised and enumerated security challenges in NFV infrastructure and provided corresponding solutions for each scenario adopted from [105]. The challenges and solutions are mentioned in Table 16.

Table 16: NFV Infrastructure Security Challenges, Solutions, and Requirements by Hawilo, Shami, and Hasal (2014) [105, Tab. 5]

Security Challenges	Solutions and Requirements
Unauthorized access and data leakage	Virtual machines are only available for authentication controls.
Shared computing resources: CPU, memory, etc.	Data should be encrypted and accessed only by the Virtual Network Functions (VNFs)
Shared logical-networking layer and shared physical Network Interface Cards (NICs)	Secure networking techniques should be adopted, such as Transport Layer Security (TLS), IPSec, and SSH.

The enumeration of security challenges in [105] is structured based on the division of NFV into different functional domains, namely the hypervisor, which is the virtualization environment, and the computing, infrastructure, and application domain. In Table 16 security challenges of only the first three domains are listed. However, as it would go beyond the scope of this thesis to go into each of these individual security challenges, they are omitted and are only mentioned here for the sake of completeness.

VNFs are typically deployed on top of virtual machines, which consist of the emulated system including the operating system, storage, and network functionalities [106]. Thus, it is important to not only secure the virtualised software environment but also the underlying system. Most virtual machines are deployed on Linux Kernel-based Virtual Machine (KVM) or VMware vSphere hypervisors [107]. Macaulay [108] discussed the topic of IoT and NFV and stated that open source software will dominate the network, namely as with KVM and other Linux-based solutions. In Rao's [109] enumeration of open source packages for network functions virtualization, Linux-based distributions led the list in virtual switching and routing software. Like any operating system, there are extensive security mechanisms in Linux-based systems. From patching the operating system to permissions and firewalls, the principles and benefits are obvious. One difference, however, is Security-Enhanced Linux (SELinux), which supports access control security policies for access file permissions, network ports and other resources [110]. In practical scenarios, SELinux is mainly used to restrict processes. That is, processes such as containers or virtual machines can be isolated so that other processes cannot access them. SELinux is a kind of an internal system firewall protecting access to processes. An alternative to SELinux is ApplicationArmor (AppArmor), the former is among others used on RedHat and the latter on Debian-based systems. AppArmor is said to be easier to administrate while SELinux is more secure [111].

Privacy: Outsourcing sensitive data to a cloud environment often comes along with the question of how the provider handles sensitive data in terms of privacy. In [112] the problem of the interrelation between security and privacy is mentioned. For example, if privacy is increased by anonymous access to cloud providers, this also affects security, as malicious users can now hide behind the cloud systems with malicious intentions. Privacy-preserving approaches in cloud computing have been summarised from various works in [112] and are depicted in Table 17.

Table 17: Privacy-Preserving Approaches in Cloud Computing by Y. Xiao and Z. Xiao (2013) [112, Tab. 2]

Approach	**Description**	**Example**
Information-Centric Security	Data objects with access-control policies.	A data outsourcing architecture combining cryptography and access control [113].
Trusted Computing	Constant expected system behaviour with hardware or software enforcement.	TCCP [114]; Hardware token [115]; Privacy-aaS [116].
Cryptographic Protocols	Employed cryptographic techniques and tools to preserve privacy.	FHE [117] and its applications [115].

The approach of a Trusted Cloud Computing Platform (TCCP) provides a closed-box execution environment for Infrastructure as a Service (IaaS) services. The main goal is to provide a trusted cloud computing platform, where system administrators are unable to access the memory of a hosted VM, even with root privileges on the physical node [114].

Together with the hardware token verification and Privacy-as-a-Service these approaches are listed in the *Trusted Computing* row in Table 17. Gentry [117] proposes the Fully Homomorphic Encryption (FHE) which enables computation on encrypted data, meaning that data may be processed without decryption by means of which cloud providers have no insight into the customers' data. Due to the presumption that processing fully encrypted data will certainly result in high latency, Sadeghi et al. [115] suggest using a combination of trusted hardware tokens and Secure Function Evaluation (SFE). Both approaches are listed in Table 17 in the row *Cryptographic Protocols*. Further details about the *Information-Centric Security* approach are not described in more detail in [112], thus for further information reference is made to [90].

4 Method

In this chapter, we will describe how our DTM was implemented and subsequently tested. First, we will discuss the test environment and our trust calculation approach. The individual parameters are revisited and the procedure for gathering the values of the sub-parameters is described. The allocation of values to degrees is shown and explained in more detail. The approach to visualise the grades will be presented and finally, an overview of the system environment is provided at the end of the chapter.

4.1 System and Coding Environment

Our focus in this work is on virtualised environments. While many competing cloud providers currently offer their services, such as AWS, Microsoft Azure, or Google Cloud Platform, we focus on open-source solutions. As a system environment, we worked on the National Centre of Scientific Research "DEMOKRITOS" (NCSRD) in Greece. Access to this system was provided to us via the Institute of Applied Information Technology (InIT) at the ZHAW. The open-source platform OpenStack is used as IaaS. The lightweight Kubernetes distribution MicroK8s [118] is used for the automated construction of containerised applications. In this system, a cluster is running based on the Kubernetes example application Sock Shop [119]. The demo example shows a web shop selling socks. The deployment of this shop consists of different Kubernetes pods within one namespace as can be seen in Figure 5. Istio has a running container to get the data out of the pods. This data is then collected by Prometheus and visualized by Grafana. This environment already existed in NCSRD environment prior to our work, and we used it as an implementation and test system for our DTM. Python was used as a programming language and GitHub Actions to create a Docker image. Microk8s was used for deploying our DTM in our test system.

4.2 Trust Calculation

We based our trust calculation on the trust score as in [17]. The authors proposed a calculation using five parameters, namely *persistence, competence, reputation, credibility, and integrity,* and then calculated the trust score by giving weight to each parameter and summing them together. Because the authors wanted to keep it simple, they used the weight of 0.2 for each parameter which leads to an average of the parameters. We changed them to the trust parameters as in [14], namely *correctness, safety, availability, reliability, performance, security, and privacy*. Afterwards, the average of those parameters was calculated as in [17]. A discussion about the sub-parameters and weights used are provided in the following sections. Table 32 summarises the most important metrics of our trust calculation.

Scale: Our scale for trustworthiness is based on [21]. The authors used a scale ranging from -1 to 1, with -1 as untrusted, 0 as neutral, and 1 as trusted. We have chosen this scale as a basis for our further approach because of its simple way of interpretation as negative numbers can be considered a bad score and positive numbers a good one. Furthermore, we increased the scale to range from -5 to 5 to have a bigger extent. This allows us to detect the effects on the trust score more clearer.

Initial Calculation: Because we used an already existing system with an active Prometheus instance, we implemented our DTM to use historical data if available. If the data was available, we calculated the grades for the last 24 hours as if our system was running. If the data was not available, we set the grade to 0 for every hour, because we defined 0 as the neutral and initial value.

Updating and Averaging Grades: We split our sub-parameters into two groups, the first one was updated daily while the second one was updated hourly. For the grade of the sub-parameters which were updated every hour, we saved the last 24 values and calculated the final sub-parameter grade by averaging over it. We used this approach to mitigate the effect of spikes in our parameters. The group assignment of the parameters is summarized in Table 32 and explained in the following sections. The trust score was updated hourly.

Multiple Pods: Because the test environment consisted of multiple pods with multiple containers in the same namespace, a solution had to be found for the corresponding trust calculation. This mainly relates to the question of whether the calculations should be used individually for each test scenario or whether all systems should be checked iteratively. We split the answer to this into three subcategories: Prometheus request, kubectl request, and external request. The sub-parameters and the corresponding request category are depicted in Table 18. Figure 5 visualises these requests including the answer categories.

Table 18: Category of Answer for each Sub-Parameter

Sub-Parameter	**Answer category**
Uptime	Prometheus request
Status Code Comparison	Prometheus request
Log Level Count	Kubectl request
Patch Level	Kubectl request
Response Time	Prometheus request
Memory Usage	Prometheus request
Disk Access	Prometheus request
CPU Usage	Prometheus request
Call Correctness	Kubectl request
Vulnerability Check	External request
Certificate Check	External request
AppArmor	Kubectl request

For Prometheus requests, containers containing the pattern 'istio' in their name were excluded. This is because these pods do not directly affect the sock shop application content and are part of the Istio service mesh. Each of the remaining containers was graded. The average of all container grades resulted in the final sub-parameter grade. This approach is depicted in *Eq. 4-1*.

$$ParameterGrade_{Prometheus} = \frac{\sum_{PodContainer} Container\ Grade}{Number\ of\ Containers} \qquad Eq.\ 4\text{-}1$$

If data could be collected with the kubectl request, we calculated a grade for each container per pod. Afterwards, the average was calculated. This resulted in the grade for the corresponding pod. The sub-parameter grades were calculated by averaging over the pod grades. This is done every time the parameter is updated. This approach is depicted in *Eq. 4-2*.

$$ParameterGrade_{kubectl} = \frac{\sum_{pods}[(\sum_{containers} container\ grade)/number\ of\ containers]}{number\ of\ pods} \qquad Eq.\ 4\text{-}2$$

All queries containing external requests were directly mapped into a single grade. Thus, they did not have to be specially considered and processed.

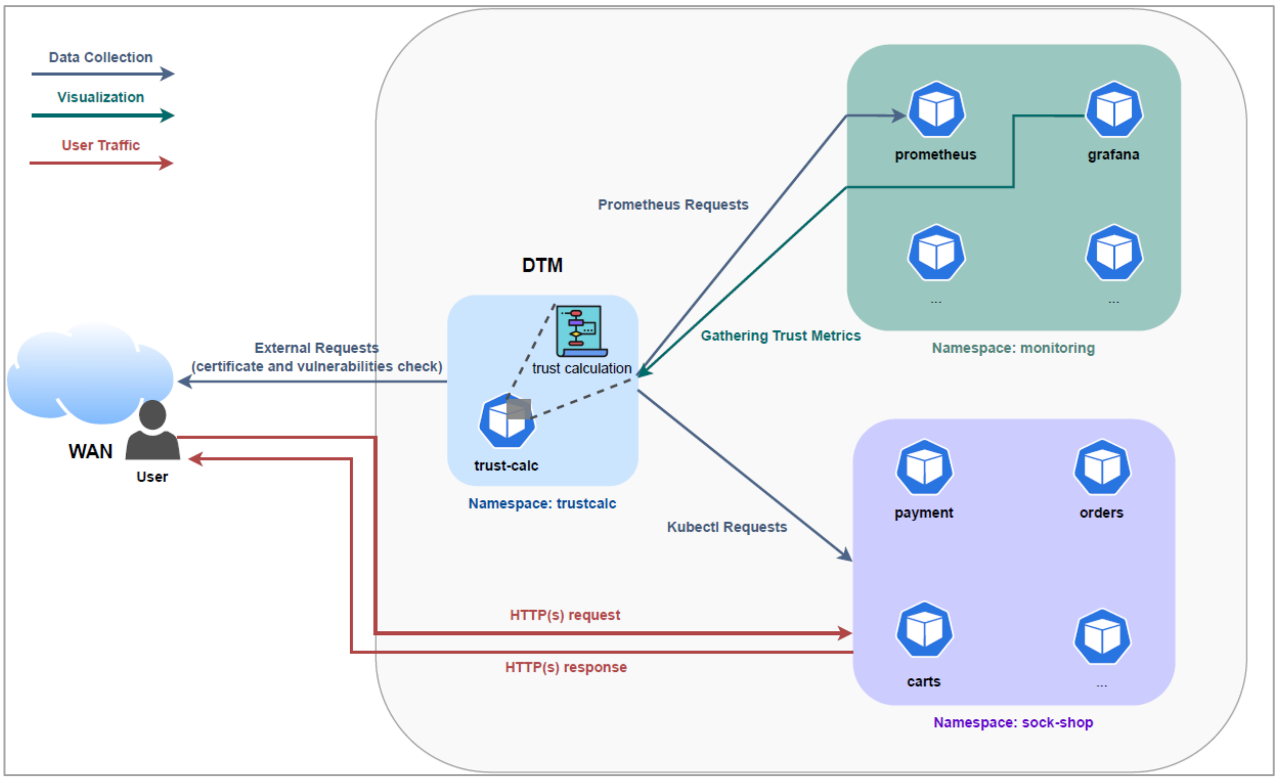


Figure 5: Visualization of Category of Answer for Gathering Trust Parameter Grades

4.2.1 Availability

For availability, we used uptime as a single sub-parameter. As depicted in Table 5, availability is described in [14] as "error-free and uninterrupted service at a given time" and reliability as "recovery time in case of system errors or failure". While in [15]–[17] availability is described as reachability of transactions, responsiveness towards a request, or accessibility and operability of a system, in our opinion these descriptions are better suited for the implementation of the availability parameter in our DTM. Reliability as interpreted by the other authors in comparison to Becker et al. [14] flows in the direction of stability and consistency or the possibility of making data available for requests. Thus, we will deal more closely with the interpretation of the availability parameter as in [14] for the implementation of the reliability parameter. We will omit the recovery time, which is not mentioned by the other authors in availability nor in reliability and it is additionally rather difficult to measure in our test system. As uptime is therefore our only sub-parameter for availability it gets a weighting of one. The uptime is calculated every hour with values queried from Prometheus. For this purpose, we queried the current value and the value from an hour ago. Afterwards, the formula depicted in *Eq. 4-3* is used to calculate the uptime in the corresponding hour.

$$\frac{uptime_{now} - uptime_{last}}{time_{now} - time_{last}} \qquad \textit{Eq. 4-3}$$

Subsequently, we graded the uptime according to the descriptions in Section Availability. We followed the availability nines listed therein according to the SLAs of Telstra Corporation [64], AWS, and Microsoft Azure [65]. This resulted in the values in Table 19.

Table 19: Scale of Uptime

Availability	Grade
99.9%	5
95%	4
90%	3
75%	0
50%	-5

We suggest *Eq. 4-4* to calculate the availability grade. *Eq. 4-4* takes the points from Table 19 and linearises the values in between these points.

$$Grade = Grade_{lower} + \frac{availability - availabilty_{lower}}{availability_{higher} - availability_{lower}} * (Grade_{higher} - Grade_{lower}) \qquad \textit{Eq. 4-4}$$

For example, if the uptime was 80% the calculation is as depicted in *Eq. 4-5*:

$$Grade = 0 + \frac{80\% - 75\%}{90\% - 75\%} * (3 - 0) = 1 \qquad \textit{Eq. 4-5}$$

The grade for uptime was calculated every hour and for the final grade, the average from the last 24 values was calculated. Also, if available by Prometheus, at initiation the values were calculated.

4.2.2 Reliability

For reliability, we used three sub-parameters: Status Code Comparison, Log Level Count, and Patch Level. This decision is based on the principles from the Section *Reliability* for the assessment of reliable systems.

Status Code Comparison: The Status Code Comparison is a comparison between the 500 HTTP request status to the 200 HTTP request status. Thus, it describes a comparison between errors and calls without errors. To calculate this, we queried Prometheus for the current number of 200 and 500 answers and the number of 200 and 500 answers an hour before. We then calculated the amount of 200 and 500 answers in the last hour and then divided the amount of 500s by the amount of 200s. This quotient was then made into a grade according to Table 20.

Table 20: Scale of Status Code Comparison

500's by 200's	Grade
< 0.25	5
< 0.5	0
> 0.5	-5

Since it can generally be assumed that if the 500 status codes outweigh the 200 status codes, the system is not reliable, the worst score was given according to this weighting. Prometheus gave us the results for multiple pods. Therefore, we calculated the grade for every pod and averaged them together for the grade of this hour. This was done every hour and, if available, at the start time of the DTM the value for the last 24 hours was calculated. The final grade of Status Code Comparison was calculated by taking the average of the last 24 calculated grades.

Log Level Count: Because the web shop consists of multiple pods, we calculated the Log Level Count for every pod and made an average over all pods to get the number of the Log Level Count. To get this value we accessed the logs of every pod and counted the number of warnings and errors. After retrieving the log level number of every pod, we calculated the overall average. For the initial calculation, we counted the number of all errors found in the logfiles. Since the age of the log files is not determined, the number of errors and warnings was set high. The scale for the initial calculation can be found in Table 21.

Table 21: Initial Scale of Log Level Count

Log Level Count	Grade
< 1000	5
< 3000	0
> 3000	-5

After the initial calculation, we saved the number of the Log Level Count, so we could constantly update the number of newly added log messages at hourly intervals. We calculated the grades during the updates with the errors and warnings from the last hour. Therefore, we used another scale, which is depicted in Table 22.

Table 22: Update Scale of Log Level Count

Log Level Count	Grade
< 10	5
< 40	0
> 40	-5

For the final grade of the Log Level Count, we averaged over the last 24 values. The last 24 values of the default vector consisted of zeros and the initial calculation was set as the newest value.

Patch Level: To calculate the Patch Level, we examined every pod for available updates. We did this by using the kubectl and exec commands and the internal update system of the corresponding Linux distribution to get the available updates. If the system is outdated and updates are available, we returned -5 as a grade. If the operating system is up to date but some of the used packages are outdated, we gave back -2.5 or 2.5 as a grade according to the number of out-of-date packages. And if everything was up to date, we set the grade to 5. To decide which distribution is up to date, we used the distribution lifecycles as a guide. If an OS version is not yet End-of-Life (EOL) respectively End of Support, it is considered trustworthy. This is depicted in Table 23.

Table 23: Comparison of Versions of Linux Distributions

Linux Distribution	Current Version
Ubuntu	18 or 20 [120]
Debian	10 or 11 [121]
Alpine	3.12 to 3.15 [122]

After calculating the grades for all pods, the averaged grade is used as the Patch Level grade. As updates are normally not installed several times a day, we calculated the Patch Level grade once per day and did not average the grade over multiple calculated grades.

The sub-parameters and their weights are shown in Table 24. As the log level count can be high despite a good and reliable system, for example when many warnings are thrown, but the system nonetheless works, we weighed this parameter half in comparison to the other two.

Table 24: Weights of Reliability

Sub-Parameter	Weight
Status Code Comparison	0.4
Log Level Count	0.2
Patch Level	0.4

4.2.3 Performance

The performance grade is calculated with the help of four sub-parameters: Response Time, Memory Usage, Disk Access, and CPU Usage. These four parameters are gathered from Prometheus and include the state-of-the-art performance values similar to common monitoring solutions as described in Chapter Performance. We weighed them equally except for response time, which we gave a doubled weighting. This was done because our test system is a web shop and response time is the parameter that is directly experienced by customers. The resulting values are shown in Table 25.

Table 25: Weights of Performance

Sub-Parameter	Weight
Response Time	0.4
Memory Usage	0.2
Disk Access	0.2
CPU Usage	0.2

Response Time: The Response Time was gathered from Prometheus and grades were calculated according to [123]. This gave the values in Table 26.

Table 26: Scale of Response Time

Response Time	Grade
< 0.5 s	5
< 1s	0
>= 1s	-5

Memory Usage: The Memory Usage was gathered from Prometheus and grades were calculated according to Table 13. We have decided to use the values according to [89] as a guide, as this contains the strictest values respectively the lowest memory usage as thresholds. This resulted in the values in Table 27.

Table 27: Scale of Memory Usage

Memory Usage	Grade
< 70%	5
< 85%	0
> 85%	-5

Disk Access: Disk Access was split into two parts: Disk Write and Disk Read. For each of the two, the data was gathered from Prometheus, and grades were calculated according to [124]. Since the Prometheus query includes the quotient of the time of the disk read respectively write query in seconds by the completed queries, percentages are assigned to the grades for the weighting of the parameters. This gave the values in Table 28.

Table 28: Scale of Disk Access

Disk Access	Grade
< 25%	5
< 50%	0
> 50%	-5

After calculating the grades for each part an average of the two, i.e., for the Disk Read and Disk Write Access was calculated, this resulted in the Disk Access grade.

CPU Usage: The CPU Usage was gathered from Prometheus and grades were calculated according to Table 12. For the allocation of the CPU usage to the highest grade, the recovery grade as in [80] was used. The overload value as in [78] was used for the worst weighting. The only available value as in [79] was used as the minor respectively medium grade. This gave the value mapping as depicted in Table 29.

Table 29: Scale of CPU Usage

CPU Usage	Grade
< 75%	5
< 90%	0
> 90%	-5

All sub-parameter grades were calculated hourly and for the final grade, the average from the last 24 values was calculated. Also, if available by Prometheus at initiation the values for the last 24 hours were calculated.

4.2.4 Correctness

For correctness, an Application Programming Interface (API) call was executed every hour. Subsequently, we compared the response to the values stored in the database. If the values corresponded to the values stored in the database, grade 5 was given. If the values did not match, grade -5 was set. Different from the other sub-parameters we only used one pod in the correctness calculation. This pod was the catalogue pod. It consists of three databases and four API endpoints. Since the API calls are very similar to each other, only two API calls were executed and compared to the database. Each call got a grade, and we took the average of the two to calculate the correctness grade. Similar to the availability parameter the correctness parameter only consists of one sub-parameter, i.e., the weight corresponds to the factor one. As with the other hourly calculated sub-parameters the final grade of correctness was calculated by taking the last 24 values and averaging them. In difference to the other parameters, the last 24 values were not calculated at initiation, because API calls cannot be executed in the past.

4.2.5 Security and Privacy

Due to the overlapping of the two parameters Security and Privacy, we decided to put them together into one parameter. As a basis for the sub-parameters, we oriented ourselves to the theoretical foundations from Chapter 3.4. Each of these parameters was updated daily because changes to the system which will affect this parameter are not done regularly, but mostly once a day maximum.

Vulnerability Check: We planned to use trivy [125] as a tool for Common Vulnerabilities and Exposures (CVE) checks because it offers the possibility to work together with microk8s, i. e. all deployed pods within a namespace would be scanned automatically. Trivy needs to update its CVE database at least once a day. This was not possible due to restrictions in the test system environment rejecting the database download. Due to this restriction, we implemented a lighter version of a vulnerability check consisting of SSL Labs Scan [126], Nikto [127], [128], and Mozilla's HTTP observatory [129]. All these products were queried via API and then the results were transferred into a grade. Because these products only check domain names which are available from the outside or standard ports for web servers, we couldn't use our test environment for the checks as access to the sock shop is only provided over its IP address and the TCP port number 30001. Nonetheless, this has no influence on the validity of the vulnerability test approach, as it is pure verification detecting enabled TLS versions and cipher suites as well as detecting misconfiguration and outdated components. After querying the three products, we gave every result a grade from -5 to 5 and then took the average over the three grades. SSL Labs Scan and Mozilla's HTTP observatory returned grades from A to F, where SSL Labs additionally returns the values T and M if the tool encounters an out-of-scope situation. This occurs in the case of certificate name mismatch (M) and if the site certificate is not trusted (T)* [130]. We transferred these grades into one of our grades as depicted in Table 30.

Table 30: Scale of SSL Labs Scan and Mozilla's HTTP observatory

Product Grade	Our Grade
A	5
B	3
C	1
D	-1
E	-3
*F, (T, M)**	-5

Nikto's result consists of a list of vulnerabilities and misconfigurations detected. We linearised this number into a grade according to *Eq. 4-6*.

$$Grade = 5 - \min\left(\frac{number}{2}, 10\right) \qquad \text{Eq. 4-6}$$

AppArmor: As stated in the Section Security, AppArmor is a security system with which specific rights to programs and files can be assigned respectively withdrawn. It was checked whether AppArmor was enabled for each pod. If enabled, we gave the grade 5, otherwise, grade -5 was assigned. After checking all pods, we made an average over all pods which resulted in the AppArmor grade. The grade was calculated once a day and no initial calculation of past values was done because we couldn't check the past enablement of AppArmor. Also, we didn't average over multiple past values and took the last calculated value as the final grade.

Certificate Check: We verified the certificate by sending an HTTP and Hypertext Transfer Protocol Secure (HTTPS) request to the corresponding domain name. The sock shop has not implemented a certificate in our test environment since the web request takes place directly via the IP address. Therefore, a test domain name outside of our test environment was used. The HTTP request in advance to the HTTPS request was used to check if the site was available. If the site wasn't available, we gave the grade 0. If the site was available, we gathered the certificate information and checked if the certificate was valid and not expired. The validity is determined similarly to the validity determination of a browser or operating system with the help of the certificates in the certificate store. If the certificate was invalid or no certificate was available, we gave the grade -5. If the certificate was valid but expired, we gave the grade 0 and if the certificate was valid and not expired, we gave the grade 5. For better understanding, this approach is depicted in Table 31.

Table 31: Scale of Certificate Check

Certification State	**Grade**
Valid and not expired certificate	5
Site unavailable	0
Valid but expired certificate	0
No certificate	-5
Invalid certificate	-5

The grade was calculated once a day and no initial calculation of past values was done, as it is not possible to check whether and which certificates were used in the past. Also, we didn't average over multiple past values, therefore the last value is used as the final grade.

Weight: Because the vulnerability check consists of three different sub-parameters which test different aspects of security, the vulnerability check is given a weighting of 0.5. The web page certificate is one of the first indices regarding security for a customer, thus we gave the Certification Check a higher weighting compared to AppArmor. Nevertheless, AppArmor and its relevance from a privacy perspective should not lose too much emphasis compared to the other sub-parameters. Therefore, we gave Certification Check a weight of 0.3 and AppArmor a weight of 0.2.

4.2.6 Safety

Due to the limited time and resources for this thesis, we omitted the safety parameter. This had nothing to do with its importance, but with the complexity to measure this attribute reliable and simple. As Becker et al. [14] described *Safety* as a measure of severity of harm to a user or its environment, the implementation of such a parameter is not self-explanatory and differs from environment to environment. As we are bound to our test environment for this thesis, there is no direct harm towards a user or its environment even under a poor system or service conditions.

Table 32 summarises all parameters and their sub-parameters including their weighting. In addition, the update and initial calculations are given and whether or how an average is calculated over the values.

Table 32: Parameter and Sub-Parameter Collection Schedule including Weights, Update, and Initial Calculation Times

Parameter	Sub-Parameter	Weight	Update Time	Initial Calculation	Averaging
Availability	Uptime	1.0	Every hour	Last 24 hours	Last 24 grades
Reliability	Status Code Comparison	0.4	Every hour	Last 24 hours	Last 24 grades
	Log Level Count	0.2	Every hour	No	Last 24 grades
	Patch Level	0.4	Every Day	No	No
Performance	Response Time	0.4	Every hour	Last 24 hours	Last 24 grades
	Memory Usage	0.2	Every hour	Last 24 hours	Last 24 grades
	Disk Access	0.2	Every hour	Last 24 hours	Last 24 grades
	CPU Usage	0.2	Every hour	Last 24 hours	Last 24 grades
Correctness	Call Correctness	1.0	Every hour	No	Last 24 grades
Security & Privacy	Vulnerability Check	0.5	Every Day	No	No
	Certificate Check	0.3	Every Day	No	No
	AppArmor	0.2	Every Day	No	No

Finally, the trust score is calculated as depicted in *Eq. 4-7*.

$$Trust\ Score = \omega_A * A + \omega_R * R + \omega_P * P + \omega_C * C + \omega_S * S \qquad \textit{Eq. 4-7}$$

Where ω_x is set to the factor of 0.2 for each parameter.

4.3 Grafana

After calculating the grades for all parameters, sub-parameters, and the trust score, we let Grafana pull these values from our DTM to display them graphically. For this, we created four graphs, one for the trust score, one for the parameters, one for the sub-parameters and the last one for non-averaged sub-parameters. In these graphs, the course of the grades can be seen over time.

4.4 System Overview

A brief overview of the test environment with the DTM can be seen in Figure 6. The illustration shows our DTM in the lower right area. It fetches its trust metrics both from

Prometheus and directly from the sock shop, which is monitored by our DTM. Prometheus also fetches metrics from the sock shop. As not all metrics can be retrieved directly from Prometheus, some must be obtained directly from the sock shop namespace by the DTM. Prometheus has an API interface that simplifies the retrieval of metrics for our DTM. To process and visualise the data, Grafana pulls the parameters and trust scores from our DTM via the JSON API plugin. This data is made available via Python Flask.

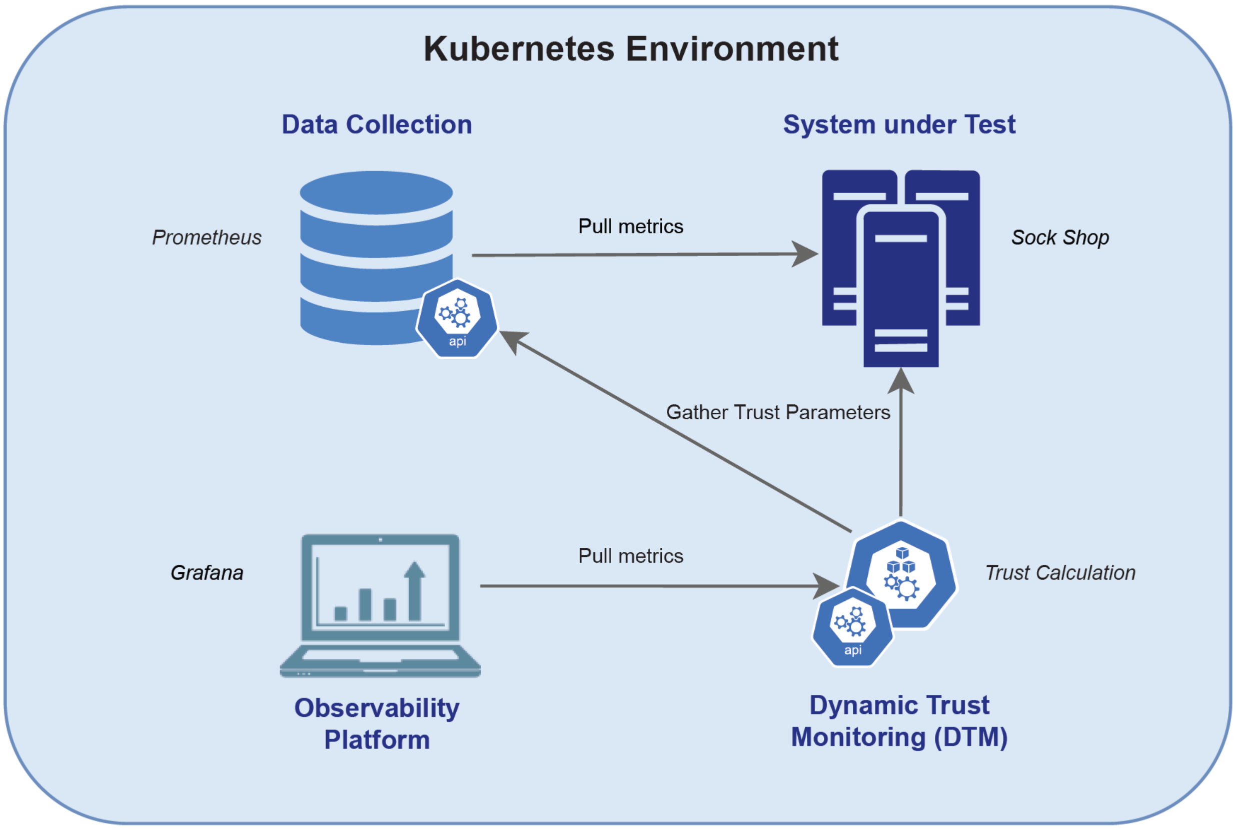


Figure 6: High-Level Overview of the DTM Test Environment

An overview of the DTM in terms of gathering the trust scores and the dynamic trust decision for mapping the values to a grade is shown in Figure 7. As depicted in Figure 5, the system under test is the Sock Shop, but values are also obtained from Prometheus and external websites for the vulnerability and certificate checks. The DTM, respectively for the parameter groups only, i.e., parameter monitoring, calculates the scores according to the parameter group and assigns them to a grade. It is checked whether this grade exceeds a certain threshold. According to this decision, the final values are assigned. This calculation refers to the individual sub-parameters and the parameter groups correspond to the average of these parameters, according to their weighting. For reasons of clarity, however, this has been omitted from Figure 7.

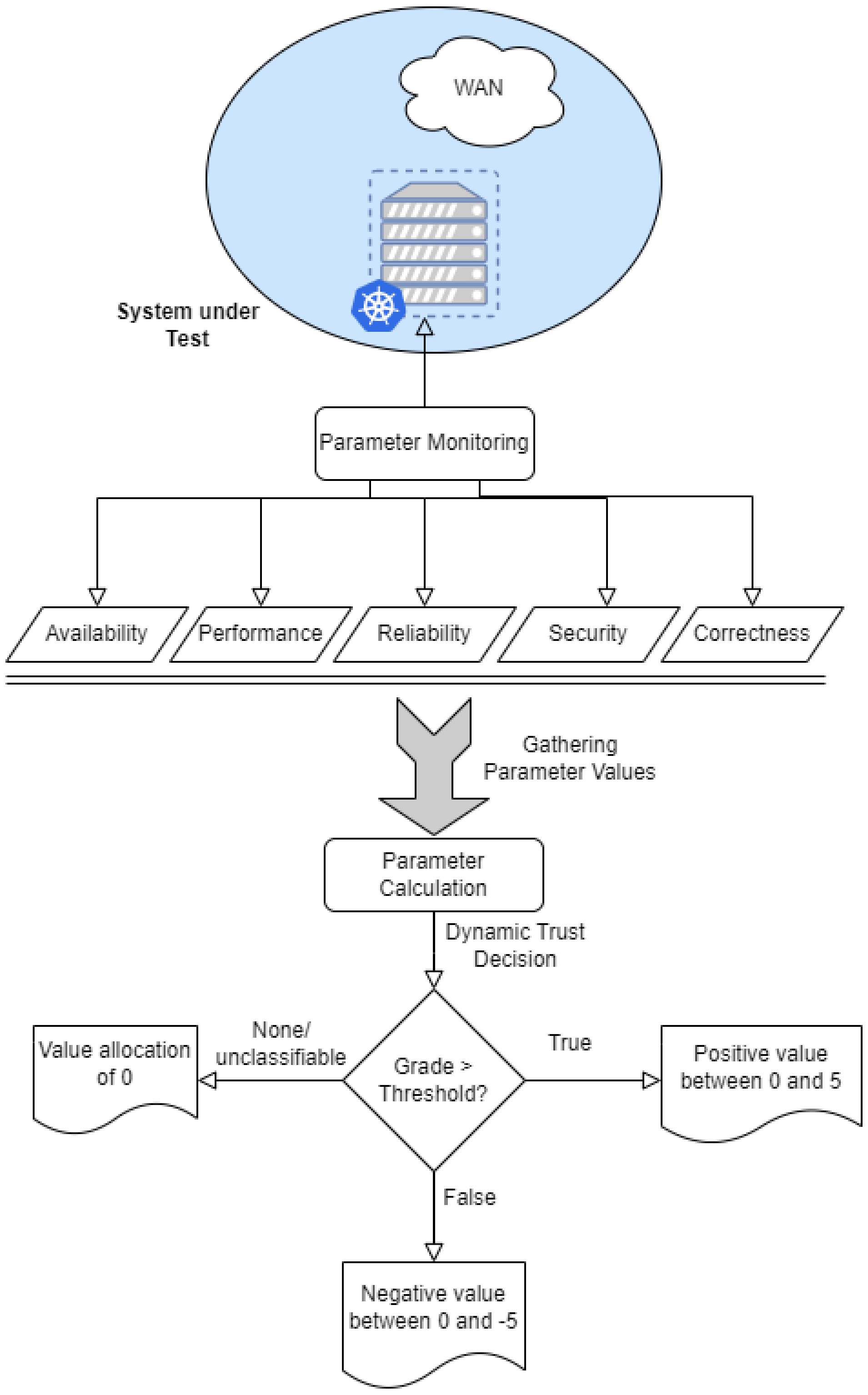


Figure 7: Flow Chart of Gathering Trust Parameter Grades and Dynamic Trust Decision

5 Experiments and Results

The experiments were conducted in stages and focused on changes in the (sub-)parameters and the resulting trust score. Due to the different parameters, it was considered how individual parameter groups could be changed, but also how cascading scenarios could be achieved. On the one hand, experiments were considered that focused on resource overloads, such as increased CPU and memory utilization and disk access rates. In addition, response error codes were deliberately generated. Due to the difficulty of consciously influencing some parameters so that they would nonetheless correspond to a realistic scenario, some parameter groups were not manipulated for these experiments. Since the implementation of the vulnerability checks has only been completed after the outcome of the first tests, they are only discussed in Section 5.5 and are therefore not included in the previous experiments. Thus, the trust score in the previous tests was obtained without the influence of the vulnerability checks. This also applies to the invalid certificate test in Section 5.3. At this point, the security grade only involved two sub-parameters, the certificate check and AppArmor. However, since the vulnerability check consists of a constant value, which does not include sporadic changes, this is negligible and does not influence the interpretation of the results. The test scenarios and approach are now explained, followed by a presentation of our results. The interpretations of the results are discussed in Section 6.

For the tests, the hourly update time was reduced to five minutes and therefore the daily update time to two hours, so that results were immediately visible and, if necessary, adjustments could be implemented without losing time. All data sets are listed individually in Appendix C: (Sub-)Parameters during Testing.

5.1 Baseline

Before the first tests were implemented, we collected initial data of the test environment in the normal state, i.e., without high load and in everyday operation. This allowed us to make comparisons with the values that emerged from the experiments. Immediately after starting the DTM, time is needed to initialise the results before the parameters converge to the normal state. This applies especially to those parameters that do not have vector values of historical data to calculate an average as a parameter grade. In the standard state, the DTM needs about one day as initialisation time. Since we have lowered the times for the data collection for the experiments, they only require 2 hours. In the normal state, the trust score reaches an average value of 3.36, this is depicted in Figure 8. The parameters except for correctness remain constant at the values shown in Table 33. The corresponding sub-parameters are listed in Table 34. Solely the correctness grade changes sporadically and reaches different values as shown in Figure 9.

Table 33: Values of the Parameters in the Normal State of the DTM

Parameter	Value
Availability	5
Performance	3.55
Reliability	1.94
Correctness	Not consistent
Security	4.88

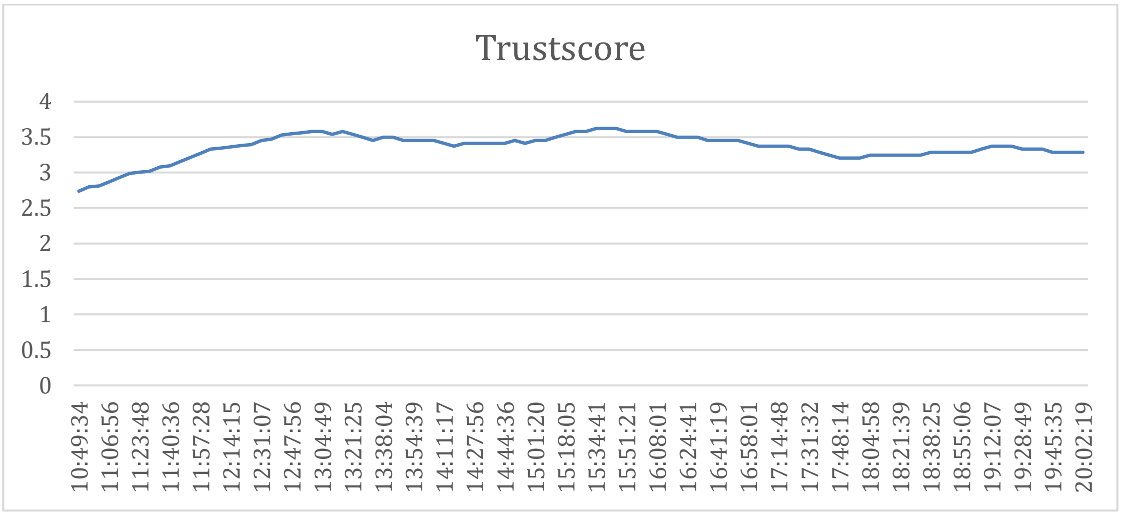


Figure 8: Baseline Trust Score

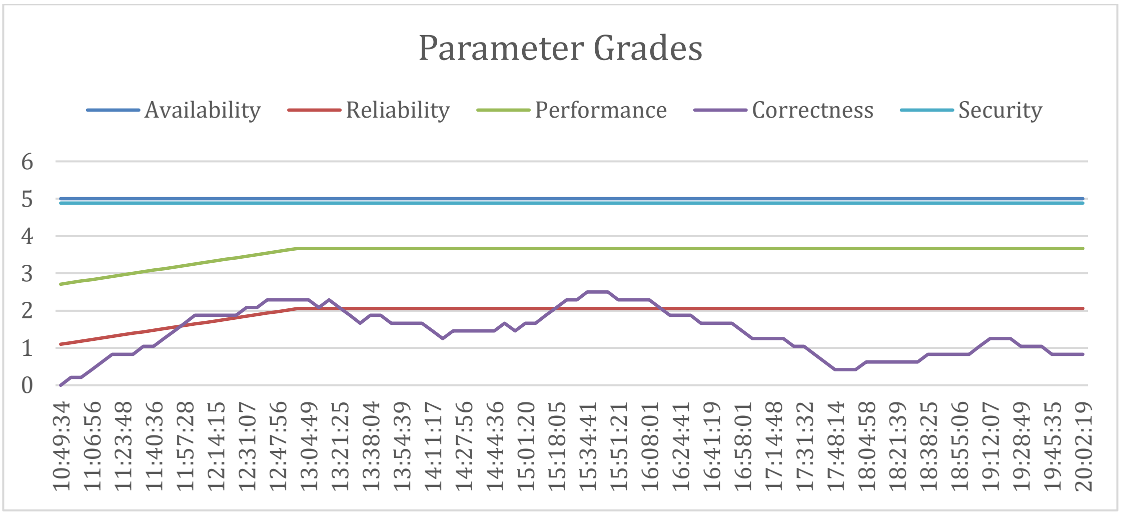


Figure 9: Baseline Parameter Grades

The sub-parameters and their values are listed in Table 34 and visually depicted in Figure 10. Since the correctness and availability parameters consist of only one sub-parameter, these values are the same as in Table 33. The sub-parameters have been colour-coded according to the parameter group so that the assignment is easier to see. The order and corresponding assignment can be seen in Table 33.

Table 34: Values of the Sub-Parameters in the Normal State of the DTM

Sub-Parameter	**Value**
Uptime	5
CPU Usage	5
Response Time	5
Disk Read	5
Disk Write	-3.33
Memory Usage	2.5
Log Level	5
Patch Level	-2.35
Status Code Comparison	5
Call Correctness	Not consistent
AppArmor	4.71
Certificate	5

Please note that since most of the sub-parameters reach constant values of 5 in the baseline, their graphs overlap in Figure 10 and are accordingly not recognisable.

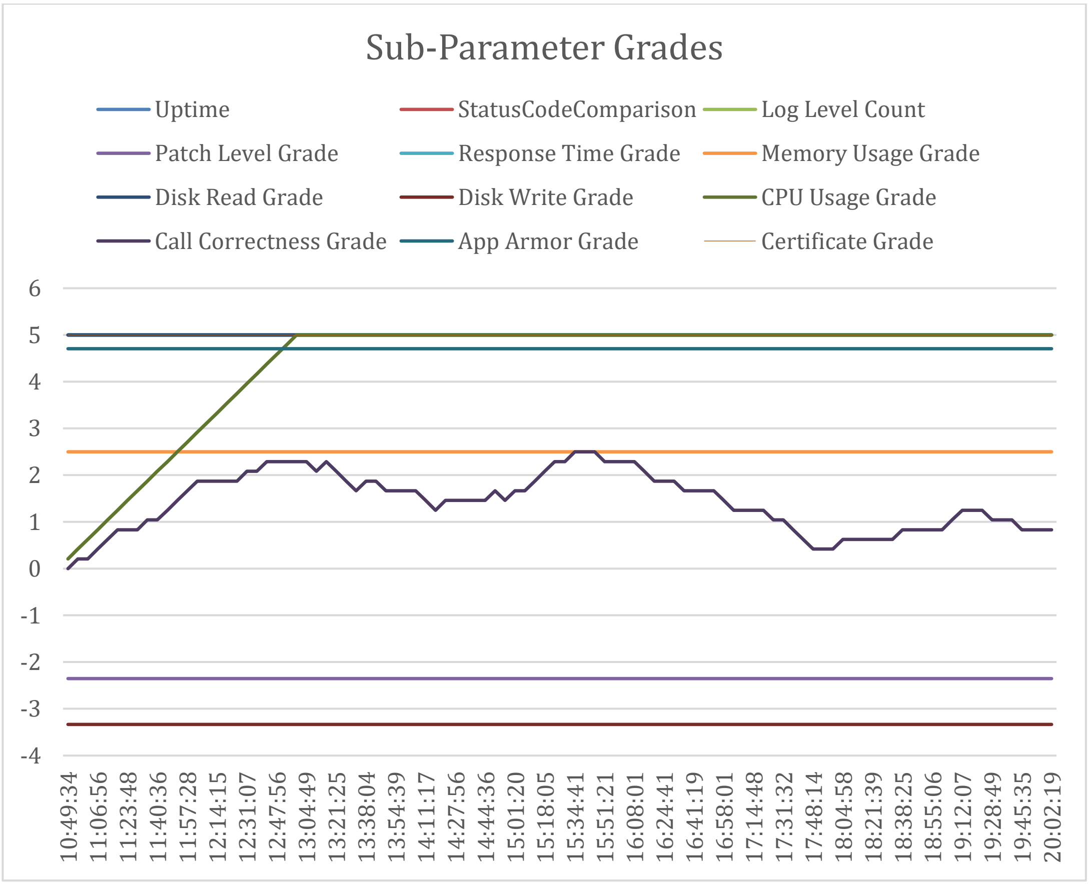


Figure 10: Baseline Sub-Parameter Grades

5.2 Denial of Service (DoS)

Based on considerations of realistic scenarios, an event from 2017 was taken as an example. A large Swiss shopping website was down on Black Friday [131]. However, due to the high demand for discounts, many users reloaded the website instead of waiting. Due to the high volume of requests, the servers of the online retailer were subjected to a further high load, causing an unintentional Distributed Denial of Service (DDoS) attack on the website, triggered by the users themselves. However, such a case can not only occur on websites but also in a similar way on services in the cloud, 5G Core, 5G Mobile Edge Computing (MEC) or on IoT devices. The idea behind a DoS test is to be able to influence as many trust parameters as possible at the same time or consecutively so that a series of scenarios can be easily recognised in the result of the trust score when it drops noticeably.

First, an approach using Istio's Traffic Management Module [132] was considered. Artificial request timeouts were generated by setting a fixed delay of 5s on a selection of pods. This generates delays not directly at the network but the application level. This approach does not affect other services generating or routing traffic via the same channel. This attempt delays the traffic but has no great influence on the individual trust metrics. Sporadic HTTP status code errors were generated, and the response time decreased, but the aim was to cascade trust metrics. For this reason, other tools were included to carry out this test. The Istio Traffic Management YAML files used can be found in the Appendix C: Istio Traffic Management.

Additionally to Istio's Request Timeouts module, the open-source Python program *Golden Eye* [133] was used as a test kit for the DoS attack. The tool allows the user to configure the number of simultaneously operating web sockets, web workers, and the HTTP method of the attack, i.e., 'get'. post' or 'random'. The difficulty of this test lies within the test infrastructure because the IP address of the Kubernetes cluster is the same as the one of the sock shop. Therefore, care had to be taken not to attack the cluster per se to the point of unresponsiveness, but to cause enough requests for results to be visible in the trust score.

The first test has been started with 1000 web workers but had to be stopped immediately because there were noticeable changes in the response time of the SSH connection and this strengthened the suspicion that the Kubernetes cluster was also affected in connection with the DoS test. Therefore, no results were gathered during the first test scenario.

The second test has been started with the default parameters, i.e., 50 web workers, 30 web sockets, and the HTTP method 'get'. This had less of an impact on the system environment. The test was run at 07:20. After a few minutes, the system became unresponsive again and the DoS test was aborted. However, the system did not recover until 09:25. The data collected during the test could nevertheless be collected. The results of the trust score and its (sub-)parameters are depicted in the following tables and graphs. (Sub-)parameters that were not changed in connection with the DoS test have been omitted from the following list. However, they are included in the calculation of the trust score.

The frontend sock shop pod appeared to be unavailable for a short period of time during the DoS attack. This had a corresponding impact on multiple parameter scores. This impact is depicted in Figure 11 for the availability, in Figure 12 for the performance, and in Figure 13 for the reliability parameter.

As depicted in Figure 11 the availability grade decreased from 5 to 4.79 at 09:00:11 and constantly maintained this value.

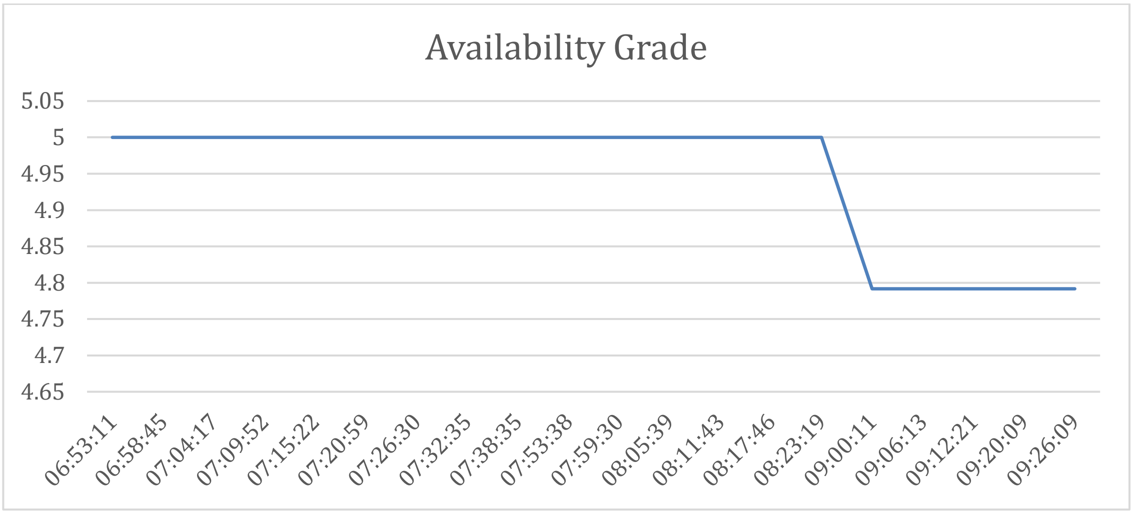


Figure 11: Availability Grade Graph throughout the DoS Test

A similar pattern can be seen in Figure 12 regarding the performance grade. Compared to Figure 11 the graph already started to decrease at 07:53:38. The value dropped from 3.75 to 3.708. Further, the value did not remain constant but continued to decrease gradually until the lowest value of 3.33 was reached at 09:20:09.

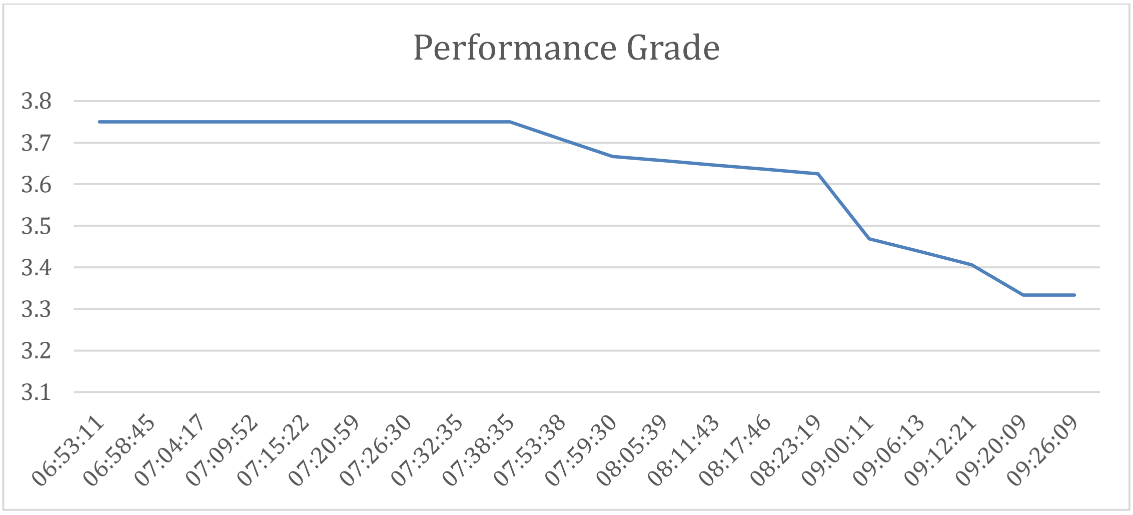


Figure 12: Performance Grade Graph throughout the DoS Test

The last grade, which was directly influenced by the DoS test, is the reliability grade. The grades started to decrease from 2.24 to 2.07 at 07:32:35. The graph reached its low point at 09:00:11 with a value of 1.99. The value remained constant until 09:26:09.

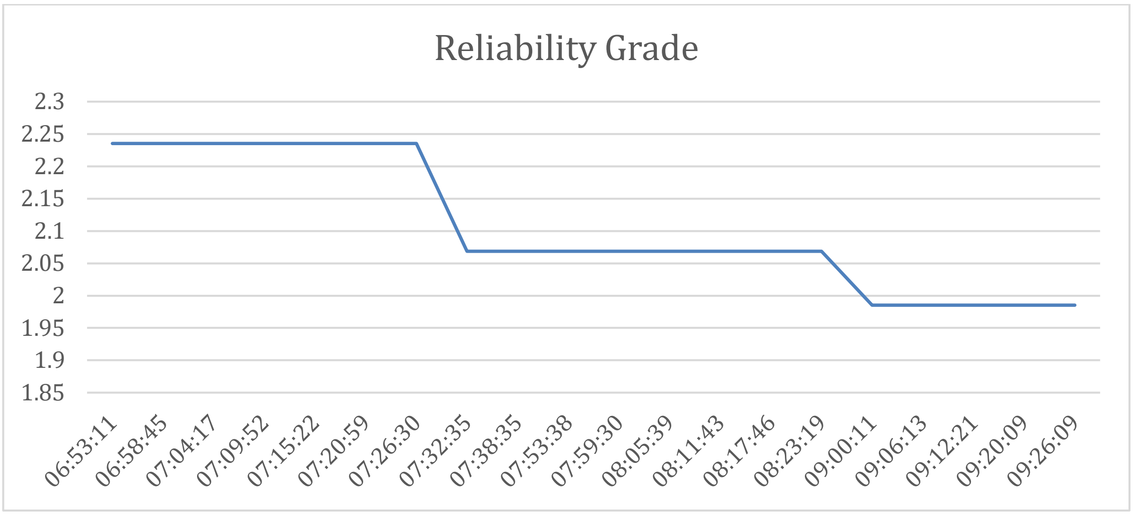


Figure 13: Reliability Grade Graph throughout the DoS Test

The parameters affected by the DoS test are listed again in Figure 14 as a comparison to each other.

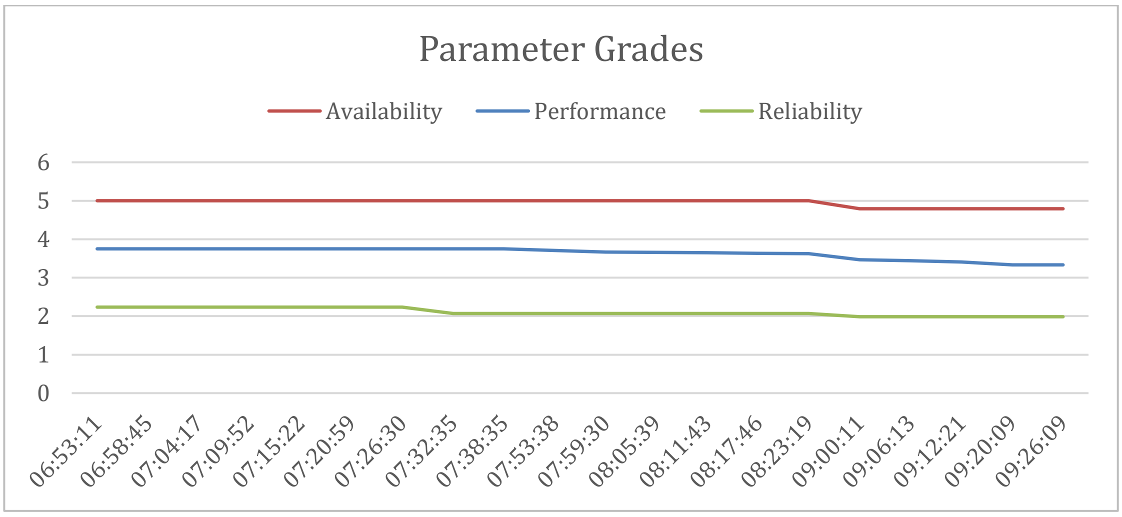


Figure 14: Comparison of the Availability, Performance, and Reliability Parameter Grades throughout the DoS Test

The individual performance sub-parameter scores are depicted in Figure 15. The DoS test had the greatest impact on the disk read score while the disk write score achieved slightly higher results throughout the attack. The test also had a marginal effect on the residual sub-parameters such as memory and CPU usage and response time.

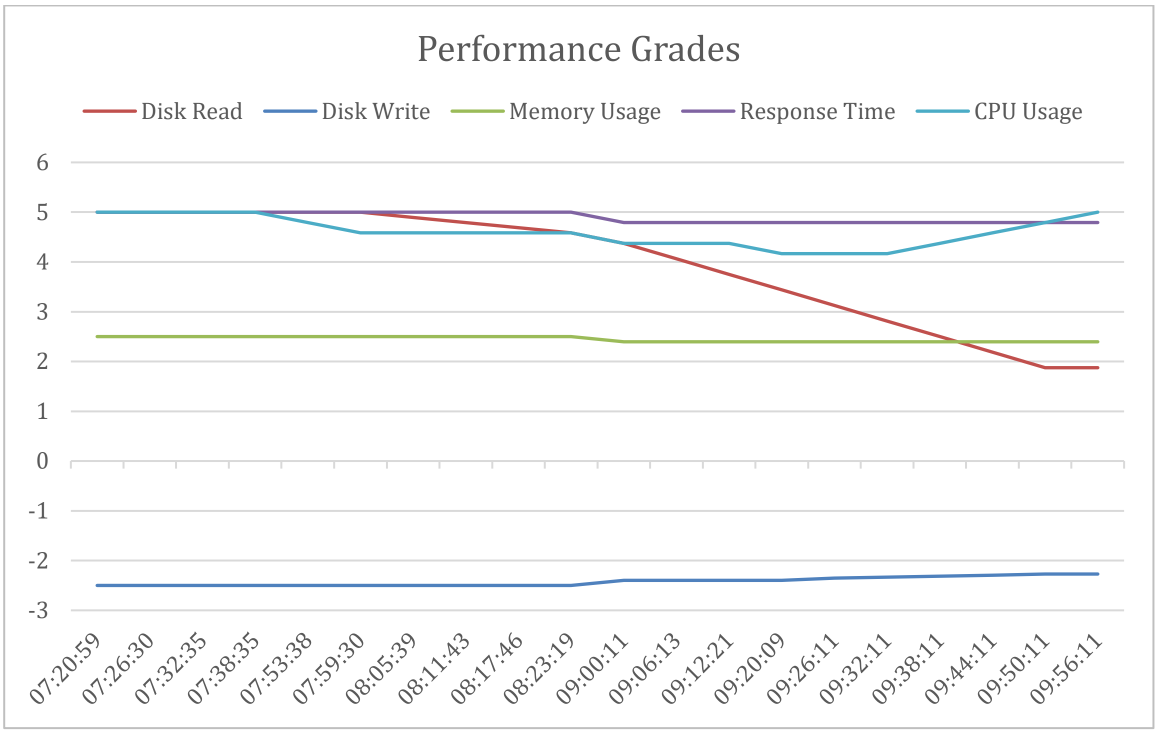


Figure 15: Comparison of the individual Performance Sub-Parameter Scores throughout the DoS Test

The uptime grade has been omitted as the availability parameter only consists of this single sub-parameter and accordingly displays the same values. Similarly, the only reliability sub-parameter affected by the DoS test has been the status code comparison grade. Thus, the other reliability sub-parameters constantly showed the same values. For this reason, these two sub-parameters are omitted here.

The trust score graph in Figure 16 shows a score of 3.63 before the beginning of the DoS test at 07:38:35. From that time onwards, the score decreased steadily. It reached its lowest value of 3.41 at 09:38:11. The trust score did not increase until 09:44:11with a value of 3.43.

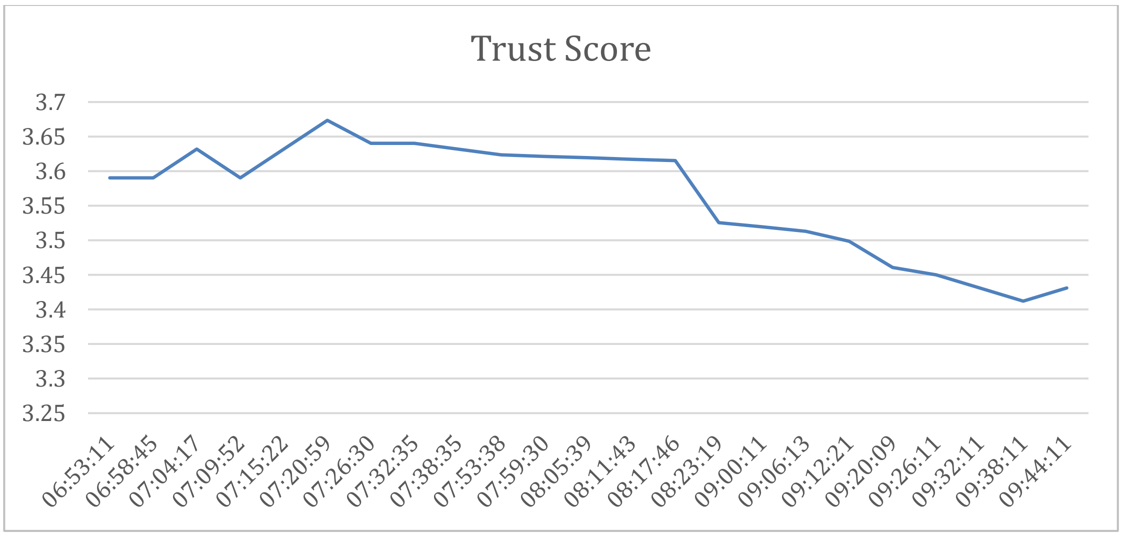


Figure 16: Trust Score Graph throughout the DoS Test

The third DoS test involved a slow increase in the parameters, i.e., the number of web workers and web sockets. The idea was to find out from which number of workers or sockets the performance slowly started to decrease without the entire environment becoming unresponsive. The test was run for a few minutes so that enough results could be obtained. Afterwards, the DoS script was cancelled and restarted with higher parameters. This approach is depicted in Table 35.

Table 35: Number of Web Sockets and Web Workers including Time of Execution of the third DoS Test

Execution Time	Number of Web Sockets	Number of Web Workers
19:15	1	1
19:45	1	2
19:56	2	2
20:16	3	3
20:37	2	3
20:52	3	2
21:07	3	4

The results of the trust score and its (sub-)parameters are depicted in the following tables and graphs. Figure 17 shows that the individual parameter values were only influenced from 20:21 respectively from 20:42 on. The reliability parameter was not affected over the entire test period.

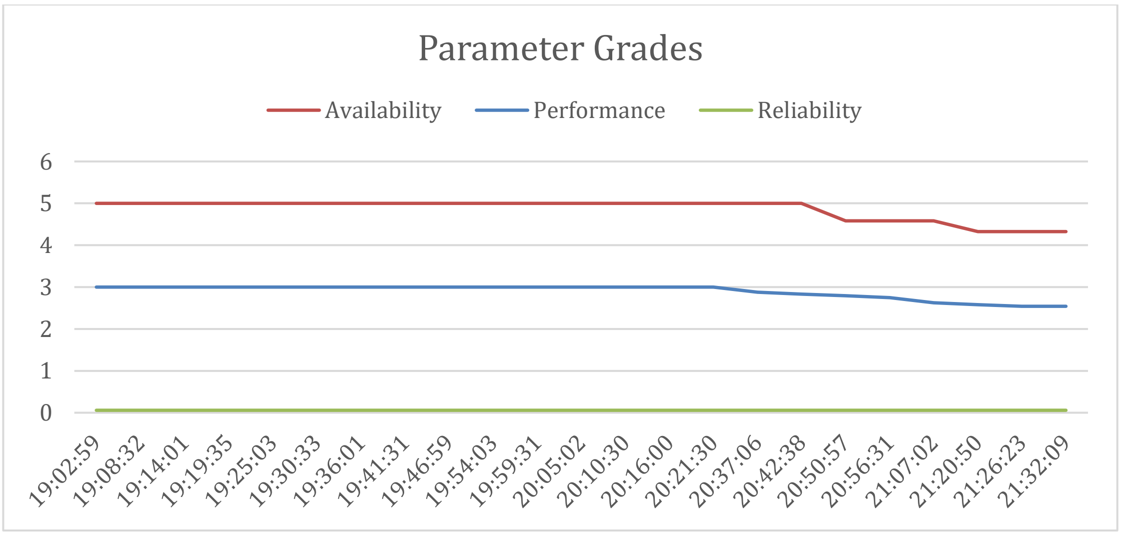


Figure 17: Comparison of the Availability, Performance and Reliability Parameter Grades throughout the third DoS Test

Figure 18 shows the different performance sub-parameters. Consistent with the values from Figure 17, it can be seen that the individual sub-parameters began to generate lower values at 20:21. These all intersected at 20:37 on the X-axis at the Y-value 0.

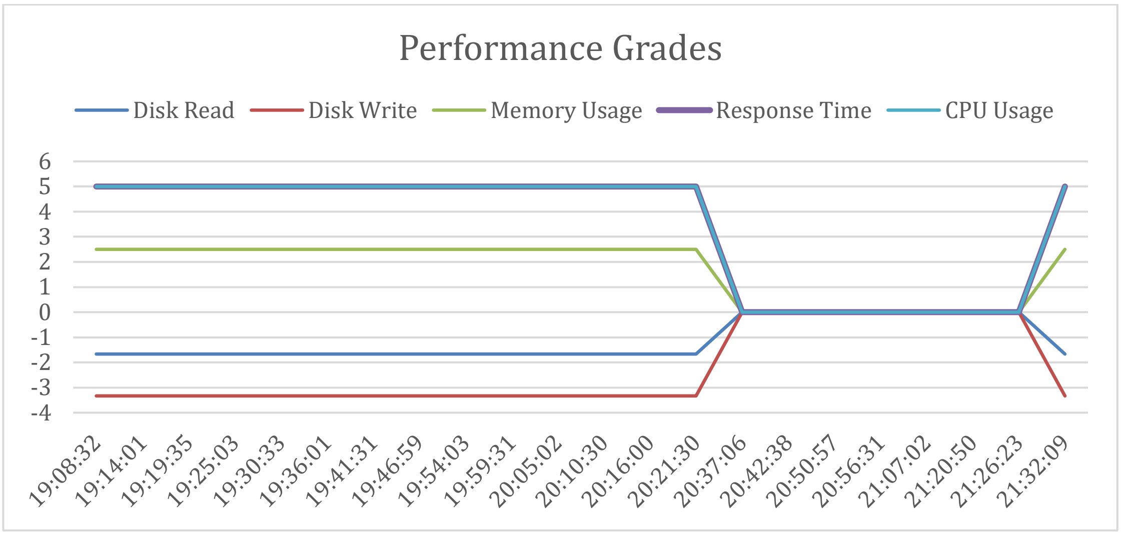


Figure 18: Comparison of the individual Performance Sub-Parameter Scores throughout the third DoS Test

Figure 19 shows how the trust score reacts depending on the input parameters of the script. It can be seen that the values not only reacted accordingly to the DoS test at 20:21 but seem to drop and rise at random. The trust score corresponded to the value 2.92 at 20:21 and 2.9 at 20:37. At 21.32 the score reached the value of 2.69.

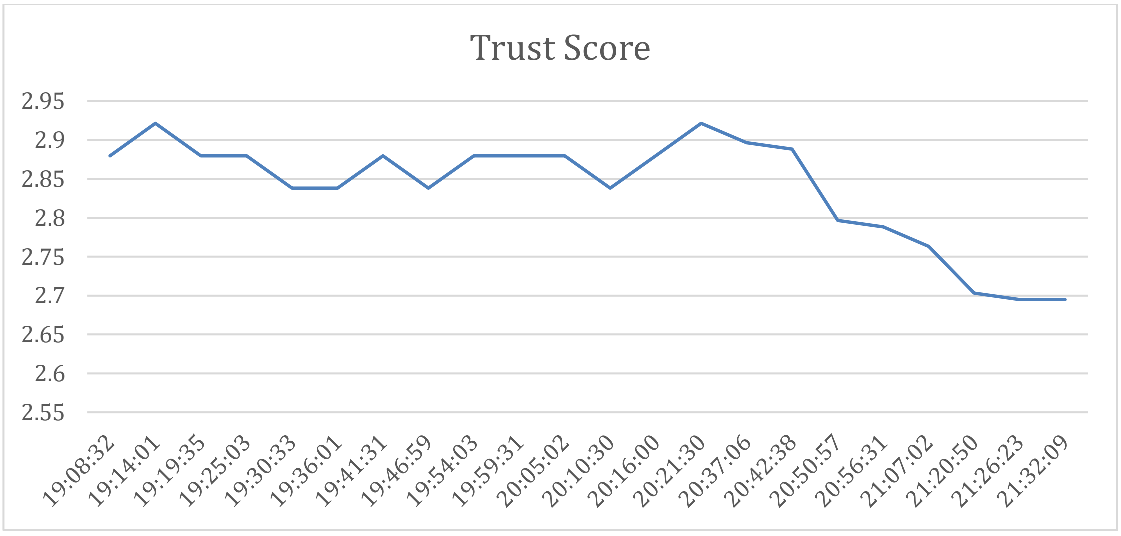


Figure 19: Trust Score Graph throughout the third DoS Test

5.3 Invalid Certificate

The following test involves checking the trust score in the absence of a valid intermediate and root certificate. The sock shop had not implemented a certificate in our test environment since the web request takes place directly via the IP address. Therefore, we decided to perform the invalid certificate test on the website of the ZHAW, which has a valid Let's Encrypt certificate. This does not influence the validity of the test, as it is pure verification using the stored certificates on the OS and is thus independent of the website. Since our container, through which we launched the tests, had no pre-installed certificates, we installed the certificates via the Dockerfile when we first deployed the container.

For the validation of the Let's Encrypt certificate, we installed the root and intermediate certificates. As the RSA intermediate certificate is cross-signed by ISRG Root X1 and TrustID X3 Root for additional client compatibility with older devices and operating systems [134], both root certificates have been installed on the container. The code snippet of the installation process of Let’s Encrypt’s Chain of Trust is depicted in Code 1.

```
RUN apt-get update \
 && apt-get install -y --no-install-recommends \
      ca-certificates \
      openssl \
 && mkdir -p /usr/local/share/ca-certificates

# install Let's Encrypt CA Root certificate
ADD https://letsencrypt.org/certs/isrgrootx1.pem.txt
/usr/local/share/ca-certificates/isrgrootx1.pem
ADD https://letsencrypt.org/certs/trustid-x3-root.pem.txt
/usr/local/share/ca-certificates/trustid-x3-root.pem

# install Let's Encrypt CA Intermediate certificate
ADD https://letsencrypt.org/certs/lets-encrypt-r3.pem
/usr/local/share/ca-certificates/lets-encrypt-r3.pem

RUN cd /usr/local/share/ca-certificates \
&& openssl x509 -in isrgrootx1.pem -inform PEM -out isrgrootx1.crt \
&& openssl x509 -in trustid-x3-root.pem -inform PEM -out trustid-x3-
root.crt \
&& openssl x509 -in lets-encrypt-r3.pem -inform PEM -out lets-
encrypt-r3.crt \
&& update-ca-certificates
```

Code 1: Installation Process of Let’s Encrypt’s Chain of Trust in Dockerfile

The first approach to impact the trust score was to delete the Let’s Encrypt root certificates from the trust store in the container while the DTM was running. This did not influence the trust score, nor did it when the intermediate certificates were also deleted.

The second approach was to deploy the Docker container without Let’s Encrypt’s root certificates. The effect on the certificate grade is depicted in Figure 20. At the start of the container, the certificate grade was in the negative range, specifically -5. This value consists of the conditions for the calculation of the certificate grade as listed in Table 31. The conditions for a non-negative value include the validity of the root or intermediate certificate. To ensure that the changes to the individual grades and the trust score can be verified when a valid certificate is available, the root certificates have been added to the trusted OS certificates after the initialisation phase of the container. The installation of the certificates is the same procedure as for the deployment of the Docker container as shown in the Dockerfile in Code 1. The gradient of the security grade can be seen between 13:42 and 13:48, the value changed from the lowest value of -5 to the highest value of 5.

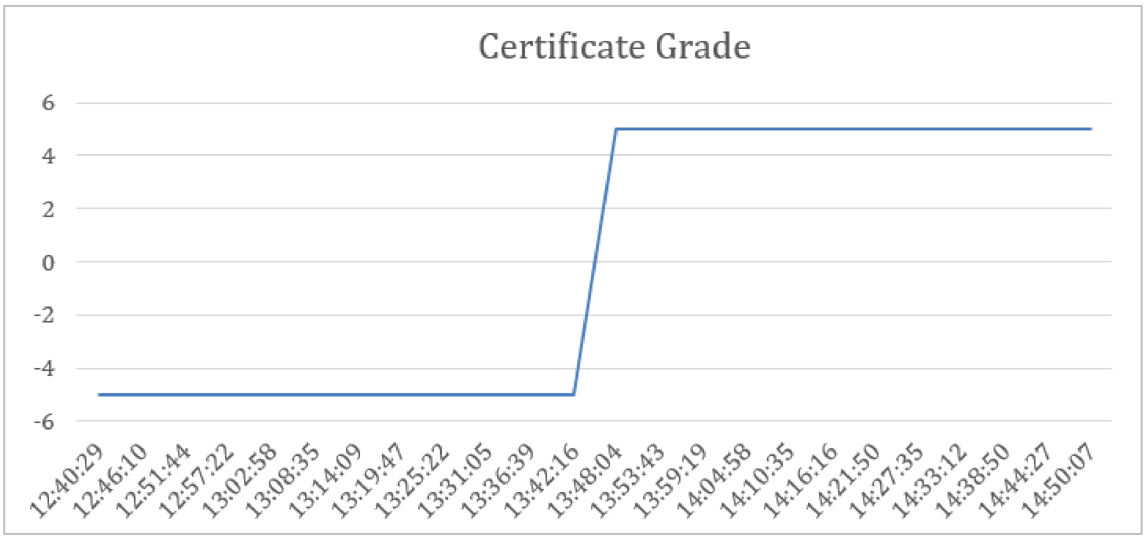


Figure 20: Certificate Grade Graph throughout the Invalid Certificate Test

As the security grade consists of both the certificate and the AppArmor grade, which has a lower weighting, i. e. 0.4, it can be seen in Figure 21 that the overall value is in the negative range. The gradient of the security grade can be seen between 13:42 and 13:48, the value changed from -1.1 to 4.88.

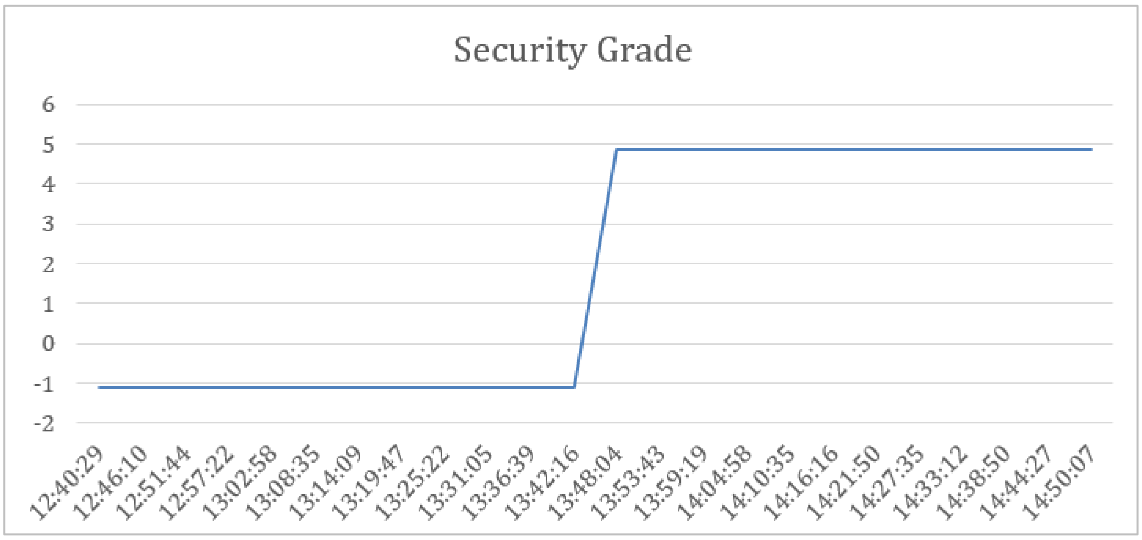


Figure 21: Security Grade Graph throughout the Invalid Certificate Test

The impact of this test on the overall trust score can be seen in Figure 22. The trust score increased from 2.43 to 3.63 between 13:42 and 13:48.

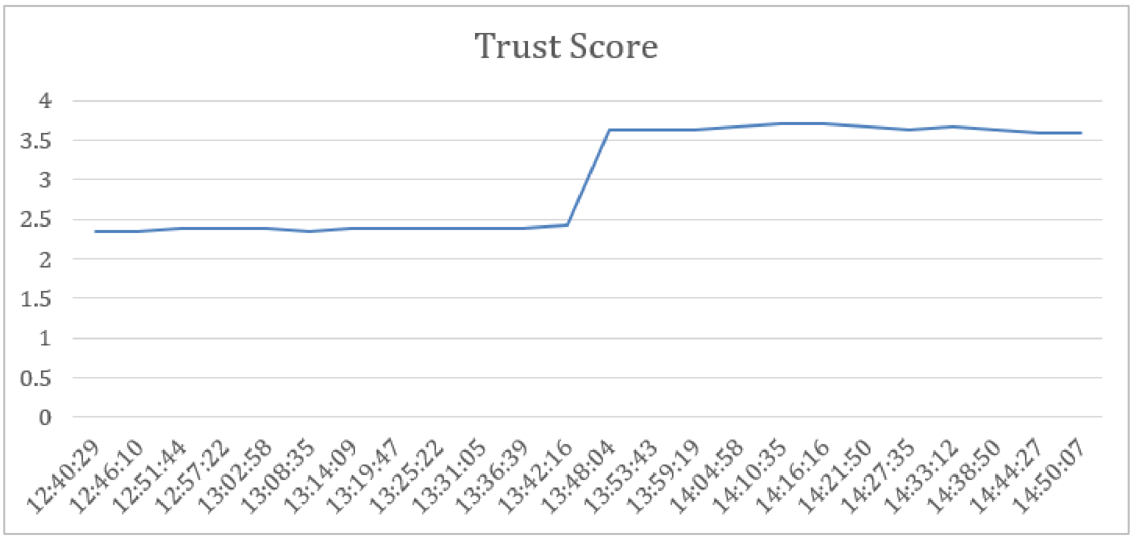


Figure 22: Trust Score Graph throughout the Invalid Certificate Test

The same test was performed without the Let's Encrypt intermediate certificate and subsequently without the intermediate and root certificates. The results were the same as when deploying the Docker container without the root certificates. Accordingly, after the initialisation phase of the container, the intermediate certificate respectively the intermediate and both root certificates were added to the trusted OS certificates. The certificate and security grades showed the same values as in the previous test approach. Accordingly, the influence of these tests on the trust score is identical.

5.4 Performance Stress Test

Another approach to influence the trust score is to execute a stress test in the pods of the sock shop namespace. This results in increased performance, i. e. the CPU and memory usage as well as the disk read, and disk write access will achieve worse grades due to the high load. A random sample of sock shop pods was selected for this test, in which the *yes* command is executed in the background. *Yes* repeatedly generates any message, default a 'y' string as output stream, until the process is killed. For the test scenario, the output is written to */dev/null*, which is a special file in Linux systems used to discard inputs. The test procedure was automated to impact several pods simultaneously. The complete script is shown in Code 2.

```
#!/bin/bash

pods=(front-end-6585d48b5c-tm6n2 catalogue-db-86c68f4757-qc24m user-db-
5569d9d7f5-wkgz6 payment-f86459795-dwvbt orders-5d8557969c-s6rgt carts-db-
7c464c9896-ptv9x user-65dcdb777-vwlmb shipping-85bcbcdf77-rgkpf)

while true
do
        for i in "${pods[@]}"
        do
                microk8s.kubectl exec -n sock-shop $1 -it -- yes >
/dev/null &
                echo $i
        done
done
```

Code 2: Bash Script for the Performance Stress Test executing the Yes Command in a List of Sock Shop Pods

The yes command to create a high CPU load is not a new approach [135]–[137]. However, according to [136], [137] this only affects one CPU core. Accordingly, the command has to be executed repeatedly depending on the number of CPU cores, which corresponds to a number of 4 per sock shop pod. Since the performance stress test will be executed several times in different pods, only one CPU core is affected in the first approach. Afterwards, the pods are tested according to the number of cores with increasing performance impact.

The first performance stress test was executed according to the configuration in Code 2. This immediately resulted in the cluster being unresponsive. The immediate abort did not result in any improvement. Thus, the entire Kubernetes instance had to be restarted. Since the containers used to collect and visualise the trust data are located on the same cluster, they no longer had any data from this test.

Further test approaches were run with individual pods from the sock shop namespace. The *yes* command was again used along with stdout redirecting the output to */dev/null*. Unfortunately, this did not affect the trust score, nor could representative results be achieved within the range of the individual performance (sub-)parameters. The short-term spikes in the area of CPU performance were visible in the corresponding affected pods, but these subsided again very quickly after a short time and never reached a higher value than 50% CPU usage. This was observed to be the same when executing the yes command several times according to the number of CPU cores. It emerged that there are corresponding cgroups, which set a value of 50% CPU usage as a limit that must not be exceeded. These groups apply to all pods in the sock shop namespace. The same applies to memory usage and other resources. Thus, the stress test cannot be executed correctly and has been omitted for further experiments.

5.5 Vulnerability

For the vulnerability tests, different domains were tested. The websites were first checked for their dissimilarities so that we could achieve wide-ranging results. The domain names were passed to our vulnerability check method using a list. With each daily update, i. e. during testing two hours, the next element from the list was reviewed. The corresponding domain names are listed below:

- moodle.zhaw.ch
- zhaw.ch
- mozilla.org
- google.com
- wikipedia.org

The parameter grade graphs of the test time range of ten hours are depicted in Figure 23.

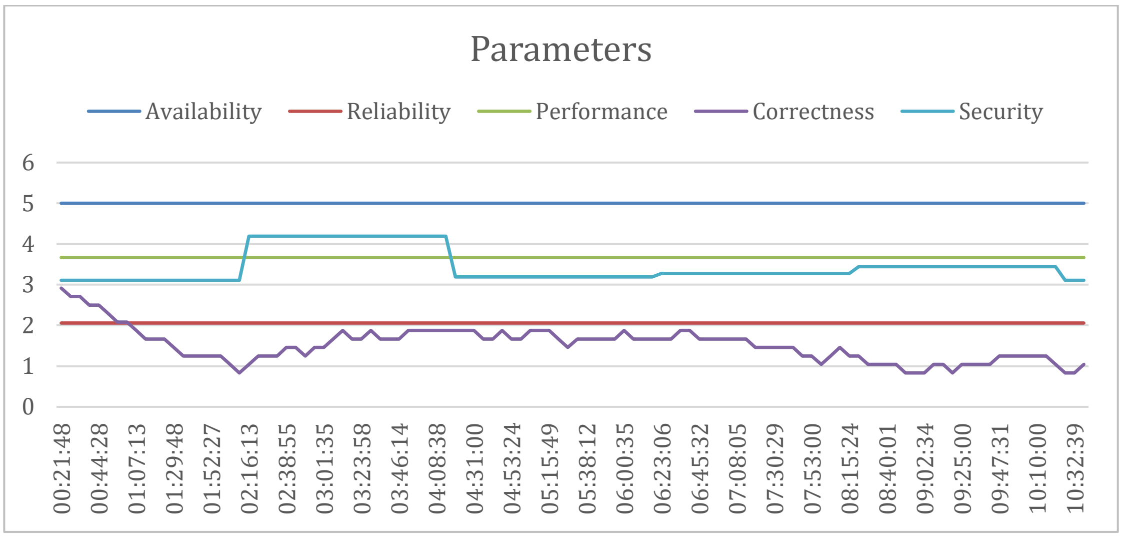


Figure 23: Parameter Grade Graph throughout the Vulnerability Check Test

As depicted in Figure 24, the parameter values affected the grade of the trust score by about 0.5 grades from the maximum to the minimum score.

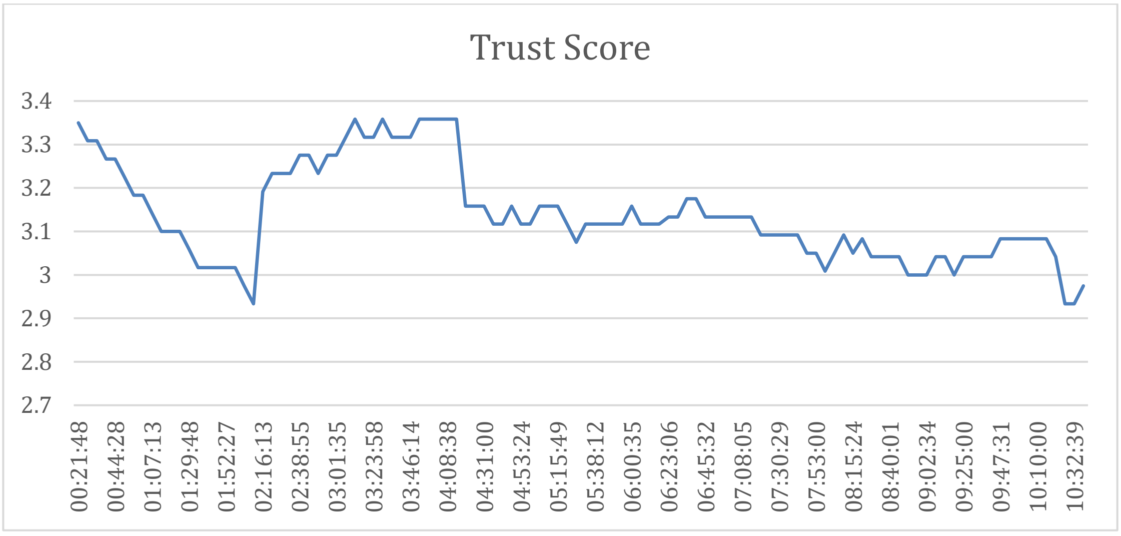


Figure 24: Trust Score Graph throughout the Vulnerability Check Test

As correctness was always fluctuating, the influence of the correctness parameter will not be discussed further in this chapter. In Figure 25 the security sub-parameters are depicted. The vulnerability check grades shifted during the test period and were directly reflected in the security grade.

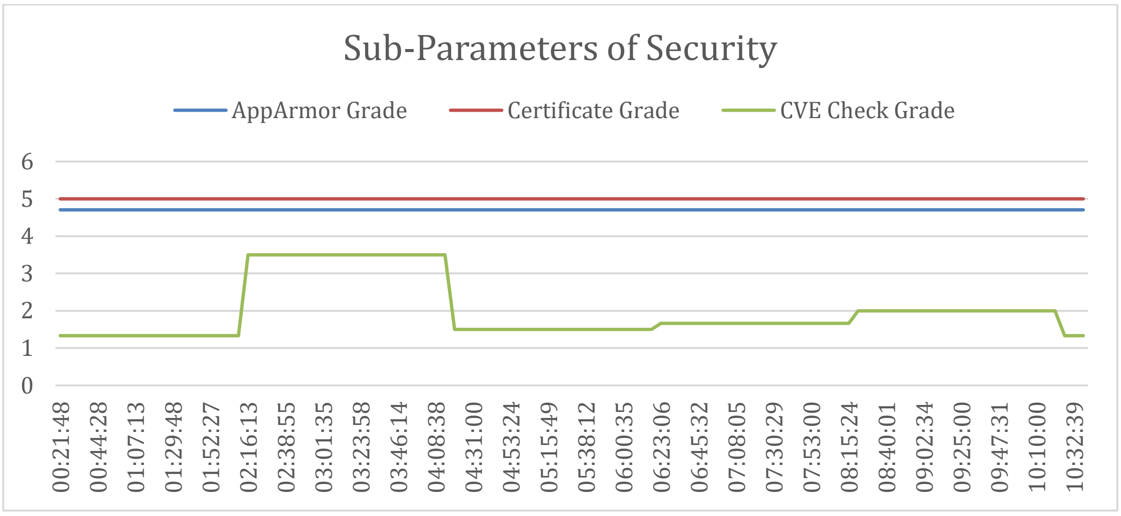


Figure 25: Sub-Parameter of Security Graph throughout Vulnerability Check Test

As the vulnerability check grade was weighted with a factor of 0.5 it can be seen that the variations in the security grade are half the variations of the vulnerability check grade. At 2:16 the vulnerability check grade changed from 1.3 to 3.5 and the Security Grade changed from 3.1 to 4.2.

When comparing the vulnerability check tools, namely SSL Labs Scan, Nikto, and Mozilla's HTTP observatory it is visible that they are independent and provide different results. This is depicted in Figure 26.

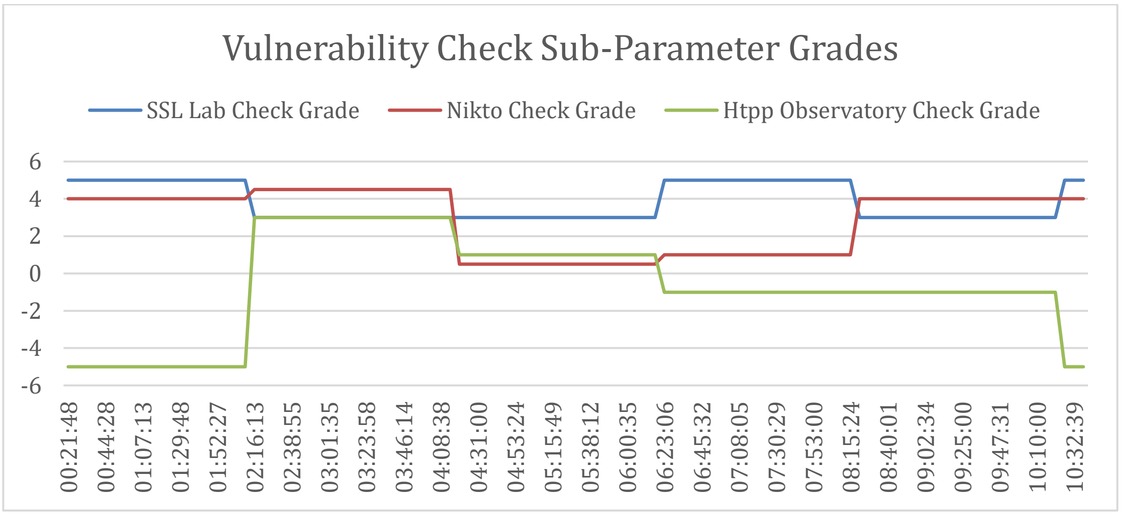


Figure 26: Vulnerability Check Sub-Parameter Grades Graph throughout the Vulnerability Check Test

5.6 Penetration Test

As another test scenario, the question arose if an attack could be detected by our DTM in terms of an impact on the trust score. We tried to answer this question with two approaches, a penetration test, and a Brute Force Attack. For both attack scenarios, the open-source program OWASP Zed Attack Proxy (ZAP) [138] was used. The Brute Force Attack and its results are described in the following Section 5.7.

The automated Scan of ZAP was used to scan the URL of the sock shop test system. The penetration test was run for about an hour and sent nearly 74'000 requests. The results of the penetration test are omitted in this thesis, as we wanted to consider this from the perspective of the defenders (e.g., customer and cloud provider) and not the attacker. The only parameter showing different values was the correctness grades. As already discussed in Section 5.1, these changes are always present and therefore not derived from the penetration test. Thus, no changes or anomalies could be detected.

5.7 Brute-Force Attack

For the Brute-Force Attack, a ZAP mode called Fuzzer was used. This mode retrieves a list of strings. For each string, ZAP sends a request containing the corresponding string value. The login mask of the sock shop was used as a target. A list of 13’390 usernames with passwords was used as login requests. The attack took about five minutes and the parameters gathered during this test period revealed no changes in the trust score, except for the usual correctness fluctuations which were not influenced by the test. This is because unsuccessful login attempts throw a 401 error, which is not verified by our DTM. Furthermore, as these kinds of errors are not logged by the sock shop the log level parameter did not detect them.

6 Discussion

In this chapter, the results of the experiments from Section 5 are construed. We will begin with the interpretations of the baseline scores and sub-parameters. Afterwards, we will discuss the fluctuations in the correctness parameter and show possible root causes that would explain this behaviour and present results without the correctness grade as well as with adjusted values. We will then revisit the experiments and interpret their influence and impact on the trust score. Finally, a conclusion is drawn as to whether the impact of the experiments on the trust score in our DTM is appropriate in a production environment. Since no results could be obtained from the performance stress test, this experiment is omitted for further discussion.

We are aware that many more interpretations could be drawn from the results and experiments. Due to time constraints, we had to limit the discussion to a few essential findings that seemed most relevant.

6.1 Baseline and (Sub-)Parameter Interpretations

Most of the (sub-)parameters achieved expected results. The exceptions are the patch level, disk write, and the correctness grade. The latter will be discussed in more detail in the next chapter, so we will focus on the first two (sub-)parameters in this chapter.

The patch level grade achieved negative values as most of the pods in the sock-shop namespace are not patchable by the default user. In addition, however, access via the root user is blocked. This is intentional by design as the root user is not created by default from the Kubernetes deployment files. This restriction is intended because in a production scenario, pods with an old application or OS version are simply replaced by new ones, and patch management is therefore not needed. Since most of the pods cannot be checked and the ones that could be successfully verified are outdated, the test results are in a negative range.

The disk write grade also achieved negative values for the most part. However, this fluctuated in comparison to the patch level grade. It is noticeable that the value achieved better results during the DoS tests. The assumption is that the value, as checked by our DTM, delivers correct results and the disk write access in the test system does not correspond to the values that would be necessary for good performance as shown in Table 15. During the DoS test, this value increased while the Disk Read Access value decreased. This change was to be expected as during a system overload the disk has no data to write and therefore the disk write access value increases because it requires fewer resources.

6.2 The Correctness Fluctuation as exemplified by the DoS Test

As already evident in the previous Chapter 5, the correctness parameter returned unexpected results. The fluctuations are well visible in the graphs in Figure 9. The variations occurred during tests as well as in the normal state. Errors in the implementation of the correctness check in the DTM could be excluded. Analyses comparing the frontend data with the database values via API calls showed that values actually do not match to some extent. The first assumption was that these fluctuations were caused by third-party tests, as other people were using the same test environment. However, as the fluctuations also occur when no tests are carried out by third parties and the correctness graph regularly returns very different values within very short intervals, this assumption was rejected. We concluded that these changes were due to poor software design of the sock shop. As the application is purely a demo and test environment, it does not need to be a highly reliable system environment where data correctness must match on a time-critical basis.

Inconsistencies can be found in the trust score of the third DoS test. For example, as shown in Figure 15 - Figure 19 between 20:16 and 20:37, higher traffic was generated by three web sockets and web workers respectively, whereby the trust score first rose short-term and only decreased again from 20:21 onwards and finally did not reach a lower value than before this execution until 20:42. The individual parameter evaluations will provide information on whether the individual parameters were influenced by the higher or lower input parameters of web sockets and web workers. Additionally, a comparison is done on the correctness grade with the trust score.

As can be seen in Figure 27 the changes on the graphs are similar. The trust score decreases respectively increases at the same time as the correctness grade. The same time ranges were used as for the figures in Section 5.2, Figure 17 - Figure 19, and in Figure 28 with a wider time range as an overview of the overall impact of the correctness grade on the trust score. We can see in Figure 27 that the correctness grade is constant from 20:21 on and the impact on the trust score arises from the DoS test. Nevertheless, the deviations on the trust score before 20:21 are attributable to the fluctuations in the correctness grade.

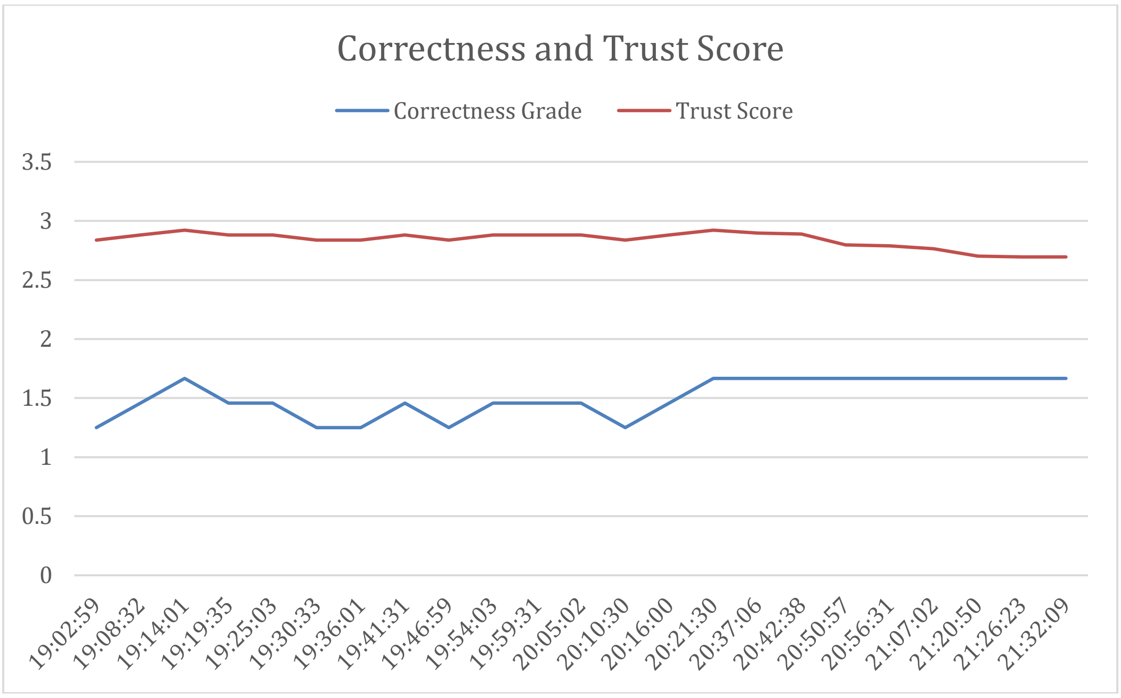


Figure 27: Comparison of Correctness Parameter Grade and Trust Score throughout the third DoS Test, Short Range

Figure 28 depicts the trust score and the correctness grade in a wider time range. The influence of the correctness grade is well visible. With an increase or decrease in the correctness grade, the same behaviour is seen in the trust score, albeit with a smaller slope.

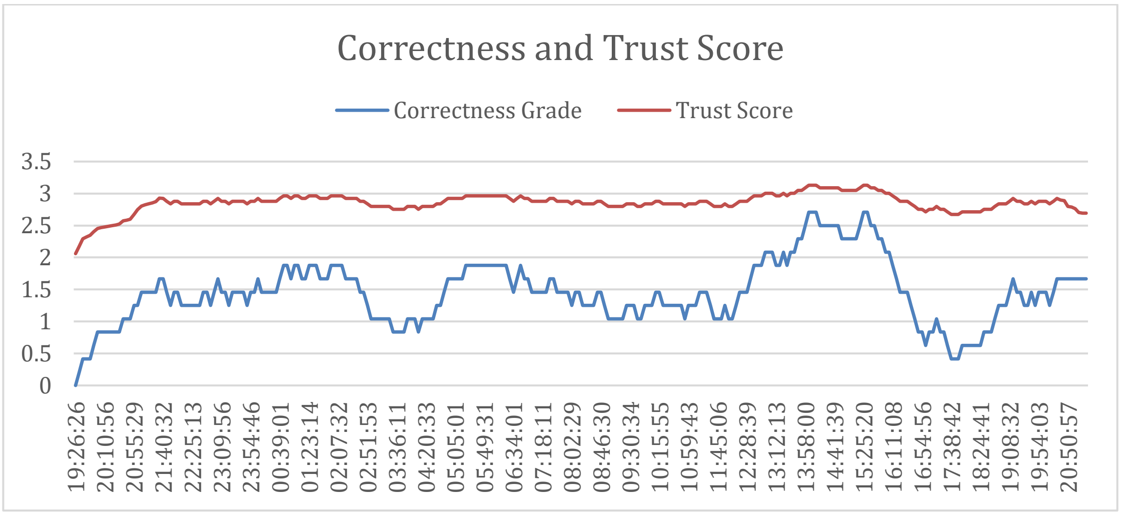


Figure 28: Comparison of Correctness Parameter Grade and Trust Score throughout the third DoS Test, Wide Range

Accordingly, for an unbiased interpretation of the results, the decision was made to remove the correctness parameter from the formula for calculating the trust score and to reinterpret the values. This adjustment is depicted in Figure 29. Since the correctness values were lower than the trust score values at all times, it can be seen that the trust score graph without correctness values is higher than the trust score graph with the correctness values. It is also evident that the fluctuations in the higher graph are no longer present, as the noise of the correctness graph is no longer present. The influence of the DoS test from 20:21 onwards is clearer due to the removal of this very noise. However, the overall influence of the DoS test was smaller than expected. If in a production scenario, the DTM were to decrease the trust score by only about 0.3 in total in the case of unresponsiveness of a system, this would not lead to a change from trustworthy to untrustworthy and accordingly for example would not lead to an adjustment of the channel to the provider. Thus, the user would not be able to rely on the DTM in this respect.

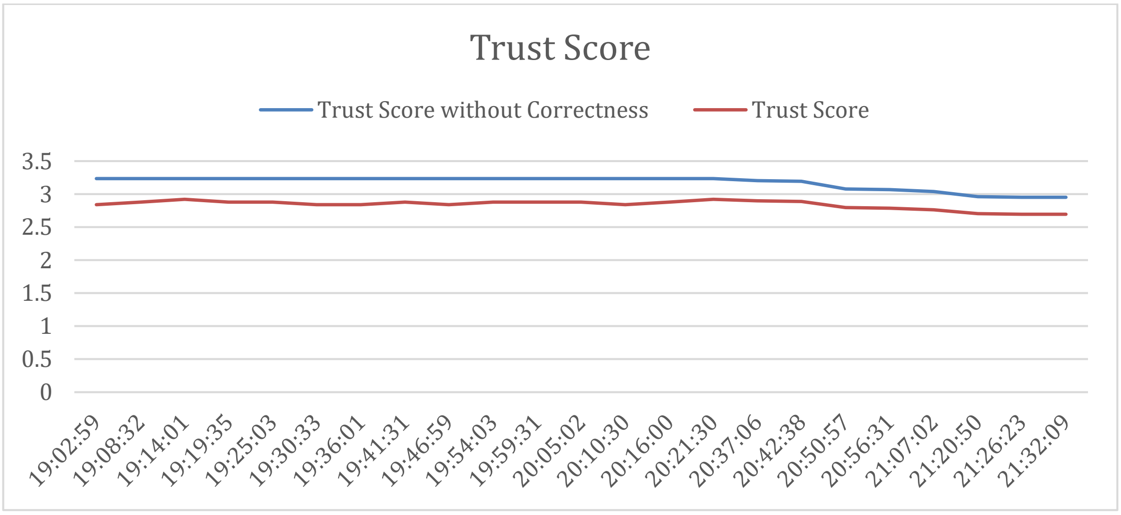


Figure 29: Trust Score Comparison without Correctness

Another approach to eliminating the noise in the correctness graph is to adjust the correctness values. Since the parameter itself is a useful and reasonable part of the trust score [14], [17], [18], it should not be completely eliminated. However, since the correctness implementation does not lead to reliable values in our test system, all correctness values were set to 5 in all collected data points. Figure 30 shows this approach. The trust score with the adjusted correctness values shows higher values than the trust score without correctness values in comparison to Figure 30. This is because all correctness values have now been set to the highest possible values, i.e., 5. Accordingly, the values were increased compared to the original trust score. The noise and thus the fluctuations are no longer present, and the graph appears smoother overall. The influence of the DoS test can be seen, similar as in Figure 30, at 20:21 with 3.58 to 3.36 at 21:32. The influence of the DoS test did not have the desired effect of adjusting the trust score from a trustworthy to a non-trustworthy score.

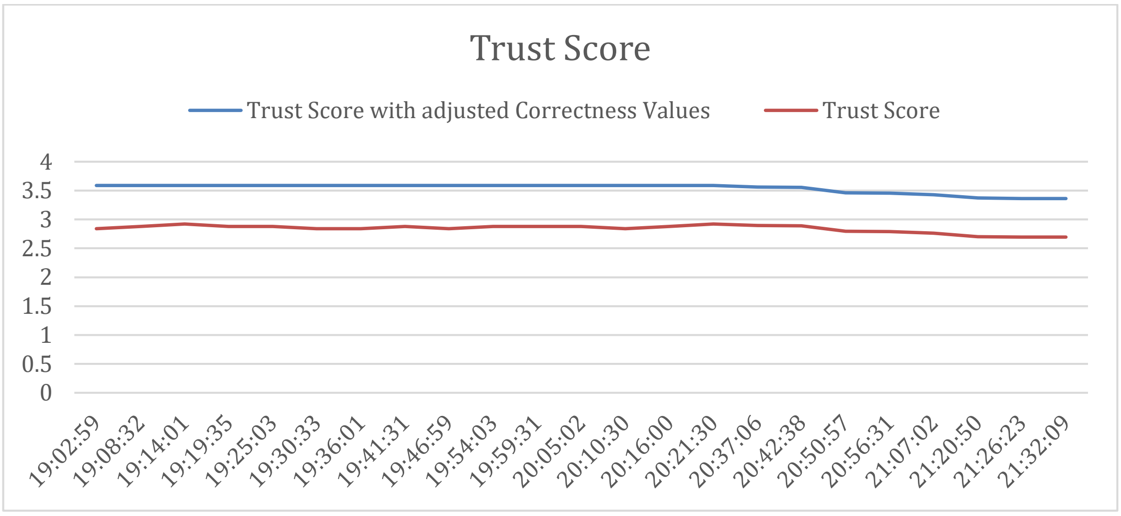


Figure 30: Trust Score Comparison with adjusted Correctness Values

The noise caused by the correctness parameter on the trust score can be observed in all experiments but is omitted for further discussion, as this has already been listed and interpreted using the DoS example. Overall, it can be assumed that the correctness parameter led to noise in all trust scores. Nevertheless, if the correctness graph is omitted or adjusted, no far-reaching change in the influence of the trust score value can be recognised. The impact is evident in the example of the DoS test but does not lead to any major changes.

6.3 Interpretation of the Security Parameter Tests

In this chapter, we will discuss the experiments which had a direct impact on the security parameter, i.e., the invalid certificate and vulnerability tests.

Invalid Certificate Test: The influence of the invalid certificate test on the trust score was compared to the DoS test large. The trust score changed from 2.43 to 3.63 between 13:42 and 13:48. This is depicted in Figure 21. This results in a difference of 1.2. In a productive scenario, this would mean that with a trust score of 2.43, assuming the same initial value as in the test environment of 3.63, the score is still in the positive and thus trustworthy range. Should the certificate have been compromised, this would lead to dangerous situations for the users in the virtualised environment. In the worst case, this could mean that data can be intercepted by attackers.

Since these tests were carried out before the vulnerability checks were implemented in the security parameter, it can be assumed that the security grade would achieve a different result when experimenting with additional sub-parameters. However, if the weighting of the vulnerability checks is considered, it can be assumed that the vulnerability check sub-parameter grades would also result in negative trust scores due to the invalid certificate. For example, both the SSL Lab Scan and Mozilla's HTTP Observatory tools, which perform checks for the implementation of secure websites, have worse values when verifying the implementation of a system with an invalid certificate. Invalid in the sense of not expired but installed on an externally accessible web server with for instance a self-signed certificate. badssl.com [139] provides various web pages with invalid certificates. We decided to scan the website with an invalid root certificate using HTTP Observatory, SSL Lab Scan, and Nikto and then map the values to our vulnerability check grades to plot the new security and trust score graph with these values. Mozilla HTTP Observatory returned a score of 'F', which corresponds to the vulnerability check grade of -5. SSL Lab Scan concluded with a score of 'T', which also corresponds to a grade of -5. Nikto returned a total of 3 findings, which corresponds to a value of 3.5. The new results are depicted in Figure 31. The figure shows the security grade and trust score graphs of the invalid certificate tests with and without the vulnerability tests. The vulnerability grades returned the scores for the invalid root webpage. i. e. from 13:19:16 to 13:42:16 and then achieved trustworthy values from 13:48:04 onwards. For the time range containing valid values, the same vulnerability check grades were used as for the check of the zhaw.ch website.

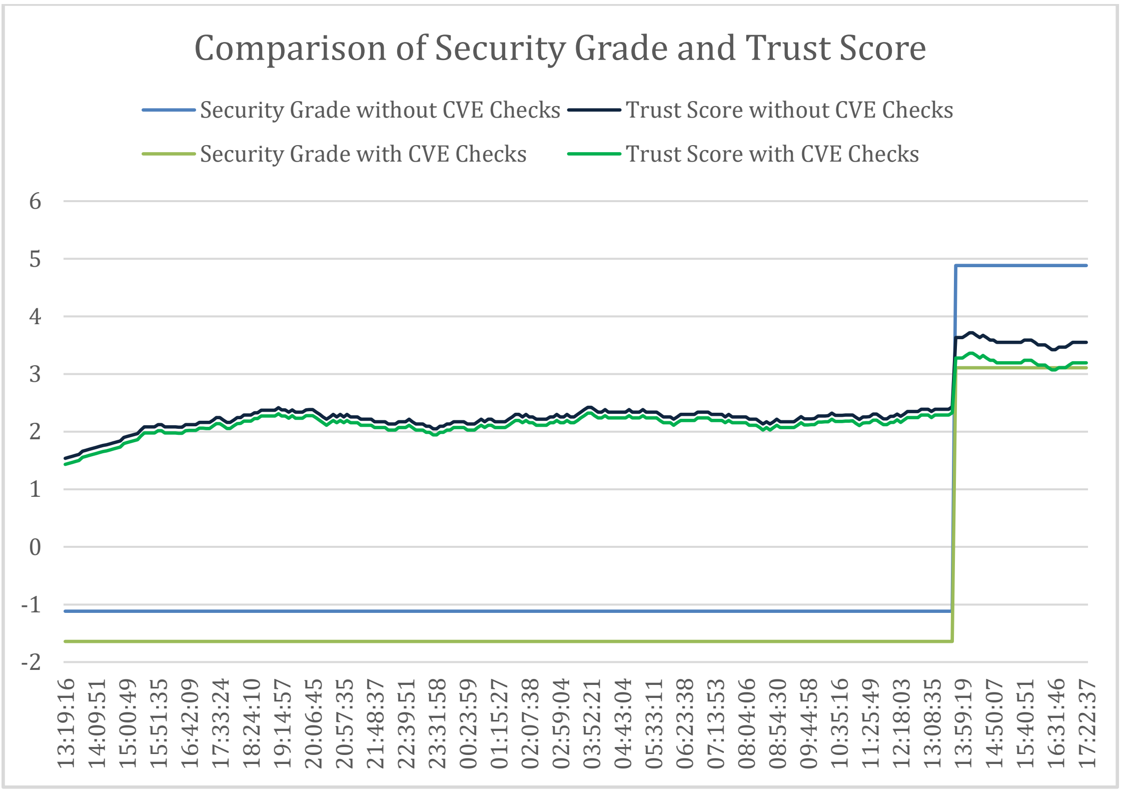


Figure 31: Comparison of the Security Grade and Trust Score Graphs with and without Vulnerability Checks

Figure 31 shows that despite the negative values of the vulnerability grades, the overall trust score does not reach negative values. Although the trust score shows overall lower results than without the vulnerability sub-parameter, this is a comparatively minimal difference compared to without the vulnerability grades.

Vulnerability Test: In the Vulnerability test we could see that a change in the grade of one, as well as all the vulnerability checks, directly influences the trust score. Nevertheless, the impact of the fluctuating correctness parameter is still visible. This is again summarized in Figure 32.

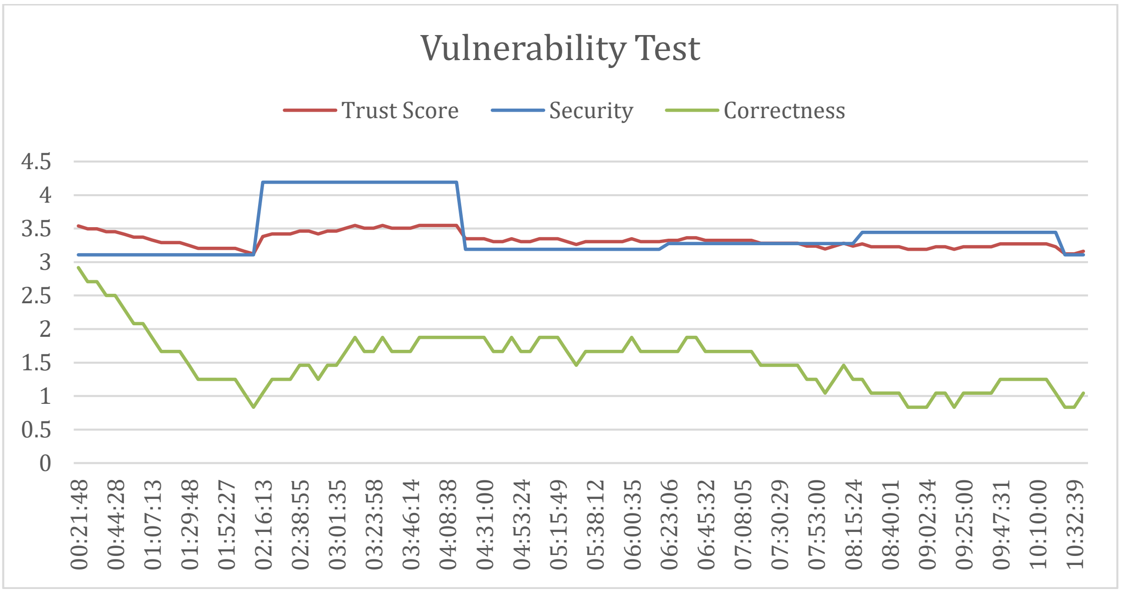


Figure 32: Trust Score, Security, and Correctness Graphs throughout the Vulnerability Test

There is a direct correlation between the trust score and the security graph. With significant changes in the security grade, the trust score altered as well. If the changes were slightly minor, they were sometimes countered by the fluctuations in the correctness grade. The different sub-parameter grades of the vulnerability check grade can be seen in Figure 26. The original values as returned by the tools are shown in Table 36.

Table 36: Sub-Grades of the Vulnerability Check Grades

Domain	**SSL Lab Scan Grade**	**Nikto Check Number**	**Mozilla's HTTP Observatory Grade**
zhaw.ch	A	2	F
mozilla.org	B	1	B
google.com	B	9	C
wikipedia.org	A	8	D
moodle.zhaw.ch	B	2	D

These results show that the three different tools use different approaches for their calculations. Nikto returns the misconfigurations and vulnerabilities found as a number (e.g., 10) and SSL Lab Scan and Mozilla's HTTP Observatory return a grade (e.g., between A-F). SSL Lab Scan and Mozilla's HTTP Observatory focus on analysing the configuration of a Secure Sockets Layer (SSL)/TLS enabled web server as they analyse the utilized available methods to secure a website (e.g., enabled TLS versions or Cipher Suites). Nikto's and SSL Lab Scan and Mozilla's HTTP Observatory grades do not correlate with each other as they use different approaches.

6.4 Attack Detection

With the two experiments Penetration Test and Brute Force Attack, we wanted to test if our DTM could detect attacks on the supervised system. As we never deliberately put measures into place to detect attacks, we expected the system to not be able to detect the two attacks and our test results showed this. Nevertheless, if an attacker is successful in infiltrating the system and starts using the system for other purposes our system will detect the changes in CPU Usage and Disk access. Therefore, a successful attack could be detected by our system.

6.5 Interpretation of the Trust Score

The DTM is currently a Proof of Concept (PoC). This means that it has been feasible to demonstrate the possibility of developing an executable DTM which performs calculations, mappings, and decisions about trusted and untrusted systems based on our proposed trust metrics and thresholds. It furthermore implies, that the DTM should not be used in a productive environment at this stage, as some improvements should be carried out in advance. Nevertheless, the system can be used as a basis. In this section, we address the constraints and issues of the prototype DTM. For this purpose, we divide the sub-section into expected limitations and unexpected issues. The expected limitations are those that we expected to encounter when testing the DTM. The unexpected issues include all difficulties and limitations which we did not consider in advance, and which emerged during testing.

6.5.1 Expected Limitations

The trust score has some limitations which we wanted to address but which were omitted because of time constraints. The first one concerns the data to be only collected for the last 24 hours, which means that an incident is forgotten by our DTM 24 hours after it occurred.

The second limitation addresses the warm-up period. This period takes our DTM as long as the historical data it keeps. A warm-up period is acceptable for a productive system, but for it to be as long as the data should be retained is not reasonable. As an example, if the DTM should calculate the trust score of the data from the last 30 days, the warm-up period would take 30 days as well. This is because, for some parameters, the variables consist of vectors with 30*24 entries. Each individual record of the vector contains data for an hour of one of these 30 days. Since the initialisation of the variable fills the vector with 0 values, these values must all be overwritten first, which would take 30 days.

The third limitation concerns persistency as the collected data is not retained when restarting the system. Meaning that every time someone restarts the system all the historical data is lost, and the warm-up period must be re-initialised. A well-built production DTM should be able to retrieve the data, avoiding the need for a warm-up period. This implementation was not realised in the DTM prototype as we want to consider the consequences of the different tests for the initialisation phase as well.

The fourth limitation addresses the problem of the differentiation between a manual or expected and unexpected downtime. If the system-under-test, i. e. a system, which the DTM monitors, is down for maintenance work, the impact on the trust scores would be the same as in the case of an unexpected system failure. Ideally, these two cases should be possible to distinguish by the DTM. As an example, this could be implemented using a maintenance mode, similar to monitoring systems. In a maintenance window scenario, where the administrator patches the systems, the DTM should subsequently generate better values, since security-relevant patches may have been installed and thus some parameters would achieve higher values.

The fifth and last limitation concerns the DTM running as a single process application. This already led to problems during the testing period where the daily update took longer than expected and blocked the hourly update. The daily and hourly updates should be developed as two individual processes for a production DTM. An attempt has already been made to implement multiprocessing. Since the multiprocessor prototype is still experiencing difficulties, it is located in a separate branch on GitHub.

6.5.2 Unexpected Issues

During the testing period of our DTM, we found some limitations that we were not aware of before. The most important one is that an untrusted system can regain trust much faster than we intended. If we do small changes, for example replacing an untrusted one with a trusted certificate, the overall security grade increases immediately. This is not the desired effect, as negative incidents in the DTM should be “remembered” for a longer period of time. As an example, if a system was unresponsive for a week, it should not regain full trust two days after the incident.

Moreover, negative grades are forgotten after 24 hours, which helps to restore trust. In other experiments, we have seen that it is difficult to lose trust after a single incident. For example, contrary to expectations, the trust score hardly reacted to the DoS attack. Therefore, the underlying trust calculation still needs to be improved. We tried this during our testing period by giving a higher weight to containers with lower results compared to the last poll. This change still did not provide the expected results. A way to address this issue would be to split the trust calculation. For example, one could consider the 5 main parameters as individual trust score parameters. If one of these parameters would score a negative value, this would affect all other parameters. This means that individual sub-parameters would have a greater direct influence on the overall trust.

7 Conclusion and Future Research

Within this thesis, it was demonstrated that the monitoring of microservices in virtualised infrastructures is not a straightforward task. The literature suggests that software-based systems are difficult to monitor concerning security aspects [103], [104]. The increased complexity results in a difficult decision on how to monitor NFV infrastructures. Nevertheless, it could further be revealed that security is not the only aspect to be taken into account. The use of different trust parameters was considered based on [14]–[18] and implemented accordingly in our DTM PoC. We believe that these proposed trust parameters, which were suggested in [14] and further in [15]–[18], are appropriate for a DTM. In addition, the trust formula based on [21] used and adapted according to our implementation, is also a suitable approach for calculating the trust score. The tests and interpretation of these led us to the conclusion that a prototype DTM is feasible based on the current state of trustworthiness monitoring. The main difficulty is processing all proposed parameters from [14]–[18] in a timely manner.

Difficulties were also found concerning data collection and data retention. The problem of whether and when a system regains its full level of trustworthiness after an incident remains unanswered in the current state of this work. As discussed in Chapter 6, we were able to find the cause of the issues and limitations in our DTM and are able to conclude that these problems do not stem from the parameters used or from the application of the formula. It would be conceivable that if the parameters were split into individual trust calculations, this would lead to more accurate results. This way, the parameters are not all dependent and would not influence each other. However, this has the consequence that negative trust scores would be obtained more easily and thus a single parameter would have an influence on the overall trust score. This would have to be tested in another PoC.

Furthermore, the implementation would require a comparison with ours. To the best of our knowledge, we are not aware of a similar DTM having been implemented before. Thus, no work could be used as a basis, and accordingly, no references are available to compare our current results as well as the proposed hypothetical ones with others. Nonetheless, we believe that our DTM implementation as well as the lessons learned from the PoC could provide important results and motivate further research in the direction of trustworthiness in virtualised network environments. Even if the DTM is not yet suitable for use in production environments, it is known which constraints need to be improved.

Future research includes the examination of whether the division into processes for the calculation of the individual parameter groups, would lead to better results. It should also be mentioned that the fast recovery time of our DTMs is also not advantageous in production scenarios. For example, if a system is unresponsive for more than a week, it is not reasonable that the trust score would gain full trustworthiness on the second day after the system is accessible again. Thus, a broader historical approach to the DTM would have to be considered. If the limitations mentioned in Chapter 6 are adjusted, we could expect to achieve good and reliable results as trust metrics. Such a DTM can furthermore be run without any additional external input, unlike the proposals from [25], [28]–[30]. Known limitations exist, but additional approaches are suggested that lead us in the right direction on how to achieve suitable results.

Another important research direction is a more methodological selection of threshold values and their optimization for more technically sound trust scoring. At the moment, these values are determined based on related work and a heuristic approach. Similarly, the weights of different parameters could be optimized for a more realistic and accurate trust score calculation in our DTM. Those weights may depend on the specific characteristics of the monitored service chain or could be optimized using some pre-generated or collected datasets via Machine Learning approaches.

8 References

[1] V. R. Tadinada, "Software Defined Networking: Redefining the Future of Internet in IoT and Cloud Era", in *2014 International Conference on Future Internet of Things and Cloud*, Aug. 2014, pp. 296–301. doi: 10.1109/FiCloud.2014.53.

[2] M. D. Ananth and R. Sharma, "Cloud Management Using Network Function Virtualization to Reduce CAPEX and OPEX", in *2016 8th International Conference on Computational Intelligence and Communication Networks (CICN)*, Dec. 2016, pp. 43–47. doi: 10.1109/CICN.2016.17.

[3] K. Rafique, A. W. Tareen, M. Saeed, J. Wu, and S. S. Qureshi, "Cloud computing economics opportunities and challenges", in *2011 4th IEEE International Conference on Broadband Network and Multimedia Technology*, Oct. 2011, pp. 401–406. doi: 10.1109/ICBNMT.2011.6155965.

[4] C. Suraci, G. Araniti, A. Abrardo, G. Bianchi, and A. Iera, "A stakeholder-oriented security analysis in virtualized 5G cellular networks", *Comput. Netw.*, vol. 184, p. 107604, Jan. 2021, doi: 10.1016/j.comnet.2020.107604.

[5] M. Miyazawa, M. Hayashi, and R. Stadler, "vNMF: Distributed fault detection using clustering approach for network function virtualization", in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, May 2015, pp. 640–645. doi: 10.1109/INM.2015.7140349.

[6] S. Vrijders *et al.*, "Reducing the complexity of virtual machine networking", *IEEE Commun. Mag.*, vol. 54, no. 4, pp. 152–158, Apr. 2016, doi: 10.1109/MCOM.2016.7452280.

[7] T. Komperda, "Virtualization security", *Infosec Resources*, Dec. 17, 2012. https://resources.infosecinstitute.com/topic/virtualization-security-2/ (accessed May 31, 2022).

[8] M. Alenezi and M. Zarour, "On the Relationship between Software Complexity and Security", *Int. J. Softw. Eng. Appl.*, vol. 11, no. 1, pp. 51–60, Jan. 2020, doi: 10.5121/ijsea.2020.11104.

[9] Y. Javed, M. Alenezi, M. Akour, and A. Alzyoud, "Discovering the relationship between software complexity and software vulnerabilities", *J. Theor. Appl. Inf. Technol.*, vol. 96, pp. 4690–4699, Jul. 2018.

[10] I. Chowdhury and M. Zulkernine, "Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities", *J. Syst. Archit.*, vol. 57, no. 3, pp. 294–313, Mar. 2011, doi: 10.1016/j.sysarc.2010.06.003.

[11]"10 5G Healthcare use cases", *STL Partners*. https://stlpartners.com/articles/digital-health/10-5g-healthcare-use-cases/ (accessed May 31, 2022).

[12]"5G Healthcare How will 5g affect healthcare?", May 31, 2017. https://www.ericsson.com/en/reports-and-papers/5g-healthcare (accessed May 31, 2022).

[13] A. Simmons, "5G Use Cases in 10 Different Industries", *Dgtl Infra*, Dec. 07, 2020. https://dgtlinfra.com/5g-use-cases-in-10-different-industries/ (accessed May 31, 2022).

[14] S. Becker *et al.*, "Trustworthy software systems: a discussion of basic concepts and terminology", *ACM SIGSOFT Softw. Eng. Notes*, vol. 31, no. 6, pp. 1–18, Nov. 2006, doi: 10.1145/1218776.1218781.

[15] S. M. Habib, S. Ries, and M. Muhlhauser, "Cloud Computing Landscape and Research Challenges Regarding Trust and Reputation", in *2010 7th International Conference on Ubiquitous Intelligence & Computing and 7th International Conference on Autonomic & Trusted Computing*, Xi'an, Shaanxi, China, Oct. 2010, pp. 410–415. doi: 10.1109/UIC-ATC.2010.48.

[16]H. Hassan, A. I. El-Desouky, A. Ibrahim, E.-S. M. El-Kenawy, and R. Arnous, "Enhanced QoS-Based Model for Trust Assessment in Cloud Computing Environment", *IEEE Access*, vol. 8, pp. 43752–43763, 2020, doi: 10.1109/ACCESS.2020.2978452.

[17]A. J. John Joseph and M. Mariappan, "A novel trust-scoring system using trustability co-efficient of variation for identification of secure agent platforms", *PloS One*, vol. 13, no. 8, pp. e0201600–e0201600, Aug. 2018, doi: 10.1371/journal.pone.0201600.

[18]S. Romdhani, G. Vargas-Solar, N. Bennani, and C. Ghedira-Guegan, "QoS-based Trust Evaluation for Data Services as a Black Box", *ArXiv211012895 Cs*, Oct. 2021, Accessed: Apr. 10, 2022. [Online]. Available: http://arxiv.org/abs/2110.12895

[19]F. Azzedin and M. Maheswaran, "Towards Trust-Aware Resource Management in Grid Computing Systems", in *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02)*, Berlin, Germany, 2002, pp. 452–452. doi: 10.1109/CCGRID.2002.1017189.

[20]M. K. Goyal, A. Aggarwal, P. Gupta, and P. Kumar, "QoS based trust management model for Cloud IaaS", in *2012 2nd IEEE International Conference on Parallel, Distributed and Grid Computing*, Solan, India, Dec. 2012, pp. 843–847. doi: 10.1109/PDGC.2012.6449933.

[21]T. Sun and M. K. Denko, "A Distributed Trust Management Scheme in the Pervasive Computing Environment", in *2007 Canadian Conference on Electrical and Computer Engineering*, Vancouver, BC, Canada, 2007, pp. 1219–1222. doi: 10.1109/CCECE.2007.311.

[22]P. Govindaraj, N. Jaisankar, and School of Computer Science and Engineering, VIT University, Vellore, India, "A Review on Various Trust Models in Cloud Environment", *J. Eng. Sci. Technol. Rev.*, vol. 10, no. 2, pp. 213–219, Jun. 2017, doi: 10.25103/jestr.102.24.

[23]J. Li, B. Mao, Z. Liang, Z. Zhang, Q. Lin, and X. Yao, "Trust and Trustworthiness: What They Are and How to Achieve Them", in *2021 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, Kassel, Germany, Mar. 2021, pp. 711–717. doi: 10.1109/PerComWorkshops51409.2021.9430929.

[24]J. Huang and D. M. Nicol, "Trust mechanisms for cloud computing", *J. Cloud Comput. Adv. Syst. Appl.*, vol. 2, no. 1, p. 9, 2013, doi: 10.1186/2192-113X-2-9.

[25]H. Zhu, B. Feng, and R. H. Deng, "Computing of Trust in Distributed Networks", 056, 2003. Accessed: May 21, 2022. [Online]. Available: http://eprint.iacr.org/2003/056

[26]R. Shaikh and M. Sasikumar, "Trust Model for Measuring Security Strength of Cloud Computing Service", *Procedia Comput. Sci.*, vol. 45, pp. 380–389, 2015, doi: 10.1016/j.procs.2015.03.165.

[27]M. Alhamad, T. Dillon, and E. Chang, "SLA-Based Trust Model for Cloud Computing", in *2010 13th International Conference on Network-Based Information Systems*, Takayama, Gifu, Japan, Sep. 2010, pp. 321–324. doi: 10.1109/NBiS.2010.67.

[28]F. Corradini, F. De Angelis, F. Ippoliti, and F. Marcantoni, *A Survey of Trust Management Models for Cloud Computing*. May 2015. [Online]. Available: https://www.researchgate.net/publication/279497649_A_Survey_of_Trust_Management_Models_for_Cloud_Computing

[29]S. Singh and D. Chand, "Trust evaluation in cloud based on friends and third party's recommendations", *2014 Recent Adv. Eng. Comput. Sci. RAECS*, 2014, doi: 10.1109/RAECS.2014.6799600.

[30]D. Kong and Y. Zhai, "Trust Based Recommendation System in Service-oriented Cloud Computing", in *Proceedings of the 2012 International Conference on Cloud and Service Computing*, USA, Nov. 2012, pp. 176–179. doi: 10.1109/CSC.2012.34.

[31]J. H. Abawajy and A. M. Goscinski, "A Reputation-Based Grid Information Service", in *Computational Science – ICCS 2006*, vol. 3994, V. N. Alexandrov, G. D. van Albada, P. M. A.

Sloot, and J. Dongarra, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 1015–1022. doi: 10.1007/11758549_135.
[32]J. Borowski, K. Hopkinson, J. Humphries, and B. Borghetti, "Reputation-Based Trust for a Cooperative Agent-Based Backup Protection Scheme", *IEEE Trans. Smart Grid*, 2011, doi: 10.1109/TSG.2011.2118240.
[33]B. W. Boehm, "Verifying and validating software requirements and design specifications", *IEEE Softw.*, pp. 75–88, 1984.
[34]W. Lowrance, *Of acceptable risk: science and the determination of safety.* William Kaufman Inc., 1976.
[35]B. Wolford, 'What is GDPR, the EU's new data protection law?', *GDPR.eu*, Nov. 07, 2018. https://gdpr.eu/what-is-gdpr/ (accessed Jun. 04, 2022).
[36]R. J. Anderson, *Security Engineering : A Guide to Building Dependable Distributed Systems*. John Wiley & Sons, 2001.
[37] "What is Software-Defined Networking (SDN)? | VMware Glossary", *VMware*. https://www.vmware.com/topics/glossary/content/software-defined-networking.html (accessed Jun. 06, 2022).
[38] "Software-Defined Networking (SDN) Definition", *Cisco*. https://www.cisco.com/c/en/us/solutions/software-defined-networking/overview.html (accessed Apr. 03, 2022).
[39] "What is network virtualization? ", *redhat.com*, Mar. 23, 2021. https://www.redhat.com/en/topics/virtualization/what-is-network-virtualization (accessed Apr. 27, 2022).
[40]U. Mulligan, "ETSI - Standards for NFV - Network Functions Virtualisation | NFV Solutions", *ETSI*. https://www.etsi.org/technologies/nfv (accessed May 24, 2022).
[41] "What is Network Functions Virtualization (NFV)? & ETSI | ThousandEyes", 2022. https://www.thousandeyes.com/learning/glossary/nfv-network-functions-virtualization (accessed May 24, 2022).
[42] "What are Virtual Network Functions (VNF)? Management & Orchestration", *thousandeyes.com*, 2022. https://www.thousandeyes.com/learning/glossary/vnf-virtual-network-functions (accessed Apr. 27, 2022).
[43]M. Jacobs and E. Kaim, "What are Microservices? - Azure DevOps", *microsoft.com*, Jun. 14, 2021. https://docs.microsoft.com/en-us/devops/deliver/what-are-microservices (accessed Apr. 27, 2022).
[44]J. Lewis and M. Fowler, 'Microservices', *martinfowler.com*, Mar. 25, 2014. https://martinfowler.com/articles/microservices.html (accessed Mar. 30, 2022).
[45]IBM Cloud Education, 'microservices', *ibm.com*, Mar. 30, 2021. https://www.ibm.com/de-de/cloud/learn/microservices (accessed Apr. 27, 2022).
[46]M. Bruce and P. Pereira, *Microservices in Action*. Manning Publications, 2018. Accessed: Apr. 24, 2022. [Online]. Available: https://livebook.manning.com/book/microservices-in-action/chapter-6/52
[47]D.-H. Luong, H.-T. Thieu, A. Outtagarts, and Y. Ghamri-Doudane, "Cloudification and Autoscaling Orchestration for Container-Based Mobile Networks toward 5G: Experimentation, Challenges and Perspectives", in *2018 IEEE 87th Vehicular Technology Conference (VTC Spring)*, Jun. 2018, pp. 1–7. doi: 10.1109/VTCSpring.2018.8417602.
[48]G. Liu, N. Li, J. Deng, Y. Wang, J. Sun, and Y. Huang, "The SOLIDS 6G Mobile Network Architecture: Driving Forces, Features, and Functional Topology", *Engineering*, vol. 8, pp. 42–59, Jan. 2022, doi: 10.1016/j.eng.2021.07.013.
[49]Bernardos, Carlos J. and Uusitalo, Mikko A., "European Vision for the 6G Network Ecosystem", Zenodo, Jun. 2021. doi: 10.5281/ZENODO.5007671.

[50] ctrfantennas, "What Is A 5G Core Network? | C&T RF Antennas Inc | Antenna Manufacturer", Nov. 23, 2021. https://lcantennas.com/what-is-a-5g-core-network/ (accessed May 21, 2022).
[51] P. Manuel, M. Barr, and T. S. Somasundaram, "A Novel Trust Management System for Cloud Computing - IaaS Providers", *J. Comb. Math. Comb. Comput.*, vol. 79, pp. 3–22, Nov. 2011.
[52]M. Blaze, J. Feigenbaum, and J. Lacy, "Decentralized trust management", in *Proceedings 1996 IEEE Symposium on Security and Privacy*, May 1996, pp. 164–173. doi: 10.1109/SECPRI.1996.502679.
[53]Y. Alghofaili and M. A. Rassam, "A Trust Management Model for IoT Devices and Services Based on the Multi-Criteria Decision-Making Approach and Deep Long Short-Term Memory Technique", *Sensors*, vol. 22, no. 2, p. 634, Jan. 2022, doi: 10.3390/s22020634.
[54] "Cornell University, Proofs of Program Correctness", *Cornell University*, 2008. https://www.cs.cornell.edu/courses/cs312/2004fa/lectures/lecture9.htm (accessed May 15, 2022).
[55]M. Jeefri A, "Data As Commodity: For Data Science Professional - DataScienceCentral.com", *Data Science Central*, Jul. 03, 2020. https://www.datasciencecentral.com/data-as-commodity-for-data-science-professional/ (accessed May 15, 2022).
[56]M. Allinson, "How has Data Become the World's Most Valuable Commodity? ", *Robotics & Automation News*, Jul. 22, 2021. https://roboticsandautomationnews.com/2021/07/22/how-has-data-become-the-worlds-most-valuable-commodity/44267/ (accessed May 15, 2022).
[57]A. Aaltonen, C. Alaimo, and J. Kallinikos, 'The Making of Data Commodities: Data Analytics as an Embedded Process", *J. Manag. Inf. Syst.*, vol. 38, no. 2, pp. 401–429, Apr. 2021, doi: 10.1080/07421222.2021.1912928.
[58]C. Tozzi, "How to Measure Data Quality – 7 Metrics to Assess Your Data", *Precisely*, May 10, 2021. https://www.precisely.com/blog/data-quality/how-to-measure-data-quality-7-metrics (accessed May 15, 2022).
[59]R. L. Sarfin, "Data Quality Dimensions: How Do You Measure Up? (+ Free Scorecard) ", *Precisely*, Apr. 30, 2021. https://www.precisely.com/blog/data-quality/data-quality-dimensions-measure (accessed May 15, 2022).
[60] "Using Key Metrics to Measure Workplace Safety Performance in Health Care", *Manchester Specialty Programs*, Feb. 01, 2019. https://www.manchesterspecialty.com/using-key-metrics-to-measure-workplace-safety-performance-in-health-care/ (accessed May 06, 2022).
[61]N. G. Leveson, "Complexity and Safety", in *Complex Systems Design & Management*, O. Hammami, D. Krob, and J.-L. Voirin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 27–39. doi: 10.1007/978-3-642-25203-7_2.
[62] "The Cost of Downtime", *Andrew Lerner*, Jul. 16, 2014. https://blogs.gartner.com/andrew-lerner/2014/07/16/the-cost-of-downtime/ (accessed Apr. 30, 2022).
[63]D. Stanford, S. Wray, and R. Millsap, "Business Metrics - Microsoft Azure Well-Architected Framework", *docs.microsoft.com*, Dec. 13, 2021. https://docs.microsoft.com/en-us/azure/architecture/framework/resiliency/business-metrics (accessed Apr. 30, 2022).
[64]M. Lewis, 'Telstra guarantees 5G network uptime with Enhanced Enterprise Wireless", *MobileCorp*, Sep. 15, 2021. https://www.mobilecorp.com.au/blog/telstra-guarantees-5g-network-uptime-with-enterprise-enhanced-wireless (accessed May 24, 2022).

[65]Ohad Shushan, "AWS vs. Azure vs. GCP | Detailed Comparison", May 08, 2020. https://www.cloudride.co.il/blog/aws-vs.-azure-vs.-gcp-detailed-comparison (accessed May 25, 2022).
[66]P. Ranaweera, A. Jurcut, and M. Liyanage, "MEC-enabled 5G Use Cases: A Survey on Security Vulnerabilities and Countermeasures", *ACM Comput. Surv.*, vol. 54, no. 9, pp. 1–37, Dec. 2022, doi: 10.1145/3474552.
[67]H. Leligou, T. Zahariadis, L. Sarakis, E. Tsampasis, A. Voulkidis, and T. Velivasaki, *Smart Grid: a demanding use case for 5G technologies*. 2018, p. 220. doi: 10.1109/PERCOMW.2018.8480296.
[68]E. Maçon-Dauxerre, "How 5G Enables Advanced Metering Infrastructure for Smarter Utilities", *Telit*, Nov. 07, 2019. https://www.telit.com/blog/how-5g-enables-advanced-metering-infrastructure-smarter-utilities/ (accessed Jun. 05, 2022).
[69]C. M. Lonvick, 'The BSD Syslog Protocol', Internet Engineering Task Force, Request for Comments RFC 3164, Aug. 2001. doi: 10.17487/RFC3164.
[70]R. Kuć, "Logging Levels: What They Are & How to Choose Them", *Sematext*, Oct. 08, 2020. https://sematext.com/blog/logging-levels/ (accessed May 01, 2022).
[71] "Ineffective Patch Management Risks", *eci.com*, Nov. 29, 2018. https://www.eci.com/blog/16080-the-risks-of-ineffective-patch-management.html (accessed Jun. 05, 2022).
[72]D. Weber, 'Why it's So Hard to Implement IoT Security | SecurityWeek.Com", *securityweek.com*, Jan. 15, 2019. https://www.securityweek.com/why-its-so-hard-implement-iot-security (accessed May 21, 2022).
[73] "Assessing Security Vulnerabilities and Applying Patches | Cyber.gov.au", *cyber.gov.au*, Oct. 2011. https://www.cyber.gov.au/acsc/view-all-content/publications/assessing-security-vulnerabilities-and-applying-patches (accessed Jun. 05, 2022).
[74]A. Haider, R. Potter, and A. Nakao, "Challenges in Resource Allocation in Network Virtualization", . *May*, p. 9, 2009.
[75]H. Shukur, S. Zeebaree, R. Zebari, D. Zeebaree, O. Ahmed, and A. Salih, "Cloud Computing Virtualization of Resources Allocation for Distributed Systems", *J. Appl. Sci. Technol. Trends*, vol. 1, no. 3, pp. 98–105, Jun. 2020, doi: 10.38094/jastt1331.
[76]H. Halabian, "Distributed Resource Allocation Optimization in 5G Virtualized Networks", *IEEE J. Sel. Areas Commun.*, vol. 37, no. 3, pp. 627–642, Mar. 2019, doi: 10.1109/JSAC.2019.2894305.
[77]J. Ellingwood, "An Introduction to Metrics, Monitoring, and Alerting | DigitalOcean", *digitalocean.com*, Dec. 05, 2017. https://www.digitalocean.com/community/tutorials/an-introduction-to-metrics-monitoring-and-alerting (accessed Apr. 30, 2022).
[78] "Configuring a CPU threshold monitor - Progress Community", *community.progress.com*, Apr. 16, 2020. https://community.progress.com/s/article/Configuring-a-CPU-threshold-monitor (accessed Apr. 30, 2022).
[79] "monitor cpu-usage threshold", *hpe.com*, 2017. https://techhub.hpe.com/eginfolib/networking/docs/switches/5950/5200-4005_fund_cr/content/499752694.htm (accessed Apr. 30, 2022).
[80] "Configuring the Thresholds of CPU Usage and Memory Usage - NE05E and NE08E V300R003C10SPC500 Configuration Guide - System Management 01 - Huawei", *support.huawei.com*. https://support.huawei.com/enterprise/en/doc/EDOC1100058923/bfc52bc9/configuring-the-thresholds-of-cpu-usage-and-memory-usage#dc_vrp_dev_cfg_0004 (accessed May 08, 2022).

[81]A. Cooke, "Monitoring thresholds determine IT performance alerts", *TechTarget*, Apr. 16, 2018. https://www.techtarget.com/searchitoperations/tip/Monitoring-thresholds-determine-IT-performance-alerts (accessed Apr. 30, 2022).
[82]"Memory Usage monitor", *solarwinds.com*. https://documentation.solarwinds.com/en/success_center/ipmonitor/content/memory_usage.htm (accessed Jun. 05, 2022).
[83]"Monitoring memory", *ibm.com*, Mar. 16, 2021. https://prod.ibmdocs-production-dal-6099123ce774e592a519d7c33db8265e-0000.us-south.containers.appdomain.cloud/docs/en/radfws/9.6.1?topic=memory-monitoring (accessed Jun. 05, 2022).
[84]ManageEngine, "Network Monitoring Software by ManageEngine OpManager", *ManageEngine OpManager*, 2022. https://www.manageengine.com/network-monitoring/ (accessed May 08, 2022).
[85]Forecast, "What is Utilization Rate? " https://www.forecast.app/faqs/what-is-utilization-rate (accessed May 08, 2022).
[86] "Memory usage calculation for Linux", *support.site24x7.com*. https://support.site24x7.com/portal/en/kb/articles/how-is-memory-utilization-calculated-for-a-linux-server-monitor (accessed May 08, 2022).
[87] "Utilization rate: what it is, how to calculate it accurately", Apr. 28, 2022. https://timelyapp.com/blog/utilization-rate (accessed May 08, 2022).
[88] "Total CPU usage for Linux monitor". https://support.site24x7.com/portal/en/kb/articles/how-is-cpu-utilization-calculated-for-a-linux-server-monitor (accessed May 08, 2022).
[89]Trend Micro Incorporated, "Alert: The memory warning threshold of Manager Node has been exceeded | Deep Security", *help.deepsecurity.trendmicro.com*, 2022. https://help.deepsecurity.trendmicro.com/aws/memory-warning-threshold.html (accessed May 08, 2022).
[90] "Using Ipswitch WhatsUp Gold v15", *docs.ipswitch.com/*. https://docs.ipswitch.com/NM/79_WhatsUp%20Gold%20v15/03_Help/index.htm?threshold_memory_utilization.htm?toc.htm (accessed May 08, 2022).
[91] "Disk metrics - Azure Virtual Machines", *microsoft.com*, Aug. 23, 2021. https://docs.microsoft.com/en-us/azure/virtual-machines/disks-metrics (accessed Jun. 05, 2022).
[92] "Leistungsmesswerte für nichtflüchtigen Speicher prüfen | Compute Engine-Dokumentation", *Google Cloud*, May 23, 2022. https://cloud.google.com/compute/docs/disks/review-disk-metrics?hl=de (accessed Jun. 05, 2022).
[93] "8 Crucial Database Performance Metrics | Scout APM Blog', *scoutapm.com*, Jan. 28, 2021. https://scoutapm.com/blog/database-performance-metrics (accessed Jun. 05, 2022).
[94]M. Petrovic, "SQL Server disk performance metrics – Part 1 – the most important disk performance metrics", *SQL Shack - articles about database auditing, server performance, data recovery, and more*, Mar. 12, 2014. https://www.sqlshack.com/sql-server-disk-performance-metrics-part-1-important-disk-performance-metrics/ (accessed May 08, 2022).
[95] "Input/Output operations Per Second", *Wikipedia*. Jun. 15, 2021. Accessed: May 08, 2022. [Online]. Available: https://de.wikipedia.org/w/index.php?title=Input/Output_operations_Per_Second&oldid=212976329

[96]M. Grande, "Diagnosing Disk Performance Issues - Concurrency", *www.concurrency.com*, Sep. 05, 2019. https://www.concurrency.com/blog/september-2019/diagnosing-disk-performance-issues (accessed May 08, 2022).
[97]T. Nagy, "How to Identify IO Bottlenecks in MS SQL Server", *www.mssqltips.com*, Mar. 17, 2011. https://www.mssqltips.com/sqlservertip/2329/how-to-identify-io-bottlenecks-in-ms-sql-server/ (accessed May 08, 2022).
[98]S. Godard, "iostat(1) - Linux manual page", *man7.org*. https://man7.org/linux/man-pages/man1/iostat.1.html (accessed May 08, 2022).
[99]B. Vinayaga Sundaram, R. M, P. M, and V. S. J, "Encryption and hash based security in Internet of Things", in *2015 3rd International Conference on Signal Processing, Communication and Networking (ICSCN)*, Mar. 2015, pp. 1–6. doi: 10.1109/ICSCN.2015.7219926.
[100]E. Mossa, "Security enhancement for AES encrypted speech in communications", *Int. J. Speech Technol.*, vol. 20, no. 1, pp. 163–169, Mar. 2017, doi: 10.1007/s10772-017-9395-3.
[101]X. Li, "Application of Data Encryption Technology in Computer Network Communication Security", *J. Phys. Conf. Ser.*, vol. 1574, no. 1, p. 012034, Jun. 2020, doi: 10.1088/1742-6596/1574/1/012034.
[102]J. M. Borky and T. H. Bradley, "Protecting Information with Cybersecurity", *Eff. Model-Based Syst. Eng.*, pp. 345–404, Sep. 2018, doi: 10.1007/978-3-319-95669-5_10.
[103]D. M. Firoozjaei, J. Jeong, H. Ko, and H. Kim, "Security challenges with network functions virtualization":, *Future Gener. Comput. Syst.*, vol. 67, Jul. 2016, doi: 10.1016/j.future.2016.07.002.
[104]W. Yang and C. Fung, "A survey on security in network functions virtualization", in *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, Jun. 2016, pp. 15–19. doi: 10.1109/NETSOFT.2016.7502434.
[105]H. Hawilo, A. Shami, M. Mirahmadi, and R. Asal, "NFV: State of the Art, Challenges and Implementation in Next Generation Mobile Networks (vEPC) ", *IEEE Netw.*, vol. 28, Sep. 2014, doi: 10.1109/MNET.2014.6963800.
[106]D. V. Khera, "Are you aware of the security risks of Network Function Virtualization? ", *Cybersecurity Magazine*, Aug. 01, 2019. https://cybersecurity-magazine.com/are-you-aware-of-the-security-risks-of-network-function-virtualization/ (accessed May 25, 2022).
[107] "Network Functions Virtualization (NFV) & Virtualized Network Functions (VNF)", *A10 Networks*, 2022. https://www.a10networks.com/glossary/what-is-nfv-network-functions-virtualization-vnf-virtualized-network-functions/ (accessed May 25, 2022).
[108] T. Macaulay, *RIoT Control: Understanding and Managing Risks and the Internet of Things*. Morgan Kaufmann, 2016.
[109] S. Rao, "Open Source Packages for Network Functions Virtualization", *The New Stack*, Feb. 15, 2016. https://thenewstack.io/opensource-virtual-network-functions-part3/ (accessed May 25, 2022).
[110] P. T. Tuyen, "AppArmor vs SELinux", *Artificial Intelligence Kiosk*, Sep. 2022. https://www.omarine.org/blog/apparmor-vs-selinux/ (accessed May 15, 2022).
[111] L. Vrabec, "Technologies for container isolation: A comparison of AppArmor and SELinux", *Enable Sysadmin*, Sep. 22, 2020. https://www.redhat.com/sysadmin/apparmor-selinux-isolation (accessed May 15, 2022).
[112] Z. Xiao and Y. Xiao, "Security and Privacy in Cloud Computing", *Commun. Surv. Tutor. IEEE*, vol. 15, pp. 843–859, Jan. 2013, doi: 10.1109/SURV.2012.060912.00182.

[113] S. Vimercati and S. Foresti, "Privacy of Outsourced Data", 2009. doi: 10.1007/978-3-642-14282-6_14.

[114] N. Santos, K. P. Gummadi, and R. Rodrigues, 'Towards Trusted Cloud Computing", *Proc. 2009 Conf. Hot Top. Cloud Comput. HOTCLOUD*, p. 3, Jan. 2009.

[115] A.-R. Sadeghi, T. Schneider, and M. Winandy, 'Token-Based Cloud Computing", in *Trust and Trustworthy Computing*, Berlin, Heidelberg, 2010, pp. 417–429. doi: 10.1007/978-3-642-13869-0_30.

[116] W. Itani, A. Kayssi, and A. Chehab, "Privacy as a Service: Privacy-Aware Data Storage and Processing in Cloud Computing Architectures", in *2009 Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing*, Chengdu, China, Dec. 2009, pp. 711–716. doi: 10.1109/DASC.2009.139.

[117] C. Gentry, "Fully homomorphic encryption using ideal lattices", in *Proceedings of the 41st annual ACM symposium on Symposium on theory of computing - STOC '09*, Bethesda, MD, USA, 2009, p. 169. doi: 10.1145/1536414.1536440.

[118] *MicroK8s*. [Online]. Available: https://microk8s.io/

[119] "Microservices Demo: Sock Shop". https://microservices-demo.github.io/ (accessed May 10, 2022).

[120] "EOL Ubuntu | End of Life (EOL) | Lifecycle". https://endoflife.software/operating-systems/linux/ubuntu (accessed Jun. 05, 2022).

[121] "EOL Debian | End of Life (EOL) | Lifecycle". https://endoflife.software/operating-systems/linux/debian (accessed Jun. 05, 2022).

[122] "Alpine release branches | Alpine Linux". https://alpinelinux.org/releases/ (accessed Jun. 05, 2022).

[123] T. Hamilton, "What is Response Time Testing? How to Measure for API, Tools", Mar. 02, 2020. https://www.guru99.com/response-time-testing.html (accessed Jun. 05, 2022).

[124] B. Candler, "Interpreting Prometheus metrics for Linux disk I/O utilization", *Medium*, Oct. 10, 2021. https://brian-candler.medium.com/interpreting-prometheus-metrics-for-linux-disk-i-o-utilization-4db53dfedcfc (accessed Jun. 05, 2022).

[125] "aquasecurity/trivy: Scanner for vulnerabilities in container images, file systems, and Git repositories, as well as for configuration issues and hard-coded secrets". https://github.com/aquasecurity/trivy (accessed Jun. 05, 2022).

[126] "ssllabs-scan/ssllabs-api-docs-v3.md at master · ssllabs/ssllabs-scan", *GitHub*. https://github.com/ssllabs/ssllabs-scan (accessed Jun. 05, 2022).

[127] C. Kumar, "How to find Web Server Vulnerabilities with Nikto Scanner ? ", *Geekflare*, Aug. 26, 2016. https://geekflare.com/nikto-webserver-scanner/ (accessed Jun. 05, 2022).

[128] sullo, *nikto*. 2022. Accessed: Jun. 05, 2022. [Online]. Available: https://github.com/sullo/nikto

[129] *Mozilla HTTP Observatory* -. Mozilla, 2022. Accessed: Jun. 05, 2022. [Online]. Available: https://github.com/mozilla/http-observatory/blob/0b7928ee9d09a3f5dffdb81f27d3e5f80ef2024c/httpobs/docs/api.md

[130] "SSL Server Rating Guide · ssllabs/research Wiki", *GitHub*. https://github.com/ssllabs/research (accessed Jun. 08, 2022).

[131] D. Schurter, "Grossansturm - So zwangen Black-Friday-Schnäppchenjäger die Schweizer Online-Shops in die Knie", *bz Basel*. https://www.bzbasel.ch/panorama/vermischtes/so-zwangen-black-friday-schnappchenjager-die-schweizer-online-shops-in-die-knie-ld.1467477 (accessed Jun. 06, 2022).

[132] 3 Minute Read Page Test, "Request Timeouts", *Istio*. https://istio.io/latest/docs/tasks/traffic-management/request-timeouts/ (accessed Jun. 09, 2022).
[133] J. Seidl, *GoldenEye*. 2022. Accessed: May 27, 2022. [Online]. Available: https://github.com/jseidl/GoldenEye
[134] "Chain of Trust - Let's Encrypt", Oct. 02, 2021. https://letsencrypt.org/certificates/ (accessed Jun. 01, 2022).
[135] J. Chapin, "How to stress test your CPU on Linux", *Linux Tutorials - Learn Linux Configuration*, Oct. 07, 2020. https://linuxconfig.org/how-to-stress-test-your-cpu-on-linux (accessed Jun. 01, 2022).
[136] L. S. Tips, "How to Create 100% CPU Load on Linux System", Jun. 15, 2021. https://www.linuxshelltips.com/create-cpu-load-linux/ (accessed Jun. 01, 2022).
[137] "Answer to "How to create a CPU spike with a bash command" ", *Stack Overflow*, Aug. 10, 2013. https://stackoverflow.com/a/18164007 (accessed Jun. 01, 2022).
[138] "The ZAP Homepage". https://www.zaproxy.org/ (accessed Jun. 07, 2022).
[139] "badssl.com". https://badssl.com/ (accessed Jun. 08, 2022).

9 Abbreviations

API	Application Programming Interface
AWS	Amazon Web Services
CVE	Common Vulnerabilities and Exposures
DoS	Denial of Service
DDoS	Distributed Denial of Service
DTM	Dynamic Trust Monitoring
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IaaS	Infrastructure as a Service
IIoT	Industrial Internet of Things
IOPS	Input/Output Operations Per Second
IoT	Internet of Things
IP	Internet Protocol
OS	Operating System
PoC	Proof of Concept
QoS	Quality of Service
NFV	Network Functions Virtualization
SDN	Software Defined Networking
SaaS	Software as a Service
SLA	Service Level Agreements
SSLA	Security Service Level Agreements
TLS	Transport Layer Security
VM	Virtual Machine
VNF	Virtual Network Functions

Appendix A: Captions

List of Tables

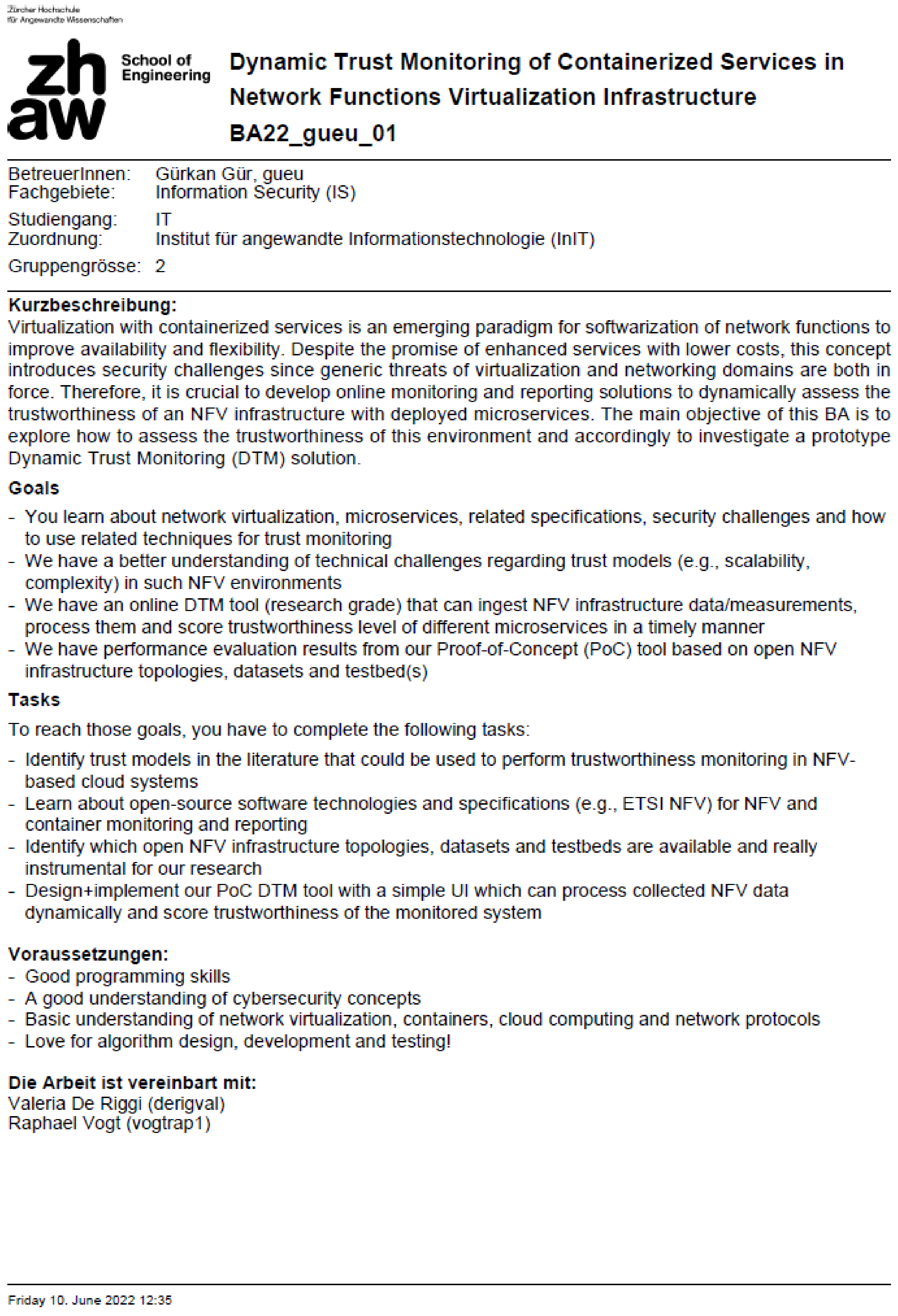
List of Figures

List of Equation

List of Code

Appendix B: Annex

Project Work Description



Zürcher Hochschule
für Angewandte Wissenschaften

zh
aw School of Engineering

Dynamic Trust Monitoring of Containerized Services in Network Functions Virtualization Infrastructure
BA22_gueu_01

BetreuerInnen:	Gürkan Gür, gueu
Fachgebiete:	Information Security (IS)
Studiengang:	IT
Zuordnung:	Institut für angewandte Informationstechnologie (InIT)
Gruppengrösse:	2

Kurzbeschreibung:
Virtualization with containerized services is an emerging paradigm for softwarization of network functions to improve availability and flexibility. Despite the promise of enhanced services with lower costs, this concept introduces security challenges since generic threats of virtualization and networking domains are both in force. Therefore, it is crucial to develop online monitoring and reporting solutions to dynamically assess the trustworthiness of an NFV infrastructure with deployed microservices. The main objective of this BA is to explore how to assess the trustworthiness of this environment and accordingly to investigate a prototype Dynamic Trust Monitoring (DTM) solution.

Goals

- You learn about network virtualization, microservices, related specifications, security challenges and how to use related techniques for trust monitoring
- We have a better understanding of technical challenges regarding trust models (e.g., scalability, complexity) in such NFV environments
- We have an online DTM tool (research grade) that can ingest NFV infrastructure data/measurements, process them and score trustworthiness level of different microservices in a timely manner
- We have performance evaluation results from our Proof-of-Concept (PoC) tool based on open NFV infrastructure topologies, datasets and testbed(s)

Tasks

To reach those goals, you have to complete the following tasks:

- Identify trust models in the literature that could be used to perform trustworthiness monitoring in NFV-based cloud systems
- Learn about open-source software technologies and specifications (e.g., ETSI NFV) for NFV and container monitoring and reporting
- Identify which open NFV infrastructure topologies, datasets and testbeds are available and really instrumental for our research
- Design+implement our PoC DTM tool with a simple UI which can process collected NFV data dynamically and score trustworthiness of the monitored system

Voraussetzungen:
- Good programming skills
- A good understanding of cybersecurity concepts
- Basic understanding of network virtualization, containers, cloud computing and network protocols
- Love for algorithm design, development and testing!

Die Arbeit ist vereinbart mit:
Valeria De Riggi (derigval)
Raphael Vogt (vogtrap1)

Friday 10. June 2022 12:35

GitHub Repository

The code used to calculate the trust calculation in our lab environment can be found in the following GitHub repository:

https://github.com/BA22gueu01/TrustCalculation

Appendix C: Data Sets

Istio Traffic Management – Request Timeouts

Istio’s delay fault injection on the catalogue Sock Shop pod.

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: catalogue
  namespace: sock-shop
spec:
  hosts:
  - catalogue
  http:
  - fault:
      delay:
        fixedDelay: 5s
        percent: 100
    route:
    - destination:
        host: catalogue.sock-shop.svc.cluster.local
        subset: v1
```

The corresponding destination rule for delay fault injection in the catalogue Sock Shop pod.

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: catalogue
  namespace: sock-shop
spec:
  host: catalogue
  trafficPolicy:
    tls:
      mode: ISTIO_MUTUAL
  subsets:
  - name: v1
    labels:
      version: v1
```

Istio’s delay fault injection on the carts Sock Shop pod and the corresponding destination rule.

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: carts
  namespace: sock-shop
spec:
  hosts:
  - carts
  http:
  - fault:
      delay:
        fixedDelay: 5s
        percent: 100
    route:
    - destination:
        host: carts.sock-shop.svc.cluster.local
        subset: v1
```

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: carts
  namespace: sock-shop
spec:
  host: carts
  trafficPolicy:
    tls:
      mode: ISTIO_MUTUAL
  subsets:
  - name: v1
    labels:
      version: v1
```

Istio's delay fault injection on the orders Sock Shop pod and the corresponding destination rule.

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: orders
  namespace: sock-shop
spec:
  hosts:
  - orders
  http:
  - fault:
      delay:
        fixedDelay: 5s
        percent: 100
    route:
    - destination:
        host: orders.sock-shop.svc.cluster.local
        subset: v1
```

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: orders
  namespace: sock-shop
spec:
  host: orders
  trafficPolicy:
    tls:
      mode: ISTIO_MUTUAL
  subsets:
  - name: v1
    labels:
      version: v1
```

Istio's delay fault injection on the front-end Sock Shop pod and the corresponding destination rule.

```yaml
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: front-end
  namespace: sock-shop
spec:
  host: front-end
  trafficPolicy:
    tls:
      mode: ISTIO_MUTUAL
  subsets:
  - name: v1
    labels:
      version: v1
```

(Sub-)Parameters during Testing

The data sets were originally gathered as JSON files from the DTM. For reasons of simplicity, the remaining datasets were deposited on the GitHub repository under https://github.com/BA22gueu01/TrustCalculation/tree/main/thesis/results.

The trustscore.json file from the Baseline test is shown here as an example of a data set section. The first column shows the timestamp and the second column the corresponding value, i.e., in this example the trust score.

Sample One: Truststcore.json

Time	Trust Score
10:49:34	2.738235294
10:55:09	2.796568627
11:01:18	2.813235294
11:06:56	2.871568627
11:12:36	2.929901961
11:18:11	2.988235294
11:23:48	3.004901961
11:29:22	3.021568627
11:35:01	3.079901961
11:40:36	3.096568627
11:46:13	3.154901961
11:51:48	3.213235294
11:57:28	3.271568627
12:03:02	3.329901961
12:08:41	3.346568627
12:14:15	3.363235294
12:19:53	3.379901961
12:25:29	3.396568627
12:31:07	3.454901961
12:36:41	3.471568627
12:42:22	3.529901961
12:47:56	3.546568627
12:53:35	3.563235294
12:59:10	3.579901961
13:04:49	3.579901961
13:10:20	3.538235294
13:15:54	3.579901961
13:21:25	3.538235294
13:27:00	3.496568627
13:32:30	3.454901961
13:38:04	3.496568627
13:43:34	3.496568627
13:49:07	3.454901961
13:54:39	3.454901961
14:00:13	3.454901961
14:05:44	3.454901961
14:11:17	3.413235294
14:16:48	3.371568627
14:22:26	3.413235294
14:27:56	3.413235294
14:33:30	3.413235294
14:39:02	3.413235294
14:44:36	3.413235294
14:50:07	3.454901961
14:55:45	3.413235294
15:01:20	3.454901961
15:06:57	3.454901961
15:12:29	3.496568627
15:18:05	3.538235294
15:23:35	3.579901961
15:29:09	3.579901961
15:34:41	3.621568627
15:40:15	3.621568627
15:45:47	3.621568627
15:51:21	3.579901961
15:56:53	3.579901961
16:02:29	3.579901961
16:08:01	3.579901961
16:13:35	3.538235294
16:19:06	3.496568627
16:24:41	3.496568627
16:30:13	3.496568627
16:35:47	3.454901961
16:41:19	3.454901961
16:46:57	3.454901961
16:52:28	3.454901961
16:58:01	3.413235294
17:03:40	3.371568627
17:09:15	3.371568627
17:14:48	3.371568627
17:20:24	3.371568627
17:25:56	3.329901961
17:31:32	3.329901961
17:37:04	3.288235294
17:42:41	3.246568627
17:48:14	3.204901961
17:53:49	3.204901961
17:59:22	3.204901961
18:04:58	3.246568627
18:10:31	3.246568627
18:16:06	3.246568627
18:21:39	3.246568627
18:27:15	3.246568627
18:32:48	3.246568627
18:38:25	3.288235294
18:43:58	3.288235294
18:49:34	3.288235294
18:55:06	3.288235294
19:00:57	3.288235294
19:06:29	3.329901961
19:12:07	3.371568627
19:17:39	3.371568627
19:23:16	3.371568627
19:28:49	3.329901961
19:34:25	3.329901961
19:39:59	3.329901961
19:45:35	3.288235294
19:51:09	3.288235294
19:56:46	3.288235294
20:02:19	3.288235294