About Single-Sign-On

Gabriele Biagini

April 2020

Contents

1 Overview 3

2 Policy 4

3 Mechanism 5

3.1 Service Registration 6

3.2 User Operations 7

3.2.1 Registration 7

3.2.2 Login 8

3.2.3 Claims Declaration 8

3.2.4 Claims Validation 8

3.3 Services Requests 9

4 Conclusions 10

1 Overview

The purpose of this project is to build a **Single-Sign-On platform** inspired by the **MILS** (Multiple Independent Levels of Security/Safety) security architecture, which is built upon the secrecy based **Bell-LaPadula model**, and **William Wulf**'s works.

Bell and LaPadula developed their model to enforce access control in government and military applications, introducing many important concepts like a hierarchical classification of Clearance and Sensitivity levels for Subjects and Objects, their contribution in the security field was truly valuable.

MILS is a "high assurance security architecture"[1] that supports the coexistence of non-trusted and trusted components, based on verifiable separation mechanisms and controlled information flow.

William Wulf, in his work about *HYDRA - The kernel of a multiprocessor operating system*, refers the need of a strong separation between protection and security when referring to access control systems, a concept that lacks in Bell-LaPadula principles.
In his paper he stated:

> "Protection is a Mechanism, security is a Policy.
>
> A system utilizing such a mechanism may be more or less secure depending upon policies governing the use of the mechanism and upon the reliability of the program which manipulate the protected entities. [...]
>
> A particular consequence of this philosophy is to discard the notion of "ownership". While ownership is useful concept for certain security strategies, to include a concept at the most primitive levels would lead to the exclusion of the construction of certain other classes of truly secure systems."[2]

Let us now introduce the policy and the mechanism.

2 Policy

The policy is a **Claim-Based Access Control** for Subjects and Objects abandoning the hierarchical Role-Based system:

- Subjects:
  - Have one or multiple *claims* that declare what the Subject is or is not, defining what *realm(s)* it can have access to;
  - Can register to the SSO independently and verify their claims by submitting valuable information that vouch for the declared claims;
  - Once the authentication process is performed, Subjects are given an **Access Token**, that expires after a pre-defined and configurable time, for submitting requests for Objects.
- Objects: are defined into a distinct *realm*, grouped by the nature of the resource.

A MILS system employs a set of properties, commonly acronym-ed as "**NEAT**"[3]:

- Non-bypassable: each component of the system must use a pre-defined and non-modifiable communication path/pattern;
- Evaluatable: any component must be evaluated to the level of assurance required of that component;
- Always-invoked: each and every access/message is checked by the appropriate security monitors;
- Tamper-proof: the system prevents unauthorized changes to the security monitor code, configuration and data.

The *NE* properties imply a strong *separation of duty* for each component of the SSO.

The *AT* properties imply the need of several *security-managers* with a defined *scope* that are constantly monitoring every request in their context and are being constantly monitored by all others security-managers to prevent hacking or tweaking.

3 Mechanism

MILS systems are in-line with current Cloud development patterns so the SSO revolves around 2 micro-services built from scratch, a Service Mesh Agent and a Secret Manager:

- **Subjects Registry**:
  - The **Services/Realms Registry** module maps the relationships between Services and Realms;
  - The **User Registry** module is responsible for Users registration requests storing the Subject's credentials and claims.
- **Gateway**: main entry-point that handles incoming requests for services verifying the validity of the Access Token and the necessary claims to access the service realm;
- **Service Mesh Agent**: creates a graph of the registered services for discovery resolution via explicit connection policy declarations and is responsible for encrypting the SSO-service communication;
- **Secret Manager**: the mesh configuration (coordinates, APIs and certificates) for registered services is stored in a vault.

The communication pattern of choice between the micro-services is gRPC and the isolation/bounded-context of each component makes it independently testable to guarantee the level of assurance required.

Several state-of-art architectural patterns are applied for reliability in distributed environments:

- *Service Discovery*: with Self-Registration and Service Registry;
- *API Gateway*: a service that provides each client with unified interface to services;
- *Health Checking*: the service mesh configuration is a representation of alive and dead services;
- *Circuit Breaking*: if the requested service is present in the service mesh configuration, a rollback will be performed;
- *Secure Service Communication*: TLS certificates are used for mutual authentication between the SSO and the requested service.

3.1 Service Registration

The following diagram describes the process:

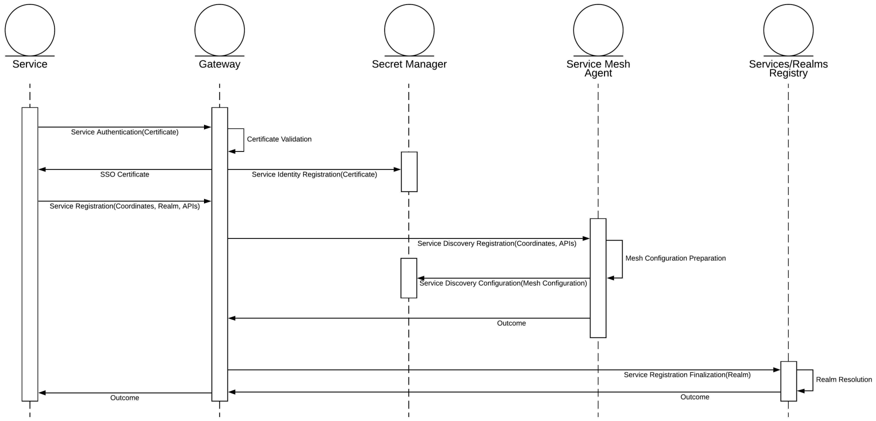
Service
Gateway
Secret Manager
Service Mesh Agent
Services/Realms Registry
Service Authentication(Certificate)
Certificate Validation
SSO Certificate
Service Identity Registration(Certificate)
Service Registration(Coordinates, Realm, APIs)
Service Discovery Registration(Coordinates, APIs)
Mesh Configuration Preparation
Service Discovery Configuration(Mesh Configuration)
Outcome
Service Registration Finalization(Realm)
Realm Resolution
Outcome
Outcome

Figure 1: Service Registration - Sequence Diagram

The service and SSO perform a mutual authentication by issuing each other's certificate; the SSO stores the service certificate in the vault and updates the service graph. The realm gets resolved via the Service Registry pattern.

3.2 User Operations

All interactions with the Users Registry are referenced in the subsequent diagram:

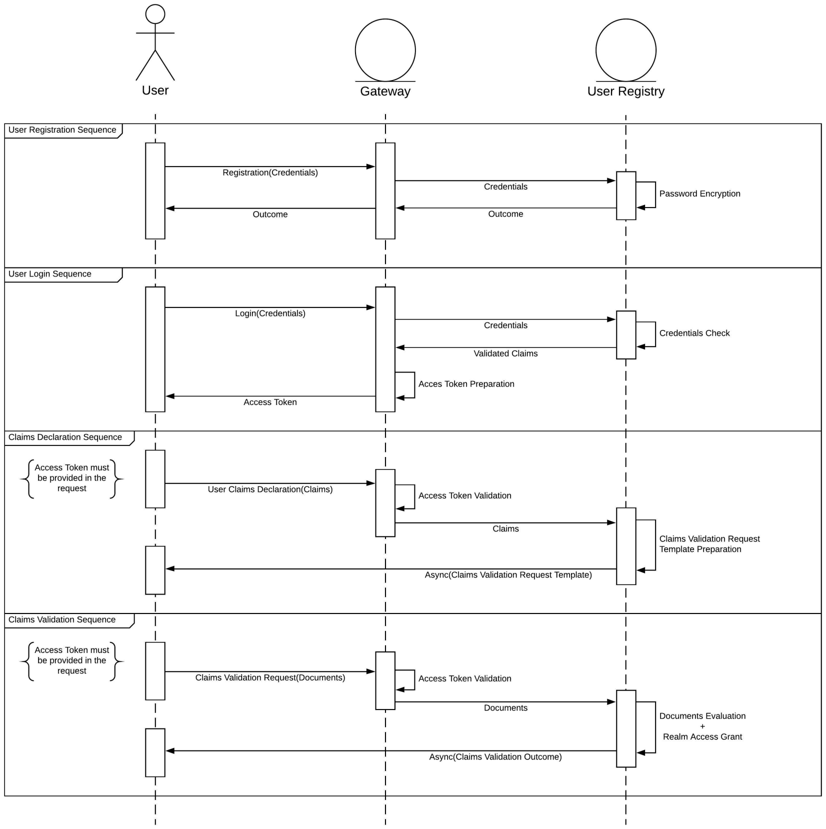
User
Gateway
User Registry
User Registration Sequence
Registration(Credentials)
Credentials
Password Encryption
Outcome
Outcome
User Login Sequence
Login(Credentials)
Credentials
Credentials Check
Validated Claims
Acces Token Preparation
Access Token
Claims Declaration Sequence
Access Token must be provided in the request
User Claims Declaration(Claims)
Access Token Validation
Claims
Claims Validation Request Template Preparation
Async(Claims Validation Request Template)
Claims Validation Sequence
Access Token must be provided in the request
Claims Validation Request(Documents)
Access Token Validation
Documents
Documents Evaluation + Realm Access Grant
Async(Claims Validation Outcome)

Figure 2: User Operations - Sequence Diagram

3.2.1 Registration

The password is hashed via an implementation of the PBKDF2 key-derivation function described in the “RFC-8018”[4].
The function takes the password as an input and proceeds to calculate the derivation function by using 5 declared parameters:

- Pseudo-Random Function: an efficiently computable function that emulates a *random oracle*, in this case is used HMAC-SHA1;

- Password: the password itself;
- Salt: randomly generated at every function call, with a length of 64 bits as per general recommendation;
- Number of iterations: in this case 65536 (which produces far more entropy than common web Password-Managers use, eg. LastPass makes 5000 iterations);
- Key length: in this case 128.

The output of the function are the **salt** and the **hashed password**.

3.2.2 Login

The Access Token for users is an “implementation”[5] of the JWT (JSON Web Token) described in the “RFC-7519”[6], which contains the user’s identity and a representation of it’s verified claims.

3.2.3 Claims Declaration

Registered users can independently submit their claims and will receive a Template for the documentation needed for verify them. Each realm will have its own Template which can be extended in many sub-Templates based on the authorities that can verify the claim.

3.2.4 Claims Validation

For certain realms the process of claims validation can be sometimes automated: the documentation to prove the user’s affinity with the realm, if defined by a ”contract”, can be easily evaluated.

Some fields make use of proprietary formats which can only be judged by members of the realms, but if we take into account the possibility to register new services, third verified parties can take care of this.

3.3 Services Requests

Users can interact with the registered services (if they have the necessary verified claims) via the SSO's Gateway that will take care of forwarding the request in a secured context:

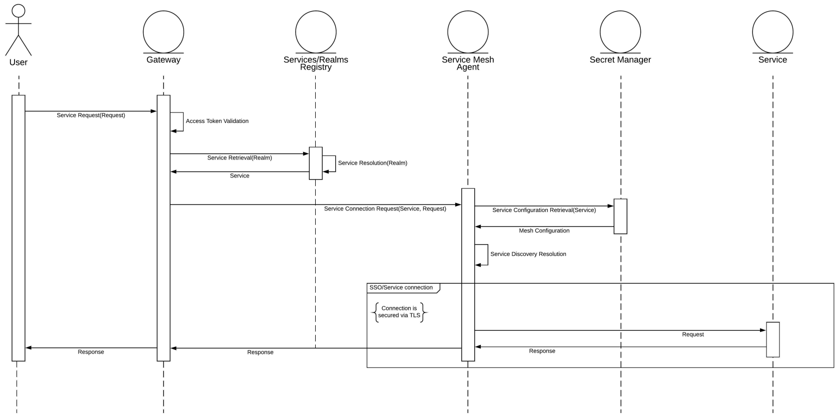

Figure 3: Service Request Forwarding - Sequence Diagram

4 Conclusions

The main difference between the solution and a traditional SSO like “Kerberos” [7] is about granting **service availability and connection responsibility** as core features.
Kerberos provides the User with the ticket needed to authenticate with a service, but the subsequent interactions with it are entirely delegated to the user.

In the presented SSO solution the responsibility is not delegated and takes care of the security of the communication channel with the service forwarding the user’s request.
In a distributed environment, if the registered services have interactions within their contexts besides the SSO flow-control and implement the Service Mesh Agent, the SSO can still be aware of what is going on.
The service graph will always be updated and the health-check monitor will catch eventual failures, circuit-breaking the requests and securing the users from attacks.

References

[1] J. Rushby. Design and verification of secure systems. www.csl.sri.com/papers/sosp81/sosp81.pdf, 1981.

[2] W. Wulf. HYDRA - The kernel of a multiprocessor operating system. *Communications of the ACM*, 1974.

[3] H. Blasu. EURO-MILS: Secure european virtualisation for trustworthy applications in critical domains. https://zenodo.org/record/45164, 2013.

[4] IETF. RFC-8018 - PKCS #5: Password-based cryptography specification version 2.1. tools.ietf.org/html/rfc8018, 2017.

[5] Auth0. jwt.io.

[6] IETF. RFC-7519 - JSON Web Token (JWT). tools.ietf.org/html/rfc7519, 2015.

[7] MIT. web.mit.edu/kerberos/.