# Towards Secure and Leak-Free Workflows Using Microservice Isolation

Loïc Miller
*ICube*
*University of Strasbourg*
Strasbourg, France
loicmiller@unistra.fr

Pascal Mérindol
*ICube*
*University of Strasbourg*
Strasbourg, France
merindol@unistra.fr

Antoine Gallais
*ICube*
*UPHF / INSA Hauts-de-France*
Valenciennes, France
antoine.gallais@uphf.fr

Cristel Pelsser
*ICube*
*University of Strasbourg*
Strasbourg, France
pelsser@unistra.fr

***Abstract*—Companies like Netflix increasingly use the cloud to deploy their business processes. Those processes often involve partnerships with other companies, and can be modeled as workflows. This shift towards the cloud environment has led to more and more data leaks and breaches, resulting in huge losses of money for businesses like the movie industry, as well as a loss of user privacy for businesses dealing with user data like the pharmaceutical industry.**

**In this paper, we show how those workflows can be enforced while preventing data exposure. Following the principles of zero-trust, we develop an infrastructure using the isolation provided by a microservice architecture, to enforce owner policy. We show that our infrastructure is resilient to the set of attacks considered in our security model. We implement a simple, yet realistic, workflow with our infrastructure in a publicly available proof of concept. We then verify that the specified policy is correctly enforced by testing the deployment for policy violations, and estimate the overhead cost of authorization.**

***Index Terms*—data leak, data breach, workflow, microservices, authorization, security**

## I. Introduction

Data leaks and breaches are increasingly happening. With more and more businesses using public clouds to process data [1], and this data being frequently moved around, exposures are more likely to happen than ever. Those exposures are perceived as huge losses of money for businesses like the movie industry [2], and as a loss of user privacy for applications dealing with user data [3]. Malicious actors have been responsible for most incidents, but accidental exposure of data on the Internet (e.g. misconfigured databases, backups, end points, services) has put the most records at risk [4].

Even though both data breaches and data leaks result in data being exposed to unauthorized entities, the way this data is exposed is different. On one hand, data breaches refer to the unauthorized access of data by exploiting flaws in the security of the breached system. Data breaches can happen with data at rest [5] where attackers exploit a flaw to gain access to the data, or in transport [6], where attackers exploit a vulnerability to eavesdrop on traffic. On the other hand, data leaks refer to the exposure of data belonging to an entity, due to the way this data is processed by this entity, by a mistake [7], or caused by malicious behavior [8].

The recent rise of microservices as a paradigm, and their increased use in building large, cloud-based enterprise applications [9] has increased the attack surface, meaning protecting the network border is no longer sufficient. To prevent data leaks, one needs to consider attacks coming 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 [10], where all traffic flows are required to be authenticated and authorized via fine-grained policies, provides such protection.

In accordance with such principles, we aim to achieve a secure system enabling the exchange of data between non-trusted agents in the context of workflows. The data should be secured at rest and in transport and cannot be exposed by any agent in both cases. To meet our requirements for zero-trust and prevent data leaks during the execution of workflows, we rely on a secured microservice architecture.

The microservice architecture allows us to design a system preventing data exposures that is simple, modular and scalable, thanks to its loosely coupled services. This is important when considering security mechanisms quickly become challenging to configure, manage, scale and monitor when combined, with a large number of actors using different IT environments.

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, where 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 [2]. 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

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.

978-1-6654-4005-9/21/$31.00 ©2021 European Union

$C_4$ sends their output back to the owner.

We isolate in containers the environment in which the agents execute their tasks. 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.

Considering the problem at hand (Sec. II), we specify our threat and security models. We describe our solution and its companion proof of concept (Sec. III). We finally evaluate the authorization overhead (Sec. IV), review related works (Sec. V and conclude (Sec. VI).

## II. 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.

## III. Infrastructure and Proof of Concept

We propose an infrastructure to protect a workflow execution from the threats expressed in Sec. II. As we need a way to prevent data leaks, we need to control the communications an agent can engage in. To achieve this, we need to control the environments the agents will be using, to make sure that all the actions of an agent follow a policy enforced by the owner. In this infrastructure, agents of our workflow are mapped to containers, which are then used in conjunction with an orchestrator, a service mesh and policy engines to enforce the policy of the owner.

Figure 1 illustrates how we arrange the elements of the microservice architecture[1]. We use the microservice architecture to make our solution pratical and easily deployable. 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 $C_1$). Each agent is a pod, containing multiple containers: 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[2]. We use Docker [11] for our containers, Kubernetes [12] for our orchestration layer, Istio [13] as our service mesh. Finally, we rely on Envoy [14] and Open Policy Agent [15] for the proxy and policy sidecars respectively.

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 us-central1-f 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. According to the pod type (source, destination or bystander) and the interface (loopback or external), we need to verify different things:

- **Source / Destination loopback**: we verify that a communication between the source and the destination is occurring (i.e., correct IP addresses and ports). We need to verify that the request in the capture corresponds to the request we are testing for (GET or POST). The response needs to be in accordance with the policy: in this case, '403 Forbidden' if the policy was 'deny' and '200 OK' (GET) or '201 OK' (POST) if the policy was 'allow'.

[1] See https://arxiv.org/abs/2012.06300 for more details.

[2] See https://github.com/loicmiller/secure-workflow.

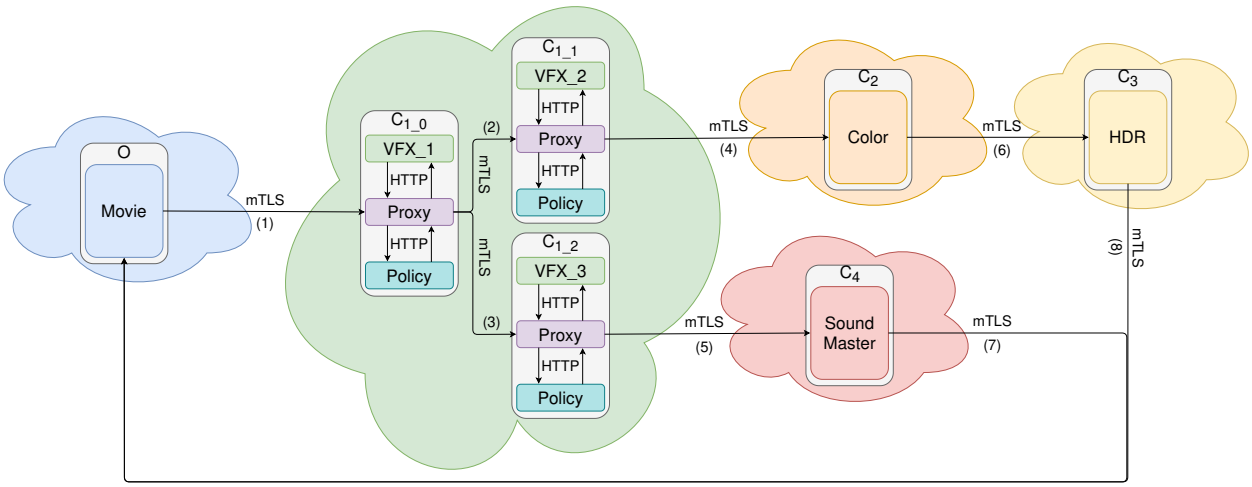


Fig. 1: Our proposed secure infrastructure. Each actor is represented by a cloud, and each agent by a box. Each box corresponds to a Pod, containing the service, the proxy and the policy sidecar.

- **Source / Destination external interface**: we verify that a communication between the source and the destination is occurring (correct IP addresses and ports). We need to verify that the traffic is encrypted by `mTLS`, and not passed in clear text.
- **Bystander loopback and external interface**: we verify that no communication between the source and the destination is occurring, whether encrypted or unencrypted.

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[2].

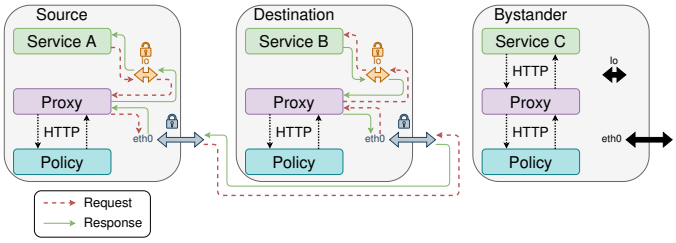


Fig. 2: Detailed view of pods and the communication flow.

## IV. 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).

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. More in details, the *effect size* for this analysis, $d = 1.985$, was found to exceed Cohen's convention for a large effect ($d = 0.80$). Running a *post hoc power analysis* also reveals a high statistical power, $1 - \beta > 0.999$.

*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).

The `no opa` policy deployment corresponds to having no OPA container at all. The `all allow` policy deployment corresponds to having no rules and allowing all communications by default. The `minimal` policy deployment corresponds to having the default minimal number of rules to enforce the workflow of the PoC. The `+100` and `+1000` correspond to the minimal policy being inflated respectively with 100 additional rules (+147%) and 1000 additional rules (+1470%), with additional rules being obligatorily evaluated by OPA.

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

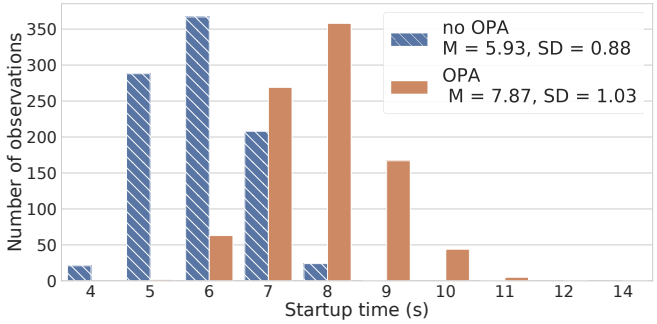


Fig. 3: Distribution of startup time in deployments with/without OPA.

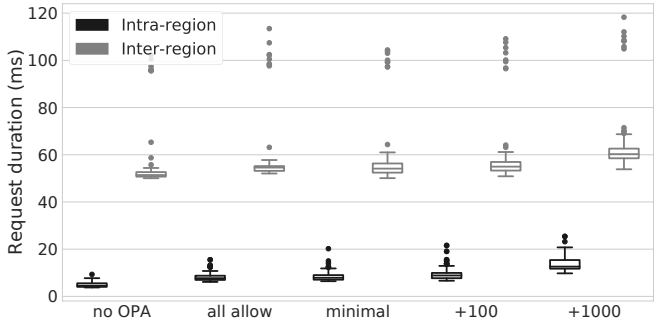


Fig. 4: Spread of request duration in intra and inter-region communications by policy size.

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: $F(4, 795) = 15.23, p < 0.001, \eta^2{}_p = 0.07$[3].

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. This effect is minor in inter-region communications. In summary, the added security provided by the workflow enforcement costs pods two seconds of startup time on average, and either 7% or 65% of the variance in request duration.

## V. Related Works

Existing works provide guidance on overall security requirements and strategies for microservices [16], as well as guidance on more specific microservices components like service mesh [9], [17] or containers [18], [19]. Chandramouli [16] provides guidance on security strategies for implementing core features of microservices, as well as countermeasures for microservices-specific threats. We follow the guidelines and recommendations presented in these works. Contrary to those works, we propose a complete infrastructure, accompanied by a real-world deployment, as well as both a security and a performance evaluation of this deployment.

Weever et al. [20] investigate operational control requirements for zero-trust network security, and then implement zero-trust security in a microservice environment to protect and regulate traffic between microservices. They focus on implementing deep visibility in the service mesh, and do not propose a security or a performance evaluation. Hussain et al. [21] propose and implement a security framework for the creation of a secure API service mesh using Istio and Kubernetes. They then use a machine learning based model to automatically associate new APIs to already existing categories of service mesh. Contrary to our work, they use a central enterprise authorization server, in opposition to our policy sidecars. Zaheer et al. [22] propose eZTrust, a policy-driven perimeterization access control system for containerized microservices environments. They leverage eBPF to apply per-packet tagging depending on the security context, and then use those tags to enforce policy, in opposition to our enforcement of policy which relies on policy sidecars local to the services.

On the side of formal analysis of data leaks in workflows, Accorsi and Wonnemann [23] proposed a framework for the automated detection of leaks based on static flow analysis by transforming workflows into Petri nets. Some papers propose data leak protection, by screening data and comparing fingerprints [24]–[30]. Segarra et al. [31] propose an architecture to securely stream medical data using Trusted Execution Environments, while Zuo et al. investigate data leakage in mobile applications interaction with the cloud [32].

## VI. Conclusion

In this work, we described and implemented a secure architecture that prevents data leaks and protects business intelligence in the cloud. More specifically, in accordance with the principles of zero-trust, we achieve a secure system that enables the exchange of data between non-trusted agents while guaranteeing this data is secure at rest, in transport and cannot be leaked by any agent in both cases. The workflow is defined by the owner and enforced using policy sidecars, which controls the agents participating in the workflow. 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 experiments finally show that our approach scales well with increasing workflow complexity.

In the future, we plan to study how changes in the workflow impact the security of the system. Some work could also be done on the removal of trust requirements, by adding Trusted Execution Environments to our infrastructure, or on how to handle frequent user interactions with the system. This would provide us with a fully secure environment, so that even an actor with administration rights on the machine cannot peek at the data. Even though the data in our infrastructure is encrypted at rest, this would also give us another guarantee that the data of the owner and the business intelligence of the contractors are secure for any processing task.

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

## REFERENCES


[1] Nick Galov, "Cloud Adoption Statistics for 2020," Jan. 2020, [Online; accessed 28-February-2020]. [Online]. Available: https://hostingtribunal.com/blog/cloud-adoption-statistics/

[2] S. Byers, L. Cranor, D. Korman, P. McDaniel, and E. Cronin, "Analysis of security vulnerabilities in the movie production and distribution process," in *Proceedings of the 3rd ACM workshop on Digital rights management*. ACM, 2003, pp. 1–12.

[3] P. R. Clearinghouse. (2020) Data breaches. [Online]. Available: https://privacyrights.org/data-breaches

[4] R. B. Security. (2019) Data breach quickview report 2019 q3 trends. [Online]. Available: https://pages.riskbasedsecurity.com/hubfs/Reports/2019/Data\%20Breach\%20QuickView\%20Report\%202019\%20Q3\%20Trends.pdf

[5] J. Stempel and J. Finkle. (2017) Yahoo says all three billion accounts hacked in 2013 data theft. [Online]. Available: https://www.reuters.com/article/us-yahoo-cyber/yahoo-says-all-three-billion-accounts-hacked-in-2013-data-theft-idUSKCN1C82O1

[6] T. Seals. (2018) Thousands of mikrotik routers hijacked for eavesdropping. [Online]. Available: https://threatpost.com/thousands-of-mikrotik-routers-hijacked-for-eavesdropping/137165/

[7] KrebsonSecurity. (2019) First american financial corp. leaked hundreds of millions of title insurance records. [Online]. Available: https://krebsonsecurity.com/2019/05/first-american-financial-corp-leaked-hundreds-of-millions-of-title-insurance-records/

[8] C. Lecher. (2019) Google reportedly fires staffer in media leak crackdown. [Online]. Available: https://www.theverge.com/2019/11/12/20962028/google-staff-firing-media-leak-suspension-employee-termination

[9] R. Chandramouli and Z. Butcher, "Building secure microservices-based applications using service-mesh architecture," National Institute of Standards and Technology, Tech. Rep., 2020.

[10] E. Gilman and D. Barth, *Zero Trust Networks*. O'Reilly Media, Incorporated, 2017.

[11] Docker. (2019) Docker. [Online]. Available: https://www.docker.com/

[12] Kubernetes. (2020) Kubernetes. [Online]. Available: https://kubernetes.io/

[13] Istio. (2020) Istio. [Online]. Available: https://istio.io/

[14] Envoy. (2020) Envoy. [Online]. Available: https://www.envoyproxy.io/

[15] Open Policy Agent. (2020) Open policy agent. [Online]. Available: https://www.openpolicyagent.org/

[16] R. Chandramouli, "Security strategies for microservices-based application systems," Tech. Rep., 2019.

[17] A. El Malki and U. Zdun, "Guiding architectural decision making on service mesh based microservice architectures," in *European Conference on Software Architecture*. Springer, 2019, pp. 3–19.

[18] M. Souppaya, J. Morello, and K. Scarfone, "Application container security guide (2nd draft)," National Institute of Standards and Technology, Tech. Rep., 2017.

[19] R. Chandramouli and R. Chandramouli, *Security assurance requirements for linux application container deployments*. US Department of Commerce, National Institute of Standards and Technology, 2017.

[20] C. de Weever and M. Andreou, "Zero trust network security model in containerized environments," 2020.

[21] F. Hussain, W. Li, B. Noye, S. Sharieh, and A. Ferworn, "Intelligent service mesh framework for api security and management," in *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. IEEE, 2019, pp. 0735–0742.

[22] Z. Zaheer, H. Chang, S. Mukherjee, and J. Van der Merwe, "eztrust: Network-independent zero-trust perimeterization for microservices," in *Proceedings of the 2019 ACM Symposium on SDN Research*, 2019, pp. 49–61.

[23] R. Accorsi and C. Wonnemann, "Strong non-leak guarantees for workflow models," in *Proceedings of the 2011 ACM Symposium on Applied Computing*, 2011, pp. 308–314.

[24] X. Shu and D. D. Yao, "Data leak detection as a service," in *International Conference on Security and Privacy in Communication Systems*. Springer, 2012, pp. 222–240.

[25] M. Farhatullah, "Alp: An authentication and leak prediction model for cloud computing privacy," in *2013 3rd IEEE International Advance Computing Conference (IACC)*. IEEE, 2013, pp. 48–51.

[26] X. Shu, D. Yao, and E. Bertino, "Privacy-preserving detection of sensitive data exposure," *IEEE transactions on Information Forensics and Security*, vol. 10, no. 5, pp. 1092–1103, 2015.

[27] F. Liu, X. Shu, D. Yao, and A. R. Butt, "Privacy-preserving scanning of big content for sensitive data exposure with mapreduce," in *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, 2015, pp. 195–206.

[28] X. Shu, J. Zhang, D. D. Yao, and W.-c. Feng, "Fast detection of transformed data leaks," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 3, pp. 528–542, 2015.

[29] X. Shu, J. Zhang, D. Yao, and W.-c. Feng, "Rapid screening of transformed data leaks with efficient algorithms and parallel computing," in *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, 2015, pp. 147–149.

[30] T. LeVasseur and P. Richard, "Data leak protection system and processing methods thereof," Sep. 5 2017, uS Patent 9,754,217.

[31] C. Segarra, R. Delgado-Gonzalo, M. Lemay, P.-L. Aublin, P. Pietzuch, and V. Schiavoni, "Using trusted execution environments for secure stream processing of medical data," in *IFIP International Conference on Distributed Applications and Interoperable Systems*. Springer, 2019, pp. 91–107.

[32] C. Zuo, Z. Lin, and Y. Zhang, "Why does your data leak? uncovering the data leakage in cloud from mobile apps," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 1296–1310.