Towards Practical Dynamic Trust Monitoring of Containerized Services in NFV Infrastructure

Valeria De Riggi, Raphael Vogt, Onur Kalinagac and Gürkan Gür
Inst. of Applied Information Technology (InIT)
Zurich University of Applied Sciences (ZHAW)
Winterthur 8401, Switzerland
valeria@deriggi.ch, raphael.vogt@grosswies.ch, {kalo, gueu}@zhaw.ch

Abstract—**Although Network Function Virtualization (NFV) and containerized services embedded therein are an already active research field while becoming increasingly widespread in practice (e.g., 5G networks), the trust and security challenges still deserve more attention. To tackle relevant issues for this aspect, our work 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 and evaluated. These parameters are subsequently included in a trust calculation framework for their confidence analysis. Finally, a Dynamic Trust Monitoring (DTM) solution, namely MicroDTM, that supervises the trustworthiness of containerized services in an NFV infrastructure is proposed. By collecting and processing the trust parameters, a containerized service environment is evaluated according to trustworthiness for different scenarios. In addition to performance analysis, improvements and extensions necessary to use the system in a practical environment are identified.**

I. Introduction

Service providers, network operators and companies with ICT infrastructure have been able to reduce capital and operating expenditure costs with the help of virtualised networks and cloud computing [1]. From an economic point of view, getting rid of own hardware or consolidated ICT environments is lucrative. However, virtualised network environments, which are used today in cloud environments or mobile infrastructure such as 5G networks have the common feature of being complex constructs [2]. As virtualised service environments are highly software-based, they also offer a larger attack surface in terms of software vulnerabilities and new attack vectors [3]–[5]. Actionable and dynamically-assessed trust in the infrastructure and software becomes a challenging problem in this setting. Therefore, practical mechanisms such as trust-awareness and monitoring solutions in addition to conventional security measures for securing and protecting these systems are imperative.

The deployment of these pervasive digital systems is driven by many mission critical applications being transformed into connected ubiquitous services. For instance, 5G networks found numerous use cases such as in the areas of health, manufacturing or transport systems, e.g., in autonomous vehicles or in smart city context [6]. The requirements imposed by those applications also lead to the question of how to dynamically monitor and appropriately measure the trustworthiness of containerized services and NFV infrastructures. That capability is crucial for taking mitigation actions in deteriorating security posture, achieving resilience or optimizing the placement of critical services (e.g., in the selection of network slices). It is expected to be even more instrumental with the introduction of native cloud architectures and Containerized Network Functions (CNFs) in future networks such as 6G [7]. Moreover, these networks are expected to have an open, multi-party and service-based architecture where the infrastructure includes not just the network operator assets, but third-party services and devices working together to realize advanced services. Therefore, it is crucial to monitor and quantify trust for deployed assets and services from different entities, since zero-risk security cannot be achieved [6].

The main objective of this work is to address the question of how the trustworthiness of these complex virtual network environments can be evaluated practically and with lean schemes. We first assess related work and explain and lay out basic terms as they are used in our work. Subsequently, we address theoretical aspects of trustworthiness and present the Microservices Dynamic Trust Monitoring (MicroDTM) solution[1]. The open-source MicroDTM Proof-of-Concept (PoC) is run in a test environment which presents a reference NFV framework and deployed microservices therein. The adopted trust metrics are then evaluated and assessed in terms of effectiveness. Finally, a review of the experimental results is described and conclusions are drawn.

II. Background and Related Work

In the technical literature, the definition of trust varies as demonstrated in [8]–[11]. Before discussing the definitions of the individual MicroDTM trust attributes, various concepts of trust from different related work are presented in this section. [8] describes trust as a belief with dynamic fixed value tied to a context and time. The authors assign dynamic values of this "firm belief" using a trust range consisting of six grades. 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. uses [8]'s definition of trust as a basis for their Quality of Service (QoS)

[1]MicroDTM source code is available at https://github.com/BA22gueu01/MicroDTM as an open-source project.

DOI: 10.1109/ICC45041.2023.10279302

TABLE I
TRUST NOTATIONS ACCORDING TO SUN AND DENKO [10].

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

based trust mode in [9]. In that regard, trust is described as “an entity based on reliability and firm belief based on attribute of the entity”. The authors state that numerous works consider behaviour-based algorithms as well as rank-based trust models to map their models to real paradigms but data center and QoS parameters are not considered. In [10] three stages are shown, where the first two stages are for calculating and periodically updating the trust value, and the last is to delete expired entries. The definitions used for the trust value computations are depicted in Table I. 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. For instance, $T_A(B) = 1$ means Device A has complete trust to Device B while -1 means complete mistrust.

The direct trust computation is represented by:

$$T_A(B) = f(O_A(B)) \tag{1}$$

where f represents the particular function transferring the direct observation value to the corresponding trust value. The indirect computation is represented as:

$$T_A(B) = \begin{cases} w_1 \cdot f(O_A(B)) + w_2 \cdot R_A(B) \\ \quad \text{if } O_A(B) \neq 0, \quad where\ w_1 + w_2 = 1, \\ \qquad\qquad\qquad\qquad w_1 > 0, and\ w_2 > 0 \\ w_3 \cdot R_A(B) \\ \quad \text{if } O_A(B) = 0, \quad where\ 0 < w_3 \leq 1 and \\ \qquad\qquad\qquad\qquad\qquad w_3 \geq w_2 \end{cases} \tag{2}$$

where w_1 and w_2 represent the weights by which the observation and recommendation value scale. $R_A(B)$ represents the recommendation value and results from the average of trust values given to device B by all neighbouring devices of A as:

$$R_A(B) = \frac{1}{n} \sum_{i=1}^{n} T_{D_i}(B) \tag{3}$$

where $T_A(D_i) > 0$. This is to prevent device A from accepting recommendations from untrustworthy devices. Furthermore, the trust value should be updated and maintained periodically for reliability and safety reasons after the initial trust calculation. The update calculation is based on the current behaviour and previous trust value as:

$$T_A(B) = \begin{cases} 1, & \text{if } T'_A(B) + C_A(B) \geq 1 \\ -1, & \text{if } T'_A(B) + C_A(B) \leq -1 \\ T'_A(B) + C_A(B), & \text{otherwise} \end{cases} \tag{4}$$

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 where $-1 \leq C_A(B) \leq 1$.

Trust attributes differ depending on the reference which defines trust and related concepts [12]–[14]. [15] describes the trustworthiness of software systems with the help of six key attributes with the attribute QoS subdivided into three quality characteristics, namely *Availability*, *Reliability* and *Performance*. We can define these six attributes as follows:

a) 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.

b) Safety implies that nothing bad happens. The term bad in this context describes a measure of the probability and severity of harm to a user or its environment. This parameter was omitted from the implementation of our MicroDTM due to the complexity of its measurability.

c) Availability is the probability that a system can provide and fulfil a certain service at a certain time without errors or interruptions.

d) Reliability depends 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.

e) Performance refers to the ability of a system to process a certain request in a given unit of time.

f) Security: 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.

III. MICRODTM DESIGN AND IMPLEMENTATION

For MicroDTM operation, the open-source platform OpenStack is used as IaaS with the lightweight Kubernetes distribution MicroK8s [16]. The latter is used for the automated construction of containerised applications. In this system, a cluster is running based on the Kubernetes example application Sock Shop [17]. The data from different Sock Shop Kubernetes pods are then collected by Prometheus for trust calculation.

A. Trust Calculation Workflow

The MicroDTM trust workflow is shown in Figure 1. Our trust calculation is based on the trust score concept by Joseph et al. in [14]. A streamlined mechanism for trust calculation was proposed, with each main parameter equally weighted. We changed their five parameters, namely persistence, competence, reputation, credibility, and integrity, to the trust parameters listed and detailed in [15]. Our changes also include

DOI: 10.1109/ICC45041.2023.10279302

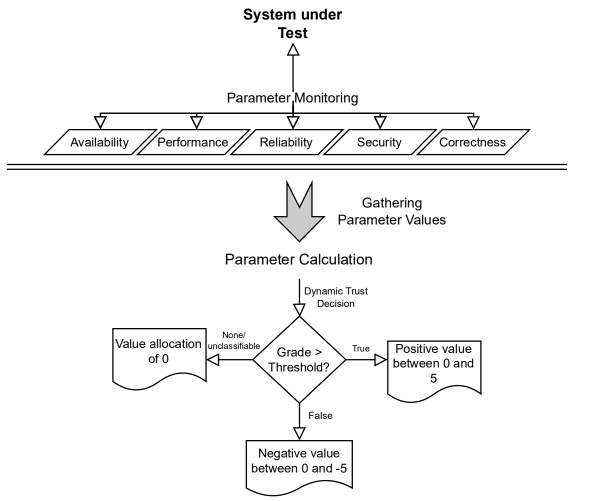

Fig. 1. Flow chart of gathering trust parameter grades and dynamic trust scoring in MicroDTM.

additional attributes for the trust calculation that we consider to be essential. A discussion about the sub-parameters and weights used is provided in the following sections. Table III summarises the most important metrics.

a) Scale: Our scale for trustworthiness is based on [10]. The authors used an intuitive scale ranging from -1 to 1, with -1 as untrusted, 0 as neutral, and 1 as trusted. Furthermore, we increased the scale to [-5,5] to have a bigger range. This allows us to detect the effects on the trust score more granularly.

b) Trust $\mathbf{T}$ *Calculation at* t *= 0:* We implemented our MicroDTM 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. That provided us a flexible deployment and test setup.

c) Updating and Averaging Grades: We split our sub-parameters into two groups: the first was updated daily and the second hourly. For the latter, we saved the last 24 values and calculated the final score by averaging. We used this approach to mitigate the effect of spikes in our parameter measurements.

d) Multiple Pods: Our test environment consisted of multiple pods with multiple containers. To counteract the problem of different number of containers per pod, hree subcategories for trust calculation were used: Prometheus request, `kubectl` request, and external request. For Prometheus requests, each container was graded. The average of all container grades resulted in the final sub-parameter grade. If data could be collected with the `kubectl` request, we calculated a grade for each container per pod. Afterwards, their mean value was calculated for the grade value of the corresponding pod. The sub-parameter grades were calculated by averaging over the pod grades. All queries containing external requests were directly mapped into a single grade.

B. MicroDTM Trust Parameters

Please note that our trust parameters are not mutually orthogonal, i.e., they may have overlapping behavior due to dependencies between system elements.

TABLE II
GRADING SCHEMA FOR RELIABILITY AND PERFORMANCE

	Reliability				Performance			
Grade	Status Code Comparison	Initial Log Level Count	hourly Log Level Count	Patch Level Count	Response Time	Memory Usage	Disk Access	CPU Usage
5	<0.25	<1000	<10	OS updated, all pods updated	<0.5s	<70%	<25%	<75%
0	<0.5	≤3000	≤40	OS updated, some pods outdated	<1s	≤85%	≤50%	≤90%
-5	≥0.5	>3000	> 40	OS outdated	≥1s	>85%	>50%	>90%

1) Availability: For availability, we used `uptime` U which is calculated every hour with values queried from Prometheus. The current value and the value from an hour ago were queried. Then we calculated the normalized uptime $\overline{U}$ as follows:

$$\overline{U} = \frac{uptime_{now} - uptime_{last}}{time_{now} - time_{last}} \tag{5}$$

We then classified the uptime according to the availability nines as per the SLAs of Telstra Corporation [18], AWS, and Microsoft Azure [19] leading to the ranged values $(Availability : Grade)$ ∈{(99.9%:5), (95%:4), (90%:3), (75%:0), (50%:-5)}

2) Reliability: For reliability, we used three sub-parameters: `Status Code Comparison`, `Log Level Count`, and `Patch Level`. The scoring per individual sub-parameter grade is depicted in Table II.

a) Status Code Comparison: This is a comparison between the `500` HTTP request state, i.e. calls with errors, and the `200` HTTP request state, i.e. calls without errors. We queried Prometheus for the current number of these responses and the number of responses an hour earlier. We then calculated the amount of `200` and `500` responses in the last hour and then divided the amount of `500`s by the amount of `200`s. This quotient was then made into a grade according to Table II.

b) Log Level Count: To get this value we accessed the logs of every pod and counted the number of warnings and errors. Afterwards, 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 thresholds for errors and warnings were set high. 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. For the Log Level Count grading, two different scales were used as depicted in Table II.

c) Patch Level: To calculate the Patch Level, we examined every pod for available updates by using the `kubectl` and `exec` commands and the internal update system of the corresponding

DOI: 10.1109/ICC45041.2023.10279302

Linux distribution. 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, the system returned -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.

As the Log Level Count can be high even in a good and reliable system we weighed this sub-parameter half in comparison to the other two, i.e., the Status Code Comparison and Patch Level parameters.

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 standard performance values similar to common monitoring solutions. The contribution of each parameter is context-dependent including the application. We weighed them equally except for response time, which we gave a doubled weight, because our test system is a web shop and response time is the parameter that is directly experienced by customers.

Response Time: These grades were calculated based on [20].

Memory Usage: These grades were calculated based on [21].

Disk Access: Disk Access was split into two parts: `Disk Write` and `Disk Read`. Grades were calculated according to [22]. Since the Prometheus query includes the quotient of the time of the disk read to the write query in seconds for the completed queries, percentages are assigned to the grades for the weighting of the parameters. After calculating the grades for each part an average of the two was calculated. This resulted in the `Disk Access` grade.

CPU Usage: For mapping the CPU usage to the highest grade, the recovery grade in [23] was used. The overload value in [24] was used for the worst weighting. For the neutral weighting, the medium threshold in [25] was used.

4) Correctness: For Correctness, an 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, the grade 5 was given. Otherwise, the grade -5 was set.

5) Security: Security is a wide topic which can be integrated into trust calculation in various schemes. We have picked three important elements for our PoC.

a) Vulnerability Check (security testing based): We implemented a vulnerability check scheme consisting of SSL Labs Scan [26], Nikto [27], [28], and Mozilla's HTTP observatory [29]. All these products were queried via their APIs and then each result was mapped to a grade from -5 to 5. As the final grade, we took the average of these 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)* [30]. Nikto's result consists of a list of vulnerabilities and misconfigurations detected. We linearised this number $\mathcal{V}$ into a grade according to (6):

TABLE III
MICRODTM PARAMETERS AND THEIR WEIGHTS IN THE TRUST CALCULATION.

Parameter	**Sub-Parameter**	**Weight**	**Update Time**	**Initial Calculation**	**Averaging**
Availability (A)	Uptime	1.0	Hourly	Last 24h	Last 24
Reliability (R)	Status Code Comparison	0.4	Hourly	Last 24h	Last 24
	Log Level Count	0.2	Hourly	No	Last 24
	Patch Level	0.4	Daily	No	No
Performance (P)	Response Time	0.4	Hourly	Last 24h	Last 24
	Memory Usage	0.2	Hourly	Last 24h	Last 24
	Disk Access	0.2	Hourly	Last 24h	Last 24
	CPU Usage	0.2	Hourly	Last 24h	Last 24
Correctness (C)	Call Correctness	1.0	Hourly	No	Last 24
Security (S)	Vulnerability Check	0.5	Daily	No	No
	Certificate Check	0.3	Daily	No	No
	AppArmor	0.2	Daily	No	No

$$Grade = 5 - min[\mathcal{V}/2, 10] \tag{6}$$

b) AppArmor (host based): AppArmor is a security system which provides access control mechanisms for programs. If AppArmor was enabled, we gave the grade 5, otherwise, the grade -5 was assigned. After checking all pods, we calculated an average over all pods.

c) Certificate Check (PKI based): We verified the certificate by sending an HTTP and HTTPS request to the corresponding domain name. The HTTP request in advance to the HTTPS request was used to check if the site was available. If the site was not 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 similar to the validity check 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.

d) Weighting: Because the vulnerability check consists of three different sub-parameters which check different aspects of security, the vulnerability check is given a weighting of 0.5. The web page certificate is one of the first indicators regarding security for a customer. Thus we gave the `Certification Check` a higher weighting compared to AppArmor, i.e., 0.3 for Certification Check and 0.2 for AppArmor.

Table III 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.

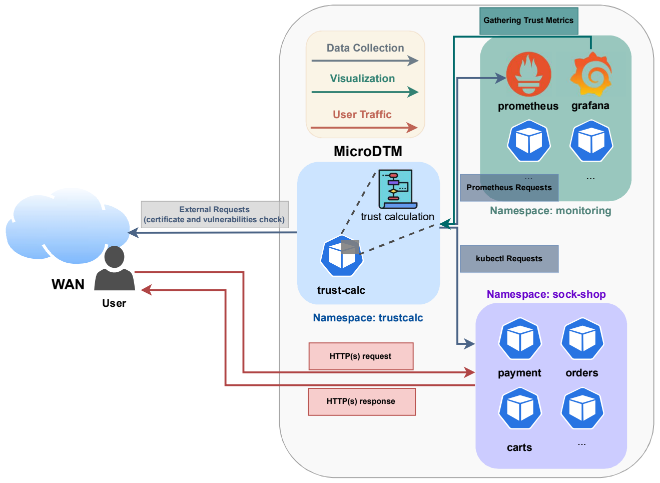

Fig. 2. MicroDTM system and deployment.

Finally, the trust score was calculated with the following formula.

$$\mathbf{T} = \omega_A * A + \omega_R * R + \omega_P * P + \omega_C * C + \omega_S * S \quad (7)$$

where T is the trust score and ω_x was set to the factor 0.2 for each parameter.

C. *Test Environment*

The overall MicroDTM system is shown in Figure 2. As test environment, the 5G testbed of the National Centre of Scientific Research Demokritos (NCSRD) in Greece was used. In this network, the open-source platform OpenStack and MicroK8s as the container environment are used. The deployment of Sock Shop application consists of different Kubernetes pods within one namespace. Istio service mesh has a running container to get the data out of the pods. This data is then collected by Prometheus and visualized by Grafana. MicroDTM was implemented in Python. Microk8s was used for deploying our MicroDTM as a Docker image in our test system. Grafana was used to display the different values of our MicroDTM graphically.

IV. Experiments and Results

The experiments were conducted in stages and focused on dynamic changes in the (sub-)parameters and the resulting trust score in MicroDTM. Due to different parameters, we considered how individual parameter groups could be changed, and also how cascading scenarios could be achieved.

A. *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 MicroDTM, a warm-up period 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 normal state, the trust score is observed to reach an average value of 3.36. The (sub-)parameters except for Correctness remain constant at the values shown in Table IV. Particularly. the Correctness grade changes sporadically and reaches different values.

TABLE IV
The system's normal state (NC: Not consistent).

Parameter	Value	Sub-Parameter	Value
Availability	5	Uptime	5
Reliability	1.94	Status Code Comparison	5
		Log Level Count	5
		Patch Level	-2.35
Performance	3.55	Response Time	5
		Memory Usage	2.5
		Disk Access	5
		CPU Usage	5
Correctness	NC	Call Correctness	NC
Security	4.88	Vulnerability Check	0.5
		Certificate Check	5
		AppArmor	4.71

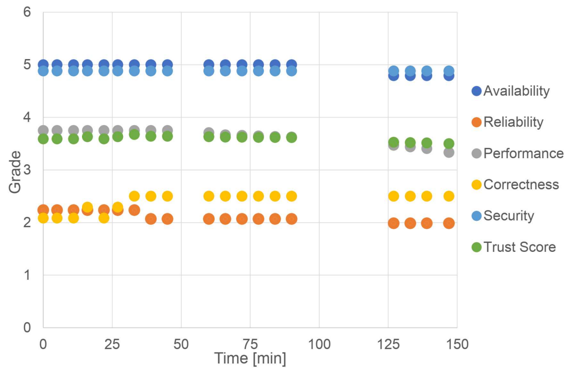

Fig. 3. Results of second DoS test.

B. *Scenario 1: Denial of Service (DoS)*

The idea behind a DoS test is to be able to influence as many trust parameters as possible. In addition to Istio's `Request Timeouts` module, the open-source Python program Golden Eye [31] was used as a test kit for the DoS attack. Three attempts of the DoS test were performed. At the first time, it was impossible to read out data because the system crashed. In the other two attempts, data could be read out. The second run is shown in Figure 3 and the third in Figure 4. During the unresponsiveness of the system, the affected sub-parameters returned the grade 0. Therefore, it looks like most of the parameters are missing. This also led to a small drop of about 0.15 in the trust score.

C. *Scenario 2: Invalid Certificate*

This 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 our university ZHAW. The results are shown in Figure 5. It can be seen that after the installation of the root and intermediate certificates the Certificate grade, Security grade and trust score increase immediately.

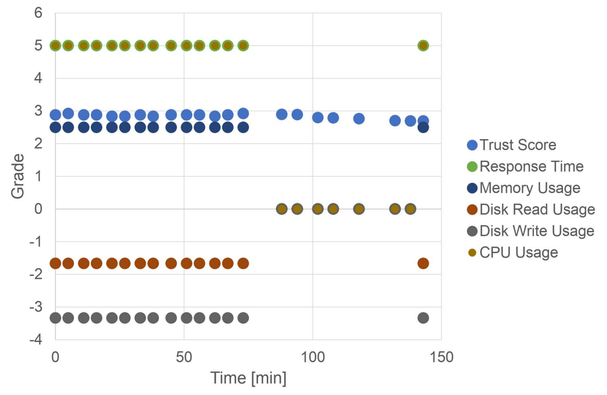

Fig. 4. Results of third DoS test.

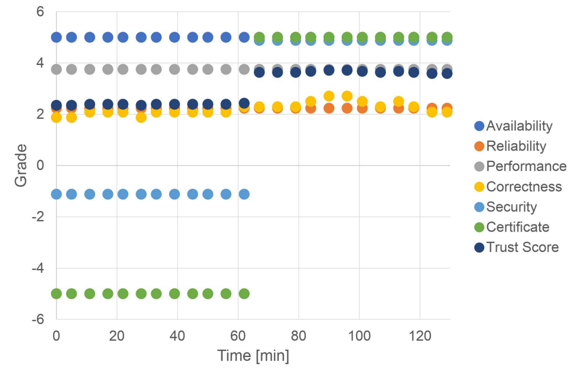

Fig. 5. Results of certificate test.

D. *Vulnerability*

For the vulnerability tests, different domains were checked iteratively for their dissimilarities so that wide-ranging results could be achieved. With each daily update, i.e., during testing for two hours, the next domain was reviewed. The corresponding domain names are listed as {moodle.zhaw.ch, zhaw.ch, mozilla.org, google.com, wikipedia.org}. As can be seen in Figure 6 the Vulnerability test influenced the Security grade as expected.

V. Discussion

a) Baseline and (Sub-)Parameter Behavior: Most of the (sub-)parameters achieved the expected results. The exceptions are the `Patch Level`, `Disk Write`, and `Correctness` grades.

The Patch Level grade achieved negative values as most of the pods in the sock-shop namespace are not patchable. 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. 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.

b) The Correctness Fluctuation: The Correctness parameter returned unexpected results. The variations occurred during tests as well as in the normal state. Errors in the implementation of the Correctness check in the MicroDTM could be excluded. Analyses comparing the front end data with the database values via API calls showed that values actually do not match to some extent. Since we concluded that these changes were due to poor application design, we omitted the effects of it in further discussions.

c) DoS: The overall influence of the DoS test was smaller than expected. If in a production scenario, the MicroDTM 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 the user would not be able to rely on the MicroDTM in this respect. This is a potential improvement for MicroDTM.

d) Interpretation of the Security Parameter Tests: The effect of the security parameter tests was bigger than the effect of the DoS test. Nevertheless, the tests still were not able to give out a negative trust score.

e) Interpretation of the MicroDTM and Trust Scoring: The MicroDTM is currently a 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. The important practical limitations in the current implementation of MicroDTM are as follows:

Memory: It has a limited "memory span" of 24 hours. This could be a configurable system parameter.

Warm-Up Period: The MicroDTM needs a warm-up period as long as the historical data it relies on.

Persistence: The collected data is not retained after system restarts. Meaning that in case of a reboot, all the historical data is lost, and the warm-up period must be re-initialised.

Single Process: The MicroDTM runs as a single process application. This already led to problems during the testing period where the daily update in the test environment took longer than expected and blocked the hourly update. The daily and hourly updates should be developed as two individual processes for a production-grade MicroDTM.

Trust Restoration: During the testing period of MicroDTM, we found that an untrusted system can regain trust much faster than we intended. If we do small changes, for example replacing an untrusted certificate with a trusted one, the overall security grade increases rapidly. This is not the desired effect,

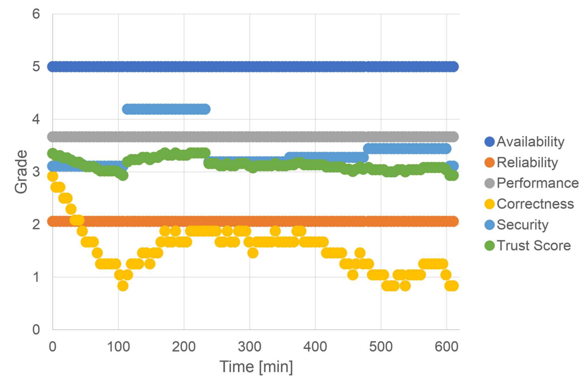

Fig. 6. Results of vulnerability test.

as negative incidents in the MicroDTM should be "remembered" for a longer period of time.

Trust Loss: In other experiments, we have seen that it is difficult to lose trust after a single incident. Therefore, the underlying trust calculation still needs to be improved in case high sensitivity to such incidents is desired.

VI. Conclusion

In this work, we have developed an open-source dynamic trust monitoring scheme based on network and service measurements for service-based 5G networks. The MicroDTM approach is also applicable for future networks since they are envisaged to heavily rely on microservices for providing elastic, pervasive and high-performance services. We identified the practical problems on how such a system could be designed (e.g., the utility of some common trust parameters) and the practical problems of trust monitoring.

Future research includes the examination of whether the division into processes for the calculation of the individual parameter groups would lead to better 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 MicroDTM. Those weights may depend on the specific characteristics of the monitored service chain or could be optimized using some pre-generated or collected data sets via machine learning approaches.

VII. Acknowledgement

The research leading to these results received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 871808 (5G PPP project INSPIRE-5Gplus). The paper reflects only the authors' views. The EC is not responsible for any use that may be made of the information it contains.

References

[1] 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)*, pp. 43–47, 2016.

[2] C. Suraci, G. Araniti, A. Abrardo, G. Bianchi, and A. Iera, "A stakeholder-oriented security analysis in virtualized 5G cellular networks," *Computer Networks*, vol. 184, p. 107604, 2021.

[3] T. Komperda, "Virtualization security." https://resources.infosecinstitute.com/topic/virtualization-security-2/, 2012. Accessed: 2022-11-03.

[4] M. Alenezi and M. Zarour, "On the relationship between software complexity and security," *International Journal of Software Engineering & Applications*, vol. 11, no. 1, pp. 51–60, 2020.

[5] Y. Javed, M. Alenezi, M. Akour, and A. Alzyoud, "Discovering the relationship between software complexity and software vulnerabilities," *Journal of Theoretical and Applied Information Technology*, vol. 96, pp. 4690–4699, 2018.

[6] C. Gaber, J. S. Vilchez, G. Gür, M. Chopin, N. Perrot, J.-L. Grimault, and J.-P. Wary, "Liability-aware security management for 5G," in *2020 IEEE 3rd 5G World Forum (5GWF)*, pp. 133–138, 2020.

[7] P. Porambage, G. Gür, D. P. M. Osorio, M. Liyanage, A. Gurtov, and M. Ylianttila, "The roadmap to 6G security and privacy," *IEEE Open Journal of the Communications Society*, vol. 2, pp. 1094–1122, 2021.

[8] 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*, pp. 452–452, 2002.

[9] M. K. Goyal, A. Aggarwal, P. Gupta, and P. Kumar, "QoS based trust management model for cloud IaaS," in *2012 2nd IEEE Int. Conference on Parallel, Distributed and Grid Computing*, pp. 843–847, 2012.

[10] 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*, pp. 1219–1222, 2007.

[11] P. Govindaraj, "A review on various trust models in cloud environment," *Journal of Engineering Science and Technology Review*, vol. 10, pp. 213–219, 03 2017.

[12] 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.

[13] S. Romdhani, G. Vargas-Solar, N. Bennani, and C. Ghedira-Guegan, "QoS-based trust evaluation for data services as a black box," in *2021 IEEE Int. Conference on Web Services (ICWS)*, pp. 476–481, 2021.

[14] 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, pp. 1–19, 08 2018.

[15] S. Becker et al., "Trustworthy software systems: a discussion of basic concepts and terminology," *ACM SIGSOFT Software Engineering Notes*, vol. 31, no. 6, pp. 1–18, 2006.

[16] Microk8s Project, "MicroK8s system." https://microk8s.io. Accessed at 2022-06-30.

[17] Sock Shop, "Microservices demo: Sock shop." https://microservices-demo.github.io/. Accessed at 2022-05-10.

[18] M. Lewis, "Telstra guarantees 5G network uptime with enhanced enterprise wireless." https://www.mobilecorp.com.au/blog/telstra-guarantees-5g-network-uptime-with-enterprise-enhanced-wireless. Accessed at 2022-05-24.

[19] O. Shushan, "AWS vs. Azure vs. GCP | detailed comparison." https://www.cloudride.co.il/blog/aws-vs.-azure-vs.-gcp-detailed-comparison. Accessed at 2022-11-10.

[20] T. Hamilton, "What is response time testing? how to measure for API, tools." https://www.guru99.com/response-time-testing.html. Accessed at 2022-06-05.

[21] Trend Micro Incorporated, "Alert: The memory warning threshold of manager node has been exceeded | deep security." https://help.deepsecurity.trendmicro.com/aws/memory-warning-threshold.html. Accessed at 2022-05-08.

[22] B. Candler, "Interpreting prometheus metrics for linux disk i/o utilization." https://brian-candler.medium.com/interpreting-prometheus-metrics-for-linux-disk-i-o-utilization-4db53dfedcfc. Accessed at 2022-06-05.

[23] Huawei, "Configuring the thresholds of CPU usage and memory usage - NE05e and NE08e v300r003c10spc500 configuration guide - system management 01 - huawei." https://support.huawei.com/enterprise/en/doc/EDOC1100058923/bfc52bc9/configuring-the-thresholds-of-cpu-usage-and-memory-usage#dc_vrp_dev_cfg_0004. Accessed at 2022-05-08.

[24] Progress Community, "Configuring a CPU threshold monitor - Progress community." https://community.progress.com/s/article/Configuring-a-CPU-threshold-monitor. Accessed at 2022-04-30.

[25] Hewlett Packard Enterprise, "monitor cpu-usage threshold." https://techhub.hpe.com/eginfolib/networking/docs/switches/5950/5200-4005_fund_cr/content/499752694.htm. Accessed at 2022-04-30.

[26] SSL Labs, "ssllabs-scan/ssllabs-api-docs-v3.md at master · ssllabs/ssllabs-scan." https://github.com/ssllabs/ssllabs-scan. Accessed at 2022-06-05.

[27] C. Kumar, "How to find web server vulnerabilities with Nikto Scanner?." https://geekflare.com/nikto-webserver-scanner/. Accessed at 2022-06-05.

[28] Sullo, "Nikto Web Server Scanner." https://github.com/sullo/nikto. Accessed at 2022-06-05.

[29] Mozilla, "Mozilla HTTP observatory." https://github.com/mozilla/http-observatory/blob/0b7928ee9d09a3f5dffdb81f27d3e5f80ef2024c/httpobs/docs/api.md. Accessed at 2022-06-05.

[30] SSL Labs, "SSL server rating guide - ssllabs/research wiki." https://github.com/ssllabs/research. Accessed at 2022-06-08.

[31] J. Seidl, "GoldenEye." https://github.com/jseidl/GoldenEye. Accessed at 2022-05-27.