Discovery of Cloud Native applications from an application security perspective

June 21, 2023

Share

Context of the blog post

This article is based on my understanding of what is a Cloud Native application. This knowledge is mainly based on the reading of the following sources:

- Book: "Cloud Native: Using Containers, Functions, and Data to Build Next-Generation Applications".
- Book: "Understanding Kubernetes in a visual way: Learn and discover Kubernetes in sketchnotes"
- Regular research on the topic from Abhay Bhargav and folks from his company WE45.

🎯 My goal was to try to identify which aspects of the security of an application change when an application is intended to be Cloud Native.

📍 From here, a Cloud Native Application will be called a CNA.

Disclaimer

🐜 As always for me, it is possible that some of my understanding/idea/hypothesis/insights were partially or totally wrong.

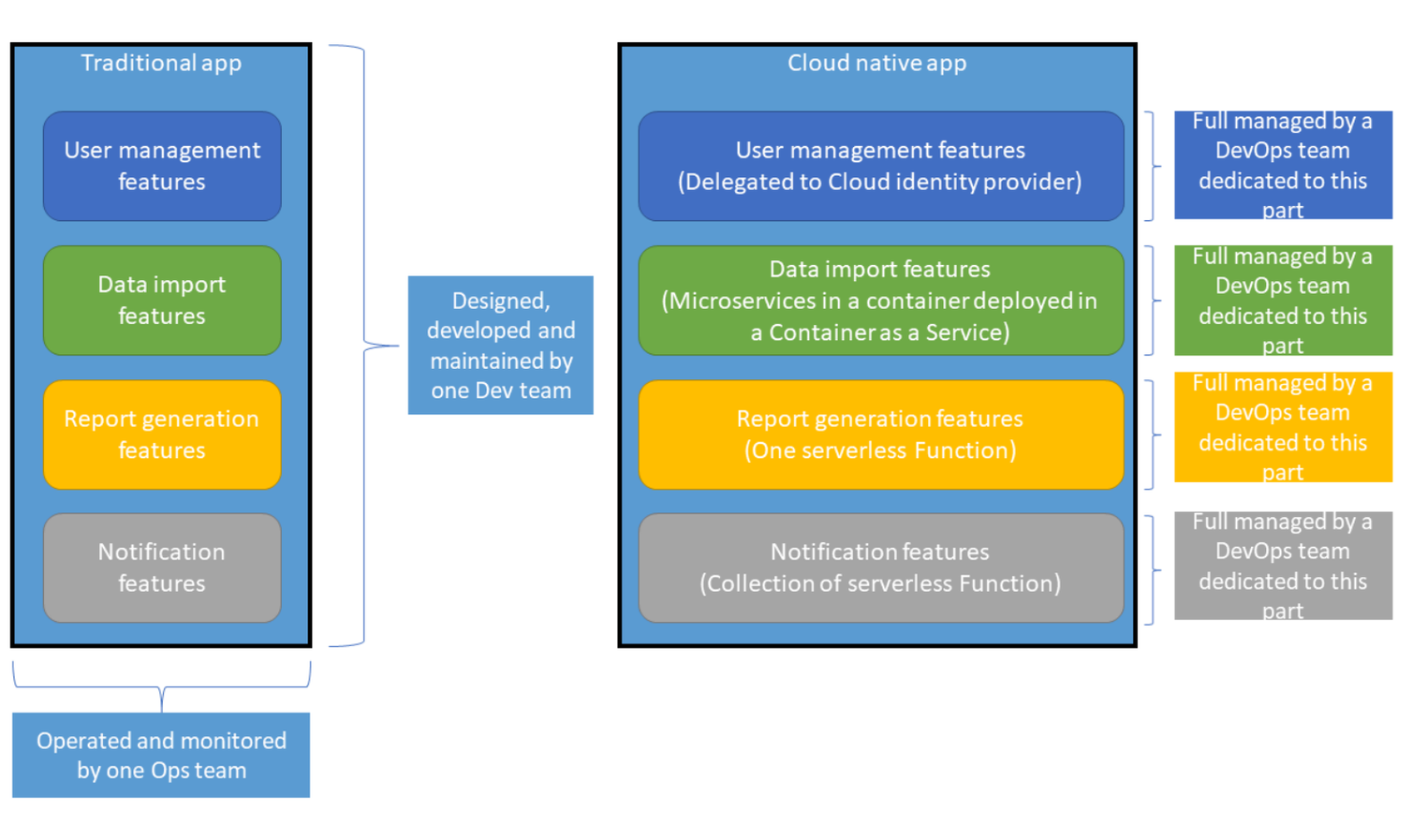
🤝 If this is the case, then, feel free to reach out to me to discuss together allowing me to enhance my understanding, fix my mistake and teach the right information.

What is considered a Cloud Native application?

📋 An application, intended to be a CNA, will have the following characteristics:

- Leverage the maximum of services provided by a Cloud provider to focus on the added value aspect to the application.
- It is composed of several dedicated parts, each one leveraging the more effective Cloud feature according to its business/technical objective: A part can be a serverless Function or a collection of Microservices hosted in a container managed by a container orchestrator.
- It is a distributed system by design.
- It is stateless to allow easy horizontal scaling in and out.
- Its design and implementation anticipate the kind of issues that can occur in Cloud-based environment, like for example, short network disruption with built-in "retry" mechanisms.

🤔 The schema below shows the difference, in terms of topology, between a traditional application and a CNA:



Topology of an CNA

The challenge from an application security perspective?

The main characteristic of a CNA is that it is composed of **several independent parts**. Therefore, to have a globally secure CNA, **all parts** of the CNA require the same security level.

Same point regarding the business/technical event logging across all parts. Indeed, from a security perspective, it is important to be able to track a communication flow through all parts of the CNA. To achieve this goal, it is required that all parts log information in the same way (format, content, etc.), with the same granularity, and have unique correlation identifiers to identify an event.

😨 Here comes a big challenge!

When we talk about a secure application (CNA or traditional one), we refer to the fact that it was designed with security in mind according to its business/technical context. This includes the following aspects:

- Authentication.
- Authoirzation.
- Input data validation and output data encoding.
- Error handling and technical information disclosure prevention.
- Protection of data in transit and at rest.
- Secure credentials/secret handling and storage.
- Security logging and monitoring.
- Monitoring of third-party components used from a security perspective to prevent vulnerable ones.
- …

🌎 Source: OWASP Top 10 Proactive Controls

Why is it a challenge?

As the CNA parts will be developed and operated by different teams, we need to define standards, security controls, libraries, framework to ensure that the security in the aspect mentioned above are handled in the same rigorous way.

This, even if all parts are using different web technologies (or even Cloud providers) and the different teams do not have the same maturity and knowledge in terms of application security as well as the development velocity/timeline.

In theory, its sound possible but unfortunately, we (like you dear reader) know what the reality in software development industry is, especially when it comes to its security aspect 😉

Therefore, let's be honest, pragmatic, and humble. As the focus of this article is modern applications topology using modern technologies, let's see how such "modern" material can help us in our challenge.

As CNA were "born" to allow company to create and deliver software quicker to follow their clients' needs (and before their competitor). Let's use this as foundation (statement learned from the book).

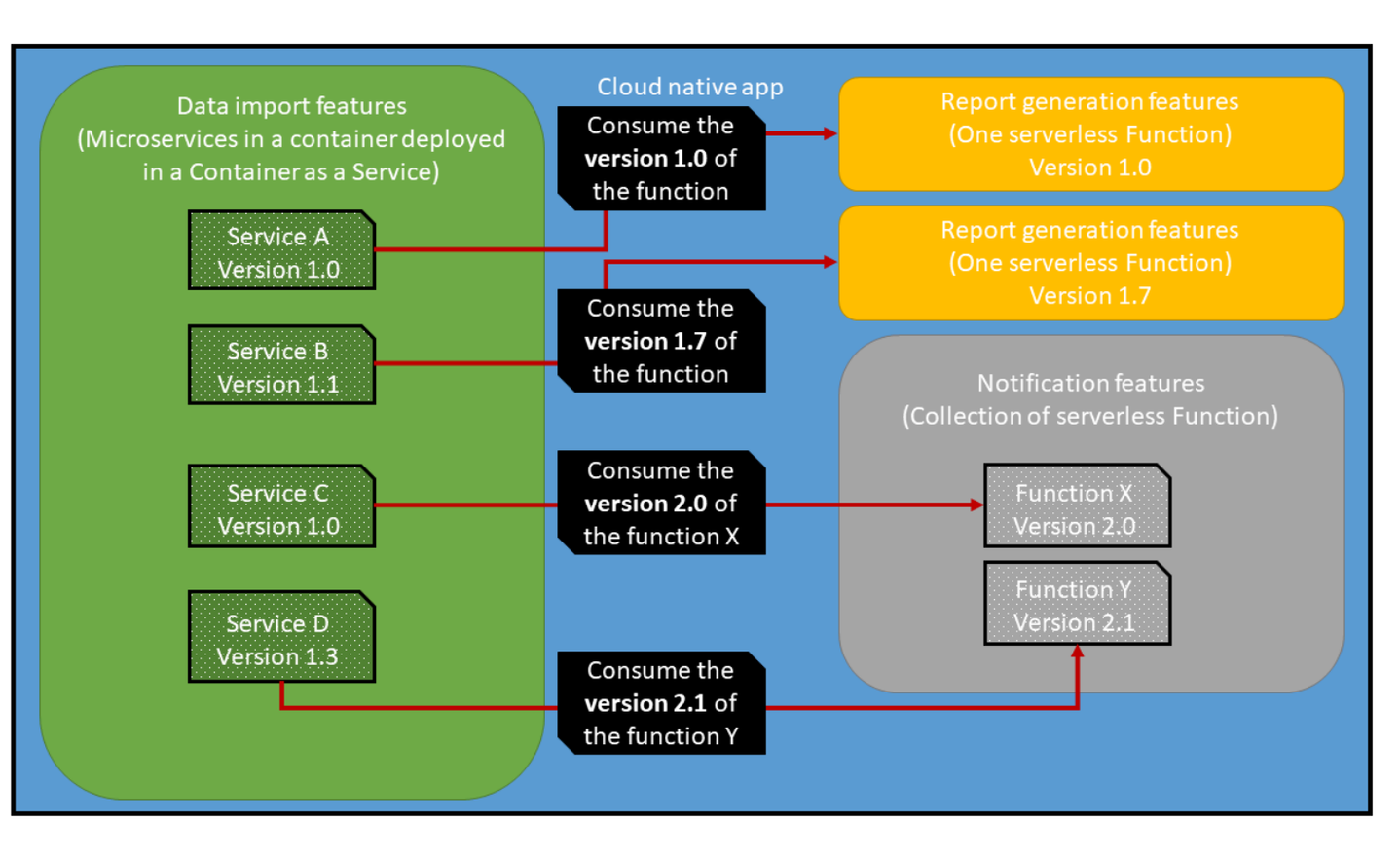
Nowadays, most of the development team already uses Agile methodologies combined with DevOps approach/mindset to create and deliver effective software in a high veloce way. Therefore, the objective is to avoid slowing them down when we add "security" related elements.

Note about the importance of the cartography in a CNA

Because CNA is composed of several parts, each collection of parts will evolve in a different timeline or velocity. The direct implication is the need to manage different versions of a part to ensure a functional whole CNA.

Thus, developing a CNA will imply that a well-planned management and cartography of the dependencies between all parts in place.

The schema below represents the idea:



Example of cartography of an CNA

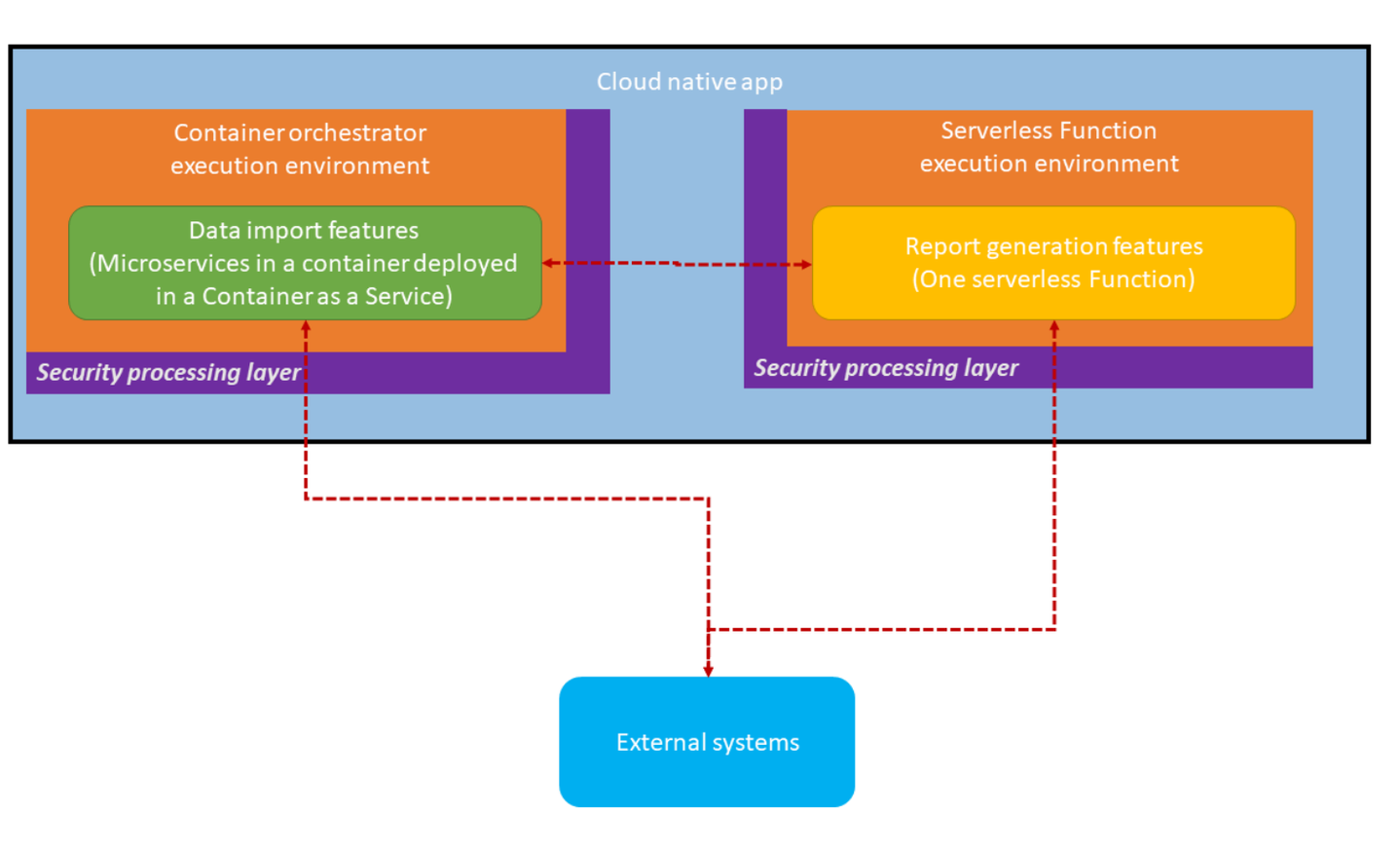
Leverage the common layer when it is possible

A common property of all parts of a CNA is the following:

- Almost all microservices will run in a container managed by a container orchestrator.
- Almost all Functions will run in a context in which processing can be applied to in/out flow.

From this observation, the idea here is to leverage the execution environment, when it is possible, to add security aspect instead of asking a DevOps team to add them at part level itself (via the code or the configuration of the part). Using this way, the DevOps team can continue to focus on the business purpose of their part of the CNA and reduce as much as possible security-related additional works/tasks.

The schema below represents the idea:



Usage of the common layer

💡 Modern container service mesh software (like Istio) and modern web application framework (like Spring Boot, ASP.Net Core, etc.) have built-in security features. Therefore, the proposed following distribution of the role can be leveraged to use a maximum of built-in security feature offered by modern technology stack:

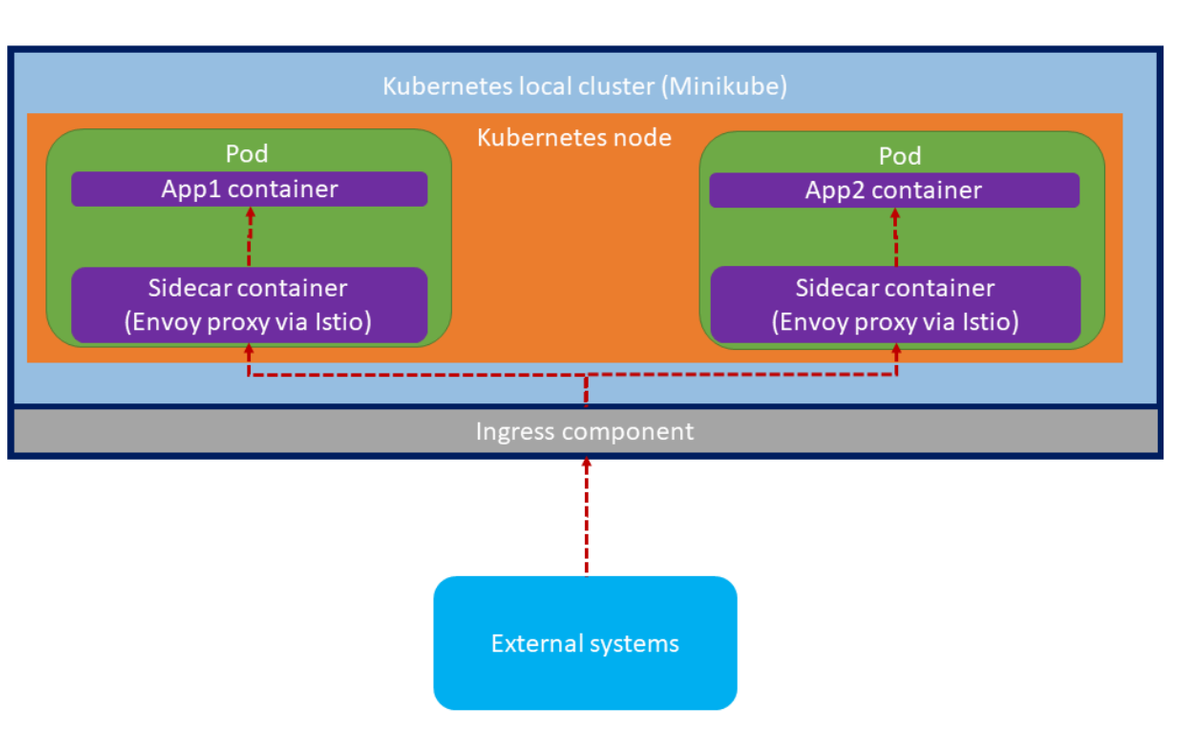
🤔 Legend:

- 🏭 Container orchestrator level.
- 📦 Service mesh level.
- 💻 Application level (code or CI/CD pipeline).

Area	Handled at which level?
Authentication.	📦
Authorization.	📦 When authorization is at service level only 📦 + 💻 When authorization is at data level
Input data validation and output data encoding.	💻
Error handling and technical information disclosure prevention.	💻 For app part 📦 For service mesh related part
Protection of data in transit and at rest.	💻 For data at rest 📦 For data in transit
Secure credentials/secret handling and storage.	🏭
Security logging and monitoring.	📦
Monitoring of third-party components used from a security perspective to prevent using vulnerable one.	💻In CI pipeline

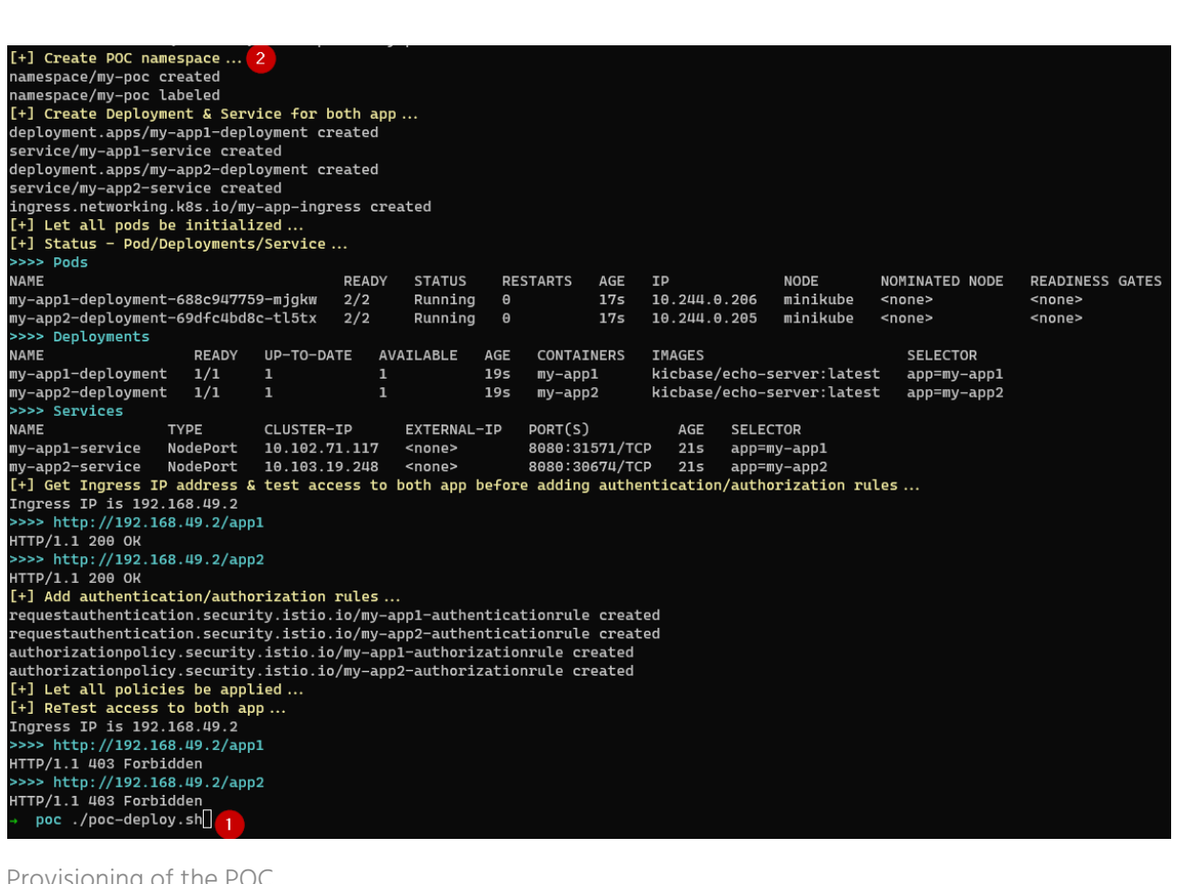
Example of leveraging the service mesh security features

To validate the elements proposed from a technical/pragmatic perspective, I put together the following proof of concept (POC):



Objective of the POC

🏭 This script was used to deploy the POC components:



Provisioning of the POC

Steps in *red circles* in the image above:

1. Command line used to run the script.
2. Execution traces.

🎯 The objective of the POC was to achieve the following security aspect, only using the Kubernetes or Istio security features, nothing implemented at application level:

- Authentication via a token JWT.
- Authorization via the claims of the token JWT.

📦 This YAML file was describing the Kubernetes deployment plan for both applications.

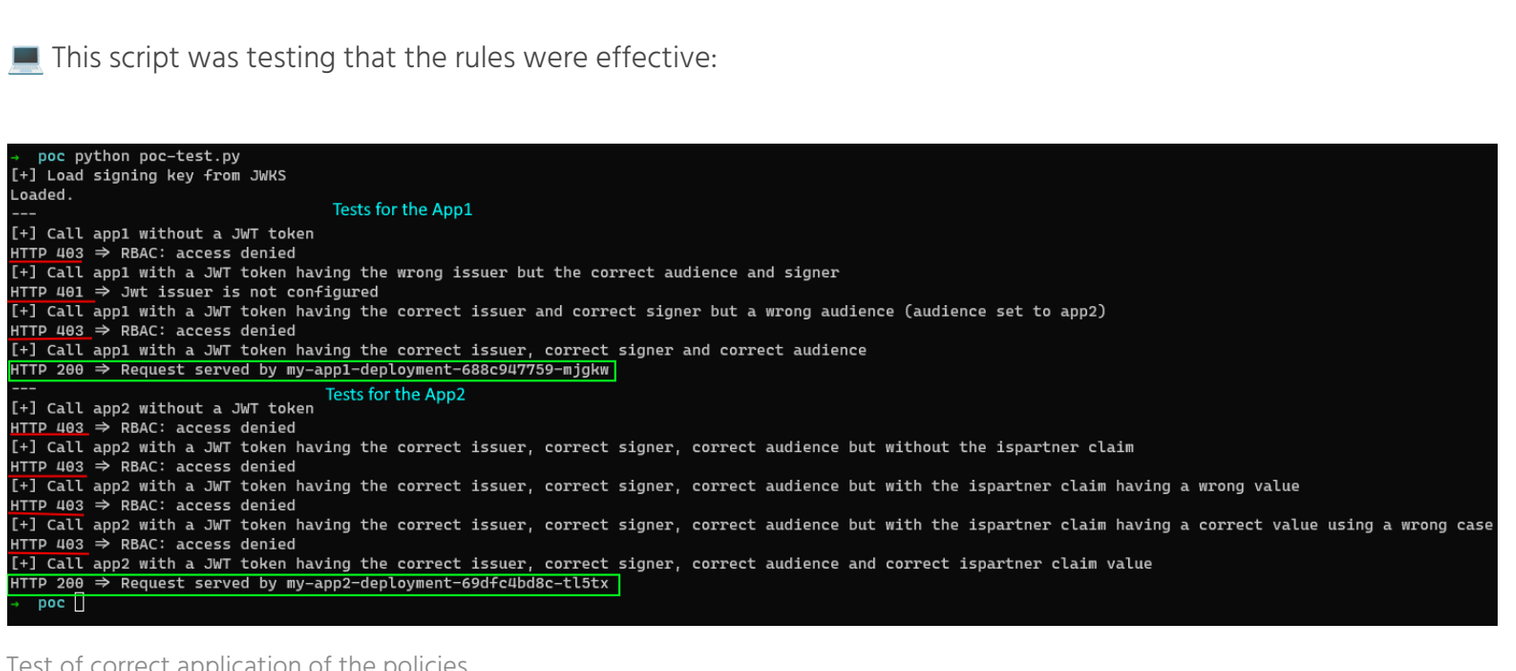
POC definition file

🔐 This YAML file was describing the Istio deployment plan for the following authentication and authorization rules:

Security provisioning

- Authentication (*RequestAuthentication entries*):
  - For both app, the JWT token provided must:
    - Been issued by "**excellium-ias**" issuer.
    - Been signed with the RSA private key associated to this public key.
  - Authorization (*AuthorizationPolicy entries*):
    - For App1:
      - A valid JWT token must be provided.
      - The **audience** claim of the token must be intended for **app1**.
    - For App2:
      - A valid JWT token must be provided.
      - The **audience** claim of the token must be intended for **app2**.
      - The custom claim named **ispartner** must be set to **Yes**.

💻 This script was testing that the rules were effective:



Test of correct application of the policies

Conclusion

Cloud Native Applications change the core structure of what is an "application" by exploding it in several parts. Each of them having its proper lifecycle, technology stack, team, and security maturity.

Traditional ways of including security in an application can still be applied. However, it will imply significantly more effort/time/cost resources. Therefore, it is important to leverage the new security features provided by the common layer to make the security level consistent and transparent for Dev and Ops teams.

💡 All materials of the POC are stored in this GitHub public repository and the POC can be fully reproduced.

Author

Dominique Righetto

Do you have any questions ? Would you like to know more about Cloud Native applications? Contact our experts !

Name *

First Last

Company *

Phone Number *

Email *

Comment or Message *

Submit

Cyber Blog Post | Application Security, Appsec, Cloud, Cloud Native, Cna, Cybersecurity, Excellium Services, Excellium Services Belgium, Excellium Services Luxembourg, IT Security, Native