UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
UNIDADE ACADÊMICA DE SISTEMAS E COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

GABRIEL SILVA VINHA

CHALLENGES IN MONITORING LARGE SCALE SECURE APPLICATIONS

CAMPINA GRANDE - PB

2019

GABRIEL SILVA VINHA

CHALLENGES IN MONITORING LARGE SCALE SECURE APPLICATIONS

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Professor Dr. Andrey Elísio Monteiro Brito.

CAMPINA GRANDE - PB

2019

V784c Vinha, Gabriel Silva.
Challenges in monitoring large scale secure applications. / Gabriel Silva Vinha. – 2019.

10 f.

Orientador: Prof. Dr. Andrey Elísio Monteiro Brito.
Trabalho de Conclusão de Curso - Artigo (Curso de Bacharelado em Ciência da Computação) - Universidade Federal de Campina Grande; Centro de Engenharia Elétrica e Informática.

1. Confidential computing. 2. Monitoring large scale secure applications. 3. Trustworthness applications. 4. User`s privacy. 5. Data protection. I. Brito, Andrey Elísio Monteiro. II. Título.

CDU:004(045)

Elaboração da Ficha Catalográfica:

Johnny Rodrigues Barbosa
Bibliotecário-Documentalista
CRB-15/626

GABRIEL SILVA VINHA

CHALLENGES IN MONITORING LARGE SCALE SECURE APPLICATIONS

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA:

Professor Dr. Andrey Elísio Monteiro Brito
Orientador – UASC/CEEI/UFCG

Professor Dr. Thiago Emmanuel Pereira da Cunha Silva
Examinador – UASC/CEEI/UFCG

Professor Dr. Tiago Lima Massoni
Examinador – UASC/CEEI/UFCG

Trabalho aprovado em: 25 de novembro 2019.

CAMPINA GRANDE - PB

Challenges in Monitoring Large Scale Secure Applications

GABRIEL SILVA VINHA and ANDREY BRITO, Universidade Federal de Campina Grande, Brazil

One of the major concerns for researchers and application developers is to safeguard user's privacy. To provision secure applications developers and researchers are making use of trusted execution environments. This enables isolation for data processing and secure communications across entities. The cost of provisioning such applications comes in availability and scalability for high load systems. Running production and large scale trusted applications can be very resource intensive. Therefore, achieving certain quality requirements for these applications and maintaining the trustworthiness is a challenge. In this paper we propose an environment system capable of provisioning secure applications in a large infrastructure for variable load demand. The system can be provisioned in any set of machines with access to Intel SGX hardware. We enabled the monitoring of application and infrastructure metrics with information specific to the security hardware, which can be later used to make scheduling and setup decisions. In a current large-scale operation, we observed that the average response times across all servers were elevate in the initial parts of experiments, but the operation latency values were around 10 ms, showing good scalability and providing confidence that the system would work satisfactorily in production.

Additional Key Words and Phrases: Confidential Computing, Monitoring, Distributed Systems, Scalability

1 INTRODUCTION

New laws and regulations such as the European Union's General Data Protection Law[1] or the Brazilian government's Lei Geral de Proteção de Dados Pessoais (general personal data protection law) have recently been published and approved by governments. This shows that security and privacy protection became major concerns in private corporations and governments. We can see it with the latest. One of the main fields where data protection is required is with energy consumption and distribution data. We see that, with smart metering and other public information it is possible to violate the privacy of users in a grid [5]. The metering information is required then to be collected and securely stored and processed. The admin-level attack model is a much common way to sabotage these processing and storage systems and breach user privacy by accessing the memory of disks from the machines designated to do the processing and storage of the above mentioned information [8].

To prevent data breach from this attack model, it is possible to make use of trusted execution environments, where the application runs in a safe hardware mode and needs to be attested before starting up and receiving data, this also involves receiving secrets such as keys and other important system information from a trusted part.

In our use case, information is very sensitive and clients of the application should be isolated from each other to prevent leakage (ex., through software bugs). All the customers data is encrypted and only decrypted inside the SGX enclave, but, in addition, the servers or processes where the data will be processed need to be isolated. Thus, although a physical node can host several servers, each server process should not be able to decrypt other customer data and the data from different customers will not be mixed in one database.

As isolation is required and clients can spend a long period between accesses keeping the exact same number of servers and clients can be overkill and result in unnecessary spending with hardware, energy, cooling, etc. Therefore we need to dynamically increase and decrease the number of servers as the load in the system increase. That way we can use an optimal number of services ready for new clients thus providing high availability. Additionally, the requests and data traffic need to be done with a good performance since the systems may be under heavy load, unexpectedly resulting in poor quality of service (QoS) for customers. For our application, QoS is defined as the response time by the system. As there are many factors that can increase the response time, such as network issues, memory leaks, high CPU utilization, etc., in this paper we consider a large variety of metrics that may harm the client's experience, including metrics that are not currently collected by cloud providers or orchestration systems, such as the Intel SGX's Enclave Page Cache (EPC) usage.

In order to achieve the desired state of resource and hardware economy, high availability and QoS requirements we propose the Electronic Client Meterign service (ECM). The ECM provides an API for metering clients to securely push encrypted information that will be processed inside SGX enclaves. It also provides a monitoring platform with state-of-the-art and cloud native monitoring tools to help the decision and scaling of resources in the system. As a consequence, we were able to reach the QoS requirements for response times in clients requests, which was a request latency of around 10ms, with peaks during log in and log out operations.

The rest of the paper is organized as follows. In Section 2 we present some background information, introducing Intel SGX, SCONE, Kubernetes and Prometheus as requirements for the development. The use case is presented and detailed in Section 3, where the requirements, implementation, and security issues for a smart grid use case are presented. This sections details as well as the monitoring of the system. In Section 4, we detail the ECM system with information about the services that composes the architecture, the platform, and some of the monitoring requirements. Finally, we conclude with the evaluation and discussion of the results in Sections 5 and 6, presenting the metrics collected and the system's functionality.

2 BACKGROUND

2.1 Intel SGX

Intel **S**oftware **G**uard e**X**tensions (SGX) is a Trusted Execution Environment (TEE) based on Intel hardware. TEEs are useful for ensuring

[1] https://gdpr-info.eu/

Authors' address: Gabriel Silva Vinha; Andrey Brito, gabrielvinha@lsd.ufcg.edu.br, andrey@computacao.ufcg.edu.br, Universidade Federal de Campina Grande, R. Aprígio Veloso, 882 - Universitário, Campina Grande, Paraíba, Brazil, 58429-900.

security of sensitive data or code from disclosure or modification. In the case of Intel SGX, it enables user-level code to allocate enclaves (i.e., private regions of memory) that are protected even from processes running at higher privilege levels. Intel SGX capabilities are available from a set of instructions introduced in off-the-shelf processors based on the Skylake microarchitecture, starting from the 6^{th} Generation Intel Core family and 5^{th} generation of some Xeon E3 family.

An application of the Intel SGX technology typically requires four main components: (*i*) the availability of the instructions set in the processor, (*ii*) the operating system driver, (*iii*) the software development kit to facilitate the access to the driver from the application code, and (*iv*) Platform Software.

The Enclave Page Cache (**EPC**) is a protected memory used to store enclave pages and SGX structures. The EPC is divided into 4KB chunks called EPC pages. EPC pages can either be valid or invalid. A valid EPC page contains either an enclave page or an SGX structure. Each enclave instance has an enclave control structure, **SECS**. Every valid enclave page in the EPC belongs to exactly one enclave instance. System software is required to map enclave virtual addresses to a valid EPC page.

Before designing a new secure application using SGX, there are some limitations that need to be kept in mind by developers, the main one being the memory limitation. When starting a machine, the SGX capable processor needs to reserve a portion of memory to itself (PRM). Also, the entire EPC must reside inside the PRM. In the current version of SGX, this portion of memory is limited to 128 *MB*. If more space is needed than what is available, a large overhead in processing time is added, due to the need to re-encrypt the data before swapping from EPC to DRAM and re-encrypting the data after swapping from DRAM to EPC. This re-encryption is needed since data in the EPC is cache-line encrypted and in the DRAM it is page-encrypted.

The SGX Linux SDK provided by Intel is only available for C/C++ development. This leaves secure application developers with no choice over what programming language to use when writing an enclave's code.

2.2 SCONE

SCONE [4] (short for Secure CONtainer Environment) is a toolset that allows containerized applications to transparently run inside Intel SGX enclaves. This considerably reduces the learning curve to implement security and integrity guarantees.

SCONE supports many popular programming languages, offers an asynchronous threading model for system calls, and also remote attestation schemes to verify containers' authenticity and securely load secrets into the container environment. As with other SGX application, besides having the applications inside SGX enclaves, it is also necessary to make sure the correct code is running, which is done by the remote attestation.

Applications can run parallel pieces of software, in SCONE this is possible without leaving the enclave. The process of context switching from and to the enclave has proven to be a large overhead to the application performance. SCONE provides application level threading, which enables it to not leave the enclave whenever a thread is blocked (e.g. by a system call), switching to a new application thread during this wait.

Instead of requiring a third party to attest the application before giving it some secret (e.g., certificate, credentials), as with applications based on the Intel SGX, SCONE manages attestation transparently. As the runtime loads an application, the SCONE runtime will attest the application and then embed the secrets into the application memory space as environment variables or command line arguments.

A variety of applications make use of SCONE's advantages of virtualization and secure enhancing such as [3, 12–14, 16].

2.3 Kubernetes

The usage of containers has become the dominant approach to orchestrate applications. Not only a larger number of IT professionals are using containers, but they are also using them in production. Two recent surveys on OpenStack [1] and Kubernetes [7] also confirm the tendency of adopting container-based orchestration. On the one hand, the OpenStack survey reports that 61% of OpenStack users that need container orchestration use Kubernetes on top of OpenStack. The Kubernetes survey reports that 75% or Kubernetes users participating in the survey use it also in production. On the other hand, according to the Kubernetes survey the main challenge is container deployments is security, reported by 43% of the users and followed by challenges in storage (38%), networking (38%), and monitoring (38%).

The basic working unit in Kubernetes is a pod -– an abstraction of a set of containers tightly coupled with some shared resources (the network interface and the storage system). With this abstraction, Kubernetes adds persistence to the deployment of single containers. It is important to note two aspects of a pod: (*i*) a pod is scheduled to execute on one machine, with all containers inside the pod being deployed on the same machine; (*ii*) a pod has a local IP address inside the cluster network, and all containers inside the pod share the same port space. The main implication of this is that two services which listen on the same port by default cannot be deployed inside a pod. A pod can be replicated along several machine for scalability and fault-tolerance purposes

2.4 Prometheus

Prometheus is an open source monitoring system and time series database platform. Prometheus also provides client libraries for numerous languages, PromDash for presentation, Alertmanager for alerting and numerous plugins. The architecture works by having each Prometheus instance independent of each other, and each instance will monitor their own subset of jobs, exporters, or anything that exposes metrics to it. Prometheus uses mainly polling techniques to get metrics, but has support of pushing by the use of an intermediate node named a push gateway. Prometheus also uses a query language called PromQL.

https://01.org/intel-software-guard-extensions

3 USE CASE: SMART GRID PERSISTENCE

3.1 Smart Grids

Smart grids are electrical grids with the ability to control the consumption, distribution and production of electricity. Smart grids rely on monitoring energy consumption on a variety of grid consumers such as households, industries, commerce and other physical locations with access to energy.

There is a limited availability of non-renewable energy sources such as coal, gas, and oil. On the other hand, renewable sources are playing a more important role for future energy supply [17]. Advanced technologies are needed to make these energy supplies more reliable and secure [2].

The main aspects of a smart grid are the following.

(1) Self-healing: the ability to detect and recover faults in the grid;
(2) Consumer empowering: when operating and planning the grid, it should be able to include the customers own equipment and behaviour;
(3) External threats tolerance: the grid should be able to detect and avoid cybernetic and physical attacks;
(4) Energy quality: the standards require certain energy quality, this needs to be provided;
(5) Variety in sources and demands: with no additional effort the grid needs to be able to integrate many sources of energy in different scales;
(6) Environmental impact reduction: using green energy sources and avoiding waste.

There are three main innovation areas to provide these characteristics in a power grid: digital control and automation in the grid, smart metering, integration with various energy sources and storage [11].

Smart meters record the measured consumption over a given measurement interval and transfer recorded values individually, or in block, in a transfer interval. In this paper we focus on the smart metering aspects of a smart grid which is an essential part in its construction. It enables the consumer to have information such as consumption and price signaling.

In our use case we consider a smart grid consisted of different consumers with their own smart meters regularly collecting energy consumption data from the physical locations they are installed in. The flow is described as illustrated in Figure 1:

(1) The clients or consumers are operated by an Energy Distribution Company. These physical locations have smart meters installed to collect their consumption information. This information can then be queried by the Energy Distribution Company or pushed by the smart meters. This information is then stored locally by the Energy Distribution Company.
(2) After the data is accumulated by the Energy Distribution Centers it is then sent via the internet to the Electronic Client Meter, using state-of-the-art communication protocols such as HTTP/{1,2}, REST protocol, SOAP protocol, etc. After the requests are received, they can be processed and stored safely in reliable storages such as Ceph.

https://ceph.io/

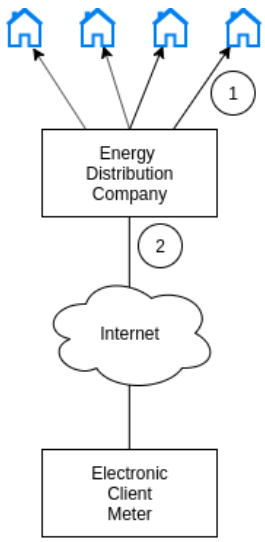

Fig. 1. Smart meters data collection architecture

3.2 Security Threats in Smart Grids

Data collected from smart meters are considered sensitive. Those who have access to the data are able to draw conclusions about the behaviour of energy consumers. We see that many organizations in Europe and across the globe are identifying risks and vulnerabilities in the private data that were not considered previously in the energy industry [15].

Some actions were taken to minimize the risk of leaks and attacks such as: limit the information push interval in 15 minutes to allow information to be used in energy saving actions. But even with the 15 minutes interval it was possible to infer holiday periods [9], electronics and appliances usage patterns [10] and religious practices [6] from the data retrieved from smart meters.

4 METHODOLOGY

To provide the necessary services for smart meters to safely persist consumers information we propose the Electronic Client Meter platform, a platform for clients to publish their metrics on reliable storages using encryption to secure private data.

The other main component is the monitoring platform, responsible to enable a view of the system state and infrastructure. Using the tools in the ECM, the monitoring platform is able to publish and save metrics over to later be compared and used by autoscalers and controllers.

4.1 Architecture

The life cycle of a server is changed as requests arrive from the clients (depicted in Figure 3). Each server serves one client during a cycle providing isolation and safely making operations in data designated to the client logged in.

The client has the information to be persisted in the ECM and it makes several HTTP requests in order to do that. The entry point for all clients is the same: a load balancer that redirects packages

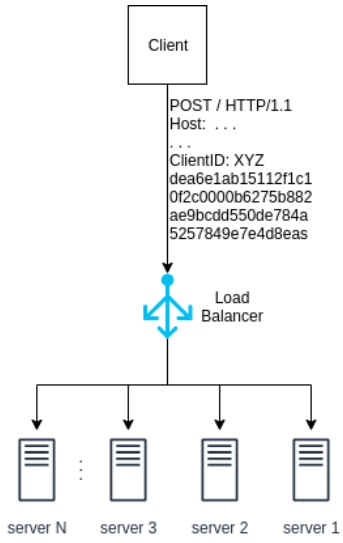

Fig. 2. Electronic Client Meter architecture

based on the parameters from the header information of the request received. It looks for the ClientID field and forwards it to the proper server. If this client ID has already logged in, it uses the same connection to the same server to forward the package, if not it uses one of the servers in the READY state.

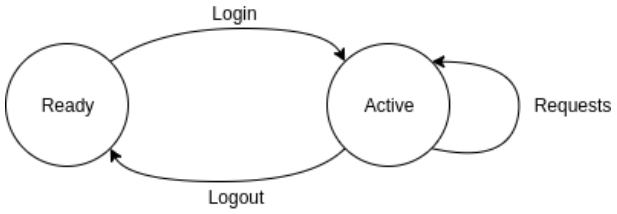

Fig. 3. Server life cycle and operations

Initially all servers are in the READY state, waiting for a client to login. After a log in request is forwarded to this server by the load balancer, the server then begins the process of unsealing the consumer data. This process requires a key exchange with the client, after that it decrypts the loaded client data with the key it received from the client, finishing the unseal process. The state is now ACTIVE as login succeeds.

When a server is in ACTIVE state, all requests made from this specific client ID is forwarded to the same server and all operations are made in the proper database. It now can handle any kind of request from the client.

After the client finishes all operations for the session, it then sends a logout request, which returns the server to READY state and the data is encrypted and unmounted.

4.2 Monitoring Platform

The scalability mechanisms are based on information retrieved from the infrastructure and from the framework components, as depicted in Figure 4. To aggregate all the needed metrics we use Prometheus and set the targets to be all the exporters we wish to collect metrics from.

Prometheus uses exporters to collect metrics and saves them in a time series database. In order to customize the metrics we want to collect from the framework we have to use custom exporters so that the information from the components can then be scraped and persisted by Prometheus. The first one is the SGX exporter, it collects information from the Linux drivers and exports them through a REST API. The main metrics exported are the following:

- Total of EPC pages;
- Number of free pages;
- Number of initialized epc enclaves;
- Number of evicted/loaded epc pages.

In order to save the mentioned metrics, the Linux SGX driver was changed to add module parameters and make arithmetic operations on them as the driver is used. The following counters were added:

- sgx_nr_total_epc_pages;
- sgx_nr_free_pages;
- sgx_nr_evicted;
- sgx_nr_alloc_pages;
- sgx_nr_enclaves;
- sgx_nr_reclaimed;
- sgx_nr_added_pages;
- sgx_init_enclaves;
- sgx_loaded_back.

Each of these metrics are added to a file in the *sys* filesystem, where the exporter collects and exposes via the Prometheus exporter REST API.

To collect node metrics such as memory usage, CPU time, I/O operations, network bandwidth, and traffic, we use Node Exporter version 0.18. We also use Kubernetes standard exporters to collect metrics from Kubelet, Docker, and overlay networking information.

eBPF is part of the Linux that allows external users to define small programs defined to run in a virtual environment. The eBPF programs allow users to define trace code. Tracing refers to performance analysis and observability tools that can produce per-event info. We use `bpftrace` to calculate the total number of page faults, context switches, exit and enter system calls, context switches and cache status.

5 SYSTEM EVALUATION

We use Grafana to visualize the new metrics and new values added for each component. The monitoring platform provide a better understanding of the environment. Thus, it can be used when making scheduling and scaling decisions resulting in a better resource utilization and use of hardware resources specially when load balancing new clients or horizontally/vertically scaling the cluster, server number or other infrastructure pattern during peak load.

`/sys/module/isgx/parameters/`

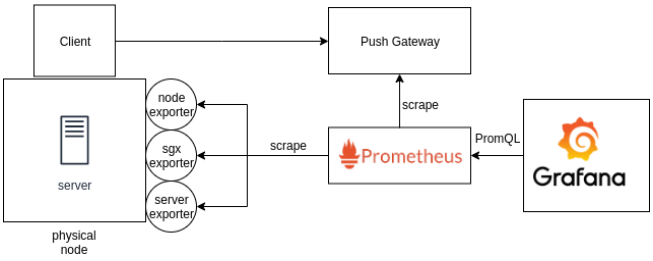

Fig. 4. Monitoring framework components

Fig. 5. Servers and states dashboard

EPC availability is important since each node has a set of servers scheduled to it. This can be seen in dashboard 5. It is possible to know how many are currently assigned to a client, ready to be assigned or stopped due to some reason. At peak load, this information can be used to provide availability for new clients as every server is isolated for one specific client during its life time. Also, to identify load patterns and see points in time where the clients increased or decreased, it is possible to see the historical number of assigned servers, as depicted in Figure 6.

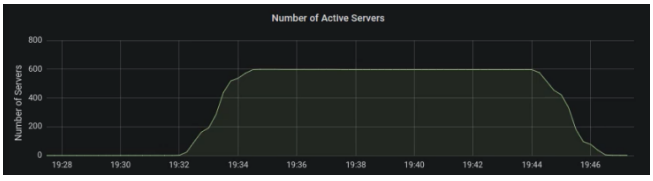

Fig. 6. Active servers over time graph

It is also useful to have application level metrics instead of infrastructure ones. This is made possible by using the server exporter that has server information based on logging, files and traffic from the port the server is listening to. The most common metric used by scalability controllers is the server response time by operation. Nevertheless, additional metrics that reflect the QoS for the specific application can also be used. The server exports the time taken to respond to the query received and Prometheus is able to save this metric via the exporter. These application metrics are the direct result from other infrastructure information, as when the nodes are stressed or the enclave page cache is full and the driver has to do pagination the response time decreases. As depicted in Figure 8, the response time QoS can be shown using the metrics collected in the servers via the exporter.

The dashboard in Figure 9 shows client level metrics. To collect these metrics we make use of Prometheus application PushGateway, which provides an exporter api so that Prometheus can scrape its metrics, but also provides an HTTP API so that authenticated external entities can push, using PromQL, non-constant metrics or limited life cycle targets. In this case the clients have limited life cycle, as they log in, complete all necessary operations and then log out. This is commonly used by production systems where client push reports of errors, operations, bugs and other information considered important during app development. The downside is that the Push Gateway endpoint has to be exposed so that client can push their metrics to it. We can see that the clients push the operations made by them and this information is shown in the dashboard.

6 CONCLUSION

With the work developed, we were able to add a rich monitoring capability to a large and complex system. This monitoring included all interactions between clients and servers as well as the whole infrastructure resource usage. This shows that the robust monitoring platform is able to save metrics from all points in the smart metering data persistence framework.

Results show that the average response times across all servers were high in the beginning due to the mapping and mounting of encrypted databases, after the servers were successfully logged in and went into ACTIVE state, satisfying QoS goals. When the life cycle comes to an end, we see that the still pending clients have higher response time because the earlier created ones were unmounting their databases and logging out.

The monitoring platform enables the replication and scalability setups or decision to be taken based on all sorts of metrics. This decision can be taken based on CPU usage for each node, memory usage or disk for IO intensive systems. All of these metrics are available via deployed node exporters or through the Kubernetes framework. Our main contribution in this work was adding the above mentioned metrics so that this decision may include security resources and combining all this exporters and dashboards into a simplified instalation process with the rest of the system. With this dashboard in hand, operators can quickly evaluate the application QoS and resource metrics, matching the resources to align with the QoS requirements.

ACKNOWLEDGMENTS

I thank God in his absolute majesty for making this possible.

To my wife Thirza for being so caring and loving in good times and in bad, in happiness as well as in sadness.

To my mother Maria for the unconditional support and my father Jeremias for being my reference until this day.

To the friends I gathered in the last three years of professional and personal growth: Clenimar, Lucas, Sergei, Oleh, Karin, Lília, Javan, Marcus, Fábio, Icaro, Joab, Whasley, Sergio and Marta.

To all my professors, but in special: Andrey, Fubica, Manel, Nazareno, Joseana, Francisco Neto, Matheus, Massoni, Reinaldo, João, Livia and Raquel.

To my advisors Andrey Brito and Christof Fetzer for their patience, wisdom and all the concerns and happiness we shared in the past years of advising.

This work considered results from the EU-Brazil SecureCloud and Atmosphere projects, funded by the European Commission and MCTIC/RNP/CTIC.

Fig. 7. SGX information dashboard

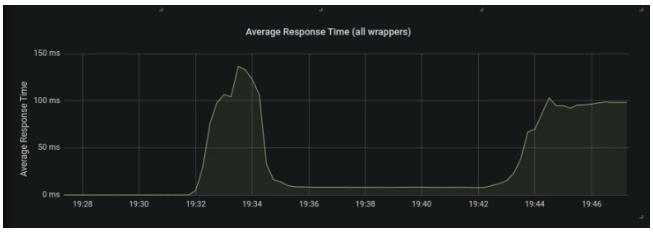

Fig. 8. Servers response time dashboard

REFERENCES

[1] [n. d.]. Openstack User Survery 2018. https://www.openstack.org/user-survey/2018-user-survey-report/. [Online; Last access: December 21st, 2018].

[2] D. Abbott. 2010. Keeping the Energy Debate Clean: How Do We Supply the World's Energy Needs? *Proc. IEEE* 98, 1 (Jan 2010), 42–66. https://doi.org/10.1109/JPROC.2009.2035162

[3] S. Arnautov, P. Felber, C. Fetzer, and B. Trach. 2017. FFQ: A Fast Single-Producer/Multiple-Consumer Concurrent FIFO Queue. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 907–916. https://doi.org/10.1109/IPDPS.2017.41

[4] Gregor F. Arnautov S., Trach B. 2016. SCONE: Secure Linux Containers with Intel SGX. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association.

[5] Alvaro Cardenas and Reihaneh Safavi-Naini. 2012. *Security and Privacy in the Smart Grid*. 637–654. https://doi.org/10.1016/B978-0-12-415815-3.00025-X

[6] S Cleemput. 2018. Secure and Privacy-friendly Smart Electricity Metering. $$Uhttps://lirias.kuleuven.be/retrieve/509996$$DPhd_CleemputSara_May25.pdf[freelyavailable]

[7] Cloud Native Computing Foundation. 2017. Cloud Native Technologies Are Scaling Production Applications. https://www.cncf.io/blog/2017/12/06/cloud-native-technologies-scaling-production-applications/. [Online; Published: December 6th, 2017; Last access: December 26th, 2017].

[8] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. *IACR Cryptology ePrint Archive* 2016, 086 (2016), 1–118.

[9] Günther Eibl., Sebastian Burkhart., and Dominik Engel. 2018. Unsupervised Holiday Detection from Low-resolution Smart Metering Data. In *Proceedings of the 4th International Conference on Information Systems Security and Privacy - Volume 1: ICISSP,*. INSTICC, SciTePress, 477–486. https://doi.org/10.5220/0006719704770486

[10] G. Eibl and D. Engel. 2015. Influence of Data Granularity on Smart Meter Privacy. *IEEE Transactions on Smart Grid* 6, 2 (March 2015), 930–939. https://doi.org/10.1109/TSG.2014.2376613

[11] Djalma M Falcão. 2009. Smart grids e microredes: o futuro já é presente. *Simpósio de Automação de Sistemas Elétricos* 8 (2009).

[12] C. Fetzer. 2016. Building Critical Applications Using Microservices. *IEEE Security Privacy* 14, 6 (Nov 2016), 86–89. https://doi.org/10.1109/MSP.2016.129

[13] Robert Krahn, Bohdan Trach, Anjo Vahldiek-Oberwagner, Thomas Knauth, Pramod Bhatotia, and Christof Fetzer. 2018. Pesos: Policy Enhanced Secure Object Store. In *Proceedings of the Thirteenth EuroSys Conference (EuroSys '18)*. ACM, New York, NY, USA, Article 25, 17 pages. https://doi.org/10.1145/3190508.3190518

[14] Do Le Quoc, Franz Gregor, Jatinder Singh, and Christof Fetzer. 2019. SGX-PySpark: Secure Distributed Data Analytics. In *The World Wide Web Conference (WWW '19)*. ACM, New York, NY, USA, 3564–3563. https://doi.org/10.1145/3308558.3314129

[15] Robert Rieman. 2019. TechDispatch 2: Smart Meters in Smart Homes. https://www.edps.europa.eu/data-protection/our-work/publications/techdispatch/techdispatch-2-smart-meters-smart-homes_en

[16] Bohdan Trach, Alfred Krohmer, Franz Gregor, Sergei Arnautov, Pramod Bhatotia, and Christof Fetzer. 2018. ShieldBox: Secure Middleboxes Using Shielded Execution. In *Proceedings of the Symposium on SDN Research (SOSR '18)*. ACM, New York, NY, USA, Article 2, 14 pages. https://doi.org/10.1145/3185467.3185469

[17] X. Yu, C. Cecati, T. Dillon, and M. G. Simões. 2011. The New Frontier of Smart Grids. *IEEE Industrial Electronics Magazine* 5, 3 (Sep. 2011), 49–63. https://doi.org/10.1109/MIE.2011.942176

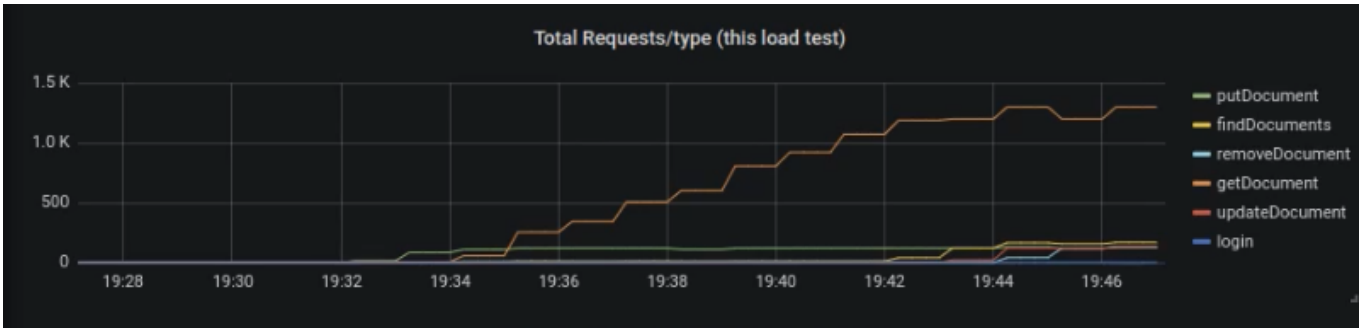

Fig. 9. Client requests dashboard