Securing Serverless Workflows on the Cloud Edge Continuum

Securing Serverless Workflows on the Cloud Edge Continuum

Gabriele Morabito*, Christian Sicari*, Lorenzo Carnevale* *member, IEEE*, Antonino Galletta* *member, IEEE*, Giuseppe Di Modica† *member, IEEE*, and Massimo Villari* *member, IEEE*
* Department of Mathematics and Computer Sciences, Physical Sciences and Earth Sciences
University of Messina, Messina, Italy
{gamorabito, csicari, lcarnevale, angalletta, mvillari}@unime.it
† Department of Computer Science and Engineering, University of Bologna, Bologna, Italy
giuseppe.dimodica@unibo.it

***Abstract*—Serverless Computing is an emergent solution that helps deploy applications in the Cloud and sometimes on the Edge, reducing the integration time and the maintenance cost of the data centers. The lack of a standard for functions and the impossibility of connecting them together in complex workflows is currently holding back the growth of Function-as-a-Service (FaaS) use. In this scenario, OpenWolf tries to overcome these issues by implementing a solution to spread functions over the Cloud-Edge Continuum and connecting them using a standardized Domain-Specific Language (DSL) to describe a serverless based workflow. In this work, we aim to enhance the OpenWolf project, solving many security threats the engine suffers, like the authenticated and authorized execution of workflows and the injection of malicious functions inside a workflow. We will validate this new version of OpenWolf in a Smart City surveillance scenario, providing validation and performance tests.**

***Index Terms*—serverless, faas, workflows, cloud-edge continuum, security, authentication**

I. Introduction

Serverless computing debuted in 2014 when Amazon released the first serverless model, AWS Lambda, with the goal of simplifying software delivery in a cloud environment. Function as a Service (FaaS) is the most well-known serverless approach, which lets developers focus on the function, while the FaaS platform is in charge of containerizing, deploying, and load balancing it. The popularity of FaaS increased quickly, and many open source solutions (such as Openwhisk [1], Knative [2], OpenFaaS [3], [4], [5]) where proposed to avoid vendor lock-in to a specific Cloud environment. These technologies are now mature enough, and recently, new projects (i.e., [6], [7]) that incorporate FaaS at the network's edge have been started, allowing for rapid response to the Internet of Things (IoT) triggers. However, applications that totally run at the edge of the network are very rare, while it is more common to have applications composed of different modules which are distributed over different tiers (Edge, Fog, Cloud) and interact between themselves. This well-known computational model is called Edge-Cloud Continuum (or *Continuum Computing*). Instead of linking and orchestrating many events and related operations, FaaS is typically applied to small and minimally connected systems. Although the lack of composability for the function could be seen as an architectural choice, it negatively affects the ease of use of serverless, particularly in the Continuum. For this reason, it is necessary a function orchestrator to overcome this issue. Big Players proposed their own orchestrators, like Amazon Step Function, Microsoft Azure Durable Functions, and IBM Composer. However, it is impossible to use these systems everywhere since they are constrained at the providers' data centers. OpenWolf [8] was recently introduced as the first open-source serverless engine capable of building serverless applications (workflows) by composing functions using three main components: a Kubernetes heterogeneous cluster, a Broker Agent, and a workflow manifest. OpenWolf attempts to introduce serverless to the Continuum layer; however, this raises significant security concerns linked to the authorization of the execution of both the entire application (the workflow) and each of its components (the functions). This work aims to improve the OpenWolf engine by introducing an authentication system to allow only authorized workflow executions. In particular, it aims to reach the following objectives:

1) **Introduce an authentication system** into the OpenWolf engine, which allows identifying the user that requests to start a workflow. This enforces the system's security because it denies unknown users access to it.
2) **Introduce an authorization system for external requests** into the OpenWolf engine, which allows the management of external requests, triggering workflow execution only if the requesting user is authorized. In this way, the system security is strengthened because

This work has been partially supported by the Italian Ministry of Health Piano Operativo Salute (POS) trajectory 2 "eHealth, diagnostica avanzata, medical device e mini invasività" through the project "Rete eHealth: AI e strumenti ICT Innovativi orientati alla Diagnostica Digitale (RAIDD)"(CUP J43C22000380001) and the Spoke 1 "FutureHPC & BigData" of the Italian Research Center on High-Performance Computing, Big Data and Quantum Computing (ICSC) funded by MUR Missione 4 Componente 2 Investimento 1.4: Potenziamento strutture di ricerca e creazione di "campioni nazionali di R&S (M4C2-19)" - Next Generation EU (NGEU).

unauthorized execution requests are blocked.

3) **Introduce an intra-workflow authorization system** into the OpenWolf engine, which allows the management of callback requests at each workflow step, handling them only if they are generated by an authorized workflow execution. In this way, it is possible to block unwanted partial execution of the workflows that could have been triggered, for example, by a malicious node inside the cluster, namely a compromised node that can reach the OpenWolf execution endpoint and executes malicious code that sends requests to it.

A. Organization of the paper

The rest of the paper is organized as follows. In section II we discuss related works about security in a serverless environment, focusing on authentication and authorization problems. In section III, basic information about the OpenWolf engine is given. The implementation of the system is described in section IV. A possible Smart City use case is described in section V. The implemented system is validated in section VI. In section VII, we assess the performance of the proposed system. Finally, in Section VIII, conclusions and future works are discussed.

II. State of the Art

When using Serverless technologies, applications can be managed without creating a server from scratch. As a result, the service provider also handles some security-related issues. However, even though Serverless apps are not running on a managed server, they still execute code. This code must be written in a secure manner; otherwise, the application may be vulnerable to traditional application-level attacks [9]–[11]. Thus, there are two major security risks in a serverless environment [12]. On the one hand, the security level is strictly dependent on the features provided by the manufacturer. On the other side, using unsafe code for serverless applications increases the attack surface. For this reason, it is really important to accurately choose the vendor service and to pay particular attention to the quality of the code, also introducing continuous monitoring of the production environment. Many approaches to improving the consumer security mindset are suggested by the researchers. In particular, they underline the importance of creating security-by-design architectures [13], implementing secure coding standards [14], introducing threat modeling and limitation of authorization [15], and automating secure deployment systems by leveraging continuous monitoring [16]. In the scientific community, one of the innovative trends is to employ Secret Sharing (SS) [17] or Nested Secret Sharing (NSS) [18] [19] techniques to handle data in Cloud Continuum environments. In a Serverless environment, following all these approaches is fundamental to guarantee thorough security measures based on layered protection [20]. In the fields of authentication and authorization of external requests, many works have been proposed. To secure access to public services, multi-factor authentication tools based on microservices with Blockchain architecture have been designed and tested, demonstrating how the overhead of a few seconds can result in evident security benefits [21].

In the case of serverless workflows, the authentication and authorization system should be based on decoupling authentication from execution with the use of a message-oriented middleware [22], [23]. In this way, blocking malicious requests as early as possible is possible by verifying the external request authorization for all the functions at the workflow entry point. For this reason, JSON Web Token (JWT) [24] and the Oauth2 protocol [25] are good solutions for authentication, authorization, and access control in a serverless environment [26].

III. Background

Recently, OpenWolf was introduced as the first solution in the Cloud Edge Continuum to enable serverless native workflows. The formal definition of workflow, or, more accurately, scientific workflow, can be found in [27]: "A scientific workflow system is a specialized form of a workflow management system designed specifically to compose and execute a series of computational or data manipulation steps, or workflow, in a scientific application". Openwolf brings this well-known concept to the Serverless layer, using different functions spread in the Cloud and in the Edge to act as a step for the workflow. Finally, the workflow structure, that is, how the functions interact with each other, is described in a manifest file, structured according to the Serverless Workflow DSL [28] designed by the Cloud Native Computing Foundation.

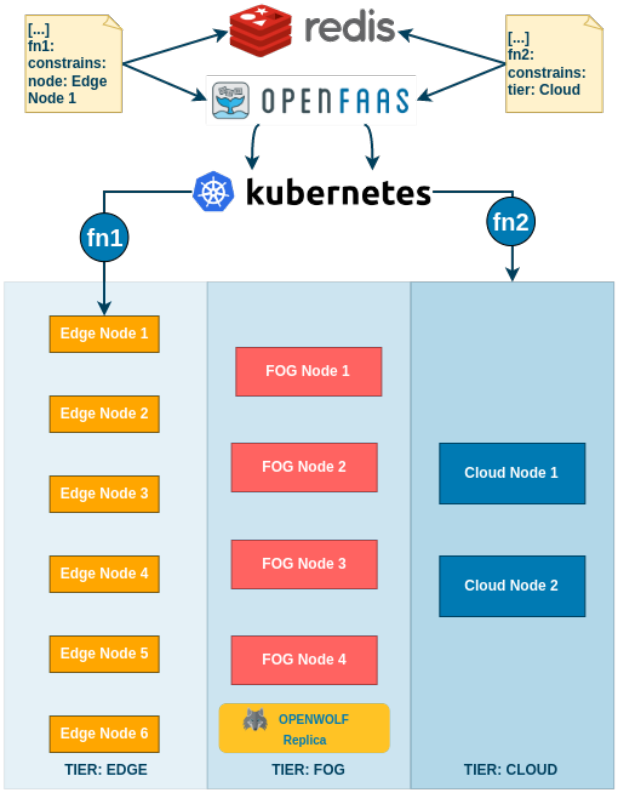

Fig. 1. OpenWolf Architecture

OpenWolf's architecture is shown in Figure 1. It stands over the K3s cluster, a tiny Kubernetes distribution well suited to be run in heterogeneous clusters composed of both ARM AMD nodes. Over the cluster, OpenWolf installs the OpenFaas engine, the OpenWolf agent, and a Redis instance. OpenFaas is the serverless engine used to build, deploy, and invoke the functions; the Agent is in charge of intercepting any function call, associating it with a specific workflow execution, and then forwarding its result to the proper destination functions in the workflow. Finally, Redis stores the manifest definitions and the workflow execution states.

IV. Implementation

The authentication and authorization mechanism designed for the deployed system is based on the Oauth2 protocol [25]. The reason behind this choice is the fact that applying the Oauth2 protocol allows for decoupling authentication and authorization routines, resulting in an important security benefit: the authorization code of the application does not have to interact with user credentials. In particular, it was decided to use the JSON Web Token (JWT) [24] for authorization purposes. In fact, the structure itself of JWT guarantees some benefits: I) it's a JSON, so it can be easily used, since most programming languages provide JSON parsers; II) it's signed using a secret, resulting in the trustness of the contained information; III) it's compact and, after being encoded, it's also smaller, resulting in a small and acceptable payload; IV) it has a standard structure (header, payload and signature) but at the same time it's versatile as you can both use standard claims and define your own ones; V) in a distributed environment it is more secure to have token authentication, that requires to set each time the request's Authorization header, than the traditional cookie-based one, where the Set-Cookie header is automatically included into the request by the browser [29]; VI) it is optimum for authenticating stateless serverless functions, because JWT itself is defined as stateless: the authorizing entity does not need to keep any state; the token alone is sufficient to authenticate a token bearer's authorization.

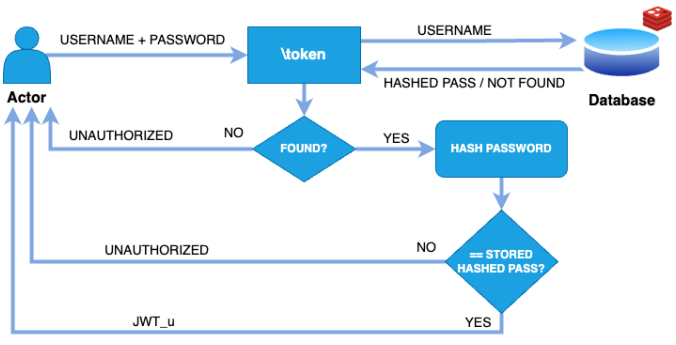

Fig. 2. User authentication

The algorithm used for the signature of the token is HMAC-SHA512 (HS512). It is a symmetric algorithm in which a single 512-bit secret key is used to generate and verify the signature. The JWT standard allows using also other algorithms, both symmetric ones, like HMAC-SHA256 (HS256), and asymmetric ones, including RSA-SHA256 (RS256), RSA-SHA512 (RS512), ES256, ES512, and many others. In this work, it was preferred to use a symmetric algorithm because, as it will be explained below, the workflow engine authentication module releases the tokens, while the execution module verifies them in the incoming requests, so there is no need to use an asymmetric algorithm, which is slower and allows third parties to verify the token using the public key.

The user authentication flow (Figure 2) is based on a traditional username and password authentication. By using his credentials, the user logs into the system to obtain a token. The user's existence is verified in a database, and if the user exists, the hashed password stored in the database is compared with the one computed from the submitted one; otherwise, the token is not released. If the hashed passwords correspond, the token (JWT_u) is released; otherwise, not. This satisfies objective 1.

The workflow execution authorization is based on a MAC (Mandatory Access Control) system, which implements zero-trust principles: access rules are manually defined by system administrators and strictly enforced by the operating system or security kernel. In this particular case, three different levels of permissions were defined: world, group, and user. World permission level refers to functions that can be executed by anyone. Group permission level is related to functions that can be executed by a particular group of users. User permission level refers to functions that can be executed only by a particular user. In addition to that, some authorization policies were defined. The authorization policy comprises the permission level and, eventually, the group or user's name related to that policy. Different Kubernetes namespaces were created for the functions. All the functions with the same execution policy were deployed in the same namespace and each namespace was labeled with the name of the policy.

The workflow execution authorization flow (Figure 3) starts when the user invokes the workflow through an API. The presence of a valid JWT_u in the request is verified; otherwise, the execution is not authorized. The *sub* claim of the token is compared with the usernames stored in the databases, and if a match is not found, the execution is not authorized. Instead, if a match is found, the user's authorization to execute all the functions of the invoked workflow is verified by comparing some labels associated with the user and the namespace of the requested functions. If the user is authorized, the workflow is triggered. This satisfies objective 2.

The workflow steps are stateless and unique entities called States that include a function. For this reason, the workflow engine triggers each step of the workflow, which listens for the callbacks of each state execution and triggers the next state executions. In this way, it is possible to manage any kind of state relationship (one-to-one, one-to-many, many-to-one, many-to-many). When a callback request is sent to the workflow engine, it is necessary to authenticate the state that sent the callback to ensure that it comes from an authorized

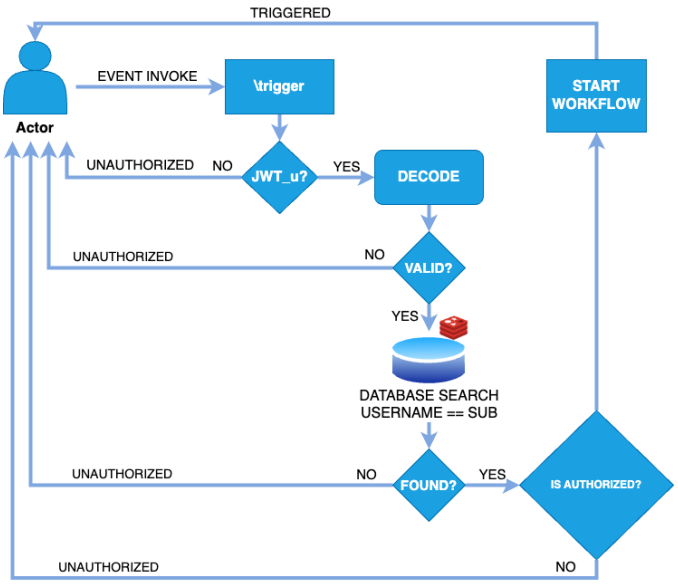

Fig. 3. Workflow execution authorization

executive of the workflow. For this reason, another token (JWT_e), released when the workflow execution is triggered, is used.

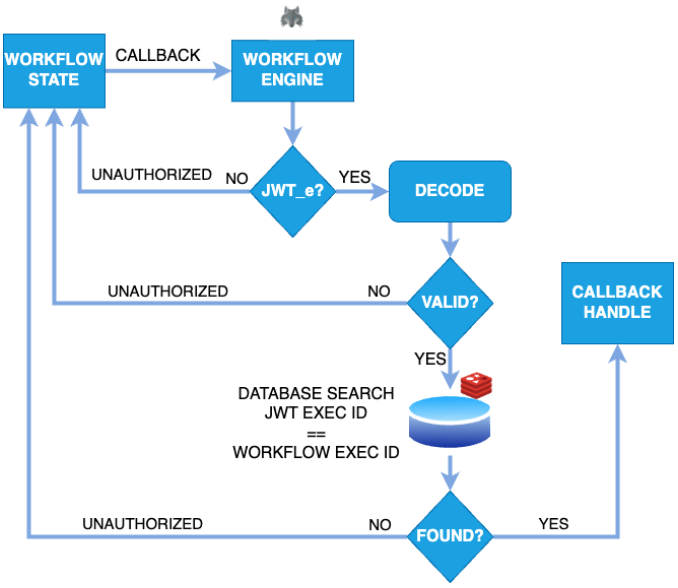

Fig. 4. Workflow States authentication

The workflow step execution authorization (figure 4) flow starts each time a callback is received by the workflow engine. The presence of a valid JWT_e in the callback request is verified; otherwise, the execution is not authorized. The *exec_id* claim of the token is compared with the workflow execution is stored in the databases and if a match is not found, the execution is not authorized. Instead, if a match is found, the execution is authorized, and the callback request is handled. This satisfies objective 3.

V. Use case

The authenticated version of OpenWolf allows the definition of workflows distributed over the Continuum. This is useful, for example, in all those scenarios where there is the need to do lighter computations at the Edge of the network, near the data sources, than heavier ones in Cloud data centers. A typical scenario where these needs are met can be represented by Smart Cities. An example of a workflow that can be applied in such a scenario could be an imaging processing workflow. In fact, it is common in Smart Cities to have image data from some Smart Cameras distributed over the city. The collected data may be processed locally using a pre-trained machine learning model to detect, for example, traffic rules violations or dangerous situations. Then, these data can be stored in Cloud data centers. In this particular scenario, the benefits introduced by the authentication system are significant: having an authenticated flow, where every single step needs to be authorized before execution, allows to avoid some malicious third party from injecting corrupted data into the system. This is very important because the collected and processed data could be used, for example, by the municipalities to make fines or by the police to intervene in case of danger. In addition to that, some functions could be used to access sensitive data, and for this reason, authorization control must be ensured. The description of a simple image processing workflow, composed of five states, each one represented by a serverless function, follows. **Collect**: utilizes a camera stream for gathering environment pictures. **Transform**: cleans the pictures from noisy data by filtering them. **Train**: trains a Convolutional Neural Network (CNN) model for analyzing the collected pictures. **Inference**: makes inference on the input data using the most recent model produced at the workflow Train state. **Show**: pushes the output data to a web page where they can be visualized. All these functions, except ”Collect”, may need to be executed either on the Cloud or on the Edge, depending on the required Quality of Service (QoS). From the performance point of view, it should be more convenient to train the neural network on the Cloud, while the inference instead depends on the requirement. For example, dangerous situations, like fights, muggings, or robberies, must be detected as early as possible, so Edge real-time computation is required. In other cases, such as traffic rules violations detection, an optimized and massive computation can be done later on the Cloud. The proposed solution allows customizing where each workflow state is executed, depending on where the serverless functions are scheduled. Thus it satisfies any kind of required QoS.

Finally, figure 5 gives a visualization of the environment we will simulate. Basically, we chose the most common way to deploy our functions, locating the data gathering and inference at the edge while the training and plotting in the cloud, leaving the fog in charge of cleaning the data. As highlighted in figure 5 and in the previous section, OpenWolf acts like a broker, and then functions do not reach each other directly but by passing through it, for this even properly locating

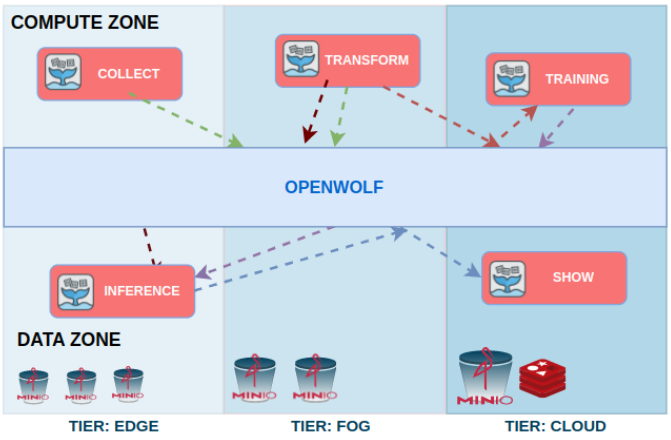

Fig. 5. OpenWolf for Image Processing

and eventually replicating OpenWolf in different tiers is an important choice to take in account.

VI. Validation

The authenticated version of OpenWolf and its authorization system were tested by registering three users and two groups: *user1* who belongs to *group1*, *user2* who belongs to *group2*, *user3* who belongs to both the groups. In addition to that, some workflows were defined: three workflows, one for each user, including functions with world-level permission policy and at least one function with the specific user permission policy; two workflows, one for each group, composed of functions with world-level permission policy and at least one function with the specific group-level permission policy; a workflow, including only functions with world-level permission policy. The tests were executed by logging into the system with each user and trying to trigger each defined workflow. The results are summarized here: I) the workflow including only functions with world-level permissions policy execution was triggered by all three users; II) the workflows including at least one function with group-level permission policy were triggered, as expected, only by the users who belonged to the specific group; III) the workflows including at least one function with user-level permissions policy were triggered, as expected, only by the specific user. In addition to that, the intra-workflow authorization system was tested by simulating the presence of a malicious node inside the Kubernetes Network that generates malicious callback requests to the OpenWolf Agent. As expected, all the requests were rejected, and unwanted partial executions of the workflows were avoided.

VII. Performance Evaluation

In this section, the performances of the implemented system are described. In particular, the execution time of a CNN workflow, composed of three states (data fetch, train, and inference), is evaluated. The testbed used for the performance evaluation is a three-node Kubernetes cluster composed of two Edge nodes and one Cloud node. In particular, Kubernetes Master, Prometheus, OpenFaas (the Gateway, the Nats, and the Queue Manager) and OpenWolf Engine (the Agent and Redis) were deployed on the Cloud node, while the Edge tier was used only for hosting functions. Our three-state workflow has been evaluated in the versions full cloud, full edge, and continuum; in the first one, all the functions of the workflows are deployed in the cloud, while in the second, all of them are located in the edge, finally, in the continuum version we balanced the functions trying to optimize the location considering the best tier they fit. The systems' characteristics are summarised in table I, while the OpenWolf parameters are summarized in table II.

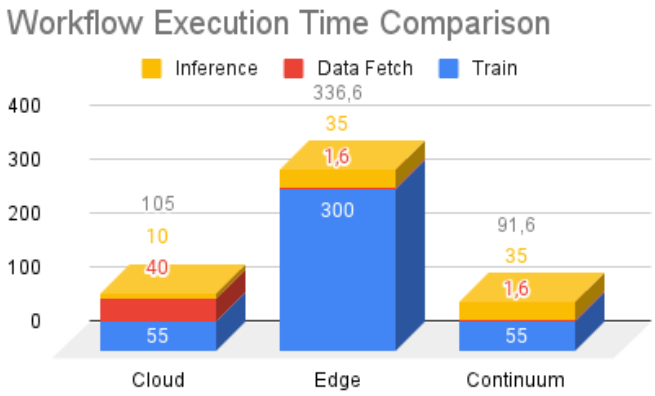

Fig. 6. Workflow Execution without Authentication and Authorization

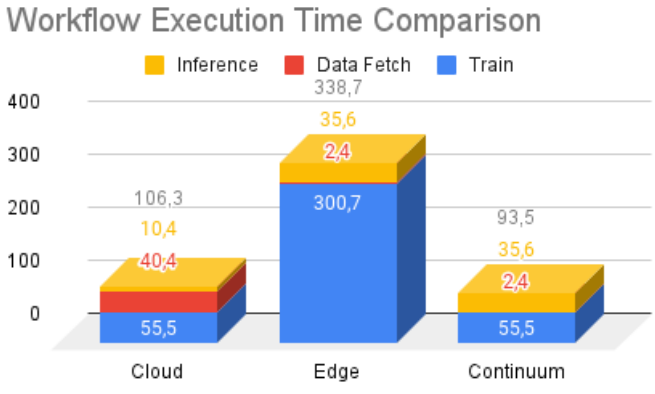

Fig. 7. Workflow Execution with Authentication and Authorization

In Figures 6 and 7 we reported the results, we obtained in terms of the response time of the workflow in the different tiers and in the Continuum environment. As expected, the Continuum guarantees the best performance with respect to the full-cloud and full-edge executions; this happens because, in Continuum, we can orchestrate the workflows to execute

TABLE I
CLUSTER'S NODES CHARACTERISTICS

Instances	Tier	Model	CPU	Memory	Operating System
1	Cloud	Openstack VM	Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz, 2-core	4 GB	Ubuntu 20
2	Edge	Raspberry Pi 4	ARM64 SoC 1.5GHz, 4-core	4 GB	Raspberry OS ARM64

TABLE II
OPENFAAS AND OPENWOLF PARAMETERS

Parameter	Value	Condition
Queue Workers	1	Ever
Function replicas	State references	States Number $<$ 60
Function replicas	(State References)/2	States Number $\geq$ 60
Max_inflight	Equal to functions replicas	Ever

functions where they best fit; in this case, it's critical training in the Cloud, and fetching data in the Edge. The authentication an authorization processes added latency to the response time of the system. In particular, in the conducted experiments, the unit cost of these processes is about 0.4 seconds in the full Cloud environment, about 0.7 seconds in the full Edge environment and about 0.5 seconds in the Continuum environment. As we can see by comparing the results in Figure 6 and in Figure 7, the entire execution of the authenticated and authorized version of the workflow is on average 1.5% slower than the non-secure one. So the latency added to the workflow execution time can be considered negligible, taking into account also the earned security benefits.

VIII. CONCLUSION

In 2022 we started the OpenWolf project, an open-source engine available on GitHub[1], that aims to act like a broker for the Serverless Continuum, which is a distributed and heterogeneous environment where FaaS functions interact each other following a Workflow Manifest file parsed and managed by OpenWolf itself. In this work, we are still improving the OpenWolf project, trying to solve some open issues related to the security of the environment; in particular, we tried to I) guarantee an authentication system to address functions, workflows, and executions, II) introduce an authorization system, able to allow and disallow the execution of workflow to particular users and group and III) keep secure the workflow state, avoiding the injection of malicious code that can affect the behavior of one or more functions in the system. We deeply described how these goals had been reached, and all the updates have been uploaded to the official repository. We verified the correctness of our work by providing a validation assessment that demonstrated that all the security threats we aimed to solve had been avoided; finally, we measured the performances of this version of OpenWolf, comparing it with the previous one; we did that to verify that the security features we added in this work had not affected the performance of the system, and as shown they didn't.

The OpenWolf project is still in early state release, and we have already planned plenty of work on it. In the next future, we plan to apply the Osmotic Computing paradigm to OpenWolf; in particular, we believe that some interesting concepts like the Software Defined Membrane and the Micro Element for the Continuum can be greatly integrated into OpenWolf to improve the security aspect of this engine further. Furthermore, we strongly believe that the serverless engine solutions proposed in the open-source context are still complementary, therefore for some use cases OpenFaas, that is, the engine used in OpenWolf, could not absolve all use cases; for this reason, we aim to integrate more serverless engine in OpenWolf, like OpenWhisk and KNative. The last work we planned to work on is defining a standard for serverless workflow. As we said in this and in the linked works, Serverless is a great opportunity for the Continuum, but the lack of a standardized workflow engine reduces the potential of these solutions; indeed, we have planned to work on a white paper with the intent of defining a common and standardized way to implement serverless workflow architectures for the Computing Continuum.

[1]https://github.com/christiansicari/OpenWolf-Serverless-Workflow

REFERENCES

[1] K. Djemame, M. Parker, and D. Datsev, "Open-source serverless architectures: an evaluation of apache openwhisk," in *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*, pp. 329–335, 2020.

[2] N. Kaviani, D. Kalinin, and M. Maximilien, "Towards serverless as commodity: A case of knative," in *Proceedings of the 5th International Workshop on Serverless Computing*, WOSC '19, (New York, NY, USA), p. 13–18, Association for Computing Machinery, 2019.

[3] D. Balla, M. Maliosz, and C. Simon, "Open source faas performance aspects," in *2020 43rd International Conference on Telecommunications and Signal Processing (TSP)*, pp. 358–364, 2020.

[4] S. K. Mohanty, G. Premsankar, and M. Di Francesco, "An evaluation of open source serverless computing frameworks," in *Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom*, vol. 2018-Decem, pp. 115–120, IEEE Computer Society, dec 2018.

[5] P. Garcia Lopez, M. Sanchez-Artigas, G. Paris, D. Barcelona Pons, A. Ruiz Ollobarren, and D. Arroyo Pinto, "Comparison of FaaS orchestration systems," *Proceedings - 11th IEEE/ACM International Conference on Utility and Cloud Computing Companion, UCC Companion 2018*, pp. 109–114, 2019.

[6] M. Ciavotta, D. Motterlini, M. Savi, and A. Tundo, "Dfaas: Decentralized function-as-a-service for federated edge computing," in *2021 IEEE 10th International Conference on Cloud Networking (CloudNet)*, pp. 1–4, 2021.

[7] T. Pfandzelter and D. Bermbach, "tinyfaas: A lightweight faas platform for edge environments," in *2020 IEEE International Conference on Fog Computing (ICFC)*, pp. 17–24, 2020.

[8] C. Sicari, L. Carnevale, A. Galletta, and M. Villari, "Openwolf: A serverless workflow engine for native cloud-edge continuum," 09 2022.

[9] T. Melamed, "Interpretation for serverless," *OWASP Top 10*, 2017.

[10] N. Mateus-Coelho and M. Cruz-Cunha, "Serverless service architectures and security minimals," in *2022 10th International Symposium on Digital Forensics and Security (ISDFS)*, pp. 1–6, 2022.

[11] M. Wu, Z. Mi, and Y. Xia, "A survey on serverless computing and its implications for jointcloud computing," in *2020 IEEE International Conference on Joint Cloud Computing*, pp. 94–101, 2020.

[12] M.-C. Nuno and M. Cruz-Cunha, "Distributed information flow control in serverless computing," in *2022 10th International Symposium on Digital Forensics and Security (ISDFS)*, 2022.

[13] V. Vallois, F. Guenane, and A. Mehaoua, "Reference architectures for security-by-design iot: Comparative study," in *2019 Fifth Conference on Mobile and Secure Services (MobiSecServ)*, pp. 1–6, 2019.

[14] T. Espinha Gasiba and U. Lechner, "Raising secure coding awareness for software developers in the industry," in *2019 IEEE 27th International Requirements Engineering Conference Workshops (REW)*, pp. 141–143, 2019.

[15] F. M. Awaysheh, M. N. Aladwan, M. Alazab, S. Alawadi, J. C. Cabaleiro, and T. F. Pena, "Security by design for big data frameworks over cloud computing," *IEEE Transactions on Engineering Management*, vol. 69, no. 6, pp. 3676–3693, 2022.

[16] W. O'Meara and R. G. Lennon, "Serverless computing security: Protecting application logic," *31st Irish Signals and Systems Conference (ISSC)*, 2020.

[17] A. Galletta, J. Taheri, and M. Villari, "On the applicability of secret share algorithms for saving data on iot, edge and cloud devices," in *2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pp. 14–21, 2019.

[18] A. Galletta, J. Taheri, M. Fazio, A. Celesti, and M. Villari, "Overcoming security limitations of secret share techniques: the nested secret share," in *2021 IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pp. 289–296, 2021.

[19] A. Galletta, J. Taheri, A. Celesti, M. Fazio, and M. Villari, "Investigating the applicability of nested secret share for drone fleet photo storage," *IEEE Transactions on Mobile Computing*, pp. 1–13, 2023.

[20] X. Li, X. Leng, and Y. Chen, "Securing serverless computing: Challenges, solutions, and opportunities," in *IEEE Network*, 2022.

[21] A. Catalfamo, A. Ruggeri, A. Celesti, M. Fazio, and M. Villari, "A microservices and blockchain based one time password (mbb-otp) protocol for security-enhanced authentication," in *2021 IEEE Symposium on Computers and Communications (ISCC)*, pp. 1–6, 2021.

[22] A. Sabbioni, C. Mazzocca, M. Colajanni, R. Montanari, and A. Corradi, "A fully decentralized architecture for access control verification in serverless environments," in *2022 IEEE Symposium on Computers and Communications (ISCC)*, 2022.

[23] C. Sicari, A. Catalfamo, A. Galletta, and M. Villari, "A distributed peer to peer identity and access management for the osmotic computing," in *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pp. 775–781, 2022.

[24] "Json web token introduction - jwt.io." https://jwt.io/introduction. Accessed: 2023-01-03.

[25] "Oauth 2.0 — oauth." https://oauth.net/2/. Accessed: 2023-01-03.

[26] M. Golec, R. Ozturac, Z. Pooranian, S. S. Gill, and R. Buyya, "ifaasbus: A security- and privacy-based lightweight framework for serverless computing using iot and machine learning," in *IEEE Transactions on Industrial Informatics*, vol. 18, pp. 3522 – 3529, 2021.

[27] C. S. Liew, M. P. Atkinson, M. Galea, T. F. Ang, P. Martin, and J. I. V. Hemert, "Scientific workflows: Moving across paradigms," *ACM Comput. Surv.*, vol. 49, dec 2016.

[28] "Serverless workflow definition language." https://serverlessworkflow.io/. Accessed: 2023-01-03.

[29] Gilles and M. De Mey, "Jwt tokens for distributed authentication." https://demey.io/jwt-tokens-for-distributed-authentication/. Accessed: 2023-01-03.