Authentication and Authorization in Cyber-Security Frameworks: a Novel Approach for Securing Digital Service Chains

Giovanni Grieco*‡, Domenico Striccoli*‡, Giuseppe Piro*‡,
Raffaele Bolla†‡, Gennaro Boggia*‡, Luigi Alfredo Grieco*‡
* *Department of Electrical and Information Engineering (DEI)*, Politecnico di Bari, Bari, Italy
Email: *name.surname*@poliba.it
† *Department of Naval, Electrical, Electronical and Telecommunications Engineering (DITEN)*,
Università di Genova, Genova, Italy
Email: *name.surname*@unige.it
‡ *Consorzio Nazionale Interunivesitario per le Telecomunicazioni (CNIT)*

***Abstract*—Digital services and digital service chains are the heart beating of the modern economy. Their composition involves several players, i.e., processes, software, devices, and many kinds of data exchanged among them. In such a scenario, it is important to guarantee data confidentiality, integrity, as well as authentication and authorization procedures between the communicating parties of a service chain. Cyber-security frameworks are explicitly designed for this purpose. They rely on the integration of different software modules, mutually interfaced to accomplish complex security tasks. Nevertheless, it is important to guarantee a high level of protection during data exchange among the modules. Currently, standardized authentication and authorization mechanisms are implemented through proprietary "As-a-Service" products, but the deployment of a mature on-premise solution is still missing. To bridge this gap, this contribution proposes an authentication and authorization module that automatically protects the information flowing among the modules of cyber-security frameworks. It guarantees resource availability only to authenticated subjects. Thus, their operations are confined in what actions they are authorized for. The proposed module has been implemented and tested in a real cyber-security framework under development into the H2020 GUARD project. Experimental tests show that the proposed module enables authentication and authorization procedure delegation among GUARD modules, which eases their implementation, while maximizing the flexibility of the set of access control policies and an efficient protection of the services.**

I. Introduction

Future economy will be driven by digital service chains. They are progressively changing the communication paradigms by creating, processing, sharing, and consuming data and contents in a digital continuum. Service composition translates into chaining several processes, software, and devices, anytime and anywhere, feeding them with relevant users data [1]. As a consequence, frontiers between different application domains are blurring, due to dynamic service deployments and withdrawals with unprecedented agility. This process is supported by the adoption and integration of existing software paradigms, e.g., cloud computing, software-defined networking, and Internet of Things (IoT) [2].

In such a scenario, data protection is a very important issue to be addressed. From one side, sensitive data need to be secured from unauthorized access. From the other, attacks and threats should be timely detected and countermeasures taken accordingly [3].

To this end, the Cyber-Security Framework (CSF) proposed in [2] can be a viable solution. It is currently under development into the H2020 GUARD project. Its aim is to collect security-related events and measurements from dynamic and evolving ICT systems and infrastructures. Data are collected and categorized by distributed third-party software agents. The power of such a CSF resides in the integration of different modules, external and internal to the platform, accomplishing heterogeneous and articulated tasks through several logical interconnections [1], [2], [4]. As a consequence, there is the need to protect information about service features, users data and exchanged information from unauthorized access, so to guarantee a high level of security among multiple domains and protocols.

With the goal to meet these requirements, this work proposes an Authentication (AuthN) and Authorization (AuthZ) module that protects data flowing among module interfaces of the CSF. It guarantees resource availability only to authenticated subjects and allows only the specific actions they are authorized for. The proposed module has been implemented in the context of the GUARD CSF developed in [5], to enable robust AuthN and AuthZ procedures while minimizing information exposure to unauthenticated and/or rogue services. In this way all CSF components delegate their security mechanisms for inter-service communications to the proposed mod-

This work was framed in the context of the GUARD project, which receives funding from the European Union's Horizon 2020 research and innovation program under grant agreement 833456. It was also supported by the PRIN project no. 2017NS9FEY funded by the Italian MIUR; it has been also partially supported by the Italian MIUR PON projects Pico&Pro (ARS01_01061), AGREED (ARS01_00254), FURTHER (ARS01_01283) and RAFAEL (ARS01_00305).

ule, consequently facilitating the overall CSF implementation through a dedicated and uniform security system that purely focuses on authentication and fine-grained authorizations. This is actually a challenge in complex CSFs, characterized by a vast variety of modules, provided by different parties, and operators that exchange heterogeneous and sensitive data.

The remainder of this paper is organized as follows. Section II summarizes the main features of the GUARD architecture. The security mechanism is presented in Section III and implemented in GUARD as described in Section IV. Performances of the proposed module are evaluated in Section V. Finally, Section VI concludes the work.

II. The Reference Architecture

The GUARD platform can be seen as a collection of security services with the aim to analyze and protect data to provide situational awareness and counteract malicious patterns. Each service provides a specialized capability to manage the acquired data, based on particular criteria. The architecture can be generally subdivided into the following macro-blocks, illustrated in Fig. 1:

- Security Services in the centralized platform, that focus on data analysis.
- Local Agents, external to the main core architecture, that extend GUARD capabilities by means of additional specialized security functions. Here the Context Broker, internal to the Core Platform, is vital in providing the status of the internal Security Services and managing Local Agent communications.
- A message broker, i.e. Kafka, for inter-service data exchange. Kafka is a distributed messaging system based on the publish/subscribe model that stores information in a distributed commit log [6].
- A Security Dashboard, to orchestrate and manage policies and security pipelines, i.e., the connections between a data source and a set of services.

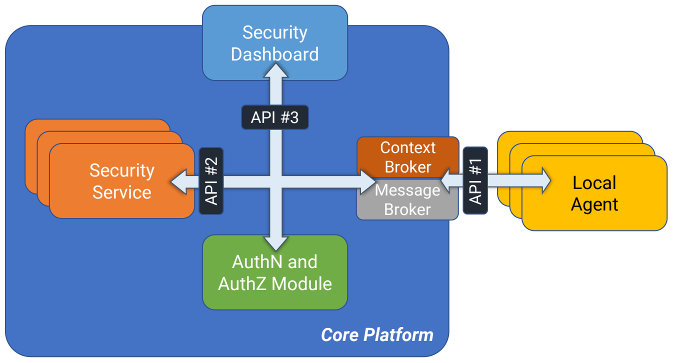

Fig. 1. Macroscopic overview of the GUARD Architecture.

The macro-blocks communicate via three Application Programming Interfaces (APIs), each one with its own role:

- API #1 for collecting data, characterized by the communication of messages structured in a common data format using a message broker. Moreover, this API provides capability control by means of a REpresentational State Transfer (REST) interface exposed by Local Agents.
- API #2 for Create, Read, Update, and Delete (CRUD) operations on data models available on the platform by means of a common REST interface.
- API #3 for delivering notification messages. Furthermore, this API provides the configuration of security services through REST and Remote Procedure Call (RPC) interfaces.

III. The Proposed Security Mechanism

The goal of the proposed AuthN/AuthZ module is to provide protection to the interfaces presented in Section II, while taking into account their differences in order to maximize the flexibility of access control policies to be encoded in an identity. A service identity can be used by the communicating services for sender validation and apply access policies based on Attribute-Based Access Control (ABAC) [7]. This access control method has been introduced by the National Institute of Standards and Technologies (NIST). It is based on granting or denying user requests based on both attributes of users and attributes of the object to be accessed to. Moreover, environmental conditions may be taken into account to further refine access policies. This access control model offers a high degree of flexibility in heterogeneous and complex architectures like GUARD [8]. In this scenario, it is necessary to issue identities from a centralized security platform apt to configure identities with all the security features in place.

To this end, the Identity Provider (IdP) authenticates services with secure identities and exposes public keys to verify them. Therefore, it is a middleware and a trusted authority between GUARD services and Security Operators, as illustrated in Fig. 2. This subsystem has a logical core that manages identities for software entities, namely Identity and Attribute Manager (IAM), and stores them in a persistent storage location, called the Identity Database. An identity is a complex object with a dictionary of attributes and a security signature to protect it from forging, replay, and tampering attacks. It is defined and maintained by Security Operators through the Administration Interface. Identities can be retrieved by services through the AuthN Endpoint by means of a secret, which is generated upon service registration on the IdP. The identity is released in the form of a limited-lifespan security token that contains related attributes and a Message Authentication Code (MAC) to preserve its integrity. Once services have been correctly identified, they can communicate with other services across the platform. On the other hand, services, that receive requests or data, validate the sender by its identity signature using the IdP public keys available from the Attestation Endpoint. These keys are used to decrypt the identity MAC and verify its digest.

Fig. 3 proposes a three-actor communication model commonly found in cloud environments, where a service sends data to a message broker, or bus, which then delivers them to other services. Connections between services and message bus

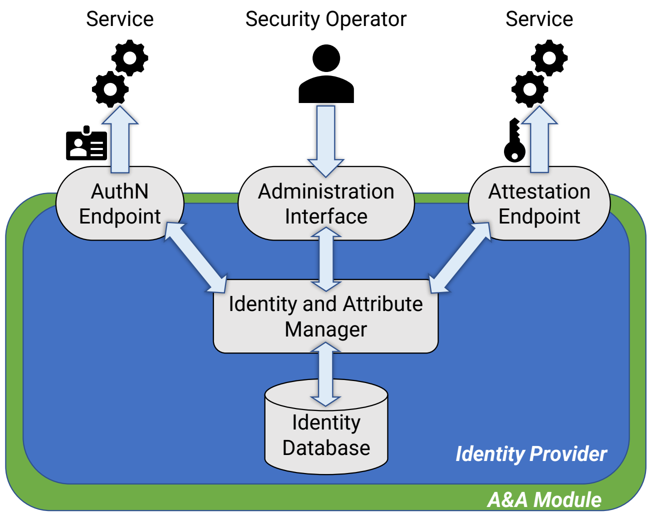

Fig. 2. A close lookup of IdP software components with their interfaces and expected communicating actors.

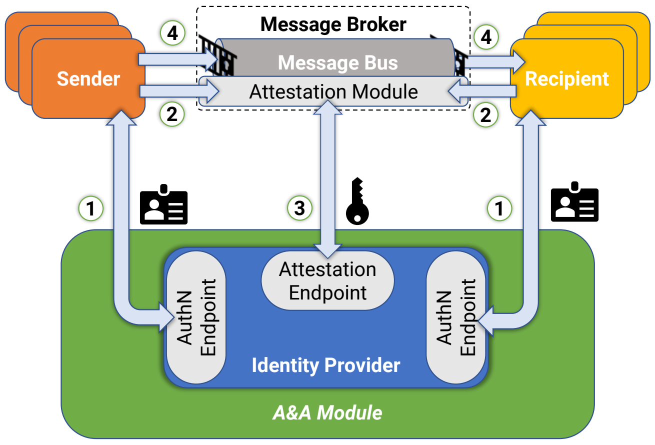

Fig. 3. Communications guarded by the proposed security mechanism.

are protected by the same security mechanism to ensure that the data flow is protected, thus minimizing the risk of their leakage. The procedure is composed of the following steps:

1) Sender/Recipient retrieves its identity by authenticating with the IdP.
2) Sender and recipient connect to the message bus and send their identity during the initial connection phase (i.e., their handshake).
3) The attestation module validates the identity and performs AuthZ procedures to protect the message bus.
4) Sender/Recipient, if correctly authenticated and authorized, can send/receive messages.

For this particular communication model, the attributes, included in identities, declare the type of messages that are delivered in the message bus, like topic names. This means that senders and recipients must respect the declarations reported on the identity to ensure an authorized data exchange through the message bus. Therefore, once they are authorized, they cannot divert their behavior and publish messages on other channels of the bus. Consequently, service interference is minimized, thus preserving the state of the message bus and their channels.

IV. Module Implementation

The experimental scenario has been developed as a standalone test bed. It has been organized in microservices using Docker, which helps building small software units in a container-based environment [9].

The security mechanism hereby implemented focuses on the initial handshake procedure between services. Their AuthN has been integrated with the proposed security mechanism with the support of the IdP. Moreover, AuthZ operations have been tightly integrated to each service to provide flexibility over adopted policies for each particular operation on a specific service.

The experimental scenario is a composition of containers, enabling the evaluation of the proposal with a set of diverse service configurations:

- WSO2 Identity Server (IS), that acts as the IdP of reference, configured for the proposed security mechanism [10].
- An Apache Kafka broker [6] equipped with the Attestation Module to provide secure communications.
- A Java-based Kafka client that uses its identity to authenticate with Kafka and publish messages in a specific topic.
- A Java-based Kafka client that consumes messages from a determined topic.

The IdP, exposed by the AuthN/AuthZ module, is based on the open source IS project developed by WSO2 [10]. It is an extensible Identity Management platform that provides secure and standardized AuthN mechanisms, identity federation, access control procedures, and APIs to automate its administration. This IdP has been identified as the ideal one to provide the basic software infrastructure for the AuthN/AuthZ module. In fact, it allows security operators to register, manage and dispose identities and attributes via a Hyper-Text Transport Protocol (HTTP)-based Administration Interface.

A service to protect must be registered on the IS as a Service Provider. In order to do so, a security operator has to configure the IS by providing essential information to recognize the service. Service Provider registration consists of a two-steps procedure:

1) Register the new Service Provider, also known as the application to protect. This operation consists in providing a service name and an optional description. At the end of the registration, the IS provides a unique identifier to refer to such service.
2) Register a new OAuth2 Inbound Authenticator associated with the Service Provider. It exposes the necessary software infrastructure of the IS for AuthN procedures that follow a specified protocol, OAuth2 in this case [10].

This procedure can be entirely automated by interfacing with the exposed IS APIs. Specifically, in order to register a new Service Provider programmatically, the procedure must send a request to the IS. It is composed of a payload, also known as Simple Object Access Protocol (SOAP) Envelope, and a header that contains the HTTP AuthZ field to authenticate with the IS. The SOAP Envelope is an XML-formatted HTTP body containing name and description of the Service Provider and its OAuth2 Inbound Authenticator to activate.

In this test bed, Kafka has been protected in order to evaluate the authenticity of each connected service and their authorizations. This scenario has the goal of protecting the message bus in case of service misuse, minimizing the potential interference of infected services to other communications if they acquire rights to publish/subscribe to other topics.

To this end, the test bed comprises a Kafka broker and service examples that produce and consume messages. The goal of this test is to demonstrate the possibility of integrating the security mechanism with Kafka. This broker already provides high-level software interfaces for AuthN via Simple Authentication and Security Layer (SASL) and OAuth2 [6], but it is limited to the validation of unsecured JSON Web Tokens (JWTs), i.e., it does not check the MAC, and it does not support external IdPs like WSO2 IS. To overcome this limitation, this test bed includes a library that provides all the necessary implementations to enable security checks between Kafka and the IS by following SASL and OAuth2 specifications. It also decodes and validates JWTs through public keys available from the IdP JSON Web Key Sets (JWKS) endpoint. Furthermore, as the local AuthZ mechanism, this extension mandates that each service declares a set of topics that it will use during execution through additional attributes encoded in the JWT. This is a confinement strategy to preserve application rights and limit what the service can do at runtime. Services must be compatible with this new procedure to be authorized by the broker, otherwise the broker will abruptly close the connection. The broker validates the identity and elaborates an AuthZ response to send back to the sender. After that, the traditional Kafka session can begin by following its reference protocol.

V. Performance evaluation

The implementation of the AuthN/AuthZ module described in Section IV has been realized in the GUARD platform, to test the communication functionalities and to validate the proposal. In this context, the performance of the module has been evaluated by checking the authentication procedures and access control mechanisms between GUARD services. Different performance indicators have been chosen in terms of response time (e.g., elapsed time for service authentication and authorization and latency during message-based communications with Kafka) and resource usage (e.g., CPU Load and Memory Usage) during the authentication and authorization processes.

The related measurements, made during the establishment of the handshake procedure between the service and Kafka to send/retrieve messages, have been carried out with and without the integration of the proposed security mechanism, to measure the time and resource overhead taken by the AuthN/AuthZ module.

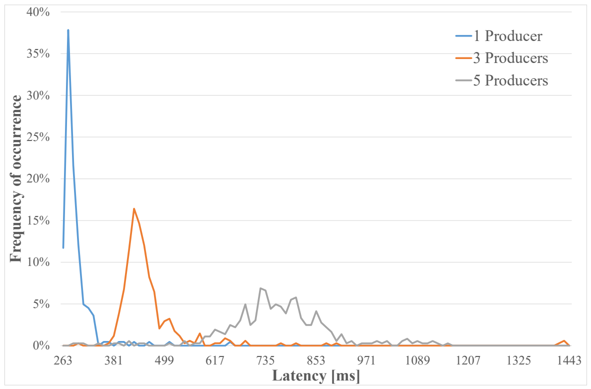

Fig. 4. Latency distribution for service authentication to Kafka, for different numbers of producers.

Simulations have been set up in a Docker environment with 8 virtual processors, 4 GB of memory, and 4 GB of swap memory on Solid State Drive. It has been configured with (i) an IdP that constitutes the AuthN/AuthZ module, (ii) a Kafka broker with the AuthN/AuthZ module extension to provide authentication and authorization functionalities, (iii) a number of Kafka clients, called *producers*, that publish data on a fixed reference topic, and (iv) one Kafka client, called *consumer*, that subscribes to the same reference topic of the producers.

Each simulation run lasts 10 minutes. During this time period, the producers publish messages in bursts of 100 messages each. After each burst, they are restarted and the process is repeated, until the end of the run. This generates a huge number of samples (about 1.1e5) for a robust analysis of the Key Performance Indicators evaluated in the following subsections.

A. *Elapsed time for service authentication and authorization*

Fig. 4 depicts the latency distribution for a number of concurrent producers (1, 3, and 5) that conclude successfully the authentication procedure to Kafka. Table I reports the mean (μ) and the variance (σ^2) of the latency obtained for service authentication and for a different number of producers.

TABLE I
Mean and variance of authentication latency for different numbers of producers.

Number of Concurrent Producers	μ $[ms]$	σ^2 $[ms^2]$
1	296.75	1541.11
3	471.23	12547.07
5	768.88	15539

As depicted in Table I, both the latency mean value and the dispersion around it increase with the number of producers. This behavior is confirmed by the curves depicted in Fig. 4, where the latency distribution is shifted towards higher values as the number of producers increases, being also sparser around the mean value.

The growth of the mean latency and the variance both derive from the complexity of the OAuth protocol used to authenticate services. The more concurrent authentication procedures are performed, the more resources are needed by the centralized IdP to accomplish those procedures simultaneously.

Regarding service authorization, Kafka has been stressed with a number of concurrent producers publishing messages on the same reference topic, and with one consumer subscribing to that topic. The role of the AuthN/AuthZ module is to check the correct authorization of each client during its publish/subscribe operations. Table II shows the mean and the variance results of the latency values obtained for service authorization.

TABLE II
MEAN AND VARIANCE OF AUTHORIZATION LATENCY FOR DIFFERENT NUMBERS OF PRODUCERS.

Number of Concurrent Producers	μ [ms]	σ^2 [ms^2]
1	0.146	0.2227
3	6.013	11211.5
5	9.162	20472

As it can be seen from Table II, authorization latency for 1 producer is very small, but it incurs in a performance penalty of 97.6% when the number of producers increases from 1 to 3, whereas the performance penalty is of 98.4% when the number of producers increases from 1 to 5. The dispersion of the measured latency around the mean highly increases for increasing numbers of producers. This is due to the producers concurrently requesting authorization to publish new data.

B. Latency overhead in message reception

To evaluate the latency overhead due to the AuthN/AuthZ module, latency results have been compared with those obtained without the integration of the module. In the latter case, the only Transport Level Security (TLS) protocol has been implemented for a secure communication with Kafka. To derive the frequency of occurrence of latency overhead, the range of the experimented latency samples has been subdivided into 2000 bins, of size 1.2 ms each. In Fig. 5, only the first 12 bins, up to 14.5 ms of latency, have been plotted. In fact, even if also higher latency values have been observed, their number is negligible and their sparsity is high. Therefore, they have not been represented in the figure for the sake of better clarity.

Given the same number of producers, the performance behavior with and without the AuthN/AuthZ module are very similar to each other. Additionally, the latency distribution, in case of the adoption of the AuthN/AuthZ module, is sparser in the range depicted in Fig. 5.

To provide a more complete analysis, Table III shows mean and variance of latency values for all the numbers of producers, considering all the results collected in the experiments.

If the AuthN/AuthZ module is adopted, the average value of the latency keeps around 5 ms, almost independently of the

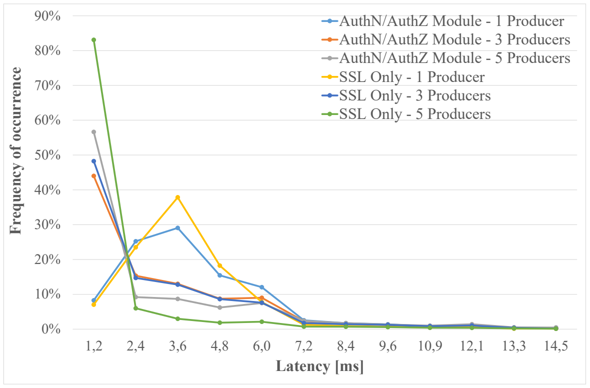

Fig. 5. Latency distribution with and without the AuthN/AuthZ module, for different numbers of producers.

TABLE III
MEAN AND VARIANCE OF LATENCY RESULTS WITH AND WITHOUT THE AUTHN/AUTHZ MODULE, FOR DIFFERENT NUMBERS OF PRODUCERS.

Number of concurrent producers	μ [ms]		σ [ms^2]	
	With the proposed module	Without the proposed module	With the proposed module	Without the proposed module
1	5.286	11.727	544.744	6769.822
3	5.811	5.738	1875.025	2238.178
5	5.332	2.476	1550.494	2132.536

number of producers. Conversely, it progressively decreases for an increasing number of producers if the module is absent. This behavior is mainly due to cache optimization of IS and Kafka module, which penalizes the use of a small number of authentication procedures (a more detailed and accurate steady-state analysis can be done in the future to evaluate the effective scalability of the mechanism). By comparing the variance results, it can be argued that if the AuthN/AuthZ module is present, the variability around the mean latency value is smaller, if compared to the case of the only TLS adoption. This behavior is mainly due to the AuthN and AuthZ procedures between the module and Kafka.

C. Resource consumption

To test the impact of the module implementation on the utilized resources, Kafka has been tested with and without the integration of the AuthN/AuthZ module, evaluating the temporal evolution of the CPU and memory load for 10 minutes of simulation. Fig. 6 and 7 depict the related results.

From Fig. 6, it can be noticed that after a start-up phase, which includes the procedure to authenticate Kafka as a trusted Service Provider in GUARD, the evolution of the CPU load is similar in the two scenarios with (blue line) and without (yellow line) the AuthN/AuthZ module. On average, Kafka with the AuthN/AuthZ module integration requires a CPU Load of 29%, whereas the base version of it requires a CPU

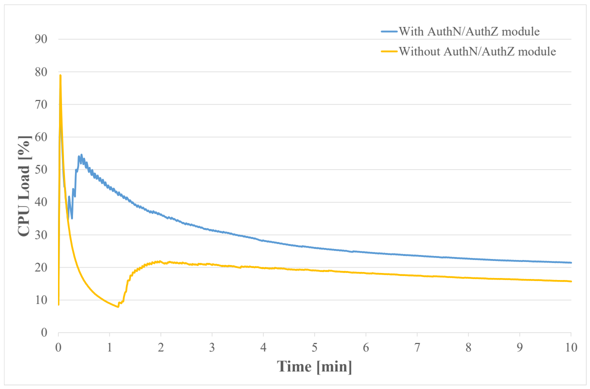

Fig. 6. Percentage of CPU load used by Kafka with and without the AuthN/AuthZ module integration.

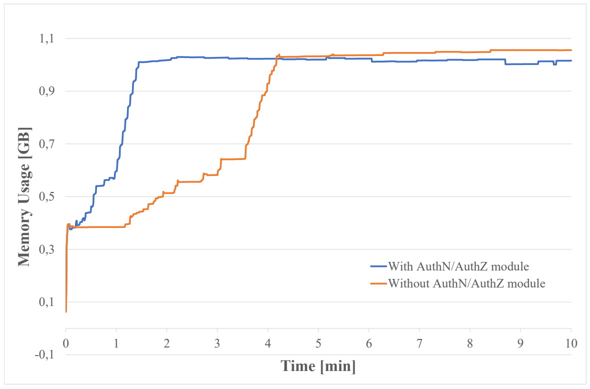

Fig. 7. Memory usage of Kafka with and without the AuthN/AuthZ module integration.

Load of 19%. So, the overhead in CPU Load introduced by the AuthN/AuthZ module is of about 36%.

Fig. 7 shows that Kafka integrating the AuthN/AuthZ module has an average memory usage of 956 MB, whereas 834 MB are required, on average, without the AuthN/AuthZ module integration. To this end, the introduced overhead is of 13%.

VI. Conclusions

In this work, an ABAC-based security mechanism has been presented to protect inter-service communications in the GUARD Cyber-Security Framework. It has been implemented as a module that provides authentication procedures and local authorization policies at each service by relying on a trusted Identity Provider managed by Security Operators. The proposed module has been successfully implemented in the GUARD environment by using a set of services and a Kafka broker for publish/subscribe messaging.

Experimental results show that latency performances of the proposed module are very close to those obtained without its integration, even if this comes at a cost of an increased resource consumption in terms of CPU and memory. Nevertheless, the proposed module brings a much higher degree of flexibility in authentication and authorization procedures, thanks to the ABAC paradigm; this is a great value added in a complex platform with many heterogeneous modules and stakeholders.

References

[1] M. Repetto, A. Carrega, and R. Rapuzzi, "An architecture to manage security operations for digital service chains," *Future Generation Computer Systems*, vol. 115, pp. 251–266, 2021. DOI: 10.1016/j.future.2020.08.044.

[2] M. Repetto, A. Carrega, and A. Duzha, "A novel cyber-security framework leveraging programmable capabilities in digital services," in *Proc. Fourth Italian Conference on Cyber Security*, ser. CEUR Workshop Proceedings, 2020, pp. 201–211.

[3] M. Panjwani and M. Jäntti, "Data protection security challenges in digital it services: A case study," in *2017 International Conference on Computer and Applications (ICCA)*, 2017, pp. 379–383. DOI: 10.1109/COMAPP.2017.8079790.

[4] M. Repetto, A. Carrega, and G. Lamanna, "An architecture to manage security services for cloud applications," in *2019 4th International Conference on Computing, Communications and Security (ICCCS)*, 2019, pp. 1–8. DOI: 10.1109/CCCS.2019.8888061.

[5] *GUARD - A cyber-security framework to GUArantee Reliability and trust for Digital service chains*, https://guard-project.eu/.

[6] *Apache Kafka*, https://kafka.apache.org/.

[7] V. Hu, D. Ferraiolo, R. Kuhn, A. Schnitzer, K. Sandlin, R. Miller, and K. Scarfone, "Guide to Attribute Based Access Control (ABAC) Definition and Considerations," National Institute of Standards and Technology (NIST), Special Publication (NIST SP) 800-162, 2019.

[8] N. Dan, S. Hua-Ji, C. Yuan, and G. Jia-Hu, "Attribute based access control (abac)-based cross-domain access control in service-oriented architecture (soa)," in *2012 International Conference on Computer Science and Service System*, 2012, pp. 1405–1408. DOI: 10.1109/CSSS.2012.354.

[9] D. Jaramillo, D. V. Nguyen, and R. Smart, "Leveraging microservices architecture by using docker technology," in *SoutheastCon 2016*, 2016, pp. 1–5. DOI: 10.1109/SECON.2016.7506647.

[10] *WSO2 Identity Server Documentation*, https://is.docs.wso2.com/en/latest/.