Protection contre les fuites de données : un environnement micro-services sécurisé

Loïc Miller[1] [†], Pascal Mérindol[1], Antoine Gallais[2] et Cristel Pelsser[1]

[1]*Université de Strasbourg,* [2]*UPHF / INSA Hauts-de-France*

Les fuites de données au repos sont malheureusement courantes et en augmentation, entrainant des pertes de profits pour les entreprises ou le non-respect de confidentialité de données personnelles sensibles. La récente montée en puissance des micro-services, en tant que paradigme de déploiement extensible, impose également la sécurisation du trafic. Un processus métier peut être modélisé sous la forme d'un *workflow* : le propriétaire des données interagit avec des sous-traitants pour réaliser une séquence de tâches.

Dans cet article, nous montrons comment ces *workflows* peuvent être appliqués tout en limitant l'exposition des données. En suivant les principes du *zero-trust*, nous proposons une infrastructure utilisant l'isolation fournie par les micro-services pour mettre en œuvre un *workflow*, dans une preuve de concept disponible en ligne. Nous vérifions ensuite que les politiques de sécurité sont correctement appliquées, et estimons le coût supplémentaire induit par les dispositifs de sécurité.

Mots-clefs : fuites de données, workflow, microservices, sécurité

1 Introduction

The recent rise of microservices as a novel deployment paradigm increases the attack surface as well as data leaks. Some attacks may come from inside the system, e.g., leaks stemming from the way data is processed or caused by a malicious employee. The zero-trust security model, where all flows are required to be authenticated and authorized via fine-grained policies, is an important step to shield from such attacks.

In accordance with such principles, we aim to achieve a secure system enabling the exchange of data between non-trusted agents. The data should be secured at rest and in transport and cannot be exposed by any agent in both cases. Our goal is to offer such guarantees in the context of workflows. We define a workflow as a sequence of tasks to be performed by a set of independent actors. The owner of the data (i.e., the instigator of the workflow) interacts with contractors to realize such a sequence. Both the owner and the contractor have agents processing the data (agents can represent an employee or an automatic service).

Let us consider a simple example of workflow (Fig. 1), where an owner in the post-production stage of making a movie employs other companies to edit the video and audio components [BCK$^+$03]. The owner (O) first sends its data to the company responsible for special effects (C_{1_x}). C_{1_x} applies special effects to the movie sequences the owner sent him, and then sends the result to the company responsible for coloring (C_2) as well as to another for sound mastering (C_3). C_2 then ships its result to the agent in charge of High Dynamic Range (HDR) (C_3) and sound mastering (C_4). Finally, both C_3 and C_4 sends their output back to the owner.

To meet our requirements for zero-trust and prevent data leaks, we rely on a secured microservice architecture. We isolate in containers the environment in which the agents execute their task. Each container is secured thanks to traffic interception and access control enforcement. An orchestrator, a service mesh and policy engines are deployed to enforce the workflow along with the access policies of the owner.

[†]This project has been made possible in part by a grant from the Cisco University Research Program Fund, an advised fund of Silicon Valley Foundation.

Related Works Existing works provide guidance on overall security requirements and strategies for microservices [Cha19]. Weever and Andreou [dWA20] implement zero-trust in a microservice environment, and implement Cilium to enable deep traffic visibility. Contrary to our work, the authors neither explore the use of authorization sidecars nor measure their cost. Accorsi et al. [AW11] use static flow analysis to automate data leak detection in workflows. In our work, we consider such leaks within the microservice paradigm and design a proof of concept to evaluate its overhead in term of performance.

Considering the problem at hand (Sec. 2), we specify our threat and security models. We describe our solution and its companion proof of concept (Sec. 3). We finally evaluate the authorization overhead (Sec. 4).

2 Threat and security model

We consider a threat model from the point of view of each actor. The owner wants to avoid the leakage of the data sent to the involved contractors. The threat here is then that an agent leaks the critical data to an unauthorized party (or that the data is accessed by an adversary). On the other hand, a contractor does not want the other actors, including the owner, to learn about their business intelligence.

Trust Model : Actors and Environment While the owner and contractors trust the organization of the contractors to not intently bypass the system, looking at a finer grain, actors do not necessarily trust their own agents or the ones of the other actors. Both the owner and the contractors need to trust the overall environment on which the workflow is deployed. The third-parties (e.g. cloud providers) hosting parts of the infrastructure are assumed not to try to gain access to the sensitive data. In summary, while the owner and contractors trust the service deployment (actors are assumed to be curious but honest), the agents are potentially malicious.

Attacker Model : External Attackers and Malicious Agents Taking into account the assets to protect and our trust model, we consider three types of attackers in our model.

— *External attacker* : External to the workflow and the location of the deployed infrastructure. Such attackers try to gain access to the data or the business intelligence from the outside.

— *Co-located attacker* : External to the workflow, but co-located at the deployment (e.g. an attacker located in one of the third-party clouds). This co-located position opens more exploit options.

— *Malicious agent* : Internal to the workflow, this attacker tries to leak the data outside.

3 Infrastructure and Proof of Concept

Figure 1 illustrates how we arrange the elements of the microservice architecture [‡]. Each actor has its own deployment space (clouds) while internal agents are depicted as boxes (e.g. C_{1_1} is an agent of contractor C_1). A control plane in each space controls the service mesh. Each agent is a pod, containing the service (i.e., the environment the agent will be using), a proxy and a policy sidecar. The access policies of a service are pushed in its associated policy sidecar. The proxy sidecar intercepts all traffic coming from and going to the associated service and the policy sidecar checks whether it shall be authorized or not.

The service mesh controller and the policy store are under the control of the owner. It specifies the policy to be enforced, preventing in particular the data from leaking outside. The data processed by the pods are stored encrypted on mounted Persistent Volumes (PVs), providing us with **data security at rest**. Pods also communicate according to the specified workflow and policy via `mTLS`, providing us with **data security in transport**. Communications inside a pod are not encrypted, but the isolation layers protect the data against eavesdroppers.

Proof of Concept We implemented the above infrastructure by reproducing the workflow of Fig. 1. The complete data, code as well as guidance to realize this Proof of Concept are publicly available [§]. We use Docker for our containers, Kubernetes for our orchestration layer, Istio as our service mesh. Finally, we rely on Envoy and Open Policy Agent for the proxy and policy sidecars respectively.

‡. See https ://arxiv.org/abs/2012.06300 for more details.
§. See https ://github.com/loicmiller/secure-workflow.

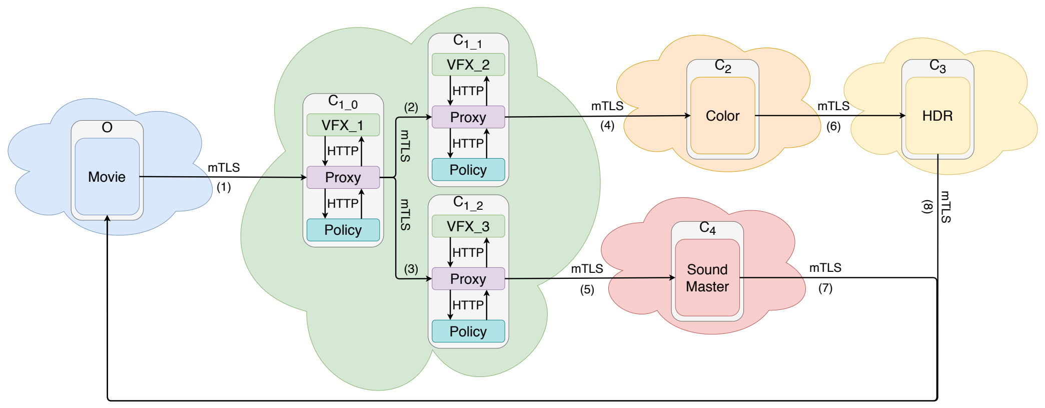

FIGURE 1: Our proposed secure infrastructure.

This infrastructure was deployed on Google Cloud Platform (GCP), using one cluster for each actor of the workflow, for a total of five clusters. Each cluster runs one **n1-standard-2** node (2 vCPUs, 7.5 GB of memory), on version **1.14.10-gke.36**, except the cluster of the owner which runs two of them, since running the control plane requires additional resources. The clusters for the owner, color and VFX are located in the us-central1-f region whereas the clusters for HDR and sound are located in us-west2-b.

Test framework We developed an evaluation framework to check the overall security : encrypted traffic, isolation inside a pod, and the policy enforcement. To do so, we capture traffic on every network interface of the service mesh and try to perform each possible communication. Fig. 2 shows the path a communication takes inside the service mesh, as well as whether traffic is encrypted or not. Traffic going to/coming from the loopback is unencrypted, whereas traffic going to/coming from the external interface should be encrypted. Our captures show that this is indeed the case. We also check that authorization is correctly enforced : we extract authorizations from the OPA policy configuration, and then generate and test the access control matrix. The complete code for the test framework is publicly available §.

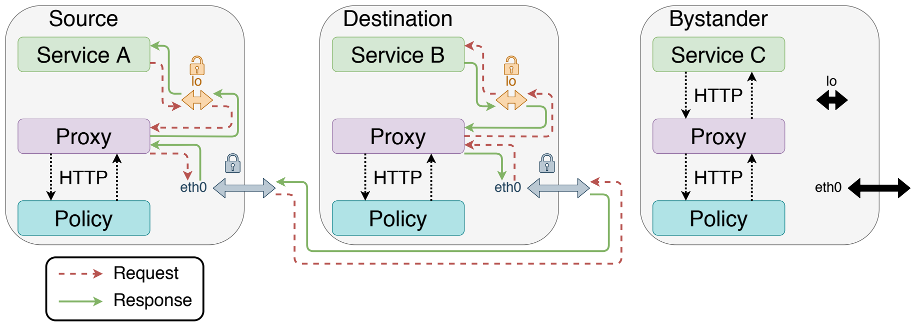

FIGURE 2: Detailed view of pods and the communication flow.

4 The Overhead of Security

In this section, we analyze the performance overhead added by the policy sidecar enforcing security. We measure the pod startup time and the request duration (between each couple of connected pods).

Startup time We first evaluate the impact of having an additional container for OPA on the startup time of pods. An independent-samples t-test was conducted to compare startup times in a deployment of our PoC with or without OPA. We gathered 130 observations per pod and per deployment ($N = 1820$ in total).

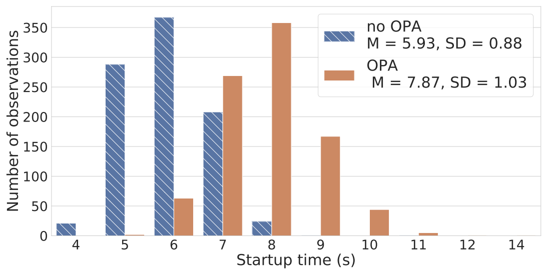

FIGURE 3: Distribution of startup time in deployments with/without OPA.

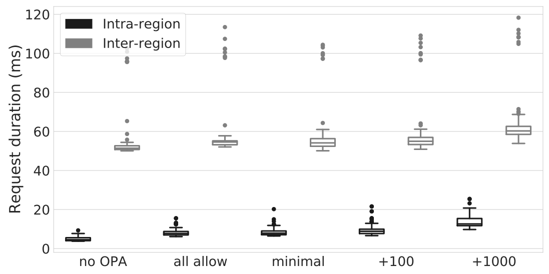

FIGURE 4: Spread of request duration in intra and inter-region communications by policy size.

Fig. 3 allows to measure the cost on the initial deployment by comparing the distribution of startup times (with or without OPA deployed). The group with the OPA sidecar exhibits significantly higher startup times compared to the group without the OPA sidecar, $t(1818) = 43.19, p < 0.001$. Pods with OPA have a substantial increase in startup time of almost two seconds on average, i.e., 32.72% of the startup time.

Request time To test whether the policy is scalable for more complex workflows, we measure the influence of policy size on communications. A *one-way between subjects ANOVA* was conducted for each type of communication (intra/inter region) to compare the effect of policy size on request duration in five increasing orders of policy size : `no opa`, `all allow`, `minimal`, `+100` (rules) and `+1000` (rules).

For each ANOVA, we gathered 40 observations per authorized communication per level of policy ($N = 1600$ in total). Fig. 4 shows the distribution of request duration for each policy size. For intra-region communications, there is a significant difference in request duration among the five scenarios of policy deployments, $F(4,795) = 364.05, p < 0.001, \eta^2{}_p = 0.65$. For inter-region communications there also exists a significant difference (in request duration) among the five scenarios of policy deployments, albeit with a lesser effect : $F(4,795) = 15.23, p < 0.001, \eta^2{}_p = 0.07$[¶].

While our results suggest that a higher policy size increases request duration, it should be noted that this size should be strongly increased in order to observe an effect. With inter-region communications in particular, incremental increases in policy size do not appear to have a significant effect on request duration.

5 Conclusion

In this work, we described and implemented a secure architecture that prevents data leaks and protects business intelligence. We achieve a secure system guaranteeing the data is secure at rest, in transport and cannot be leaked by any agent in both cases. We realized a proof of concept of our architecture to discuss its benefits and limitations, and monitored key parts of the workflow to show how the data is secured. Our infrastructure is resilient to the set of attacks considered in our security model. Our experiments finally show that our approach scales well with the increasing number of rules which reflects workflow complexity.

Références

[AW11] Rafael Accorsi and Claus Wonnemann. Strong non-leak guarantees for workflow models. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, pages 308–314, 2011.

[BCK$^+$03] Simon Byers, Lorrie Cranor, Dave Korman, Patrick McDaniel, and Eric Cronin. Analysis of security vulnerabilities in the movie production and distribution process. In *Proceedings of the 3rd ACM workshop on Digital rights management*, pages 1–12. ACM, 2003.

[Cha19] Ramaswamy Chandramouli. Security strategies for microservices-based application systems. Technical report, 2019.

[dWA20] Catherine de Weever and Marios Andreou. Zero trust network security model in containerized environments. 2020.

¶. See https ://github.com/loicmiller/secure-workflow for full data, code and statistical analysis in the form of jupyter notebooks.